WWW.IJRPR.COM

**International Journal of Research Publication and Reviews**

Journal homepage: www.ijrpr.com ISSN 2582-7421

# Comprehensive Review of Graph Search Path Planning Algorithms

***Ms. Priyanka Chemudugunta[1], Madhan. E [2], Jibin B Mathew [3], Pranav Anandkumar [4], Dhanabalan. B[5]***

[1]Assistant Professor, Department of Robotics and Automation Engineering, Karpaga Vinayaga College of Engineering and Technology, Chengalpattu District, Tamilnadu, India.

[2,3,4,5]Undergraduate student, Department of Robotics and Automation Engineering, Karpaga Vinayaga College of Engineering and Technology, Chengalpattu District, Tamilnadu, India.

**A B S T R A C T**

Path planning is a fundamental challenge in robotics and autonomous systems, requiring efficient algorithms to navigate from a start point to a destination while avoiding obstacles. This review focuses on two widely used graph search algorithms: A* and D*. Both algorithms employ graph based representations of the environment and heuristics to determine optimal paths. A* is known for its effectiveness in static environments, where it guarantees the shortest and most efficient path if the heuristic is admissible. It balances path cost and heuristic estimates to ensure optimal navigation. However, A* struggles in dynamic environments, where obstacles may change or appear unexpectedly. To address this limitation, the D* algorithm was developed, extending the capabilities of A* to handle dynamic or partially known environments. D* allows mobile robots to replan paths in real time, adjusting for changes in the environment without needing to recalculate the entire route. This adaptability makes it ideal for robots navigating in unpredictable environments. Variants of D*, such as D* Lite and Focused D*, further improve re planning efficiency by minimizing computational overhead. Overall, A* and D* are key algorithms in robotic path planning, with A* excelling in static settings and D* providing robust solutions for dynamic environments.

Keywords: Path planning algorithm, Graph search algorithms, A*algorithm, D*algorithm,

## 1. Introduction

Path planning is a crucial aspect in mobile robots focusing on the process of determining a feasible route for a robot or autonomous agent to navigate from a starting point to a designated goal while avoiding obstacles. The challenge of path planning arises from the complexity of real world environments, which can be cluttered, dynamic, and uncertain. Robots often need to operate in spaces where obstacles can appear unexpectedly, and they must adapt their paths accordingly. Therefore, an effective path planning algorithm must not only find a path efficiently but also ensure that the path is safe and optimal in terms of distance, time, or energy consumption. Path planning algorithms can be broadly classified into three main categories based on their methodologies: graph search algorithms, sampling based algorithms, and sensor based algorithms.[1]

Graph Search Algorithms: These algorithms utilize a discretized representation of the environment, often modeled as a graph or grid. They aim to explore the graph and find the optimal path by minimizing a cost function. Two notable examples of graph search algorithms are the A* algorithm, which efficiently navigates static environments, and the D* algorithm, which is designed for dynamic settings where obstacles may change over time.[2]

## 2. GRAPH SEARCH ALGORITHMS

Graph search algorithms are widely used in mobile robotics for path planning, especially when navigating complex environments. In the context of mobile robots, the environment can be modeled as a grid or a more complex graph. Each node in the graph corresponds to a specific position that the robot can occupy, while edges represent the possible movements between those positions. The weight of an edge can signify the cost of moving from one node to another, which may depend on various factors such as distance, terrain type, or the presence of obstacles.

### *2.1. A*Algorithm*

A * Algorithm is one of the mostb widely used and effective path planning algorithms in robotics. It is a graph search algorithm that finds the shortest path between two points, often used in environments where robots must navigate from a start position to a target position while avoiding obstacles. The A* Algorithm is an extension of Dijkstras algorithm enhanced by the use of heuristics to improve efficiency. In robotics environments are often shown

as grids or graphs , where each position ( or node) is connected to other possible positions. A* Algorithm is more efficiently and widely used approaches in both 2D and 3D Path Planning.[3]

### *2.1.2. Working of A* Algorithm*

The A* Algorithm searches for a path from the start node to the goal node with the help of above following steps.

