Asia Pacific Academy of Science Pte. Ltd.

Computer and Telecommunication Engineering 2024, 2(3), 2504
https://doi.org/10.54517/cte2504

Article

A new modification to the A* path-finding algorithm to improve its space and time performance

Aaron Rasheed Rababaah

College of Engineering and Applied Sciences, American University of Kuwait, Salmiya 13034, Kuwait; arababaah@auk.edu.kw

CITATION

Rababaah AR. A new modification to the A* path-finding algorithm to improve its space and time performance. Computer and Telecommunication Engineering. 2024; 2(3): 2504. https://doi.org/10.54517/cte2504

ARTICLE INFO

Received: 9 June 2024
Accepted: 13 September 2024
Available online: 26 September 2024

Abstract: We propose a new modification to the A* algorithm named AA* that significantly improves space and time complexities. In AA*'s forward pass, the node sets (open and closed) are not used, and only the local node neighborhood is saved to take the next move decision. AA* needs a backward pass to bridge and correct gaps and bad decisions made in the forward pass. The work of the backward pass is far less than that of the forward pass, as most of the task has been done. It is shown via empirical experimental work that our proposed AA* algorithm is superior to the classical A* algorithm in the typical three metrics: running time, number of probed nodes, and length of path. Furthermore, our experimental work showed that AA* is suboptimal in terms of length of path compared to the original Dijkstra's algorithm with an accuracy of 96.95%.

Keywords: A* path-finding algorithm; new modification to A*; mobile robotics; graph algorithms; grid-based path finding

1. Introduction

Path planning algorithm (PPA) is an essential operation in a number of real-world applications, including games, mobile robotics, unmanned aerial vehicles (UAVs), robotic arms, geographical maps, computer networks, sensor networks, resource allocation planning, smart vehicles, etc. [1,2]. The primary objective of PPA is to find the optimal path between source and destination nodes in an environment of nodes represented as a graph. Despite the number of algorithms of PPA that exist in the literature, most of them stem from Dijkstra's original algorithm [3]. Although Dijkstra's algorithm guarantees a globally optimal shortest distance between a source and a destination, it can be intensive in space and time. PPA has been studied for decades, and many solutions, including Dijkstra's, have been proposed. Examples of these solutions include classical Dijkstra's algorithm, A* path finding, ant colony, support vector machines, genetic algorithms, rapid random trees (RRT), local GPS, fuzzy logic, artificial potential fields, image processing, and sensor networks [2,4–15].

A typical environment of a path planning problem is depicted in **Figure 1**. It can be observed that the environment consists of a number of elements. It includes an agent that may be a robot, a vehicle, a drone, etc. This agent exists in this environment and is required to navigate through it from a source location to a destination location. This traversal is due to tasks that need to be accomplished by the agent in response to an event, command, etc. Also, the environment includes a set of obstacles that blocks the way of the agent. In our context, these obstacles are assumed to be static. The starting

and the target locations are known to the agent. Finally, the entire map of the environment is known to the agent.

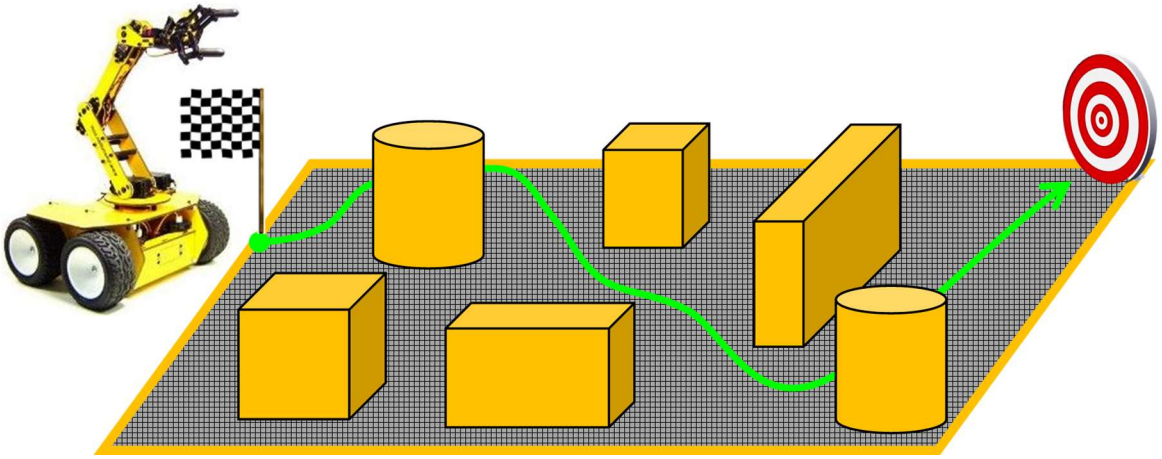


Figure 1. Typical environment of path planning problem.

Since the invention of Dijkstra's algorithm, many attempts have been proposed to improve its performance. The most popular algorithm that was proposed to improve the time and space complexity of Dijkstra's algorithm is the A* algorithm [16]. The improvement that A* made on the original Dijkstra's algorithm is the addition of a heuristic function that significantly pruned the useless paths to consider in the search space. In the technical background, there will be a more detailed and concrete discussion on how these algorithms work and what our proposed improvement is.

2. Closely related literature

The work presented a proposed method to improve the efficiency of the A* algorithm based on topological maps [17]. The paper used 2D lidar to construct the map of the environment. Furthermore, a depth camera was used to detect objects in the scene. The authors reported that their experimental work showed that the proposed method is reliable and more efficient than the original A* algorithm. A path planning technique for unmanned ships was presented by [15]. The authors assumed a known map of the environment. The authors compared the classical Dijkstra algorithm with the ant colony algorithm and concluded that their method was able to improve the efficiency of the path planning problem. The work of [7]. proposed a method to improve the A* planning algorithm for robot path planning. The authors provided a simulated experimental work that demonstrated that their proposed method is effective to produce better results than the classical A* algorithm. An improved version of the A* algorithm was used in the work of [14] for service robot path planning in restaurants. The authors reported that their method was able to avoid crowded channels and improved the effectiveness of the classical A* algorithm. Genetic algorithms (GA) were used in the work of [8] to improve the efficiency of the A* algorithm for UAVs. The authors used simulated experiments to validate their proposed method and found that it is reliable and efficient. An improved version of A* based on the dynamic window approach was proposed by [18]. The proposed method eliminated redundant path nodes, and the simulated experimental work demonstrated its effectiveness and

efficiency. A smart wheelchair system was enabled with a path finding algorithm based on the A* algorithm by [19]. The authors validated the proposed system using a simulated experimental work and found that the method is reliable. The work of [20] provided a solution for the deadlock problem in the A* algorithm when the graph is disconnected. The authors used picture matching and the A* to detect a deadlock scenario. An image processing technique for robot path finding was proposed [5]. The authors reported that their method is more effective and more efficient than current methods, as it exploits the visibility of the entire map rather than physically probing possible paths and backtracking. The work in [21] argued that many path finding algorithms are not sufficient for real-world scenarios of traffic road maps. The authors proposed more restrictions such as time duration, intersections, and lane changes. It was reported that via simulation experimental work, the proposed method was effective and reliable. A depth-first search and sub-region path planning method was introduced by [22]. The main improvement of the new method was to reduce the excessive number of turns in the paths of agricultural robots. The authors used simulated experimental work to validate their proposed method and found it to be effective. The work in [23] proposed an improvement on the A* algorithm from different aspects, including a bi-directional search and turn smoothing. The simulated and real-world experimental work showed that the proposed method is effective and efficient.

