Investigating A* Algorithm on WAZE Pathfinding System

Ho Rong Wei
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp065092@mail.apu.edu.my

Chew Jin Ni
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp064851@mail.apu.edu.my

Damon Ng Khai Weng
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp064820@mail.apu.edu.my

Tan Wen Liang
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp064341@mail.apu.edu.my

Dr.Adeline Sneha J
Senior Lecturer/School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
adeline.john@apu.edu.my

Dr. Kamalanathan Shanmugam
Senior Lecturer / School of Technology
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
kamalanathan@apu.edu.my

Juhairi Aris Muhamad Shuhili
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
juhairi.shuhili@apu.edu.my

***Abstract*— Pathfinding algorithms are used for finding the shortest path to travel between a starting point and ending points if a path exists. This paper aims to research one of the most popular pathfinding algorithms which is A*. The purpose is to find out different aspects about the traditional A* pathfinding algorithm and test its suitability for being implemented in navigation systems like Waze. The algorithm will run through multiple limited 2D static grid-based environments in cases where obstacles are either present or absent. During the trials, its heuristics factor and run-time for will be tabulated. The research revealed that A* performs most efficiently when its heuristic factor is calibrated between 0.7 to 0.8 and any value below 0.6 significantly increases its runtime. This shows evidence for A* to be suitable in navigation system but further testing with real-time data, and dynamic environments could better support its application within navigation systems.**

Keywords—A* algorithm, pathfinding algorithm, Waze path navigation system

I. Introduction

Most vehicles waste time on the road and burn more fuel due to poor traffic conditions (Chian & Kamsin, 2023), and in this age of rapid change where roads are constantly being changed, making traditional maps obsolete. This is why GPSs exist, to help us navigate through the confusing traffic and find the fastest way from our location to our destination. Other than the layout of the land and traffic data, GPSs also require a pathfinding algorithm. This report presents a comprehensive analysis of the A* algorithm, a widely utilized heuristic search algorithm renowned for its efficiency and effectiveness in pathfinding and optimization tasks. The literature review investigates into optimization principles, as well as examines the algorithm's inherent advantages. Conversely, the review also scrutinizes its disadvantages, including its sensitivity to the quality of heuristic estimates and its potential inefficiency in certain scenarios. The report also conducts a meticulous comparison between A* and alternative algorithms. Additionally, it explores upon real-life implementations of the A* algorithm across diverse applications, showcasing its practical utility in domains ranging from robotics to gaming. The report outlines the materials and requirements necessary for algorithm implementation, encompassing hardware and software prerequisites essential for executing A* efficiently. Methodologically, it describes the approach to algorithm implementation, elucidating the purpose and parameters governing its operation. Finally, the report presents the results and discussions derived from empirical evaluations, offering insights into the algorithm's performance, and discussing potential avenues for future research and optimization.

II. Literature Review

A. Strengths and Weaknesses of A*

A* inherently has its strengths and weaknesses that make it suitable for specific use cases. The classic A* is popular for being a relatively simple algorithm that can operate in a closed 2D or 3D environments as evidenced in Alsakka et. al. (2020) using A* to navigate 3D cable paths and Aziz et. al. (2022) comparing A* with Dijkstra and ACO in pathfinding for a 2D grid environment. Second, A* uses heuristics, so it always finds the shortest path if a path exists. In addition, A* operates more efficiently in environments with a smaller grid size and accurate weightage as well as only having 1 starting and ending point. This is demonstrated by Yerramilli et. al. (2021) when testing A*'s pathfinding capability in a grid map that represents a real-world location.

Consequently, due to these specifications, the traditional A* algorithm is built particularly for path finding in closed-space static environments like the one from Alaska et. al. (2020) and Rafiq et. al. (2020). This limits its use cases to situations like simple automatic navigation within game development, industrial cable planning and car navigation systems. Moreover, A* is an algorithm that relies heavily on its heuristics as since Goyal et. al. (2014) pointed out that its heuristics played a major role in its pathfinding. If the heuristics function is not properly coded to be admissible, the algorithm might fail to find the true shortest path. Moreover, the traditional A* has proven to not be efficient if there are multiple end points as mentioned by Goyal et. al. (2014), cannot be operated with real-time data as mentioned by Renoy et. al. (2015) as one of the traditional A*'s flaws; and does not work with negative edge weights, another issue that it uses up a lot of memory for dynamic environments as evidenced by Leigh et. Al. (2007) and Korkmaz & Durdu's (2018) research.