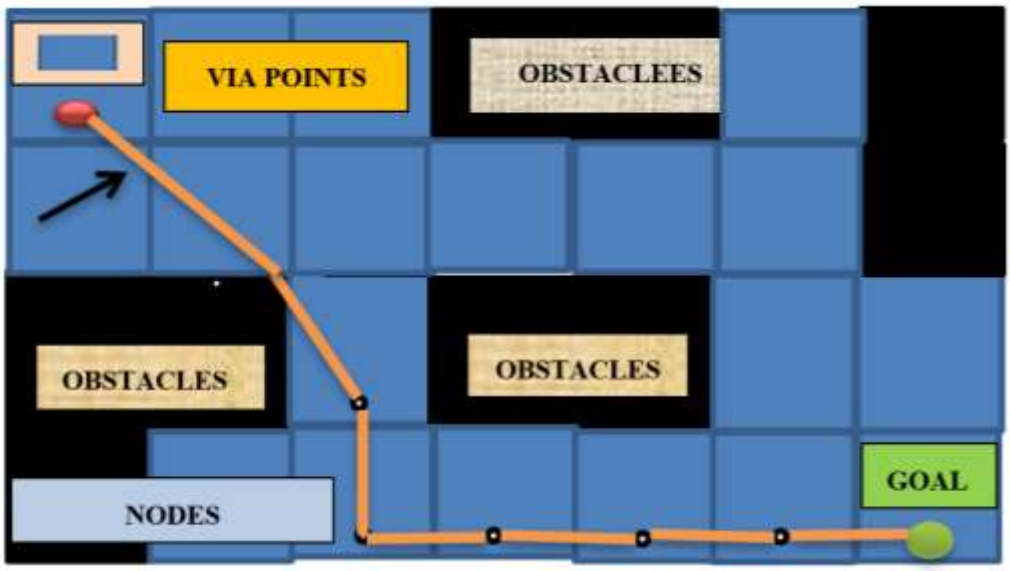



Figure.1 Graph search algorithm in grid based A* Algorithm

**STEP 1: INITIALIZE OPEN AND CLOSED LISTS**

- OPEN LIST : This list contains nodes that are candidates for exploration. It starts with the initial ( start) node.
- CLOSED LIST: This list contains nodes that have already been explored.
- A* Algorithm is a best first search algorithm that uses a heuristic to estimate the cost of a path from the current position to the goal.
- The goal of the A* Algorithm is to find the path with the lowest total cost from a start node to a goal node.
- The cost function f(n) for a node n can been determined by

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

where;

g (n) : The exact cost of the path from the start node to the current node n

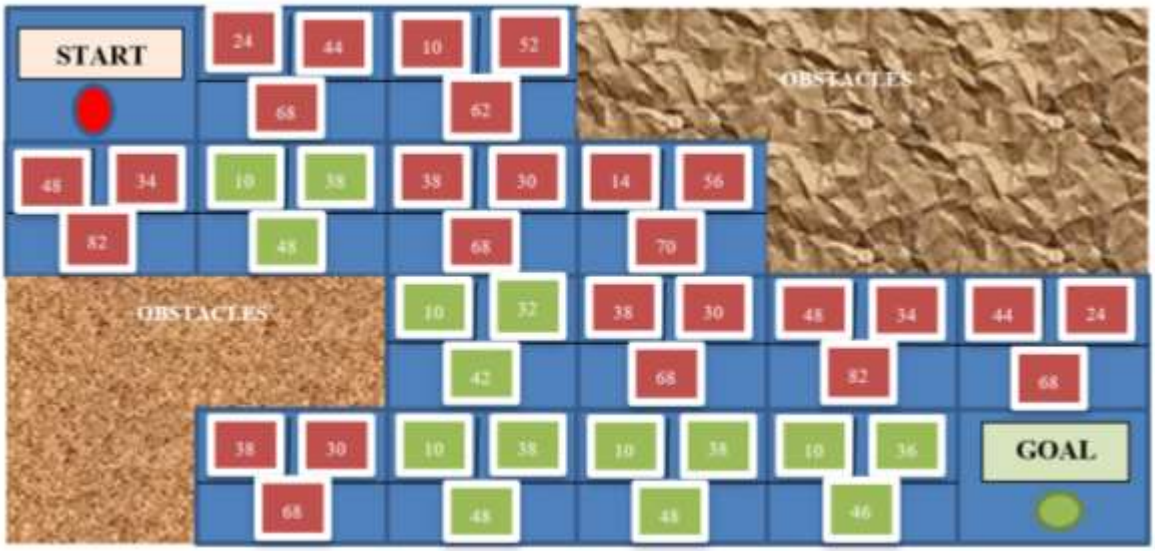
h (n) : A heuristic estimate of the cost from the current node n to the goal node