3. Technical background

In this section, the theory and fundamentals of relevant concepts and algorithms are presented. These include graphs, Dijkstra's algorithm, the A* algorithm, and our proposed modified A* algorithm.

The graph modeling of the problem of path finding is depicted in **Figure 2**. It can be observed that the graph is an undirected (bidirectional) graph and consists of a set of nodes, weighted undirected edges, a source node, and a destination node. Although graphs can be directed, our investigation will consider the less restrictive set of undirected graphs. This is because undirected graphs represent real-world scenarios more closely.

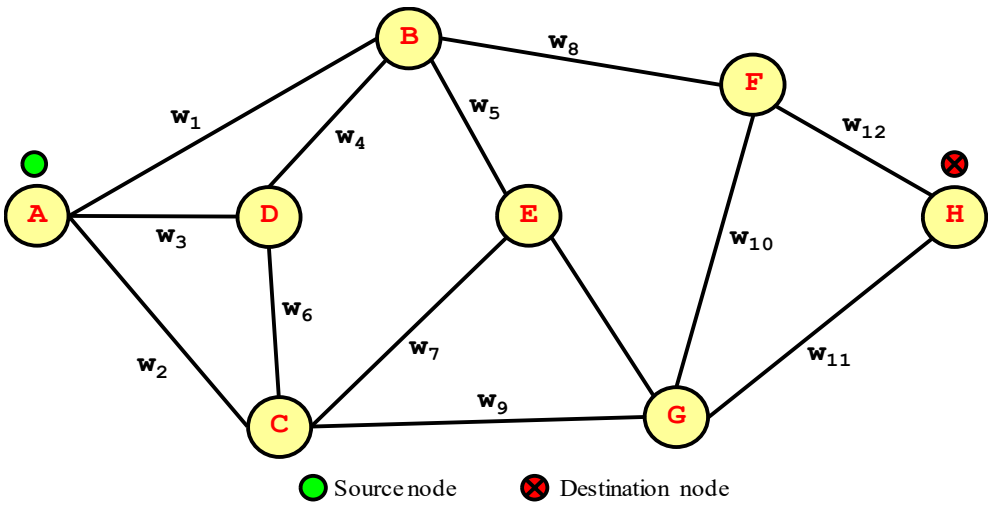


Figure 2. Graph modeling of the path finding algorithm.

A graph can be mathematically defined as a finite set of vertexes, a finite set of edges, each of which has a weight, and an incidence function that defines the connectivity among all vertexes. This concept is expressed in Equation (1).

$$G = (V, E, W) \tag{1}$$

3.1. Dijkstra shortest path algorithm

A path in a graph is defined as a finite sequence of distinct vertexes connecting a source vertex to a destination vertex [24]. This definition is given in Equation (2).

$$P(v_1, v_n) = (v_1, v_2, \ldots, v_{n-1}, v_n) \tag{2}$$

In a given graph, there are a finite number of possible paths from a source to a destination. The shortest path finding problem is interested in searching for the shortest path in this finite set of paths. Dijkstra's algorithm defines a cost function $g(x)$ based on the weight of the edge. This cost function is the basis for finding the next vertex to expand locally and eventually finding the best sequence of edges that forms the optimal path. The A* algorithm adds a heuristic function that improves both time and space complexities of the original Dijkstra's algorithm. The cost function of the A* algorithm is defined in Equation (3).

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

where:

- $f(v_i)$: the overall cost function.
- $g(v_i)$: the cumulative weight from source to v_i vertex.
- $h(v_i)$: the heuristic cost from v_i vertex to destination.

Typical heuristic functions used in A* include: Euclidean distance, Manhattan distance and diagonal distance defined in Equations (4)–(6) respectively.

$$d_{euc}(v_1, v_2) = \sqrt{(v_x^1 - v_x^2)^2 + (v_y^1 - v_y^2)^2} \tag{4}$$

$$d_{man}(v_1, v_2) = |v_x^1 - v_x^2| + |v_y^1 - v_y^2| \tag{5}$$

$$d_{dia}(v_1, v_2) = \max(|v_x^1 - v_x^2|, |v_y^1 - v_y^2|) \tag{6}$$

In Equations (4)–(6), v^1 is the current vertex and v^2 is the destination vertex. Furthermore, in all cases, it is assumed that moving from one cell to another costs the same in any direction. This is the case for 2D grids, which we assume in our investigation. Equation (3) can be generalized to account for weighted heuristic cost. As it stands, the weight of $g(x)$ and $h(x)$ is assumed to have the same influence. If it is desirable, this influence may be weighted differently and will impact the effectiveness and efficiency of the outcomes of the A* algorithm. Therefore, Equitation (7) expresses this general concept of weighted cost function.

$$f(v_i) = \alpha g(v_i) + (1 - \alpha) h(v_i) \tag{7}$$

where:

- α: a weight factor $\in [0, 1]$

According to Equation (7), if $a = 0$, A* goes back to Dijkstra's original algorithm. On the other hand, if $a = 1$, A* completely ignores the $g(x)$ term and uses only the $h(x)$ term. In the first scenario, A* is very slow but accurate, and in the second scenario, A* is very fast but inaccurate. We will show an empirical investigation of these scenarios in the experimental work section.

The graph in **Figure 2** represents the general case of any path finding environment. A special common case uses a grid-based layout. In this grid, the vertex space is broken down into cells where each cell is surrounded by an 8-cell neighborhood as shown in **Figure 3**. We label these neighbors as the set of the possible movement directions of (N "north", NE "north-east", E "east", SE "south-east", S "south", SW "south-west", W "west", NW "north-west").

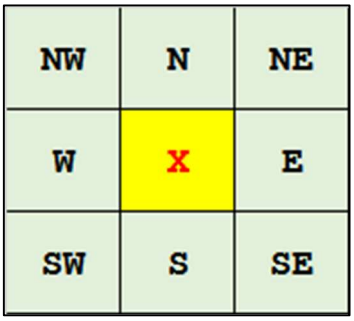


Figure 3. Neighborhood of a current cell "*x*" in a grid-based path finding environment.

A typical grid-based environment is shown in **Figure 4**. It is to be observed that color codes are significant in this map, as every color indicates the type of cell, such as ground, obstacle, open, closed, path, etc.

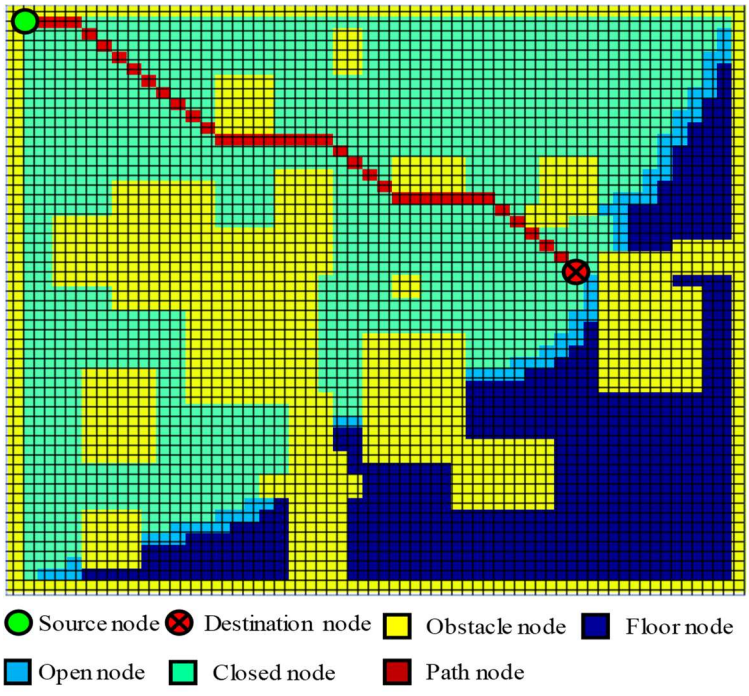


Figure 4. Typical grid-based environment for path finding algorithm.

