Volume 10, Issue 2, February – 2025 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.5281/zenodo.14915594

Evaluating the Computational Efficiency and Precision of Pathfinding Algorithms

Atharv Patil[1]

[1]University of North Carolina at Charlotte

Publication Date: 2025/02/24

Abstract: AI pathfinding is essential for autonomous transport and many other industries, for example, bot movement inside video games, navigation, and logistics in mapping applications. This paper compares DFS (depth-first search), BFS (breadth-first search), Dijkstra, and A* (A-star) algorithms. A* uses adaptive training to locate the direction needed to move from the start point to the endpoint, and proceeds to move only in that direction. Dijkstra and BFS expand in all directions, updating the least recent node in the case of BFS, or at random with Dijkstra. Lastly, DFS has a set algorithm with a priority of directions that update on the most recent node. The objective of pathfinding is to help the algorithm navigate from point A to point B in a manner where it does not cross obstacles, but still in the fastest time. Results are still ongoing in this ever-changing field; however, in conclusion, A* has proved to be the most efficient at balancing speed with results. Results from the analysis prove that A* was the fastest at completing the computing and traversing while loading the fewest number of nodes. DFS took the longest and Dijkstra loaded the most nodes and took the longest to compute.

How to Cite: Atharv Patil. (2025). Evaluating the Computational Efficiency and Precision of Pathfinding Algorithms. *International Journal of Innovative Science and Research Technology*, 10(2), 360-367. https://doi.org/10.5281/zenodo.14915594.

I. INTRODUCTION

It is fascinating to consider the methods with which autonomous vehicles navigate the same complexity that a human may operate with. AI pathfinding involves getting from one point to another and using the shortest route to get there to a location while avoiding programmed hazards and obstacles. A model is a process that is trained using data to run a task. Models are promoted to perform pathfinding by reward-based systems that give tokens to incentive behavior that matches what the user desires[1]. Models are also based on algorithms that are the base for dictating what to do. In the case of A*, for example, the model is trained using parameters that are given by the algorithm. Programs such as DFS, BFS, Dijkstra, and A* all help onboard systems traverse mazes or new maps using step-by-step algorithms. Nodes are one point used in a pathfinding map that serves as a base location an algorithm can check. Nodes are core elements for exploring new terrain with autonomous vehicles and viewing their surroundings since they rely on nodes to plot points and designate a value for them for models to apply[3]. They are also essential in things like delivery, video game[14] bot movement, shipping, and logistics. It is also a prevalent part of prominent fields including robotics such as drone routes[15]. How do different pathfinding algorithms compare in speed of computing and pathfinding efficiency? Pathfinding is important as it improves the efficiency of users of its products such as robot taxis[8]. Making it efficient not only saves time but also resources, freeing up more assets to do other work and also saving on fuel or battery charge in the case of autonomous cars[2]. Having fast computing also is important with an algorithm, as it will take much less processing power when scaled up, and will work faster for much larger processes[6]. Make faster decisions and will use knowledge from their surroundings to look ahead and see variables such as the traffic around the car and make the decision that would save the most time while still being safe and making that call faster than a human driver. Pathfinding can also improve the accuracy of GPS systems that have already been established by updating and improving upon previously recorded data for example monitoring traffic levels for future reference daily by using sensors. We created a row-first search algorithm that linearly checks cells in a rectangular shape. Additionally, in the empirical section of this paper, we are comparing the speed of pathfinding a particular route is traversed with the percentage of nodes checked and the speed at which they are computed. There will be 3 sets of trials in the empirical analysis that determine the efficiency of each program on a randomly generated maze, using differing amounts of traffic.

II. PATHFINDING MODELS

History of previous key pathfinding models. Several pathfinding models have been used to complete tasks throughout history. These first started for basic logistics, but have adapted to work with the world we know today. This is important as it provides context for how each of these programs works and the importance of where they should be used.

ISSN No:-2456-2165 https://doi.org/10.5281/zenodo.14915594

BFS (breadth first search) was first explored in 1959 by E.F. Moore, as it found the shortest path inside a maze. BFS[9] is one of the oldest algorithms that use fundamental principles. BFS searches for the nodes bordering those that have already been searched with a function of $V+E$, V demonstrated with the number of bordering vertices, and E being the edges between those vertices. Since in an open area, most of the exposed vertices are equivalent, it is still commonly used as the basis for tree structures which will then select a selection of the greater BFS area to search for the fastest path.

DFS (depth-first search) was originally developed in the 19th century by the French and was later implemented in 1957 by Polish computer scientist Kazimierz Zarankiewicz, who used it to search graphs. It always expands the most recent nodes in the order of, left, down, right, and top. This is due to the directions being the reverse of the order they were added in. If not able to expand in one direction, DFS looks at the next one in the order until the solution is, filled[17]. DFS does not always return the fastest result and typically will have to check a large, wide range if the point is not to the left of it. "It is assumed that a Depth First search starts at A, the left edges in the given graph are chosen before right edges, and the search remembers previously-visited nodes, and it does not repeat them" (2, Kaur). Since edges and vertices are searched in one path, DFS's time to compute can be simplified to $E+V$, where E is equivalent to the number of edges and V, the number of vertices.