**STEP 2 : NODE EXPANSION**

- While the open list is not empty we should do the following methods.
- Remove the node from the open list that has the lowest f (n) value.
- If this node is the goal, the algorithm terminates and returns the path.
- Otherwise, move the node to the closed list and explore its neighbours.

**STEP 3 : NEIGHBOUR EVALUATION**

- For each neighbour of the current node, calculate its g (n) and f (n).
- If the neighbour is already in the closed list skip it.
- If the neighbour is in the open list but the newly computed g (n) update the values and reinsert the node into the open list.
- If the neighbour is not in the open list , add it with calculated f (n) value.




**STEP 4 : REPEATATION**

- Move the current node to the closed list.
- Repeat the process, selecting the next node from the open list with the lowest f (n) , until the goal is found on the open list is empty ( indicating no path exists).

**HEURISTIC FUNCTION**

- A * Algorithm uses a heuristic to estimate the cost of reaching the goal from any node.
- The most common heuristic is the Euclidean distance or Manhattan distance, depending on the type of grid.

**Manhattan Distance –** Used in grids with horizontal and vertical movements.

h (n) = | x goal – x current | + | y goal – y current |

Example : If a robot is at point ( 2,3) and the goal is at (7,5) the Manhattan distance is

h (n) = |7-2| + |5-3| = 5+2 = 7

**Euclidean Distance –** Used when the robot can move in any direction including diagonally.

h (n) = $\sqrt{\ }(x\ goal - x\ current)2 + (\ y\ goal - y\ current\ )2$

Example : If a robot is at a point ( 2,3) and the goal is at (7,5) the Euclidean distance is

h (n) = $\sqrt{\ }(7-2)2 + (\ 5-3)2$

h (n) = $\sqrt{\ }25 + 4$

h (n) = $\sqrt{\ }29$

h (n) = 5.39

**Advantages of A* Algorithm :**

- Optimal Solution : Guarantees the shortest path if the heuristic is admissible
- Adaptability : Can handle Complex and Dynamic Environments with static and dynamic Robot Navigation[4]

**Applications of A* Algorithm :**

- Mobile Robots
- Autonomous Vechicles.

***2.2 D* Algorithm***

- The *D algorithm* * is a path planning algorithm designed for mobile robots that operate in dynamic or partially unknown environments.

- Unlike traditional static algorithms like A*, which assume that the environment does not change once the path is calculated, D* can adapt to new information as it becomes available.
- This makes it ideal for mobile robots that need to navigate in environments where obstacles can appear or move unexpectedly.

### *2.2.1 Working of D* Algorithm*

- The D* algorithm builds on the A* algorithm's principles but introduces the ability to update the path when changes in the environment are detected.
- In A*, once a path is found, it remains fixed unless the algorithm is re-run.
- However, this is inefficient in dynamic environments, where constant recalculations may be needed.
- D* addresses this by allowing the robot to adjust the path in real time without recalculating the entire route.
- The robot starts by calculating an initial path to the goal using similar methods as A*.
- As the robot moves along the path, it continuously receives sensor data.
- If an obstacle is encountered that was not previously mapped, the robot can locally modify the path by adjusting only the parts that are affected by the obstacle.
- This makes D* more efficient than A* in dynamic settings.[5]