Since the original Dijkstra's and the A* algorithms are very similar and they have only one difference, that is, the heuristic function $h(x)$, we provide one flowchart for A* that includes Dijkstra's as well. The flowchart is depicted in **Figure 5**.

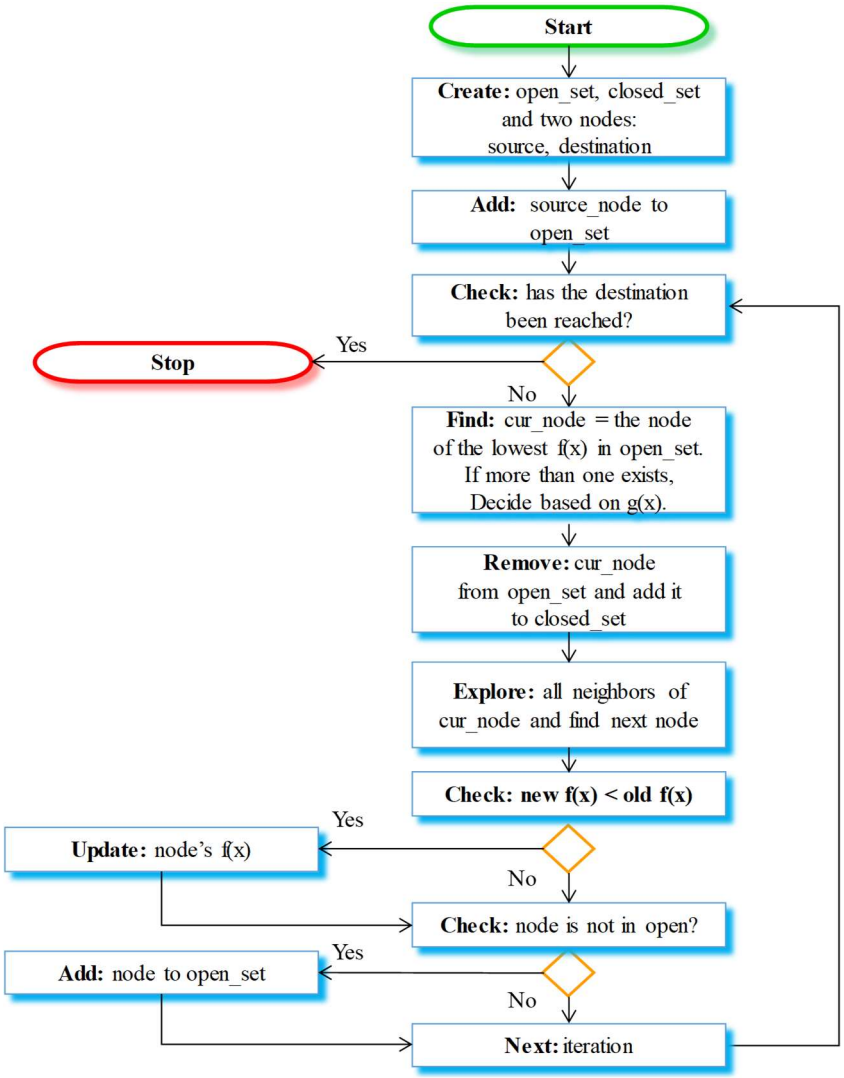


Figure 5. The flowchart of Dijkstra's algorithm for shortest path finding.

3.2. Space and time complexity

Our main focus in this study is to improve time and space complexities of the A* algorithm. As explained earlier, A* already improved the original Dijkstra's algorithm by adding a heuristic function to eliminate useless path explorations. The upper bound of Dijkstra's algorithm is given in Equation (8), assuming the use of a min-heap data structure.

$$T(E,V) = O((E+V)logV) \tag{8}$$

where:

- T: time complexity.
- E: number of edges in the graph.
- V: number of vertexes in the graph.

The complexity of A* algorithm is given in Equation (9).

$$T(E,V) = O(b^d) \tag{9}$$

where:

- b: number of expanded vertexes at each node.
- d: number of vertexes in the found optimal path.

We observe that there is an opportunity to improve both space and time complexities of the A* algorithm. For the space complexity, we do not use open and closed sets to store the current state of the algorithm. Instead, we sacrifice some accuracy for time efficiency. We will prove empirically that this sacrifice is not significant in the experimental work. On the other hand, we gain significantly on the running time efficiency.

Our proposed algorithm consists of two passes, forward and backward passes. In the forward pass we concentrate on reaching the distention as fast as possible and using the number of nodes as low as possible. We recognize that, in the forward pass, the algorithm makes bad decisions and/or has a broken path sometimes. Therefore, we devised a backward pass to fix these two problems. The pseudocode of the two passes is given in Algorithms 1 and 2, respectively.

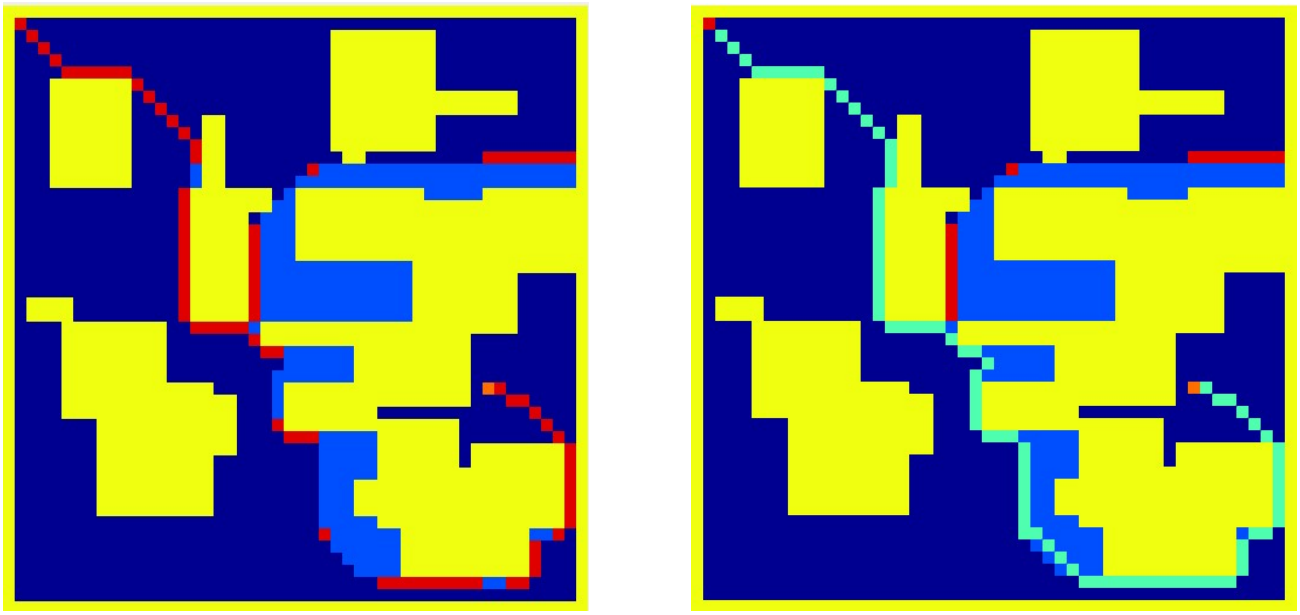
Algorithm 1 Pseudo code of the forward pass of the proposed algorithm

```
1: AA*Forward(map, src, dst)
2:     while (src.loc NE dst.loc)
3:         lst_nbr = get_neighbors(src_loc)
4:         for_each(cell in lst_nbr )
5:             if cell EQ obstacle
6:                 g(x) = ∞
7:             Else
8:                 if cell ∈ {NW, NE, SW, SE}
9:                     g(x) = g(x) + 1.4142
10:                End
11:            End
12:        End
13:        h(x) = Euclidian(x, dst)
14:        f(x) = g(x) + h(x)
15:        min_x = min(lst_nbr, f(x))
16:        if |min_x| > 1
17:            x = min(min_x, g(x))
18:        Else
19:            x = min_x[1]
20:        End
21:        map(x) = src_color
22:        map(src_color) = path_color
23:        AA*Backward(map, dst, src)
```

