Sakshi, 2026, 14:4
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

A Comprehensive Study and Analysis of Shortest Path Algorithms: Classical to Heuristic

Sakshi, Aastha, Mansi, Prince Kumar Sharma
HMR Institute of Technology and Management GGSIPU, New Delhi, India

Abstract- The Shortest Path Problem (SPP) is an important part of graph theory with Network optimization with today's system. The main objective of this research paper is to present a comprehensive analysis, ranging from classical algorithms to modern approaches. In this paper, we compared the Dijkstra, Bellman-Ford, Floyd-Warshall, Johnson's, and A (A-Star)* algorithms based on their theoretical complexity, time complexity, algorithms and real-world applications. Our research highlights that while Dijkstra's algorithm remains the main function for navigation systems, Bellman-Ford's algorithm works on graphs with negative weights. Additionally, we explore how the A* algorithm optimises the search space using heuristic functions, making it a better. Alternative to Artificial Intelligence and robotics. Through In this research analysis, we conclude that the efficiency of any The algorithm depends mainly on the network structure and specific constraints. This paper provides researchers and Developers with a clear roadmap for choosing the right algorithm for their shortest and fastest path.

Keywords: Shortest Path, Dijkstra, Bellman-Ford, A* Algorithm, Comparative Analysis.

I. INTRODUCTION

The Shortest Path Problem (SPP) is a fundamental optimisation challenge in Graph Theory. Its objective is to find a path from a source node to a destination node where the total sum of the edge weights is minimal [1]. As algorithms have moved beyond mere theoretical mathematics and reached Google Maps, Autonomous Vehicles, and Game AI. In this paper, we compared five popular algorithms Dijkstra, Bellman-Ford, Floyd - Warshall, A* (A-Star), and Johnson. This paper compares time complexity, space complexity, graphs on which they compute the algorithm. This paper examine the effects of variables like edge weights, negative weights, graph density, and negative cycles on execution time using a specialised testing framework. Our primary The goal is to determine the most effective solution for various graph sizes by comparing observed results with theoretical O(n) complexities.

Dijkstra's algorithm employs a "Greedy" strategy. It is considered most efficient for single-source shortest paths when all edge weights are positive. Dijkstra et al. (1959) presented its basic framework, which is still the basis of network routing protocols today [2]. Bellman-Ford Algorithm: Dijkstra fails when the graph has negative edge weights. Bellman (1958) and Ford (1956) developed a dynamic programming approach that not only handles negative edges but can also detect "negative cycles" [3]. However, its computational complexity is higher than Dijkstra's. Floyd-Warshall Algorithm: In some scenarios, we need the shortest path from every node to every other node, not Just from a source. Floyd-Warshall is an "All-Pairs Shortest Path" algorithm and is best for dense graphs [4].

Johnson's Algorithm: Johnson (1977) introduced a A hybrid approach that combines Bellman-Ford and Dijkstra. This algorithm outperforms Floyd-Warshall for finding All-Pairs Shortest Path in sparse graphs, because it "re-weights" the edges and neutralises negative weights [5]. A (A-Star) Search Algorithm: * An algorithm is an advanced and "informed" version of Dijkstra. It uses heuristic functions to reduce the search space, allowing faster arrival at the destination. Developed by Hart et al. (1968), this algorithm is the most popular for path-planning in Artificial Intelligence and Robotics [6].

"This review focuses on the technical problems of classical algorithms and the heuristic tactics that optimise them to bridge the gap between theoretical graph complexity and actual implementation. Our objective is to determine the most effective and fair way to implement resource and speed management in intelligent navigation systems."

II. LITERATURE REVIEW

The Shortest Path Problem (SPP) is a basic and important an optimisation challenge in graph theory that has been studied from various perspectives for many years. Its The simple objective is to find a path from a source node to a destination node where the sum of total edge weights is the lowest. Regarding previous work in this area, researchers have tested it in various situations. For example, Zhan and Noonan found that the network design has a significant impact on the algorithm's speed [7]. Similarly, Fu et al. (2006) focused their work on transportation systems where travel times are constantly changing; they demonstrated that Heuristic approaches are more effective in such a dynamic environments. Our review study extends this trend by Analysing the gap between theoretical complexities and practical execution times using experimental data from MMS [8].

Dijkstra's algorithm

OVERVIEW

Dijkstra's Algorithm (named after E. W. Dijkstra) addresses the challenges of identifying the shortest path between the source node to a destination. The algorithm calculates the shortest path distance from the source to every other vertex in the graph simultaneously. Furthermore, this is frequently categorised as a 'Single-Source Shortest Path' (SSSP) problem. [9] The first step in Dijkstra's Algorithm is that Dijkstra's divides nodes into two groups: temporary set (t) and then permanent set (p). The algorithm assigns a distance value of zero to the source node s and labels it as permanent. At each iteration, from the current node, it updates its distance using new again among all temporary nodes, it picks the one with the smallest distance and marks it as permanent (p) because it is the shortest possible path from the source node to the destination node.