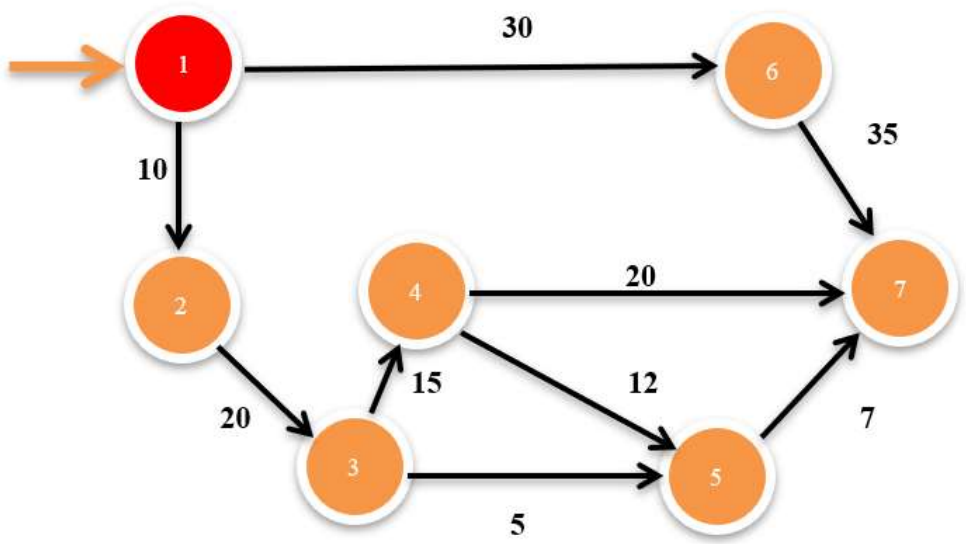
### *2.2.2 Example of D* Algorithm*

Let us assume Source vertex as 1 and we have to find the distance of all vertexes. We have to find the shortest path from source vertex to all remaining vertices.

If (d[u]+c(u,v)<d[v])

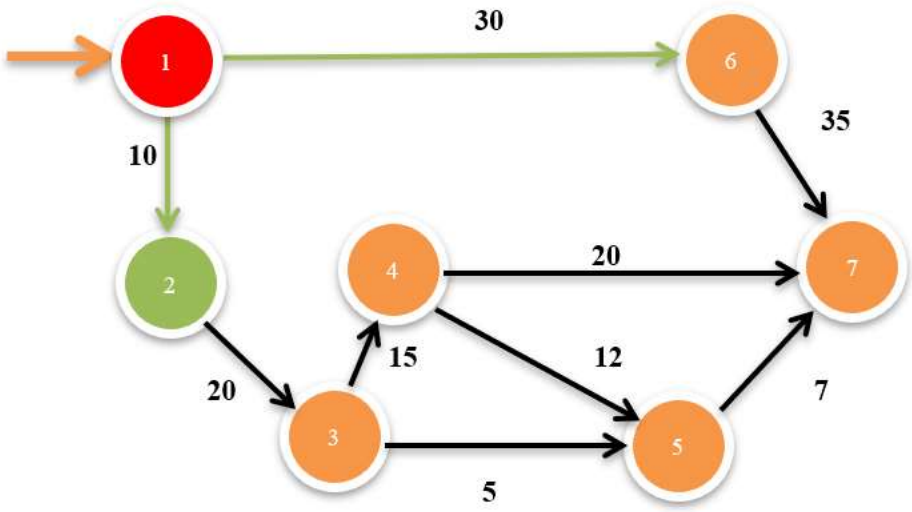
Else

d [v]=d [u]+c (u,v)