Dijkstra's algorithm was initially discovered by the Dutch computer scientist, Edsger Dijkstra, in 1959. It assigns a value to each node at random as well as a value for the connection to a new node which dictates which path will be selected. This requires viewing fewer areas than BFS and DFS in most areas. Dijkstra's time to compute can be simplified to $E \log(V)$, where E is the number of edges and V is the number of vertices. It branches out from a point of origin in all neighboring directions at the same pace, determined by the number of nodes. It is used to find the shortest path for many technology hardware components such as routers and sync connections between them.

A* (A—star) was created during the Shakey project[10] — a project that aimed to create a mobile robot that could navigate its surroundings. Essentially, A* is the same as Dijkstra in the way that it expands its search in all paths; however, it is different regarding how it rewards the movement of certain pathways. A* lengthens paths going away from the endpoint and lowers those going towards it (up until the point that no length is equivalent to 0 from any node) which helps the program find relevant pathways toward the end faster. This can be represented with the function $f(n) = g(n) + h(n)$ where n is the next node g is the length of the node from the previous point (also known as the cost), h is the preferred reward that incentives paths going towards the start point. In conclusion, all of these show a varying degree of use and function as pathfinding models. This is shown by each having their technique and principle catered to a specific category.

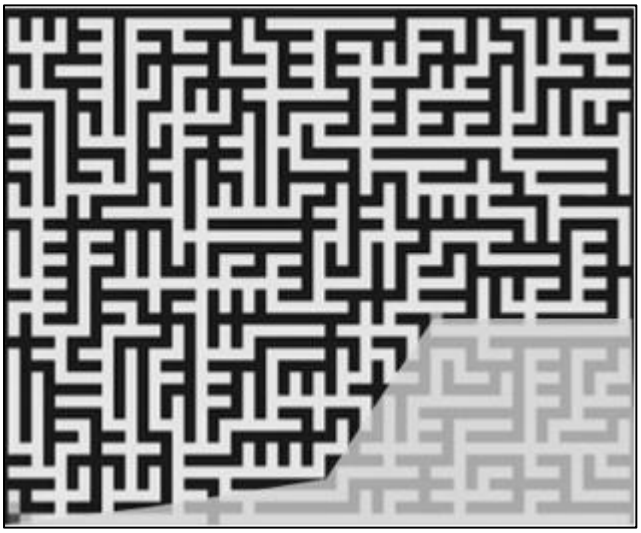

Fig 1: DFS Node Search Mapped on Grid

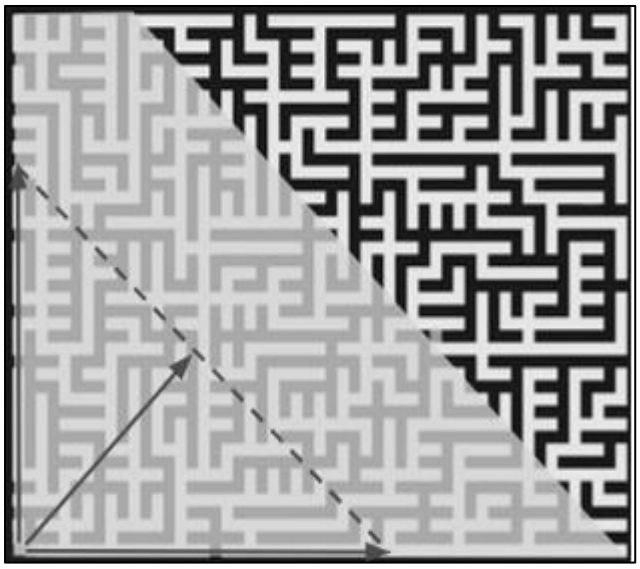

Fig 2: Dijkstra node search mapped on grid

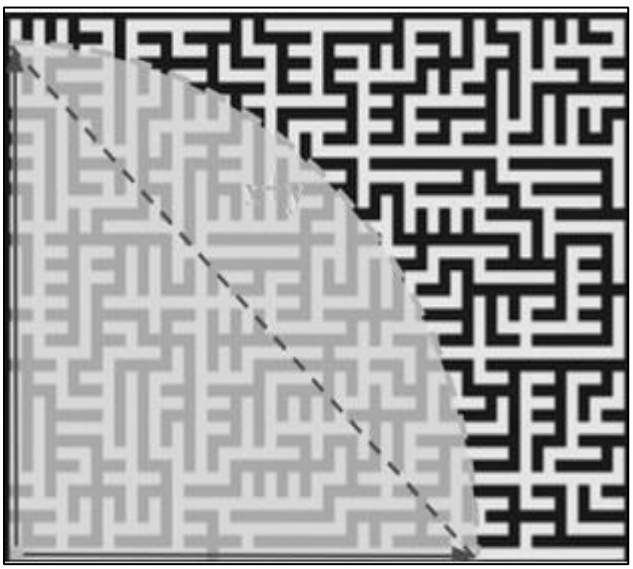

Fig 3: BFS Node Search Mapped on Grid

ISSN No:-2456-2165 https://doi.org/10.5281/zenodo.14915594

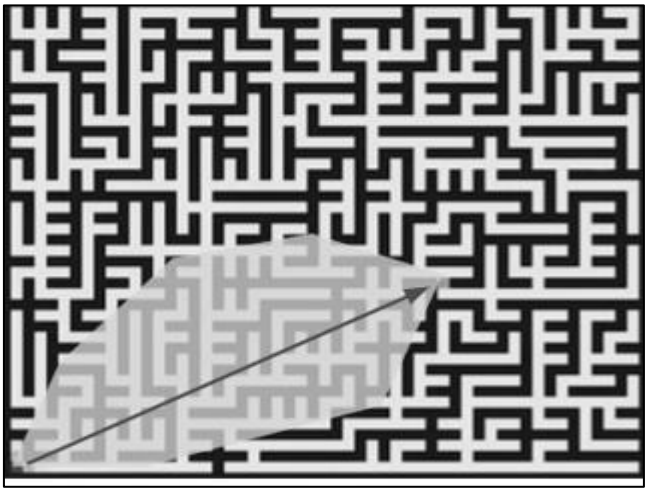

Fig 4: A* Node Search Mapped on Grid

III. APPLICATIONS

Different pathfinding algorithms developed over time have varying uses. These can include from software to raw hardware purposes.

BFS is an algorithm that serves many uses, especially for programs that require a detailed search of the targeted area. This is because it searches a larger radius than something like A* which only searches towards the end destination. It is also used in web crawlers which are models of websites laid out by web pages that the program can then sort through with pathfinding to find the easiest way to navigate within two points[9]; this is used commonly in social media algorithms to find the distance. BFS is also used to traverse unweighted graphs and map data in the same way. It is also used for routing to create a redundant system, as many outlets are identified. It is also used for basic video game pathfinding. BFS is a varied pathfinding algorithm.

DFS pathfinding[7] is used to search all points and find a route with minimal computing. It is used in mazes and topography, mapping altitudes as well. It has been used in areas where a whole plane needs to be filled such as planting crops. Moreover, DFS is also capable of solving sudoku and network topography, similar to BFS routing. Additionally, it can use topography to sort and determine various regions. It is also used to traverse tree models and section them[13]. This is used in forests in some cases to monitor health and reduce the risk of fires by identifying dry or dead trees. DFS has a use for finding redundant algorithms, and for identifying areas with certain characteristics within a provided sample.