Algorithm 2 Pseudo code of the backward pass of the proposed algorithm

```
1: AA*Backward (map, src, dst)
2:     while (src.loc NE dst.loc)
3:         lst_nbr = get_neighbors(src_loc)
4:         for_each(cell in lst_nbr )
5:             if cell EQ obstacle OR has new_path_color
6:                 g(x) = ∞
7              Else
8                  if cell ∈ {NW, NE, SW, SE}
9:                     g(x) = g(x) + 1.4142
10:                Else
11:                    g(x) = g(x) + 1
12:                End
13:                If (cell == path_color)
14:                    g(x) = 0.1 * g(x)
15:                Else
16:                    g(x) = 0.5 * g(x)
17:            End
18:            h(x) = Euclidian(x, dst)
19:            f(x) = g(x) + h(x)
20:        End
21:        min_x = min(lst, f(x) )
22:        if |min_x| > 1
23:            x = min(min_x, g(x))
24:        Else
25:            x = min_x[1]
26:        End
27:        map(x) = new_path_color
```

To demonstrate how the AA* algorithm works, show the forward and backward passes in **Figure 6**. Looking carefully at the forward pass, we can observe that AA* reached the target location but with a broken path, as the number of red-highlighted pieces of the path is visible. This is the role of the forward pass. The backward pass kicks in right after the forward pass has finished at the destination node. The role of the backward pass is to bridge these pieces together, keeping in mind the same heuristic used in the forward pass. The work of the backward pass is easier since most of the work has been done. In Lines 4–15 of the Backward pass, it gives preference to the already marked path cells in the Forward pass. Furthermore, it favors the explored cells over ground cells in case the cell was not already marked. Also, the backward pass forbids following a cell if it is an obstacle or already bridged.



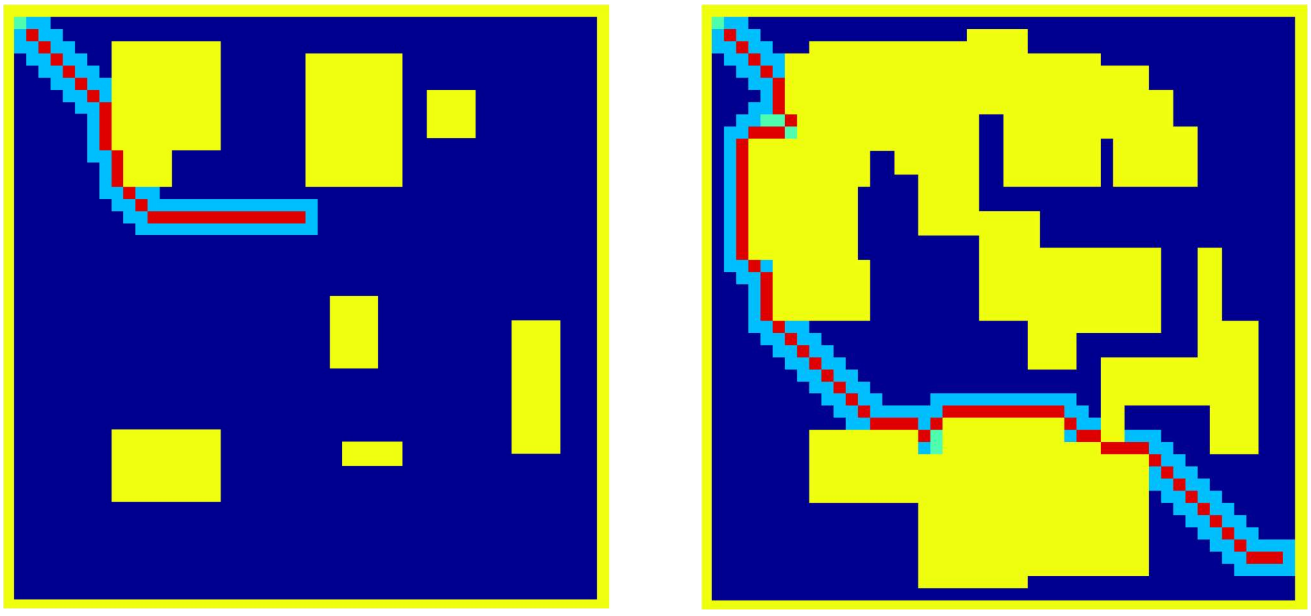
(a) **(b)**

Figure 6. (a) Sample of AA*Forward pass; **(b)** AA*Backward pass.

4. Experimental work

In this section, we present a simulator that we created to conduct our experiments, results and observations on the experiments and a discussion on the results.

4.1. Simulator



(a) **(b)**

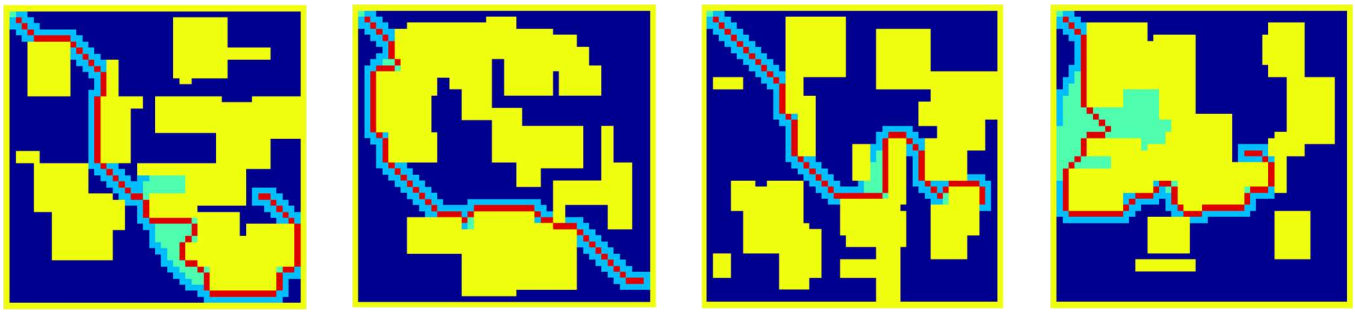
Figure 7. Sample of **(a)** simple environment; **(b)** complex environment.

To be able to conduct our experimental work, we created a 2D simulator using [25]. The simulator was effective and helpful to experiment with different parameters

and visualize the details of path finding algorithms, namely Dijkstra's, A*, and AA*. A sample view of the simulator can be seen in **Figure 4**, and it will be shown in several other figures below in the experiments subsection. Our simulator is different from others in the literature because it can automatically generate an environment with random shapes of obstacles that resembles real-world structures or buildings. The number and size of these obstacles can be set by the user so the user may test simple to challenging environments. An example of the two scenarios is shown in **Figure 7**. Other simulation environments can be found in the work of [26,27].