**STEP NO 1:** First we have to check from the Source vertex 1 what are the possible ways that can travel.

It can directly travel through the vetex from




1 ⟶6

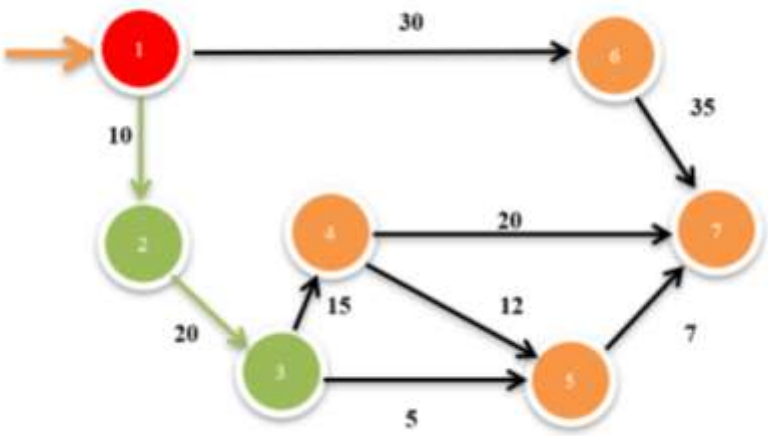
1 ⟶2

We have to put an tabular column for analysis of shortest vertex calculation

Let us compare the distance from source vertex 1 to reach vetex 6 and vertex 2 which has the least distance

Here the vertex 2 has the minimum distance so we conclude that vertex 2 has the minimum distance of travel.[6]

| Selected vertex | Visited set | D[2] | D[3] | D[4] | D[5] | D[6] | D[7] |
|---|---|---|---|---|---|---|---|
| 1 | {1} | **10** | ∞ | ∞ | ∞ | 30 | ∞ |




**STEP NO 2:** We have find the shortest vertex as Vertex 2 which has the minimum distance 10 the next shortest path can been find out from vertex 1 which travels through vertex 2. What are the possible ways that an vertex 2 can connect with other vertexes.

Here the vertex 2 has only one possible way which has to reach the vertex 3. But it cannot reach it by direct ,method of connection .It should travel from source 1 through vertex 2 to reach the vertex 3.

We have to calculate the distance from vertex 1which travels through vertex 2 which reaches vertex 3

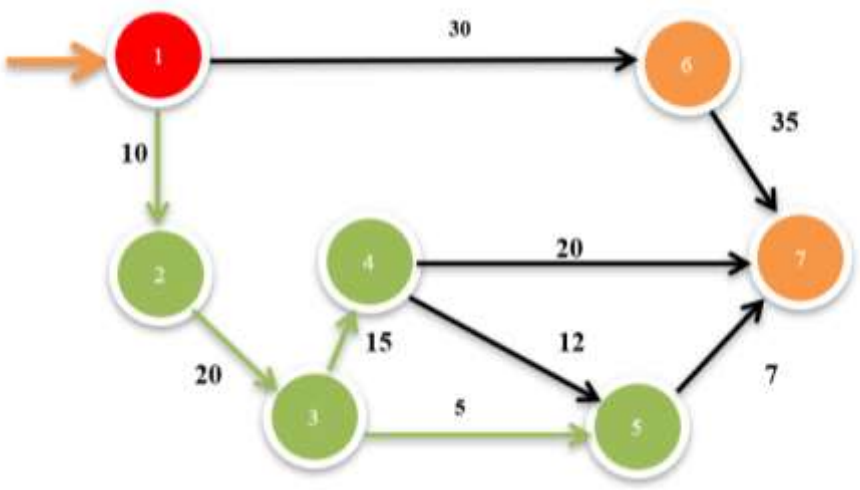
Therefore d[3]= 10 + 20 = 30

| Selected vertex | Visited set | D[2] | D[3] | D[4] | D[5] | D[6] | D[7] |
|---|---|---|---|---|---|---|---|
| 1 | {1} | **10** | ∞ | ∞ | ∞ | **30** | ∞ |
| 2 | {1,2} | **10** | **30** | ∞ | ∞ | **30** | ∞ |

Here we can choose the minimum distance as D[3] because it is given as first priority of node order.