Dijkstra is used for many applications in pathfinding of all directions. It is used in GPS navigation for paths between maps. Furthermore, it is used for network routing to search in all directions and similarly in traffic management. Dijkstra is also very commonly used in game pathfinding[4], as its circular search radius is important to locate where to go. This is effective as the simple pathfinding method requires fewer checks for node contact for larger distances, as the circular checking radius expands in all directions with distance. Dijkstra is an algorithm that is widely used to search for all directions. This application is specifically useful for monitoring distances between pre-disclosed routes such as with public roads[18], railways[12], or flight paths to find the fastest distance between points. “Bachri et al. used the Dijkstra algorithm and node combination to find the shortest path in Geographical Information Systems where the result is with node combination and Dijkstra algorithm was succeeded in finding the optimal route in the case study route in Taman Sub-district, Sidoarjo Regency, East Java, Indonesia” (51, Pardede)

A* is an application that has an increasing amount of uses in the present day. It is used in many programs such as mapping apps now to save space that is checked and factor in traffic. It is also considered in space exploration to calculate the orbit of planets during flight paths. A* is also used in many everyday uses and up-and-coming technology[11], being the standard for new autonomous vehicles and robots. A* Manhattan and A* Euclidean are two different popular algorithms of A* for specified tasks. A* Manhattan is a heuristic based on the sum of absolute differences of coordinates making it best suited for grids and city layout pathfinding. Conversely, A* Euclidean is a heuristic based on straight line distances between two points used in cases for 2d and 3d open environments, which allow diagonal movement. This indicates A* is a newer pathfinding algorithm with expanding possibilities.

IV. ROW FIRST SEARCH

RFS (row first search) is a linear pathfinding algorithm that we developed. It differs from other algorithms in terms of its linear search across rows over the whole select area even after locating the endpoint, for a thorough analysis of the possible paths. The program searches in rows across in this direction of the endpoint until it passes the x-axis point of the endpoint. Then, it moves either up or down in the direction of the endpoint until passing the y-axis of the endpoint. The results of these scanned pixels are then computed to find the route, similar to BFS. Concurrently, it is only able to analyze paths that have a straightforward approach toward the endpoint, not ones that go past it as those nodes are not scanned. RFS could be useful in scenarios with set rectangle-like shapes where every node can quickly be checked, such as in crop fields.

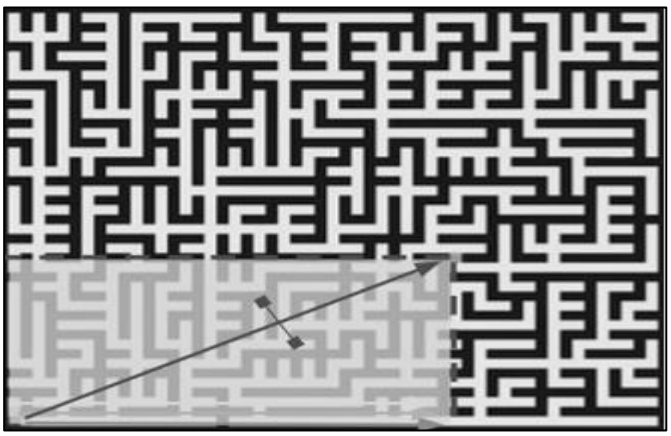

Fig 5: RFS node search mapped on grid

ISSN No:-2456-2165 https://doi.org/10.5281/zenodo.14915594

In the future, RFS could be expanded on by adding a radius around the starting and ending points. Any path going behind these coordinates would be checked and therefore may not be solvable with the current model that was run. This radius could adapt depending on the length between the set points. Additionally, more focus could be put on prioritizing nodes toward the point, in an algorithm combining more through linear search with an adaptive algorithm such as A, allowing fewer nodes to be checked, which could enable RFS used in various applications.