B. Real Life Implementation

Besides your everyday car navigation apps, the A* algorithm is still used across different of life, utilizing its ability to find the most optimal path with remarkable efficiency. The algorithm has allowed the accomplishment of many different things, from navigating through obstacles to decision making, as well as searching for different things. We can see the algorithm is being applied to robots and provided it the ability to navigate through a cluttered warehouse while avoiding obstacles. Other than that, it has been used in aspects such as checkers, providing it with the intellectual capability to make decisions and anticipate opponent's movements, or even guides the complex folding of proteins, which may potentially unlock medical breakthroughs and save more lives. Speaking of saving lives, it has been proven helpful for ambulances as the algorithm to generate an optimized path to help navigate ambulances through traffic, avoiding busy traffic to maximize time saved as time is of the essence when it comes to saving lives. Efficiency is one of the important aspects in manufacturing. With that in mind, A* has also been used to help plan factory production lines optimally to reduce times of production. Researchers has also used the algorithm to dive into the vast amounts of information present on the internet, utilizing its searching capabilities to return relevant search results, producing a smoother online experience.

C. Comparison with Different Algorithm

The literature review provides an in-depth analysis of various algorithms used for solving the shortest path problem, with a specific focus on why Waze chose to implement the A* algorithm in its navigation application. The review compares the A* algorithm with other algorithms such as Dijkstra, Bellman-Ford, and Floyd-Warshall. The real-life example of the shortest path problem will be identifying the shortest distance in navigating applications like Waze, Google Maps, Apple Maps, and many more directory applications. According to Matthieu Casanova, Waze utilises the A star algorithm to solve the shortest path problem (Sailiou, n.d.)

According to the survey from Maharshi and Ronit (2018), the A Star algorithm is a combination of the Dijkstra algorithm and the Greedy Best-First Search algorithm. It uses a heuristic function to determine a node's distance from the target node and its current state by using f(m)=c(m)+h(m) equations to calculate the cost of the node. The algorithm efficiently computes an optimal solution by combining the Uniform Cost Search function from Dijkstra. Applications in various domains, such as traffic navigation systems and games for Non-Player Characters (NPC) pathfinding, have implemented the A* algorithm due to its efficiency in route planning. Studies by Maharshi and Ronit, found that heuristic search algorithms are generally better than blind search algorithms. For example, A star and UCS, A star is more efficient and able to find more optimal paths than Uniform Cost Search. A star algorithm is the advancement of UCS by integrating heuristic functions into the algorithm. This helps in improving the A star algorithm accuracy and sensitivity of finding the optimal path, in the meantime, it also increases the memory usage and computational time due to its heuristic function. (Maharshi, Ronit, 2018)

The Dijkstra algorithm categorizes nodes into unmarked, temporary, and permanent. Research is conducted by Shrawan and Pal (2015) to compare the A star algorithm and the Dijkstra Algorithm in terms of investing their performance in searching the low-cost paths for road networks. According to the experiment results, it shows that A star performs better than Dijkstra in terms of time in every condition except in an uninformed search. (Sharawan, Pal, 2015) Not only that, the journal from Dian and Lysander also analysed that the A star algorithm performs better and faster in the large-scale map due to its heuristic compared to Dijkstra. This is due to the reason that Dijkstra explores all the directions uniformly, but A star only explores the direction of the destination. The statement can be proven by the loop count of each algorithm where Dijkstra comes with a higher loop count (Dian, Lysander, 2020)

The Bellman-Ford algorithm is designed to find the shortest path, even with negative weights, and can detect negative cycles and avoid them through the "relaxation" method (Vaibhavi, Chitra, 2014). A negative cycle in a loop will cause to decrease in the shortest path finding accuracy, creating an unreliable shortest path and an endless loop in a graph. Dijkstra is generally faster and more efficient compared to Bellman-Ford. Dijkstra algorithms consume more memory usage when executing on a large number of nodes, but Bellman-Ford may not require a complex data structure where it is more memory efficient (Samah et.al, 2020). In the context of GPS routing, the A star algorithm will be selected compared to Bellman-Ford due to its efficiency and optimality. Even though the Bellman-Ford algorithm can avoid falling into a negative cycle loop, but long processing time does not suit real-time applications and GPS routing.

The Floyd-Warshall algorithm is used for finding the shortest path between pairs of nodes in a weighted directed graph. The algorithm uses a matrix to store distance and initialize with direct edge weights or infinity which means no direct edge exists. Using a matrix helps to find all possible shortest paths where it accommodates positive and negative edge weight (Harseerat, Gopal, 2022). According to the study conducted by Ramesh and Vishwas, they conclude that the Floyd-Warshall algorithm capable of finding the most accurate result for routing purposes, but it is the slowest and time-consuming method for shortest path finding compared to the A star algorithm. Even though the Floyd-Warshall algorithm has high accuracy in finding the shortest path, it isn't a suitable algorithm for single-pair pathfinding. The Floyd-Warshall algorithm is more suitable for LPG Gas route planning, instead of real-time GPS navigation (Ramesh, Vishwas, 2023).

In conclusion, the A star algorithm has been chosen by Waze for their shortest path finding algorithm due to its efficiency and accuracy mainly because of the heuristic function. The Dijkstra algorithm is performing well, it explores all directions from the initial node, but it might consume more time. It is an alternative algorithm for shortest path finding compared to the A star algorithm. Compared to the A star algorithm, the Bellman-Ford algorithm is not suited for real-life route planning, but it is suitable for routing protocols in computer networks or telecommunications networks due to its negative cycle loop detection and high processing time. The Floyd-Warshall algorithm wasn't the best algorithm in terms of route planning due to the time complexity, even though it has the most accurate shortest path.