Mathematical Formulation

D(v)=min{dv,du + w (u,v)}[10]

where d(v) is current known shortest distance from the source node s to vertex v, d(u) is current known shortest distance from the source node s to vertex u, w (u, v) is weight (or cost) of the edge from vertex u to vertex v, d(u) + w (u, v) is distance to reach v through vertex u

ALGORITHM

The computational time complexity of Dijkstra's Algorithm is O(|V|^2) where |V| represent the number of vertices in a graph. [11]

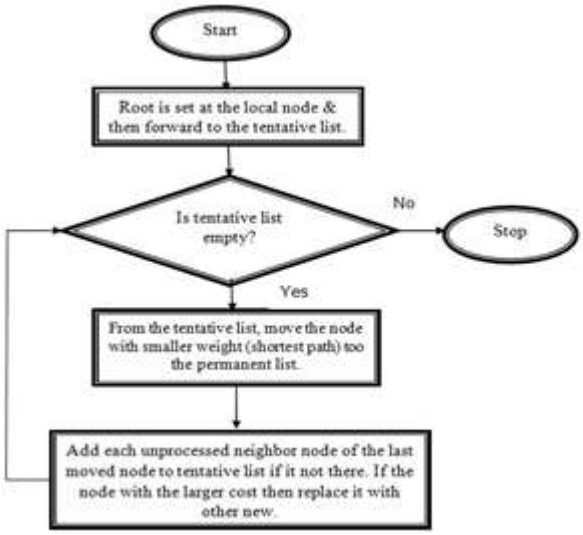

Fig 1. Flowchart of Dijkstra's algorithm [2]

Application

The Dijkstra algorithm is used in GPS to avoid traffic and find the shortest route. It also helps in emergency services like Ambulance and fire brigade operations are to be solved by the respective personnel. It is also used in Internet routing for data transfer, and it saves time and fuel. It is used in network design and optimisation. Dijkstra's algorithm does not work with negative weights and may produce incorrect results in such cases. It can also become computationally expensive for large graphs and is not well-suited for dynamic environments where edge weights change frequently. Additionally, it is limited to single-source shortest path problems.

Bellman-Ford algorithm

Overview The Bellman-Ford algorithm is a very classical A programming approach to solving shortest path problems. Bellman (1958) and Ford (1956) developed it when Dijkstra's algorithm failed on graphs with negative weights. Its main function is to find the Single-Source Shortest Path (SSSP), But its real strength lies in its versatility [3]. Working

Process: Its working is based on the simple logic of "relaxation". This algorithm scans each path (edge) of The graph (V-1) is used to confirm the shortest path. According to Bang-Jensen and Gutin (2008), its greatest strength is its ability to detect "negative cycles". If, after all attempts, the path remains narrow, the algorithm recognises that there is a cycle in the graph that could stretch the computation to infinity [13].

Mathematical Formulation

d(v)= min (d(v), d(u) + w (u, v))

Algorithm

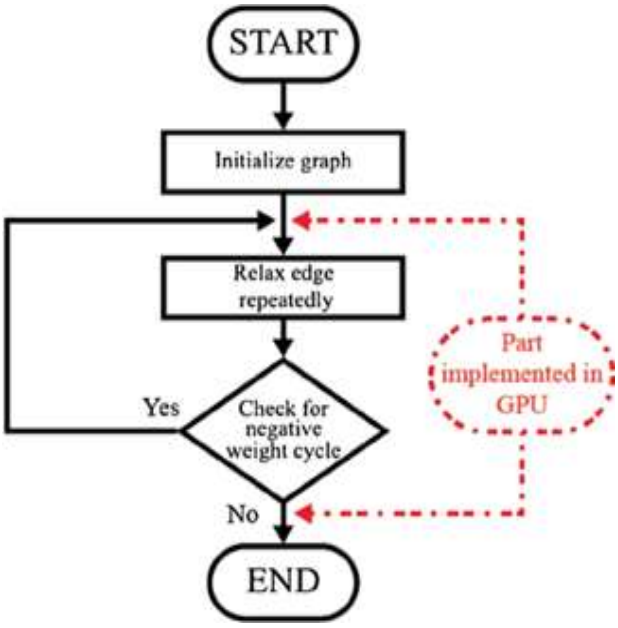

Fig 2. Flowchart of BELLMAN-Ford Algorithm [14]

Applications In the practical world, Bellman-Ford is the basis of routing protocols like RIP. Bertsekas and Gallager (1992) described how this algorithm helps to direct data packets on the Internet correctly. Additionally, it has an interesting use in the stock market and finance, where traders use it to find arbitrage (taking advantage of currency rates) [15].