V. EMPIRICAL ANALYSIS

Table 1 consists of 5 runs per trial, each with different conditions[5]. The conditions for the trial involve a trial area where the user begins starting at the bottom left and making their way to the top right. The user can move in all four directions as well as diagonally to the top right, top left, bottom right, and bottom left. Trial 1 consists of only barrier blocks that inhibit the path of movement through them. Trial 2 has blocks that slow down the path of the user to a speed of 50% normal when they are in contact. Trial 3 has on top of blocks that slow down the user by 55% also blocks that reduce the speed of movement by 75% when in contact. All the blocks are procedurally generated using noise rounded to a 0 or 1 value to around ¼ of the total pixel area each. The nodes modifier is trained to be punished for going onto slower tiled squares.

Additionally, all trials were run on the same computer for computing consistency. The setup of noise allows for random variation, allowing for fair results across all algorithms. This allows the trials to mimic a random 2d grid of obstacles and varying traffic. This is applicable to navigation within a large city, where some areas can not be traversed such as buildings, and others such as intersections, have high traffic slowing down the time of crossing between points. The results depicted below were performed by each algorithm on a grid.

Table 1: Results of Empirical Analysis

	Computing Time (sec)				Traversing Time (sec)				Scanned Nodes (%)				Nodes Checked (#)			
	T1	**T2**	**T3**	**Avg**	**T1**	**T2**	**T3**	**Avg**	**T1**	**T2**	**T3**	**Avg**	**T1**	**T2**	**T3**	**Avg**
A* Man.	0.022	0.026	0.028	0.025	5.62	6.25	8.76	6.88	4.13	4.35	4.61	4.36	536	649	681	622
A* Ecu.	0.024	0.027	0.028	0.026	6.23	7.34	9.18	7.58	5.34	6.08	6.49	5.97	842	972	1045	953
Dijkstra	0.127	0.130	0.129	0.129	8.56	9.27	10.52	9.45	76.13	76.83	76.59	76.52	11486	11588	11490	11521
RFS	0.062	0.063	0.063	0.063	7.43	8.18	9.64	8.42	33.88	34.97	34.62	34.49	5562	5682	5624	5623
BFS	0.042	0.041	0.042	0.042	7.74	8.26	9.82	8.61	76.19	74.26	76.23	75.56	11262	10865	11306	11144
DFS	0.027	0.027	0.027	0.027	126.52	152.78	173.85	151.05	30.26	31.05	30.85	30.72	4613	4686	4628	4642

- Average results rounded to the thousandths for computing
- Average results rounded to hundredths for traversing
- Average results rounded to the hundredths of a percent for scanned nodes
- Average results rounded to the nearest node