D. *Optimization of A* algorithm*

Several aspects of A* algorithm can be optimised to improve the performance of the WAZE navigation system, including pathfinding speed, memory usage, path robustness and multi-target search. Mi at el. proposed RPT and JPS algorithms to be integrated with A* algorithm to filter unnecessary nodes, minimizing the execution time by 67% and pathfinding speed by 47% (Mi, Xiao, & Huang, 2023). Liu et al. presented fusion algorithm, which reduce the pathfinding time by approximately 70% (Liu, Wang, & Xu, 2022). It managed to choose the best available option instead of revisiting the starting node, and lessen the path branching by exploring smaller range of path points to reduce data redundancy and path planning time. Waze can benefit from faster pathfinding algorithms to calculate optimum routes swiftly and generate real-time navigation updates, particularly in complex environments.

Besides, Tang et al. suggested an improved A* algorithm formula, $F(n) = G(n) + H(n) + C(n)$, which lowered memory consumption by 64.38%. It utilized turning points and key point setting method to improve path robustness and prevent the algorithm from being trapped (XiangRong, Yukun, & XinXin, 2021). Huang et al implemented JPS and pruning methods such as Triangle Inequality Theorem into conventional A* algorithm, reducing its memory space by 14.98% and nodes by 3.14% (Huang, Li, & Bai, 2022). Memory space reduction can be advantageous to WAZE to operate more efficiently and smoother in users' smartphones to improve user experience.

Moreover, Yao et al. increases path robustness by applying unbelievable points, virtual human motion, inner searching markers and repeated path search so that it will not be trapped in obstacles (Yao, Binbin, & Qingda, 2009). It addresses WAZE's challenge with unexpected road closures due to constructions or events and traffic congestion, preventing cars from getting trapped in such conditions. Xiang et al. also introduced the integration of greedy algorithm with A*algorithm to achieve multi-target pathfinding which allows the users to visit several destinations, like the add stops features in WAZE (Xiang, Lin, & Ouyang, 2022).

III. MATERIALS AND METHOD

A. *Materials*

i. Software Requirements

To execute the source code given, visual studio code is utilised for this research paper. Visual Studio Code is a code editor where users can run the code snippets within the file directly. The given source code is in Python format, python is selected as the development language due to its readability and conciseness. Python has five main components like UI, code editor, file's view, compiler and debugger. (Youssef, 2022) It is easier to write and maintain the code, which is beneficial for complex algorithms like A Star for shortest path finding. Other than that, it has rich standards of library where it provides plenty of modules and packages. In this source code given, it has installed and imported two libraries which are pygame and random. Pygame is used for creating a graphical user interface for pathfinding visualisation. Meanwhile, random is used for generating random numbers, in this case, it creates a random maze for pathfinding. The A Star algorithm is executed in the Windows operating system.

ii. Hardware Requirements

The hardware that has been used for executing the given A star algorithm for pathfinding in this paper is the NVIDIA GeForce RTX3050 for the Graphic Processing Unit (GPU). In the source code, the A star algorithm itself doesn't heavily rely on GPU acceleration. The Central Processing Unit (CPU) used in this research paper is 12th Gen Intel(R) Core (TM) i5-12500H. The performance of the A star algorithm can be affected by the capabilities of the CPU. A star algorithm includes various computations like node evaluation, heuristics calculations, priority queues, and so on. The speed of processing those computations depends on the processing power of the CPU. The computer uses 16GB RAM to perform the A-star algorithm. It involves keeping track of open and closed sets, node information storage, and computations that require memory..

B. *Methodology*

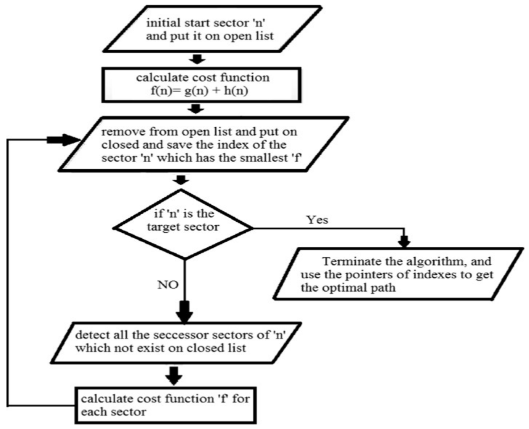

initial start sector 'n' and put it on open list
calculate cost function f(n)= g(n) + h(n)
remove from open list and put on closed and save the index of the sector 'n' which has the smallest 'f'
if 'n' is the target sector
Yes
Terminate the algorithm, and use the pointers of indexes to get the optimal path
NO
detect all the seccessor sectors of 'n' which not exist on closed list
calculate cost function 'f' for each sector