4.2. Experiments

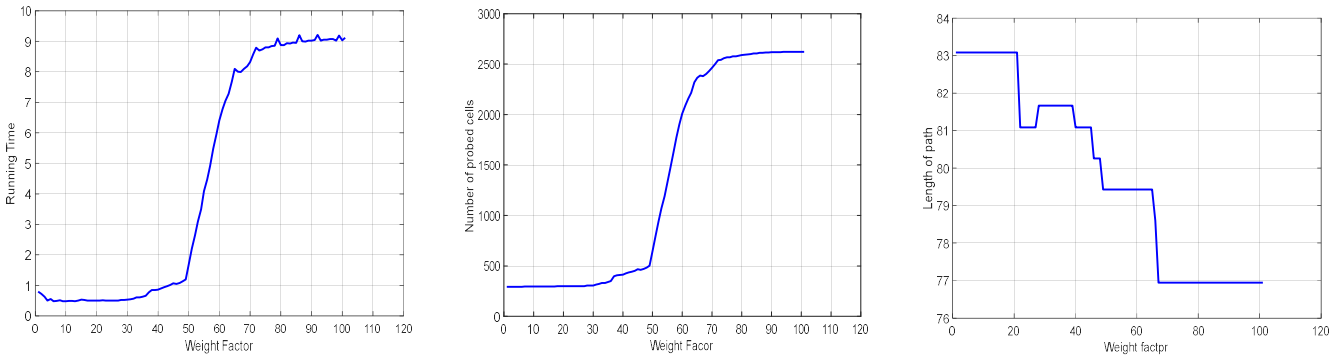
Our experiments were performed in two stages. The first stage addressed Equation (7) as the weight factor α was graduated between 0–1 in 0.01 increments using 5 different environments shown in **Figure 8**.



(a) **(b)** **(c)** **(d)**

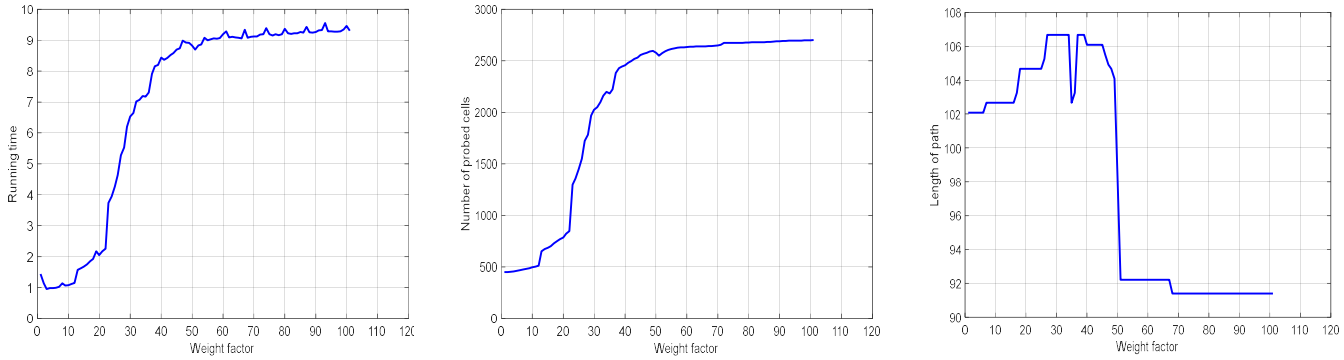
Figure 8. Used environments in stage 1 of the experimental work.

In each experiment we performed, three performance metrics were measured as follows: execution time of the algorithm, number of probed cells, and length of the found path. **Figures 9–13** show the results of 500 experiments using 5 different complex environments shown earlier in **Figure 8**.



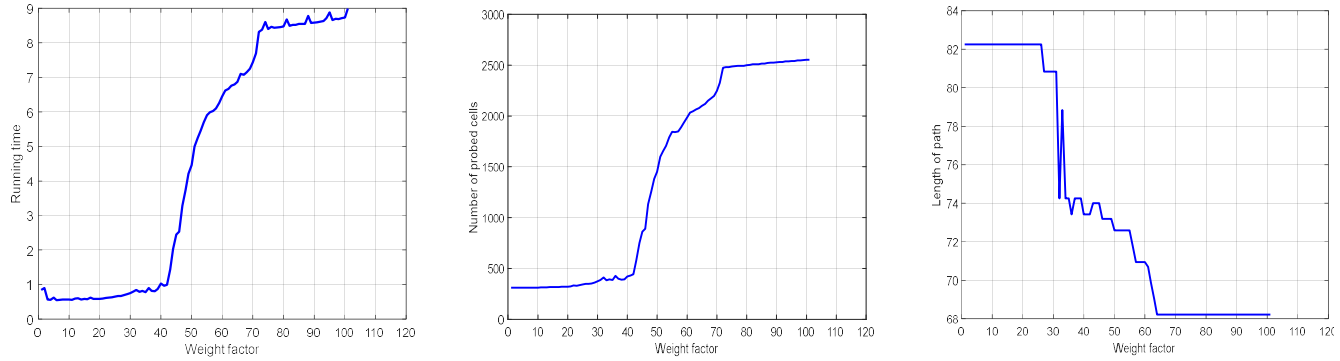
(a) **(b)** **(c)**

Figure 9. Experimental results of 1st environment: running time, number of probed cells and length of found path.



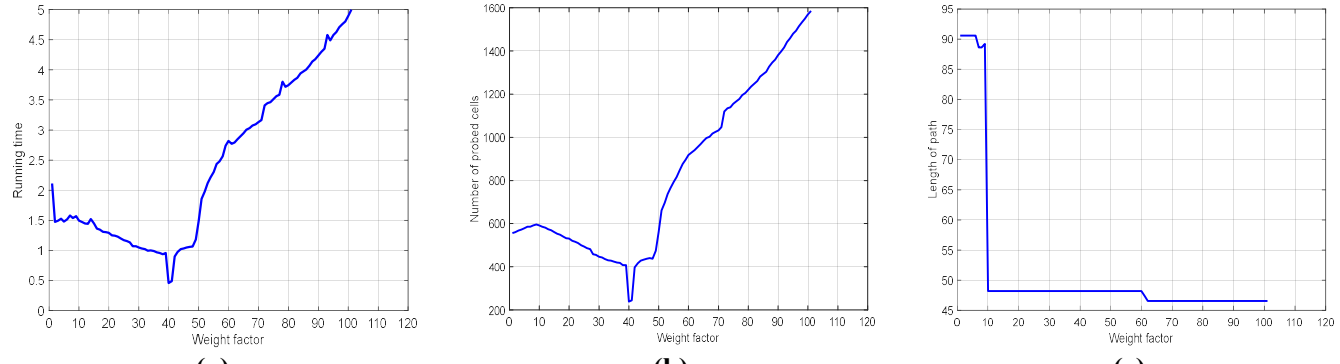
(a) **(b)** **(c)**

Figure 10. Experimental results of 2st environment: running time, number of probed cells and length of found path.