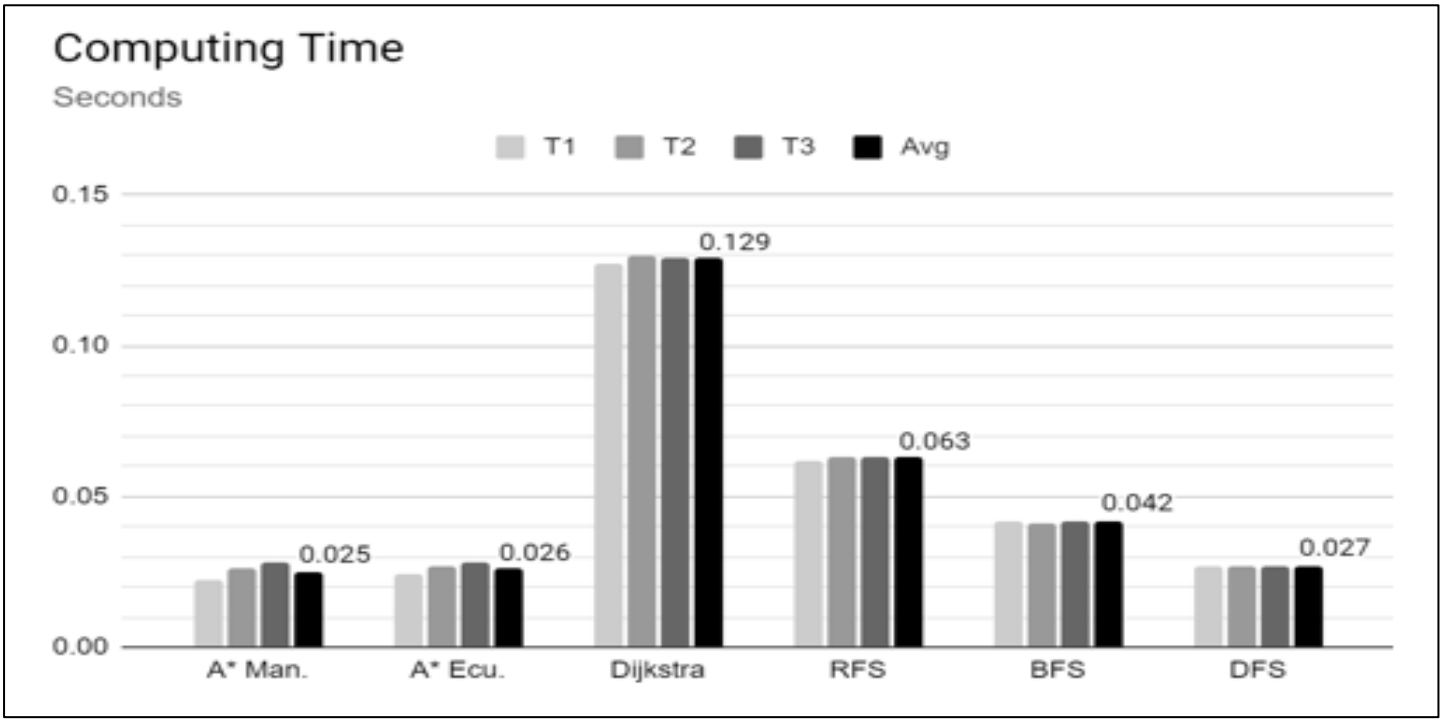

Fig 6: Computing Time of Algorithms (Sec)

ISSN No:-2456-2165 https://doi.org/10.5281/zenodo.14915594

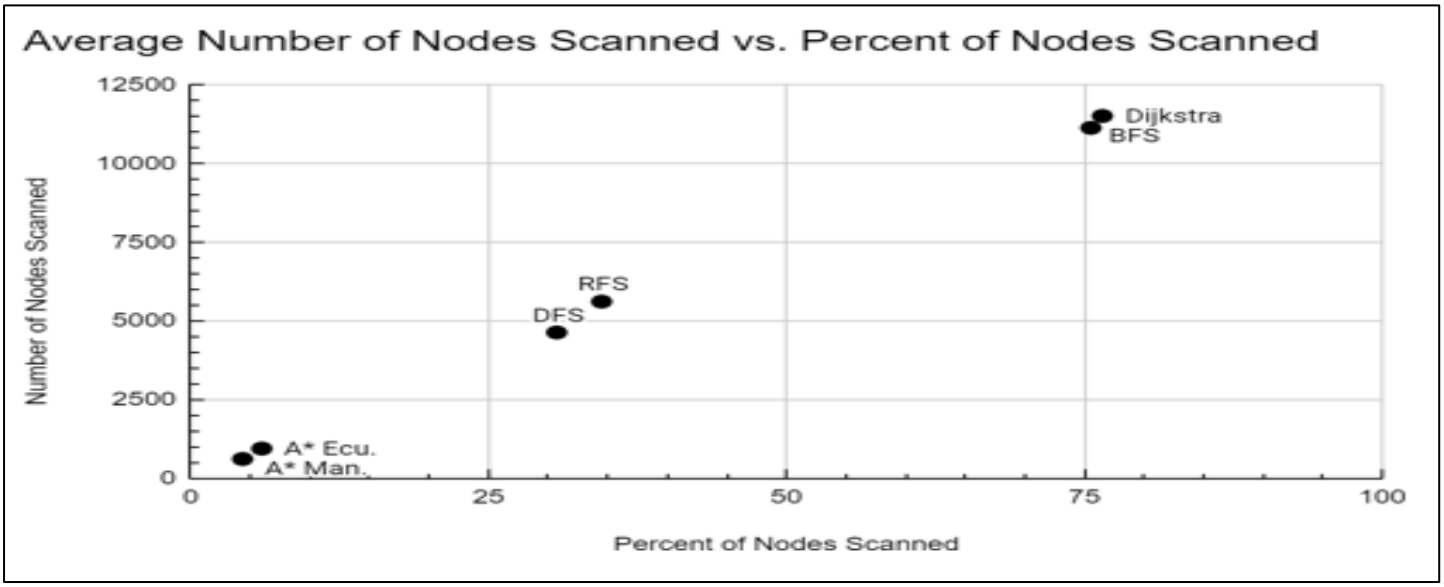

Fig 7: Avg Number of Nodes vs % Scanned of Algorithms

The computing time shows algorithms that searched more nodes that were irrelevant towards the position of the ending node took longer to compute. The computing time shows that Dijkstra took the longest along across all the algorithms, with an average of 0.129 seconds. This is because, on a rectangular grid, Dijkstra searches mostly towards the axes of the starting node, farthest from an end path node that is Orthodiagonal, and had to search all nodes to find the ending path. RFS took the second longest as it had to compute many terms in an ordered list before checking every time if it had exceeded the limit past the endpoint. This was contrasted by BFS which searched with an arc toward the end node and did not have to frequently check leading to faster computing time. DFS was faster than BFS as it was able to not have to continuously check for position while scanning nodes, and had to continue on a linear path. Both A* algorithm variations were the fastest, with A* Manhattan being the fastest with an average of 0.025 seconds. A* in particular was able to adapt to the slower nodes that were added as it was able to factor them into its distancing height map and was trained to avoid them, making computing faster. This also meant the breadth of the A* path was wider, due to the interruptions discouraging certain paths of the path; it was therefore more branched out. This led to A* having a significantly varying number of nodes checked, from 536 to 1045. Furthermore, accounting for the speed of the node in A* made its computing speed slower than DFS (due to calculations having to be made using the height map) on the third trial even though DFS searched more than 4500 nodes compared to the two A* algorithms searching less than 1000 nodes on average.