Figure 1: A algorithm Block Diagram (Zidane & Ibrahim, 2018)*

Figure 1 shows the block diagram of an A* algorithm. Pathfinding of A* algorithm is initialized by storing the starting node into an open list. The f, g, and h value of the node is then calculated. G value is the actual value from current node to child node. The final cost of the path will be stored in g value. H value is the heuristics value from start point to end point, which is computed using Euclidean Distance, also known as Pythagoras' theorem. F value is the sum of g value and h value. During exploration, the algorithm generates child nodes by navigating the neighboring position. If the neighboring node is an obstacle, it will skip the calculation and proceed to the next neighboring node to evaluate their f value. The neighboring nodes are then stored in a priority queue in the open list, sorted by their f value ascendingly. The node with the lowest f value will be selected as the current node. After that, the current node is inserted into the closed list to prevent the node from being revisited. The current node is also checked to determine if it is the target point. If it is not the destination point, the child nodes that are not in closed list and have not been explored, are added into the open list. The iteration continues starting from computing the f value of the child nodes until the target node is found. When the algorithm reaches the goal node or the open list is empty, backtracking is executed to retrieve the best optimal path from start node to end node.

IV. Algorithm Implementation

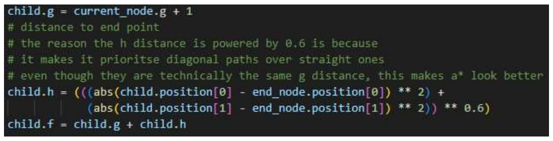
The source code from James Robinson enables the simulation of pathfinding process of A* algorithm in a virtual environment (Robinson, 2020). The pathfinding can be simulated in a simple map by inserting one starting point and one or more end points by using keys 1 to 9. The algorithm will initialize the pathfinding from the starting point and progressively advances to other points in ascending order based on their assigned numbers. The maximum point that can be assigned is 9 points, including the starting point. To create a more complex environment, a random maze can automatically be generated by clicking the 'm' key. The algorithm will cross over the obstacles when navigating the map until the targets are found. The optimal path is returned after it reaches the targets. Customized maps can also be developed by adding the obstacles manually to the blank map. The pathfinding process of A* algorithm can be observed, and by importing time module into the source code, the run time of A*algorithm pathfinding can also be calculated.

A. Purpose

The A* algorithm serves the fundamental purpose of finding the optimal path from a designated starting point to a specified goal within a graph or network structure efficiently. Its significance spans across various aspects including route planning, robotics, and artificial intelligence, where identifying the shortest or most efficient route is significant. A* has the ability to keep a perfect balance between completeness and optimality, offering a versatile solution for various real-world problems that require pathfinding and optimization tasks. Its effectiveness lies in navigating through complex networks while intelligently leveraging heuristic estimates to guide the search process towards the goal, making it a cornerstone algorithm in computational problem-solving. (GeeksforGeeks, 2023)

B. Parameters

Critical to the functionality of the A* algorithm are its parameters. To be more specific, there are the heuristic function and the cost function. The heuristic function provides an estimate of the cost from the current node to the goal node, facilitating informed decision-making within the algorithm. Meanwhile, the cost function determines the expense associated with traversing from one node to another in the graph, influencing the overall optimization process. These parameters play a significant role in guiding A*'s exploration of potential paths, allowing it to prioritize those that are deemed most promising while maintaining computational efficiency. The heuristic function's value represents an approximation of the remaining cost from a given node to the goal, thereby shaping the algorithm's decision-making process and ultimately impacting its ability to find optimal solutions effectively. (GeeksforGeeks, 2023)



```python
child.g = current_node.g + 1
# distance to end point
# the reason the h distance is powered by 0.6 is because
# it makes it prioritse diagonal paths over straight ones
# even though they are technically the same g distance, this makes a* look better
child.h = (((abs(child.position[0] - end_node.position[0]) ** 2) +
            (abs(child.position[1] - end_node.position[1]) ** 2)) ** 0.6)
child.f = child.g + child.h
```

Figure 2: Parameters for A Algorithm*

The figure above shows the parameters for the A* algorithm. The parameters required for the algorithm are the starting node, goal node, its heuristic function (f), the cost function (g), and the total cost function (f), which is calculated using the f and g. In this case, h is set at 0.6, which can be seen in the red box located on the figure. The reasoning behind is so that it will prioritize diagonal lines instead of the straight lines as it is the optimal line and much efficient. As we change its value, the amount of time taken to generate an optimal line will be affected as well. These parameters guide the A* algorithm in exploring the graph systematically while also selecting the path intelligently towards the goal node.