**STEP NO 3:**

- We have find the shortest vertex as Vertex 3which has the minimum distance 30 the next shortest path can been find out from vertex 1 which travels through vertex 2 and then it reaches vertex 3.
- What are the possible ways that an vertex 3 can connect with other vertexes.
- Here the vertex 2 has two possible ways which has to reach the vertex 3.
- But it cannot reach it by direct ,method of connection .
- It should travel from source 1 through vertex 2 through vertex 3 to reach vertex 4 and vertex 5.[7]




We have to calculate the distance from vertex 1which travels through vertex 2 through vertex 3 to reach the two vertex 4 and 5.

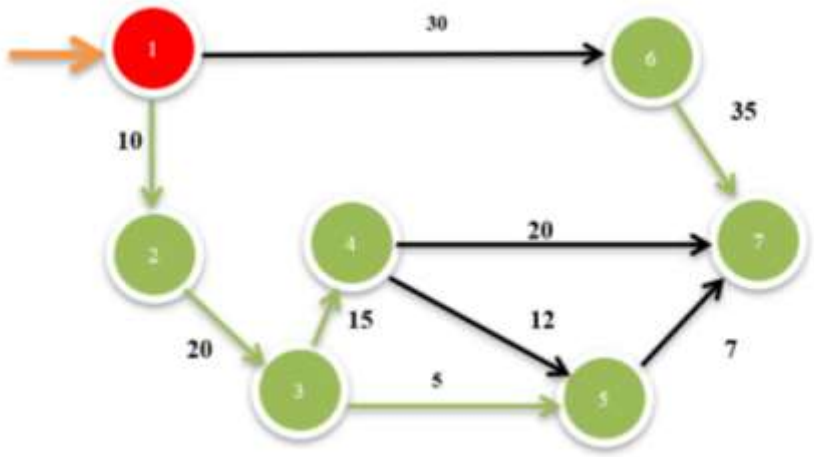
Therefore d[4]= 30 + 15 = 45

Therefore d[5]= 30 + 5 = 35

| Selected vertex | Visited set | D[2] | D[3] | D[4] | D[5] | D[6] | D[7] |
|---|---|---|---|---|---|---|---|
| 1 | {1} | **10** | ∞ | ∞ | ∞ | **30** | ∞ |
| 2 | {1,2} | **10** | **30** | ∞ | ∞ | **30** | ∞ |
| 3 | {1,2,3} | **10** | **30** | 45 | ∞ | **30** | ∞ |

From here we have the minimum distance as d[6] so we can calculate the shortest distance from vertex 6 through which are the possible ways that it can travel through.

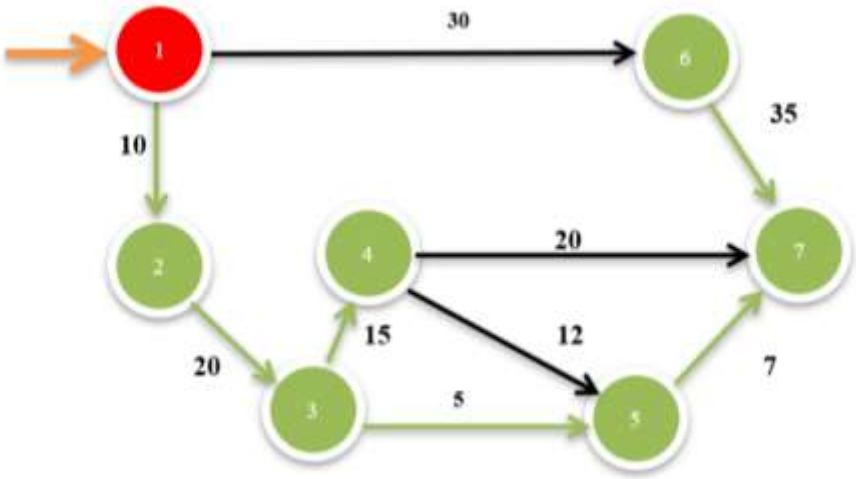
**STEP NO 4 :** Next we have to consider vertex 6 what are the possible ways that it can reaches the other vertexes.Here the vertex 6 has only one possible way to reach the other vertex 7.[8]




| Selected vertex | Visited set | D[2] | D[3] | D[4] | D[5] | D[6] | D[7] |
|---|---|---|---|---|---|---|---|
| 1 | {1} | **10** | ∞ | ∞ | ∞ | **30** | ∞ |
| 2 | {1,2} | **10** | **30** | ∞ | ∞ | **30** | ∞ |
| 3 | {1,2,3} | **10** | **30** | 45 | 35 | **30** | ∞ |
| 4 | {1,2,3,6} | **10** | **30** | 45 | **35** | **30** | 65 |

Here we have the least minimum distance on d[5] which has the minimum distance 35. Therefore we have to calculate the vertex 5 and what are all the possible ways where the vertex can reach the next vertex.