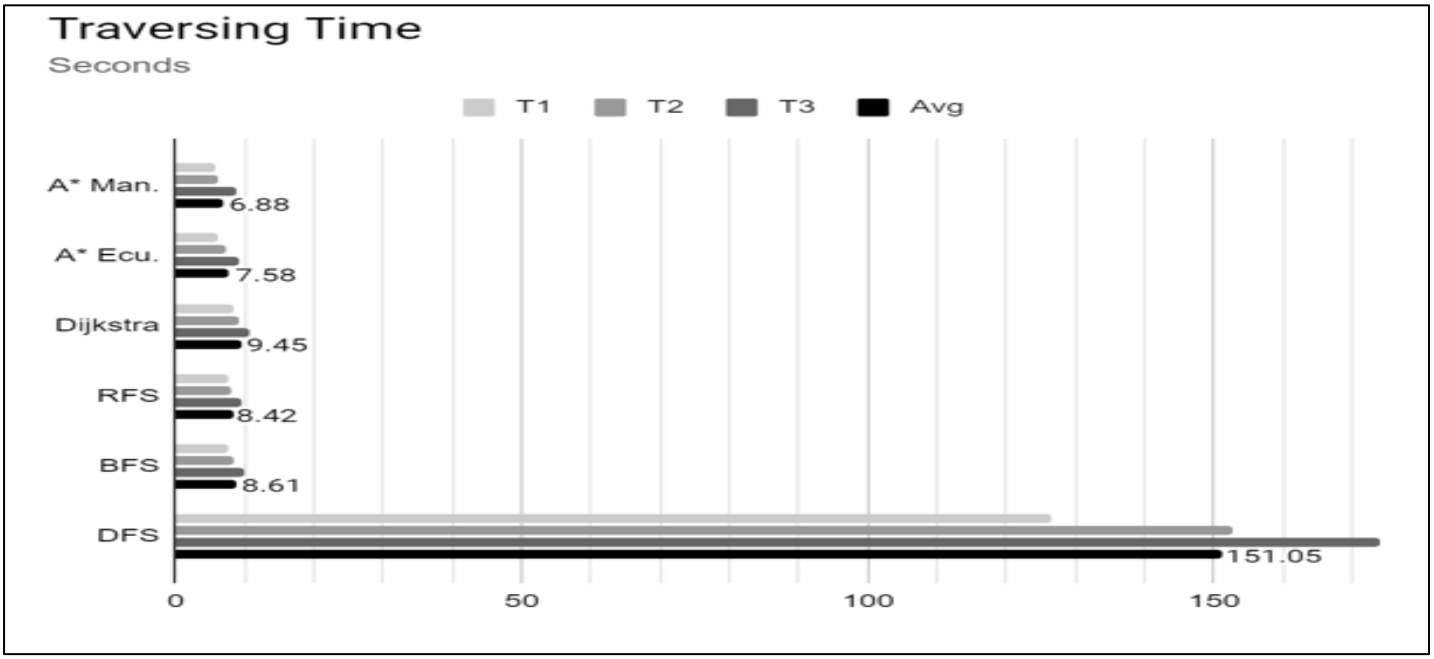

Fig 8: Traversing Time of Algorithms (sec)

ISSN No:-2456-2165 https://doi.org/10.5281/zenodo.14915594

The traversing time was largely impacted by the amount of traffic, with each consecutive trial having more traffic and therefore taking longer to traverse. The amount of time it took to navigate the grid was highest for DFS by a large margin due to the linear pattern of its search going through every possible path to the left of the shortest one. Dijkstra took an average of 9.45 seconds per path, due to it not scanning all the nodes around the finishing node, only allowing it to find paths that approached in one direction to the final node. Likewise, this was seen with RFS and BFS which scanned a similar set of nodes approaching primarily from one expected path, leading to close results in terms of speed. A* Manhattan was fastest with an average time of 6.88 and trial 1 had the fastest time of 5.26 seconds. A* Euclidean had an average of 7.58 seconds due to less punishment in the model for diverting routes compared to Manhattan. The height map-like reward systems for the A* allowed them to have the fastest traversing speeds by searching the most crucial nodes to the fastest path. A* was generally able to perform the best in all categories as it was best suited for a grid-like pathfinding simulation. A paper from Trilogi University16 found similar results when comparing BFS, A*, and Dijkstra. On their second level of pathfinding on a 2d grid, A* scanned the least nodes, less than 23 than that of BFS and Dijkstra and was the fastest to compute while still finding the optimal solution. "A* is the best algorithm in pathfinding, especially in Maze game/grids. This is supported by the minimal computing process needed and a relatively short searching time" (6, Permana).