V. Results and Discussion

A. Results

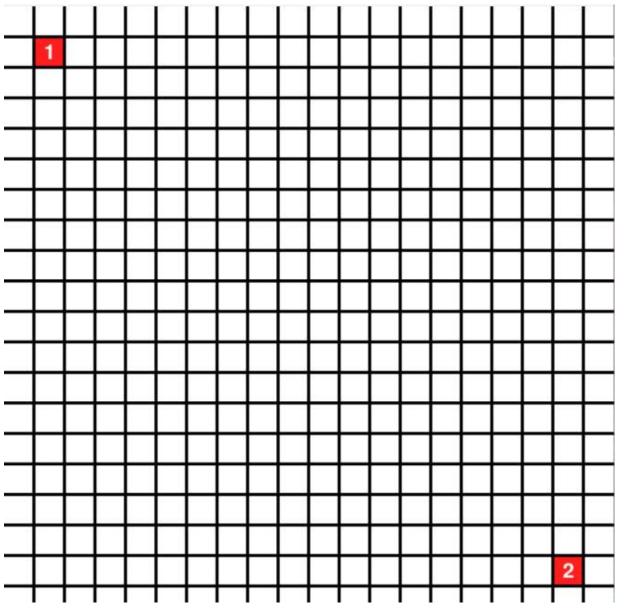
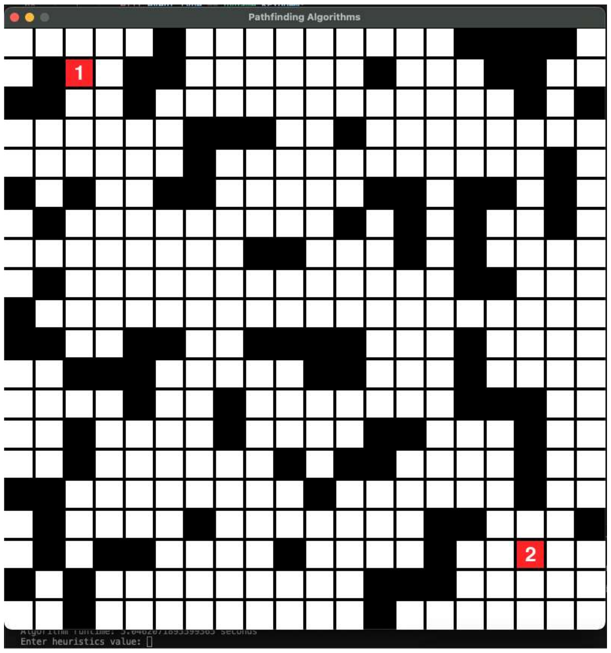
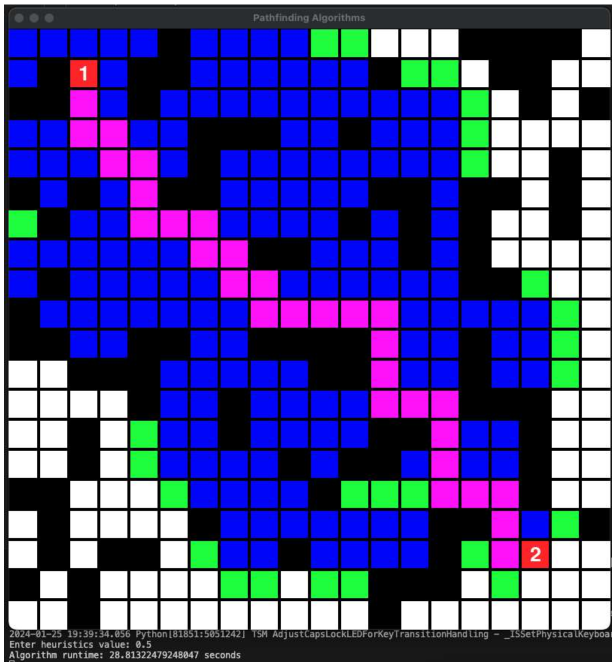


Figure 3: The map, start point and end point without obstacles.



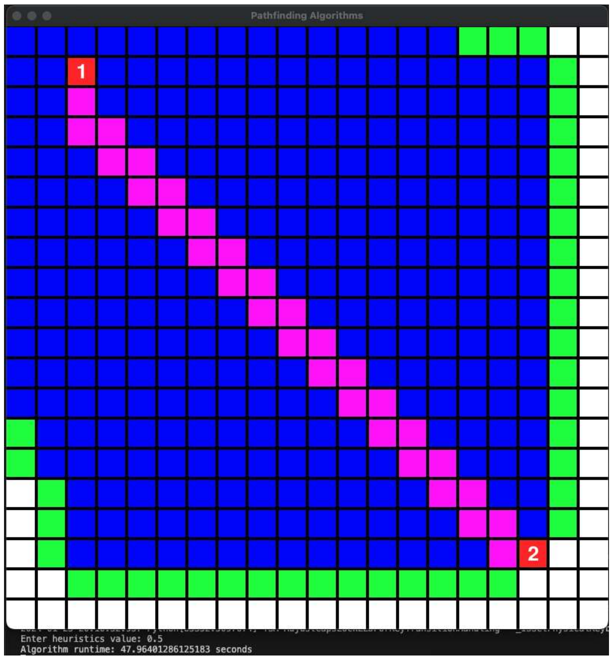
```
Enter heuristics value: 
```

Figure 4: The map, start point and end point with obstacles.