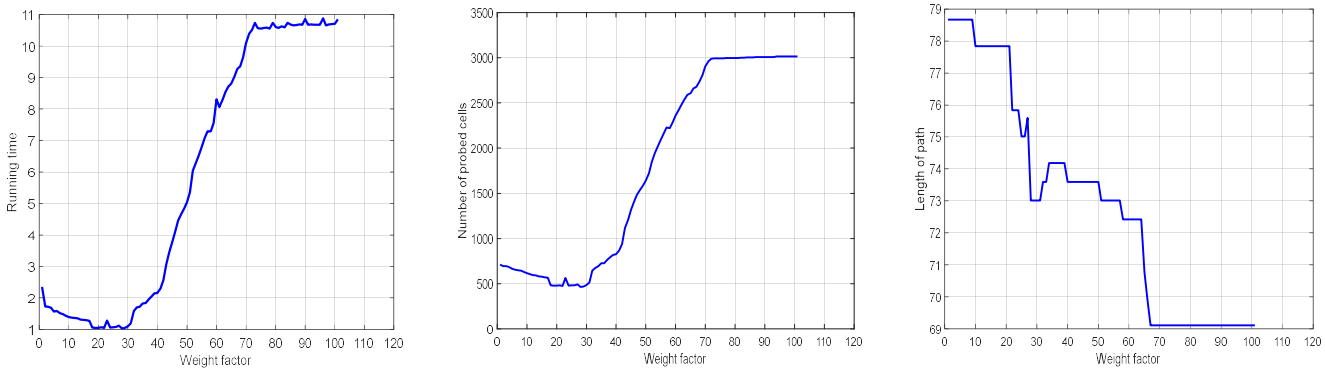
(a) **(b)** **(c)**

Figure 11. Experimental results of 3st environment: running time, number of probed cells and length of found path.



(a) **(b)** **(c)**

Figure 12. Experimental results of 4st environment: running time, number of probed cells and length of found path.



(a) **(b)** **(c)**

Figure 13. Experimental results of 5st environment: running time, number of probed cells and length of found path.

The second stage of our experimental work addressed our newly proposed AA* algorithm, which improves the performance of the A* algorithm. We conducted 500 experiments to investigate the performance of our AA* compared to A*. **Figures 14–17** show all the results of the aforementioned metrics, and **Tables 1** and **2** aggregate these results as the mean and the standard deviation for more convenient comparison.

Furthermore, the AA* algorithm was tested using challenging environments by simulating 2500 experiments to see the success/failure rate. A sample experiment is shown in **Figure 12,** and the results of all 2500 experiences are plotted in **Figure 14**. The results are further discussed in the next section of the discussion of results.

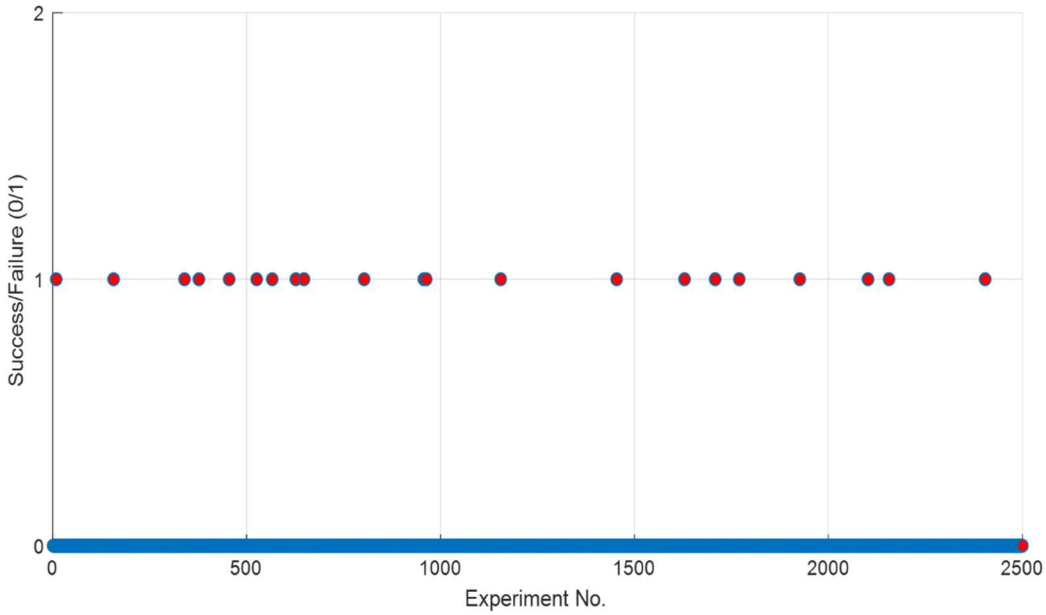


Figure 14. Testing AA* for success/failure in challenging environments: 0 = success, 1 = failure.

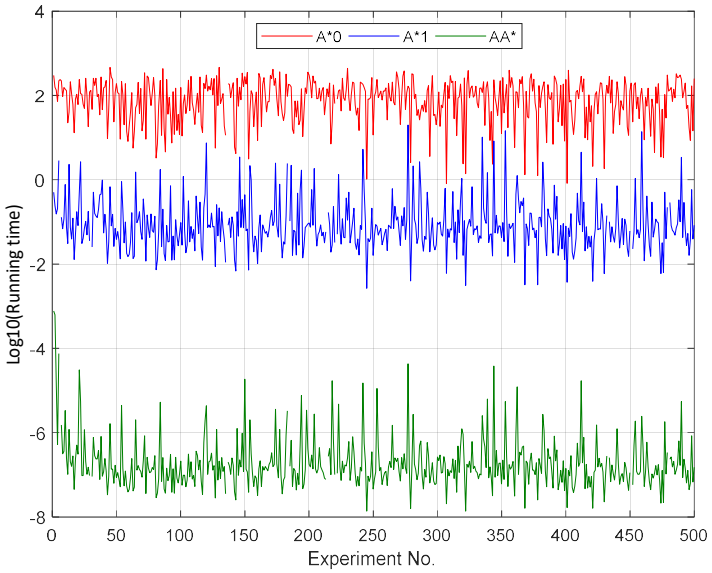


Figure 15. A*0, A*1 and AA* compared based on running time.

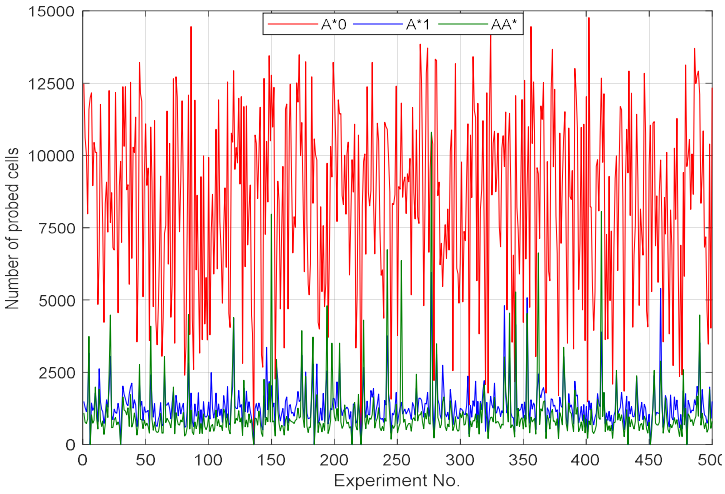


Figure 16. A*0, A*1 and AA* compared based on number of probed cells.