The results also demonstrate RFS to be a viable contender for a linear searching algorithm. This is because it checks the area of the nodes completely between the two points rather than A* which uses an adaptive path. This is useful for applications in areas where mistakes can afford to be made, such as probe surgery. RFS also had the best traveling time with the exclusion of A* and would be closer to it in speed given more training with the node network. On average, it was noticed to have a slower learning curve than the other models with the node network. This was probably due to the large amounts of nodes it would repetitively scan each time, most of which were irrelevant to the solution.

VI. CONCLUSION

Pathfinding algorithms contrast in speed of computing and pathfinding efficiency. They use many different models and training sets to realize what they are. They also each have designated uses tailored to what they are required for. The algorithms of DFS, BFS, Dijkstra, and A* have allowed many advancements in the field of pathfinding and the applications they are suited to. The results show that A* was the fastest algorithm with DFS and Dijkstra having elements of the slowest. They also displayed how RFS could be a viable algorithm to expand on for thorough linear search purposes. RFS could be expanded upon by making the algorithm use elements of A* to not search nodes that were straying too far from the endpoint, while still maintaining a row-by-row linear search for fast computing. These results highlighted the navigation of traffic changing the pace of computing and traversing time. This was applicable as it could highlight the need for different pathfinding models for different regions, as this empirical analysis was mostly testing an environment similar to that of a city. Testing also showed how all algorithms met previous expectations outlined for their uses, reinforcing their importance in their sectors. In the future, these algorithms could be advanced by searching fewer nodes, which would reduce computation time, and this could be implemented with more node network training data to have more experienced programs. Additionally, punishments and rewards during training could be increased to have the programs only search the proper nodes necessary for the optimal path. Moreover, in different scenarios, such as in 3d and in cases where nodes are arranged in various angles. There could be more emphasis placed on the grid-like training, to have programs familiar with basic pathfinding which can be applied to different scenarios, possibly through machine learning. How will these algorithms be advanced in the future[6] and how might they shape our understanding of how to best navigate from point A to B? How will the introduction of new software such as enhanced machine learning affect the needs for these algorithms to fulfill in the forthcoming years?

ACKNOWLEDGMENTS

We would like to thank Dr. Srikanth Krishnan for his help on this project as well as Billy Gao.

REFERENCES

[1]. Morgan, Graham, et al. "Game Engineering - Newcastle University." 2: Pathfinding; Game Engineering;, Newcastle University, research.ncl.ac.uk/game/mastersdegree/gametechnologies/aitutorials/2pathfinding/. Accessed 2 Aug. 2024. https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/aitutorials/2pathfinding/AI%2020Simple%20Pathfinding.pdf

[2]. Team, Lark Editorial. "Some Common Pathfinding Algorithms." Lark, Lark Suite, 26 Dec. 2023, www.larksuite.com/en_us/topics/ai-glossary/some-common-pathfinding-algorithms. https://www.larksuite.com/en_us/topics/ai-glossary/some-common-pathfinding-algorithms AI, Pathfinding, algorithms This article talked about the history and use of pathfinding programs. It also mentioned how they work with the pros and cons provided.

[3]. Botea, Adi, et al. "Pathfinding in Games." DROPS, drops.dagstuhl.de/entities/document/10.4230/DFU.Vol6.12191.21. Accessed 25 July 2024. https://drops.dagstuhl.de/entities/document/10.4230/DFU.Vol6.12191.21. Pathfinding, Games, AI Talks about the use of pathfinding for bot characters in commercial games. Also mentions the main elements of a basic approach to pathfinding which includes having a map, ways to sense surroundings, and an algorithm to be the brain and evaluate the action.

ISSN No:-2456-2165 https://doi.org/10.5281/zenodo.14915594

[4]. Graham, Ross, et al. “Pathfinding in Computer Games.” ARROW@TU Dublin, arrow.tudublin.ie/itbj/vol4/iss2/6/. Accessed 2. Aug. 2024. https://arrow.tudublin.ie/itbj/vol4/iss2/6/ Computer, Pathfinding This article is about the design of AI pathfinding technologies concerning game design. Use of top-down simulations similar to my part of the secondary data analysis.

[5]. “Pathfinding Using AI Algorithms.” Baza Wiedzy Politechniki Warszawskiej, 1 Jan. 1970,repo.pw.edu.pl/info/bachelor/WUTc1a5c774c8442e78b129cbde56fc1fa/. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwj47JSAxNWHAxV2STABHauKEX0QFnoECBUQAQ&url=https%3A%2F%2Frepo.pw.edu.pl%2Fdocstore%2Fdownload%2FWUT8aeb20bbb6964b7da1cfefbf2e370139%2F1-s2.0-S0952197617301227-main.pdf&usg=AOvVaw15nWVVZFQgr6W7gB91Mc7s&opi=89978449 Neural Network, AI Talks about the detection of body parts which is useful in being able to differentiate different objects when pathfinding. It also provides a graph on the tree of detection.

[6]. Stout, Bryan. “Smart Moves: Intelligent Pathfinding.” *Gamasutra - Smart Moves: Intelligent Pathfinding*, Game Developer Magazine, July 1997, folk.idi.ntnu.no/agnar/it272/pekere/local/pathfinding.htm. Pathfinding, Algorithms Shows the differences in training costs for maneuvering models around obstacles. Demonstrates how pathfinding logic of punishments and incentives work to change.