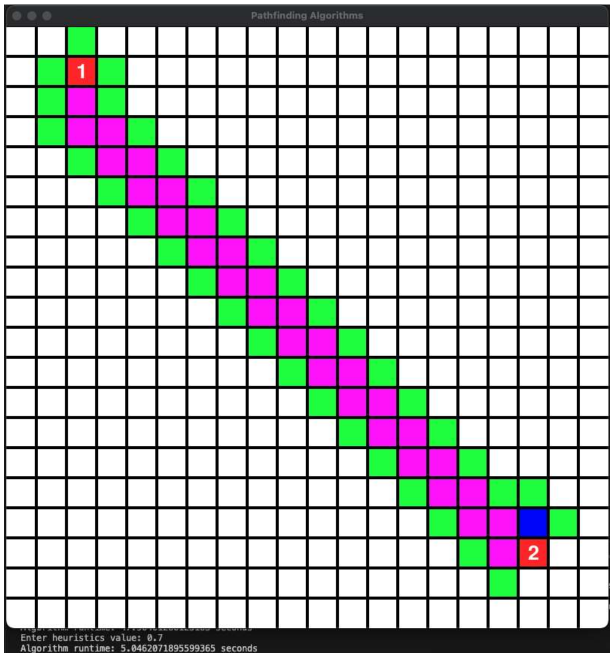
```
2024-01-25 19:39:34.056 Python[81851:5051242] TSM AdjustCapsLockLEDForKeyTransitionHandling - _ISSetPhysicalKeyboa
Enter heuristics value: 0.5
Algorithm runtime: 28.81322479248047 seconds
```

Figure 6: A* behavior with H (0.5) in a map with obstacles



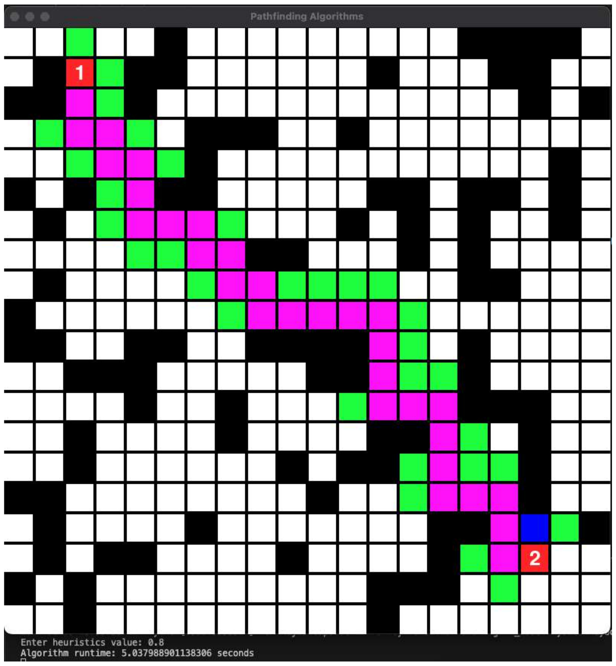
```
Enter heuristics value: 0.5
Algorithm runtime: 47.96401286125183 seconds
```

Figure 5: A* behavior with H (0.5) in a map without obstacles



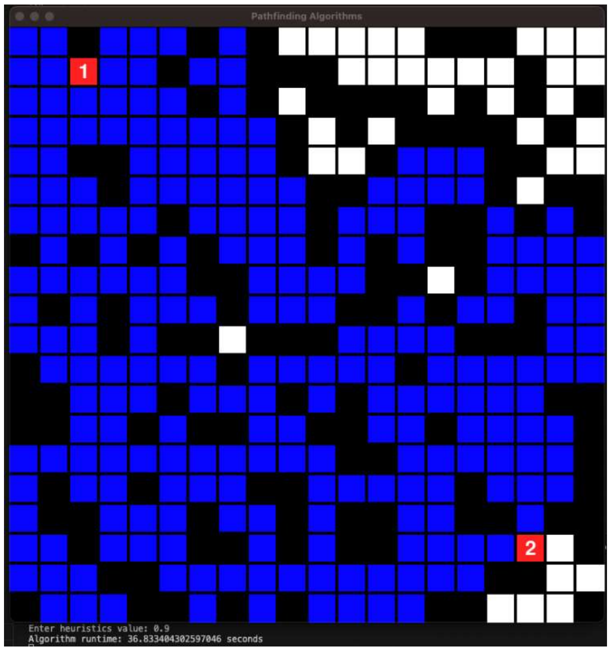
```
Enter heuristics value: 0.7
Algorithm runtime: 5.0462071895599365 seconds
```

Figure 7: A* behavior with H (0.7) in a map without obstacles



```
Enter heuristics value: 0.8
Algorithm runtime: 5.037988901138306 seconds
```

Figure 8: A behavior with H (0.8) in a map with obstacles*



```
Enter heuristics value: 0.9
Algorithm runtime: 36.833404302597046 seconds
```

Figure 9: A enters deadlock state with H (0.9) in more complex map*

TABLE I. A* ALGORITHM RUN TIME FOR DIFFERENT H DISTANCE VALUE IN MAP WITHOUT OBSTACLES.