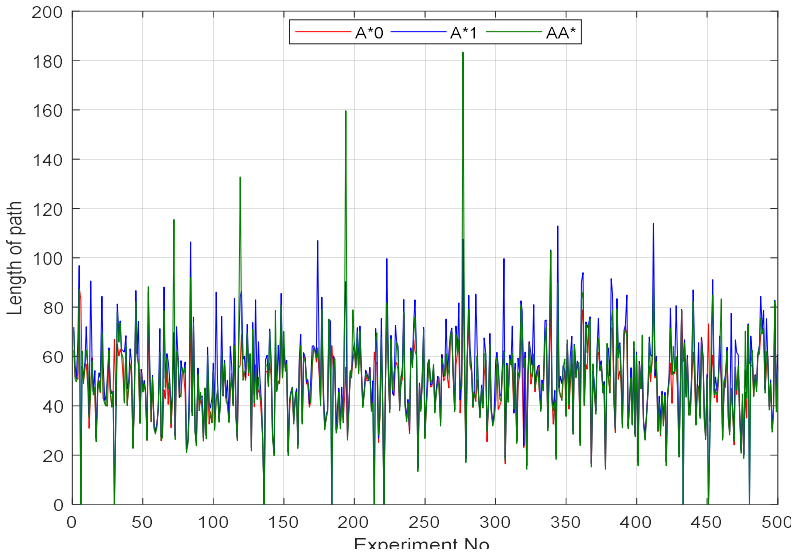


Figure 17. A*0, A*1 and AA* compared based on length of path.

Table 1. Summary of mean results in **Figures 14–16**.

Algorithm	Running time (sec)	Number of probed cells (each)	Length of path (each)
A*0	7.2230	8448	48
A*1	0.4200	1339	52
AA*	0.0016	1059	49

Table 2. Summary of standard deviation results in **Figures 14–16**.

Algorithm	Running time (sec)	Number of probed cells (each)	Length of path (each)
A*0	3.058	3055.8	13.89
A*1	0.406	688.5	18.91
AA*	0.003	1079.6	19.00

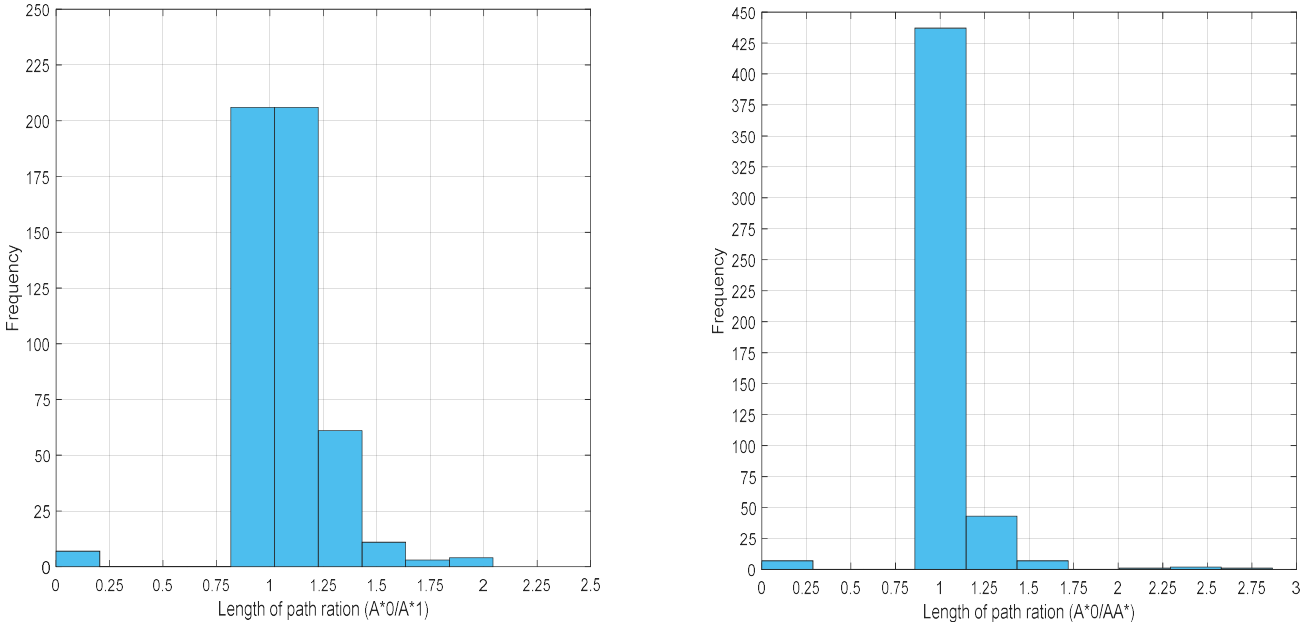
4.3. Discussion of results

In this section, we discuss our observations on the results of all conducted experimental work and attempt to explain our findings.

1) In the weighted cost A* experiments, all results are consistent with our expectations. This can be seen in **Figures 9–13**.
2) For the A* runtime results, all results in **Figures 9–13** showed that running time increases by increasing the value of *a*.
3) For the A* number of cells results, all results in **Figures 9–13** showed that the number of probed cells increases by increasing the value of *a*.
4) For the A* length of path results, all results in **Figures 9–13** showed that the length of the path decreases by increasing the value of *a*.
5) The observation in 4 indicates that although Dijkstra's algorithm (A*0) is inefficient in terms of running time and number of probed cells, it always produces the shortest path.
6) In **Table 1**, the mean summary of **Figures 9–13** indicates that our proposed algorithm AA* is superior in running time as it scored a mean running time of 0.0016 sec, which is 262.5% faster than A*.
7) In **Table 1**, the mean summary of **Figures 9–13** indicates that our proposed algorithm AA* is superior in number of probed cells, as it scored a mean number of probed cells of 1059 cells, which is 26.44% fewer probed cells than A*.
8) In **Table 1**, the mean summary of **Figures 9–13** indicates that our proposed algorithm AA* is superior in length of path, as it scored a mean length of path of 1059 cells, which is 6.12% fewer cells in the path than A*. Furthermore, this metric indicates that our AA* algorithm is suboptimal compared to the A*0 (original Dijkstra) as the length of the path is longer than the optimal found in A*0.
9) In **Table 1**, the standard deviation summary of **Figures 9–13** indicates that our proposed algorithm AA* is the most stable in running time, as it has the lowest scatter compared to A*0 and A*1. In fact, the STD of AA* is 135.33% better than that of A*.
10) Our experimental work provides empirical evidence that the proposed algorithm AA* is effective by achieving a sub-optimal path compared to A*0 but better than the A*1 algorithm. Furthermore, besides demonstrating a competitive length

of path, the experimental work provided evidence that our algorithm is superior in running time as well.

11) According to the histogram in **Figure 18a**, our calculations showed that the A*1 algorithm has a length of path accuracy of 91.19% compared to the optimal path of the original Dijkstra's A*0 algorithm.
12) According to the histogram in **Figure 18b**, our calculations showed that the AA* algorithm has a length of path accuracy of 96.95% compared to the optimal path of the original Dijkstra's A*0 algorithm.
13) It is to be reported that we observed that our proposed algorithm occasionally fails to find a path in very complex obstacle settings while A*0 and A*1 never fail. That is the only disadvantage we could report based on our extensive experimental validation. An example of this scenario is demonstrated in **Figure 19**.



(a) **(b)**

Figure 18. Length of path ration comparison. **(a)** A*0/A*1; **(b)** A*0/AA*.

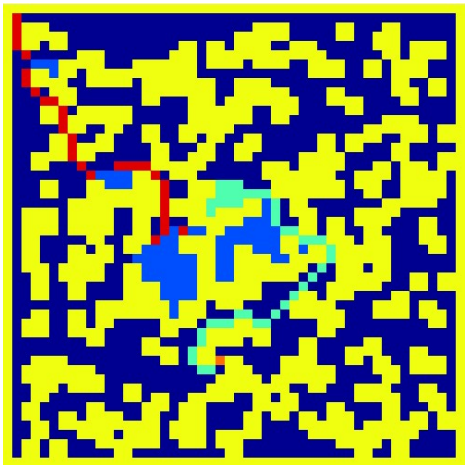


Figure 19. Example of occasional failure of the AA* algorithm in a very complex environment.