The vertex 5 can reach only one vertex 7.[9]



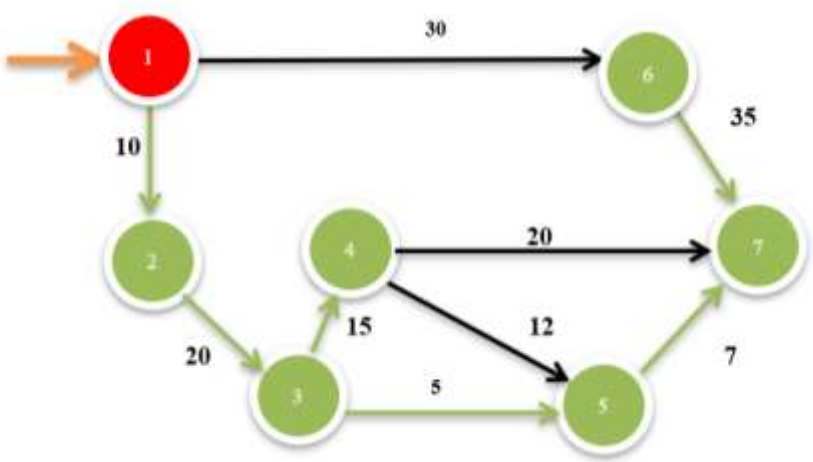
| Selected vertex | Visited set | D[2] | D[3] | D[4] | D[5] | D[6] | D[7] |
|---|---|---|---|---|---|---|---|
| 1 | {1} | **10** | ∞ | ∞ | ∞ | **30** | ∞ |
| 2 | {1,2} | **10** | **30** | ∞ | ∞ | **30** | ∞ |
| 3 | {1,2,3} | **10** | **30** | 45 | 35 | **30** | ∞ |
| 4 | {1,2,3,6} | **10** | **30** | 45 | **35** | **30** | **65** |
| 5 | {1,2,3,6,5} | **10** | **30** | **45** | **35** | **30** | **42** |

Here we have the least minimum distance on d[4] which has the minimum distance 45. Therefore we have to calculate the vertex 4 and what are all the possible ways where the vertex can reach the next vertex.

The vertex 4 can reach only one vertex 7 and vertex 5[10]

**STEP NO 5 :** Next we have to consider vertex 4 what are the possible ways that it can reaches the other vertexes.

Here the vertex 4 has two possible way to reach the other vertex 7 and vertex 5.




| Selected vertex | Visited set | D[2] | D[3] | D[4] | D[5] | D[6] | D[7] |
|---|---|---|---|---|---|---|---|
| 1 | {1} | **10** | ∞ | ∞ | ∞ | **30** | ∞ |
| 2 | {1,2} | **10** | **30** | ∞ | ∞ | **30** | ∞ |
| 3 | {1,2,3} | **10** | **30** | 45 | 35 | **30** | ∞ |
| 4 | {1,2,3,6} | **10** | **30** | 45 | **35** | **30** | **65** |
| 5 | {1,2,3,6,5} | **10** | **30** | **45** | **35** | **30** | **42** |
| 6 | {1,2,3,6,5,4} | **10** | **30** | **45** | **35** | **30** | **42** |

Therefore the D * Algorithm can been done and the minimum cost of distance can been calculated as follows