H Distance Value	**Run time**
0.1	58.7944
0.2	58.6837
0.3	58.6808
0.4	59.9777
0.5	47.9640
0.6	6.1523
0.7	5.0462
0.8	5.8242
0.9	5.3859
1.0	5.3940
1.1	5.4140
1.2	5.4163

TABLE II. A* ALGORITHM RUN TIME FOR DIFFERENT H DISTANCE VALUE IN MAP WITH OBSTACLES

H Distance Value	**Run time**
0.1	42.1373
0.2	42.2490
0.3	41.5854
0.4	38.4340
0.5	28.8132
0.6	5.2446
0.7	4.9509
0.8	4.9457
0.9	4.9588
1.0	4.9212
1.1	4.9453
1.2	4.9658

B. Discussion

Based on results from the 2 tables above, we discovered that the classic A* algorithm performs optimally when its heuristics value is set to 0.7 and 0.8 for pathfinding with or without obstacles respectively. Hence, determining the admissible range to be from 0.7 to 0.8. Consequently, anything lower than 0.6 greatly increases its runtime and hence decreases its efficiency in path finding; the worse performing trials occur when the heuristics value is set to 0.2 or 0.4 in cases with or without obstacles respectively.

Although the collected data shows faster runtime in cases where the heuristics value is calibrated higher than 0.8, those values were where were cases that A* will get trapped in a path-finding deadlock occur, the algorithm would reach a dead-end and it essentially stops functioning and does not further explore any nodes or map the shortest path between the starting and ending point. Besides that, when calibrating higher than 0.8, there were even special cases where A* continues searching for shorter routes even when it has explored the nodes closest to the ending point and subsequently entering a deadlock.

In regards to how changing the heuristics value would affect the behaviour of the classic A* algorithm. The higher the heuristics value, the narrows the direction of searching, and the higher the chances of entering a deadlock state.

Therefore, if we relate our trials to navigating through traffic in a city or rural area, calibrating the heuristics value to 0.8 would be the best choice. This is to prevent deadlocks from high values, and preventing too much exploration from low values.

VI. CONCLUSION

This paper concludes that the A* pathfinding algorithm is indeed suitable to be implemented for pathfinding within navigation systems with results performing the best when setting the heuristics value from 0.7 randomly generated grids. But due to the limitations of using randomly generated, small and static grid environments, the results only support the original algorithm's application for navigation in a small scale where the environment does not change, which does not cover cases of outdated or faulty data that are common for modern navigating systems.

The research can be improved by comparing different path finding algorithms including A* or its variants while using real-world data for generating its virtual environment. This will identify which of them is a more suitable algorithm to be used in a real-time navigation system such as Waze or Google Maps.

ACKNOWLEDGMENT

I extend my sincere appreciation to Ms. Adeline Sneha John Chrisastum, my dedicated lecturer, whose guidance, and expertise greatly enriched this research paper. Her insightful feedback and continuous support were instrumental in shaping the direction of this work. I would also like to acknowledge the invaluable resources provided by the APU library, both in terms of physical materials and electronic databases. Additionally, I express my gratitude to the online sources that served as essential references, augmenting the comprehensiveness of this study. This research is a collective effort, and I am grateful for the contributions of all involved.

REFERENCES

Alsakka, F., Khalife, S., Nomir, M., Mohamed, Y., & Hermann, R. (2020). Comparison of Shortest Path Finding Algorithms for Cable Networks on Industrial Construction Projects. *Proceedings of the International Symposium on Automation and Robotics in Construction. 37*, pp. 409-416. Kitakyushu, Japan: IAARC Publications.

Aziz, A., Tasfia, S., & Akhtaruzzaman, M. (2022). A Comparative Analysis among Three Different Shortest Path-finding Algorithms. *3rd International Conference for Emerging Technology (INCET)*, 1-4.

Chian, C. Y., & Kamsin, D. I. (2023). Implementation of Intelligent Transportation System using Smartphone Sensors to Improve Traffic Conditions. *Journal of Applied Technology and Innovation 7(1)*, 52-56, https://dif7uuh3zqcps.cloudfront.net/wp-content/uploads/sites/11/2023/01/21104542/Volume7_Issue1_Paper10_2023-1.pdf.

Dian, Lysander. (2020). *Analysis of Dijkstra's Algorithm and A* Algorithm in*. Retrieved from Journal of Physics: https://iopscience.iop.org/article/10.1088/1742-6596/1566/1/012061/pdf#:~:text=A*%20algorithm%20is%20just%20like,just%20explore%20all%20possible%20ways.

GeeksforGeeks. (2023, March 8). *A* Search Algorithm*. Retrieved from www.geeksforgeeks.org: https://www.geeksforgeeks.org/a-search-algorithm/

Goyal, A., Mogha, P., Luthra, R., & Sangwan, N. (2014). PATH FINDING: A* OR DIJKSTRA'S? *International Journal in IT & Engineering, 2*(1), 1-15.