Limitations: However, not everything is perfect. Cormen et al. (2009) pointed out that its time complexity is O (V E), making it significantly slower than Dijkstra's. In large networks or dense graphs, this algorithm consumes a lot of processing power, so it is only chosen when handling negative weights is necessary [1]. Conclusion: In short, Bellman-Ford follows a "Slow but Steady" approach. Where Dijkstra prioritises speed, Bellman-Ford focuses on accuracy and reliability. This is still important in complex scenarios where graphs are prone to negative values.

Floyd Warshall Algorithm

Overview

The Floyd-Warshall algorithm is a dynamic programming approach used to compute the shortest path between all pairs of vertices in a weighted graph. It is particularly effective for dense graphs where multiple shortest path queries are required simultaneously unlike a single source algorithm. This method determines the shortest path between every possible pair of nodes in the graph [16]. Cormen et al. Describe the Floyd-Warshall algorithm as a dynamic

A programming technique for solving the all-pairs shortest path problem, iteratively improving the shortest path estimates between vertex pairs through intermediate vertices. [1] At each step, it checks whether including an additional vertex as an An intermediate node results in a shorter path. This process continues until all vertices have been considered as intermediate points. Warshal 1962 presented a related method for computing the transitive closure, which forms the basis of the Floyd-Warshall algorithm [16].

Mathematical Formulation

$$D^{k[i,j]} = min\left(D^{k-1[i,j]}, D^{k-1[i,j]} + D^{k-1[k-1]} \right)$$

where D(k) = shortest distance from vertex i to j using vertices 1 to k [4]

Algorithm

The algorithm computes shortest paths between all pairs of vertices in a single execution. It is simple and easy to implement, especially using matrix representation and it works very well with a negative-weight graph. This method is based on dynamic programming, ensuring systematic and optimal updates. It is particularly efficient for dense graphs where the number of edges is large.

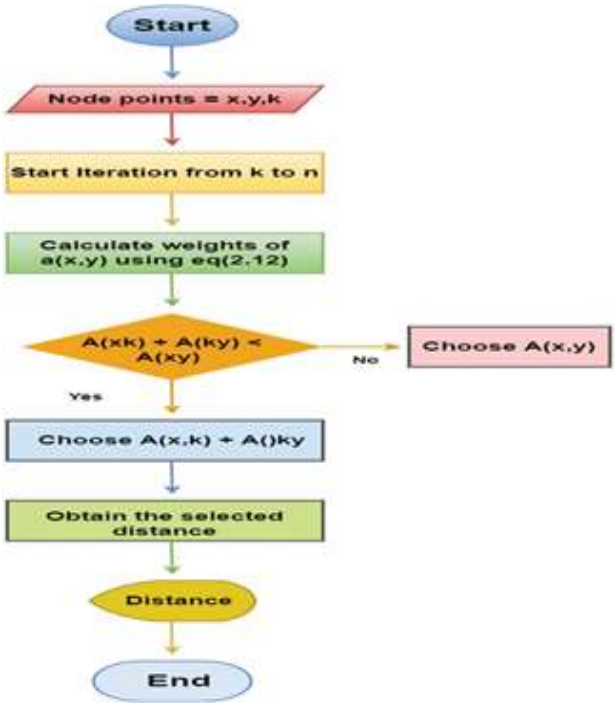

Fig3: Flowchart of Floyd Warshall Algorithm [17]

APPLICATION

The Floyd-Warshall algorithm helps in network routing. This algorithm helps in finding the path between all nodes in a communication network. It is also used in transportation systems, such as routes of airlines and traffic management, where distances between every pair of Locations must be evaluated simultaneously. It is also useful for finding transitive closure, that is, checking a graph. Their important application is in social network analysis, where it can be used to measure the closeness or connectivity between users. The Floyd-Warshall algorithm is particularly suitable for dense graphs because of its ability to handle both positive and negative edge weights (without negative cycles).

LIMITATIONS

The time complexity is not that efficient for large graphs. It requires high space complexity due to matrix storage. Moreover, the Floyd-Warshall algorithm cannot handle a negative-weight cycle as distances become undefined. It is less efficient than the Dijkstra algorithm for sparse graphs and it is not suitable for real-time, very large skill networks due to computational cost.

A (A-Star) algorithm*

OVERVIEW

The A* algorithm is considered an "informed search" strategy for shortest path problems. It was first developed by Hart et al. (1968) [6]. Greedy Best First Search is the extension of Dijkstra's Algorithm and Greedy Best First Search. In modern literature, this technique focuses on the destination. A* selects the node A with the minimum value of f(n). According to Russell and Norvig (2010) [18], the efficiency of A* gives the shortest path if the heuristic is admissible. If a heuristic is admissible it never overestimates the actual cost. This algorithm always guarantees the optimal shortest path. The A* algorithm serves as a powerful bridge between theoretical and practical applications. Where classical algorithms search without any direction, A* uses its "intelligence" to produce target-oriented results. This is why it remains an indispensable tool for intelligent navigation systems.