14) Further investigation of point# 13 above, **Figure 14** shows extensive testing of this test using 2500 experiments with various challenging environments as shown **Figure 19** and the results showed that the failure rate was 21/2500 = 0.0084 = 0.084%. Therefore, the success rate was 99.16% which undeniably remarkable.

5. Conclusion

We have presented a new proposed modification to the A* path finding algorithm, and we have demonstrated the effectiveness and efficiency. The new algorithm is named AA*. Two passes are needed to accomplish AA* objectives. In the forward pass, we employ the strategy of A* without the space and effort spent on the two traditional sets (open and closed) needed to maintain the state of the A* algorithm. The AA* algorithm, by not maintaining the state, suffers some bad decisions and broken paths. These problems are solved in the backward pass. The role of the backward pass is to bridge the broken pieces of the generated, possibly incomplete path and fix the bad decisions made in the forward pass. To validate our proposed modification, a simulator was created that turned out to be effective and helpful for this domain. A total of 500 simulated experiments were conducted, and their results showed that the new algorithm AA* is superior to the A* algorithm in the three typical metrics of running time, number of probed cells, and length of path with an accuracy of 96.95% compared to the optimal path of the original Dijkstra's algorithm. Furthermore, we ran another 500 simulated experiments to investigate the influence of the weight factor of the heuristic function and reported interesting results and observations. One minor drawback of our proposed algorithm is the occasional failure to find the path in a very complex setting of obstacles, but this finding was verified to be insignificant as the success rate was outstanding at 99.16%.

Conflict of interest: The author declares no conflict of interest.

References

1. Rababaah H, Shirkhodaie A. Guard Duty Alarming Technique (GDAT): A Novel Scheduling Approach for Target-tracking in Large-scale Distributed Sensor Networks. In: Proceedings of 2007 IEEE International Conference on System of Systems Engineering; 16–18 April 2007; San Antonio, USA. pp. 1–6.
2. Rababaah AR. Sensor networks simulation framework for target tracking applications: SN-SiFTTA. International Journal of Web Engineering and Technology. 2021; 16(2): 113. doi: 10.1504/ijwet.2021.117767
3. Dijkstra EW. A note on two problems in connexion with graphs. Numerische Mathematik. 1959; 1(1): 269–271. doi: 10.1007/bf01386390
4. Afakh ML, Masudi MI, Ardilla F, et al. Bicycle Path Planning on Omnidirectional Mobile Robot Using Fuzzy Logic Controller. In: Proceedings of 2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE); Bali, Indonesia. pp. 237–241.
5. Ambeskar A, Turkar V, Bondre A, et al. Path finding robot using image processing. In: Proceedings of 2016 International Conference on Inventive Computation Technologies (ICICT). pp. 1–6.
6. Chen J, Ye F, Jiang T. Path planning under obstacle-avoidance constraints based on ant colony optimization algorithm. In: Proceedings of 2017 IEEE 17th International Conference on Communication Technology (ICCT); 27–30 October 2017; Chengdu, China. pp. 1434–1438.
7. Ju C, Luo Q, Yan X. Path Planning Using an Improved A-star Algorithm. In: Proceedings of 2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan); 23–25 October 2020; Jinan, China. pp. 23–26.

8. Ma N, Cao Y, Wang X, et al. A Fast path re-planning method for UAV based on improved A* algorithm. In: Proceedings of 2020 3rd International Conference on Unmanned Systems (ICUS); Harbin, China. pp. 462–467.
9. Qiao S, Zheng K, Wang G. A Path Planning Method for Autonomous Ships Based on SVM. In: Proceedings of 2020 Chinese Control and Decision Conference (CCDC); 22–24 August 2020; Hefei, China. pp. 3068–3072.
10. Rababaah A, Kuscu E, Shirkhodaie A. Indoor Mobile Robot Localization Using IPS Cricket Technology. Journal of Global Information Technology (JGIT). 2014; 9(1): 18–23.
11. Shen Z, Hao Y, Li K. Application research of an Adaptive Genetic Algorithms based on information entropy in path planning. In: Proceedings of the 2010 IEEE International Conference on Information and Automation; 20–23 June 2010; Harbin, China. pp. 2013–2016.
12. Szczepanski R, Tarczewski T, Erwinski K. Energy Efficient Local Path Planning Algorithm Based on Predictive Artificial Potential Field. IEEE Access. 2022; 10: 39729–39742. doi: 10.1109/access.2022.3166632
13. Tusi Y, Chung HY. Using ABC and RRT algorithms to improve mobile robot path planning with danger degree. In: Proceedings of 2016 Fifth International Conference on Future Generation Communication Technologies (FGCT); 17–19 August 2016; London, UK. pp. 21–26.
14. Yang R, Cheng L. Path Planning of Restaurant Service Robot Based on A-star Algorithms with Updated Weights. In: Proceedings of 2019 12th International Symposium on Computational Intelligence and Design (ISCID); Hangzhou, China. pp. 292–295.
15. Zhu Z, Li L, Wu W, et al. Application of improved Dijkstra algorithm in intelligent ship path planning. In: Proceedings of 2021 33rd Chinese Control and Decision Conference (CCDC); 22–24 May 2021; Kunming, China. pp. 4926–4931.
16. Hart P, Nilsson N, Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics. 1968; 4(2): 100–107. doi: 10.1109/tssc.1968.300136.
17. Kuang H, Li Y, Zhang Y, et al. Improved A-star Algorithm based on Topological Maps for Indoor Mobile Robot Path Planning. In: Proceedings of 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC); 4–6 March 2022; Chongqing, China. pp. 1236–1240.
18. Sun T, Wang T, Sun P. Mobile Robot Dynamic Path Planning Based on Improved A* Algorithm. In: Proceedings of 2021 3rd International Conference on Robotics and Computer Vision (ICRCV); Beijing, China. pp. 24–29.
19. Şahın Hİ, Kavsaoğlu AR. Indoor Path Finding and Simulation for Smart Wheelchairs. In: Proceedings of 2021 29th Signal Processing and Communications Applications Conference (SIU); Istanbul, Turkey. pp. 1–4.
20. Hu Z, Li J. Application of Maintaining the Shortest Path Method in the Game Map Path-Finding. In: Proceedings of 2010 International Conference on Computational Aspects of Social Networks; Taiyuan, China. pp. 737–740.
21. Kim OTT, Nguyen VD, Moon S, Hong CS. Finding realistic shortest path in road networks with lane changing and turn restriction. In: Proceedings of the18th Asia-Pacific Network Operations and Management Symposium (APNOMS), 2016. pp.1–4.
22. Zuo G, Zhang P, Qiao J. Path planning algorithm based on sub-region for agricultural robot. In: Proceedings of the 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010). pp. 197–200.
23. Wang H, Qi X, Lou S, et al. An Efficient and Robust Improved A* Algorithm for Path Planning. Symmetry. 2021; 13(11): 2213. doi: 10.3390/sym13112213
24. Johsonbaugh R. Discrete Mathematics, 8th ed. Pearson; 2018.
25. Matlab. 9.4.0.813654 (R2018a). USA: The MathWorks Inc; 2018.
26. Shirkhodaie A, Rababaah H. “Multi-layered context impact modulation for enhanced focus of attention of situational awareness in persistent surveillance systems”, Proc. SPIE 7710, Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications. 2010; 771009. doi: 10.1117/12.850795.
27. Rababaah A, As’ad A, Sultan A, et al. Development of Robotic Model to Support Intelligent Vehicles Behaviors. Global Journal of Modeling and Computational Intelligence (GJMCI). 2021; 1(1): 50–59.