[7]. Anisyah, Ani Siti, et al. “Route Optimization Movement of Tugboat with A∗ Tactical Pathfinding in Spin 3D Simulation | Request PDF.” *Researchgate.Net*, research gate, Dec. 2015, www.researchgate.net/publication/307802157_Route_optimization_movement_of_tugboat_with_A_tactical_pathfinding_in_SPIN_3D_simulation. Pathfinding, Reward Systems, Simulation Demonstrates the optimization of a tugboat using A*. This is shown by using fuel as a punishment to influence the tugboat to choose the shortest path.

[8]. “View of Cooperative Pathfinding.” Aaai.org, 2024, ojs.aaai.org/index.php/AIIDE/article/view/18726/18503. Accessed 10 Aug. 2024. Pathfinding, Collective This paper shows the difference in different pathfinding algorithms of A*. Demonstrates the difference within variations of incentive models, including Manhattan and Elucid.

[9]. Geeks for Geeks. “Breadth First Search or BFS for a Graph.” Geeks for Geeks, Geeks for Geeks, 20 Mar. 2012, www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/. Accessed 10 Aug. 2024. Algorithms, Pathfinding Shows the time necessity as a punishment/reward for BFS. Also provides code in various languages for BFS to disclose how the reward system works while using a node map to simplify the algorithm.

[10]. Wikipedia Contributors. “A* Search Algorithm.” Wikipedia, Wikimedia Foundation, 28 June 2024, en.wikipedia.org/wiki/A*_search_algorithm#:~:text=13%20External%20links-,History,algorithm%20for%20Shakey’s%20path%20planning. Accessed 10 Aug. 2024. Algorithm, Formula Gives the mathematical equation for A* and the history of the development. Also provides more understanding of path trees and the reward and search system of A*.

[11]. Aglubagerry. “A* Search Real-Life Application.” Medium, Medium, 10 Oct. 2023, medium.com/@aglubagerry/a-search-real-life-application-2bd175624952. Algorithm, Applications Provides examples of the real-life applications of A*. Also depicts how A* is used in specific cases and how it is essential to those tasks.

[12]. Schulz, Frank & Wagner, Dorothea & Weihe, Karsten. (1999). Dijkstra’s Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. Algorithm Engineering. 1668. 110-123. 10.1007/3-540-48318-7_11. Algorithm, Applications, Pathfinding Compare the distances with the pathfinding of German public railroads. Describes the application of Dijkstra's algorithm in this use case.

[13]. Burger, Alewyn & de Villiers, Anton & Vuuren, J.H.. (2013). Two algorithms for secure graph domination. JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing. 85. Algorithm, Applications Uses DFS to differentiate parts of a search tree. Checks for specific categories using DFS and provides the results of the tree with symbols.

[14]. Abd Algfoor, Zeyad & Sunar, Mohd Shahrizal & Kolivand, Hoshang. (2015). A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games. International Journal of Computer Games Technology. 2015. 1-11. 10.1155/2015/736138. Pathfinding, Simulation, Reward Systems Reviews the last 10 years of pathfinding algorithms and their development. Gives areas for future growth and current areas being explored. Delves into real-life examples in robotics and video games.

[15]. Kilic, Kemal & Mostarda, Leonardo. (2021). Heuristic Drone Pathfinding Over Optimized Charging Station Grid. IEEE Access. 1-1. 10.1109/ACCESS.2021.3134459. Pathfinding, Applications Uses a grid of nodes on a 2d field as pathfinding heuristics for the indoor drone. Calculates charging times factored in for the overall speed of the indoor drone path. Additionally, this paper provides information on having multiple heuristics monitored at once using various metrics. algorithm results, as well as a history of all algorithms and their applications.

[16]. Permana, Silvester & Bintoro, Ketut & Arifitama, Budi & Syahputra, Ade. (2018). Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game. IJISTECH (International Journal Of Information System & Technology). 1. 1. 10.30645/ijistech.v1i2.7. Algorithms, Collective Compares A*, Dijkstra, and BFS to find what

ISSN No:-2456-2165 https://doi.org/10.5281/zenodo.14915594

performed best in a grid format. Analyzes the number of nodes searched compared to traveling speed on average.

[17]. Kaur, Navneet, and Deepak Garg. "Analysis of the Depth First Search Algorithms." Gdeepak.Com, Thapar University, www.gdeepak.com/pubs/Analysis of the Depth First Search Algorithms.pdf. Accessed 28 Jan. 2025. Algorithms, Analysis Analyzes DFS algorithms using visual graphs to explain the methods DFS searches by Highlights areas for the algorithm's improvement, while mentioning how it works. Provides code for DFS that was used to base the paper off.

[18]. Pardede, Sara Lutami. "A Review of Pathfinding in Game Development." Cloudfront.Net, CEPAT Journal of Computer Engineering, May 2022, d1wqtxts1xzle7.cloudfront.net/91399669 Accessed 28 Jan. 2025. Algorithms, Analysis, Pathfinding Depicts several algorithms using a 2d grid-based node network to visually simplify them. Explains the use of Dijkstra for finding the shortest path in Japanese districts using the public road system.