## 3. CONCLUSION

In summary, graph search algorithms like A* and D* play a vital role in the field of mobile robotics, particularly for path planning. [11] Both algorithms use a graph representation of the environment to find the shortest path to a destination, but they differ in their approach and suitability for different scenarios.[12] The A algorithm* is known for its efficiency in finding the optimal path in static environments. It balances exploration of new paths with cost estimation through a heuristic function, ensuring that the path chosen is not only the shortest but also the most efficient. A* guarantees optimal results when the heuristic is well chosen, making it a popular choice in applications like autonomous robots or navigation in controlled environments where obstacles remain fixed. [13] On the other hand, the D algorithm* is specifically designed to handle dynamic and changing environments. It allows robots to adjust their paths in real time when new obstacles are detected, without needing to recalculate the entire path from scratch. This ability to re-plan makes D* highly suitable for real world applications where robots encounter unexpected changes, such as in search and rescue operations, autonomous driving, or crowded warehouses. [14] Variants like D-Lite* and Focused D* further improve the efficiency of the original algorithm, allowing robots to quickly adapt to environmental changes while minimizing computational load. In conclusion, both A* and D* have their strengths depending on the application. A* is ideal for static environments, ensuring optimal and precise path finding, while D* excels in dynamic environments where continuous updates are required. Understanding these algorithms is key to developing advanced robotic systems that can navigate complex, real world environments with efficiency and intelligence.[15]

***ACKNOWLEDGEMENTS***

Special thanks to mentors and colleagues for their valuable insights and encouragement. Their support has been instrumental in shaping the ideas and content presented in this journal.

**REFERENCES**

[1].1.Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." IEEE Transactions on Systems Science and Cybernetics, 4(2), 100-107.

[2].Stentz, A. (1994). "Optimal and Efficient Path Planning for Partially-Known Environments." Proceedings of the IEEE International Conference on Robotics and Automation, 3310-3317.

[3]. Koenig, S., & Likhachev, M. (2005). "Fast Replanning for Navigation in Unknown Terrain." IEEE Transactions on Robotics, 21(3), 354-363.

[4]. Carsten, J., Ferguson, D., & Stentz, A. (2006). "3D Field D*: Improved Path Planning and Replanning in Three Dimensions." IEEE/RSJ International Conference on Intelligent Robots and Systems, 3381-3386.

[5]. Ferguson, D., & Stentz, A. (2006). "Using Interpolation to Improve Path Planning: The Field D* Algorithm." Journal of Field Robotics, 23(2), 79-101.

[6].Karaman, S., & Frazzoli, E. (2011). "Sampling-based Algorithms for Optimal Motion Planning." The International Journal of Robotics Research, 30(7), 846-894.

[7]. Likhachev, M., Gordon, G., & Thrun, S. (2004). "ARA*: Anytime A* with Provable Bounds on Sub-Optimality." Advances in Neural Information Processing Systems (NIPS), 47-54.

[8]. LaValle, S. M., & Kuffner, J. J. (2000). "Rapidly-Exploring Random Trees: Progress and Prospects." Algorithmic and Computational Robotics: New Directions, 293-308.

[9]. Nash, A., Koenig, S., & Tovey, C. A. (2010). "Lazy A*: Faster Path Planning for Any-Angle Paths." Proceedings of the AAAI Conference on Artificial Intelligence, 27(1), 17-23.

[10]. Qin, W., Dong, C., & He, W. (2016). "Improved A* Algorithm Based on Traffic Rules for Vehicle Path Planning." Procedia Computer Science, 96, 1075-1081.

[11]. MacAllister, B., & Koenig, S. (2009). "Real-Time Adaptive A*." Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 42-49.

[12]. Nilsson, S. (2004). "Finding the Shortest Path: A* Search and its Variants." Technical Report, Linköping University.

[13]. Sun, X., Koenig, S., & Yeoh, W. (2009). "Generalized Adaptive A*." Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 238-243.

[14]. Liu, Y., Zhang, Q., Li, J., & Su, Y. (2017). "Path Planning for Autonomous Robots Based on Improved A* Algorithm." IEEE Access, 5, 13101-13110.

[15]. Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2005). "Anytime Dynamic A*: An Anytime, Replanning Algorithm." Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 262-271.