Harseerat, Gopal. (2022). Research on Dijkstra's, A*, Bellman-Ford, and Floyd Warshall Path Finding Algorithm. *International Research Journal of Modernization in Enginerring Teachnolocy and Science*, 886-893.

Huang, H., Li, Y., & Bai, Q. (2022). An improved A star algorithm for wheeled robots pathplanning with jump points search and pruning method. *Complex Engineering Systems 2(3):11*, 1-12. doi:10.20517/ces.2022.12.

Korkmaz, M., & Durdu, A. (2018). Comparison of optimal path planning algorithms. *14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)* (pp. 255-258). Lviv-Slavske, Ukraine: IEEE.

Leigh, R., Louis, S. J., & Miles, C. (2007, April). Using a genetic algorithm to explore A*-like pathfinding algorithms. *IEEE Symposium on Computational Intelligence and Games* (pp. 72-79). Honolulu, HI, USA: IEEE.

Liu, L., Wang, B., & Xu, H. (2022). Research on Path-Planning Algorithm Integrating Optimization A-Star Algorithm and Artificial Potential Field Method. *Electronics 2022, 11*, 3660. doi:10.3390/electronics11223660.

Maharshi, Ronit. (2018, June). *Comparative Analysis of Search Algorithm*. Retrieved from Research Gate: https://www.researchgate.net/profile/Ronit-Patel-2/publication/333262471_Comparative_Analysis_of_Search_Algorithms/links/5ce4fd5aa6fdccc9ddc4c25c/Comparative-Analysis-of-Search-Algorithms.pdf

Mi, Z., Xiao, H., & Huang, C. (2023). Path planning of indoor mobile robot based on improved A* algorithm incorporating RRT and JPS. *AIP Advances*, 13(4): 045313. https://doi.org/10.1063/5.0144960.

Rafiq, A., Abdul Kadir, T. A., & Ihsan, S. N. (2020). Pathfinding Algorithms in Game Development. *IOP Conference Series: Materials Science and Engineering. 769.* IOP Publishing.

Ramesh, Vishwas. (2023). Advance Study of Shortest Route Problem and its application Bellman-Ford algorithm. *IJSDR Volume 8 Issue 6*, 1640-1656.

Renoy, R., Prem, R., & Patil, S. (2015, May-June). A focused dynamic path finding algorithm to pursue a moving target. *International Journal of Engineering Research and General Science, 3*(3), 1220-1224.

Robinson, J. C. (2020, April 12). *Path-Finding-Visualisation-with-Pygame*. Retrieved from github.com: https://github.com/James-Charles-Robinson/Path-Finding-Visualisation-with-Pygame?tab=readme-ov-file#a-star

Sailiou, M. (n.d.). *Automated navigation systems are still wreaking havoc on small towns' streets*. Retrieved from Algorithm Watch: https://algorithmwatch.org/en/navigation-systems-small-towns/#:~:text=%E2%80%9CWaze%20uses%20an%20A*%20path,applications'%20map%20back%20in%202010.

Samah et.al. (2020). Comparative Analysis between Dijkstra and Bellman-Ford Algorithms in Shortest Path Optimization. *International Conference on*

Technology, Engineering and Sciences (ICTES) 2020.

Sharawan, Pal. (2015). Shortest Path Searching for Road. *International Journal of Computer Science and Mobile Computing*, 513-522.

Vaibhavi, Chitra. (2014). A survey paper of Bellman-ford algorithm and Dijkstra algorithm for finding shortest path in GIS application. *International Journal of P2P Network Trends and Technology (IJPTT) – Volume 4 Issue 1* , 21-23.

Xiang, D., Lin, H., & Ouyang, J. (2022). Combined improved A* and greedy algorithm for path planning of multi-objective mobile robot. *Sci Rep 12*, 13273. https://doi.org/10.1038/s41598-022-17684-0.

XiangRong, T., Yukun, Z., & XinXin, J. (2021). Improved A-star algorithm for robot path planning in static environment. *Journal of Physics Conference Series 1792(1):012067*, 1-8. doi:10.1088/1742-6596/1792/1/012067.

Yao, J., Binbin, Z., & Qingda, Z. (2009). The Optimization of A* Algorithm in the Practical Path Finding Application. *WRI World Congress on Software Engineering, Xiamen, China*, 514-518. doi: 10.1109/WCSE.2009.412.

Yerramilli, N. S., Johnson, N. J., Reddy, O. S., & Prajwal, S. (2021). Navigation Systems Using A. *2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)* (pp. 708-712). Bangalore, India: IEEE.

Youssef, L. (2022). Docker container python IDE. *Journal of Applied Technology and Innovation*, 30-34.

Zidane, I., & Ibrahim, K. A. (2018). Wavefront and A-Star Algorithms for Mobile Robot Path Planning. *Advances in Intelligent Systems and Computing 639*, doi:10.1007/978-3-319-64861-3_7.