MATHEMATICAL FORMULATION

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

g(n) represents the actual cost from the source to the current node. h(n) is an estimated heuristic value i.e. estimated cost from the current node to the goal node.

ALGORITHM

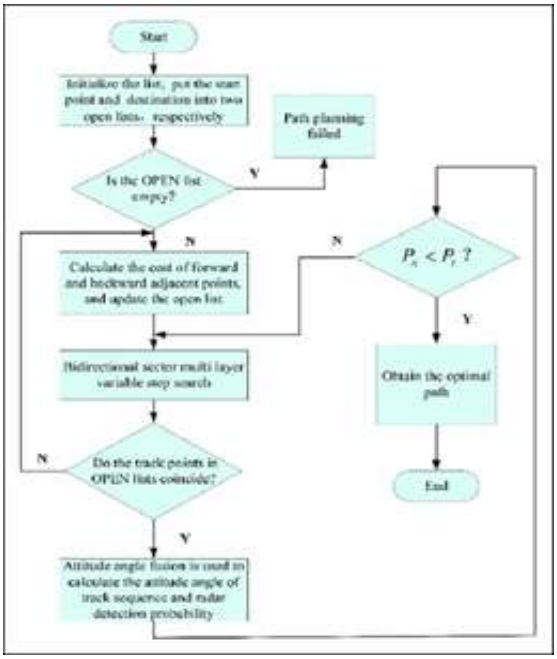

Fig4: Flowchart of A star Algorithm [19]

In modern literature, this technique focuses only on those nodes that have a high chance of leading to the destination. A* selects the node A with the minimum value of f(n).

APPLICATION

In terms of practical use, the A* algorithm has revolutionised robotics and navigation. Stentz (1994) showed how variants of A* are best suited for finding paths in dynamic environments. It is widely used in Google Maps as well as GPS. A* is also used in Artificial Intelligence and Game AI because of its low memory and ability to generate paths in microseconds. [20]

LIMITATION

Despite its many strengths, the A* algorithm has its limitations. The value of A* increases significantly in large and complex state spaces because it has to store all the generated nodes in an 'Open List'. At the same time, designing an accurate heuristic function remains a major challenge for researchers even today. [21]

Johnson's algorithm

OVERVIEW

Johnson's algorithm is mainly used for finding the shortest path between pairs of vertices in a graph. It mainly work is to work with negative weights for a faster result. It is faster than other methods that are in use in other networks. It also gives the "arc set partitions "algorithm which solves the single-source problem in an unrestricted area. It may also exist the positive edges. And their time complexity is and the space complexity is also the same. It mainly works for sparse graphs and works on directed graphs. They only work when there is no negative cycle. This method is a combination of Bellman-Ford and Dijkstra. [5][20]

MATHEMATICAL FORMULATION

$$w'(u,v) = w(u,v) + h(u) - h(v)$$

w'(u,v) is the new non-negative weight. w(u,v) is the original edge weight. h(u) h(v): Potential values for vertices u and v are typically source vertex to each vertex.

Algorithm: How to Use

First, add a node then apply the Bellman-Ford method to remove weight then re-weight the code then apply Dijkstra to the vertex to find the shortest path then give the final and shortest path distance.

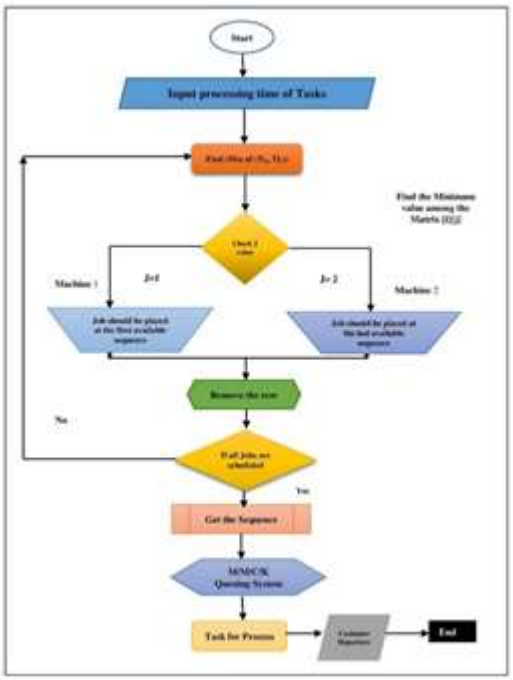

Fig 5: Flowchart of Johnson's Algorithm [21]

Limitations

If it has a negative cycle then the algorithm will fail and is complex to understand because of many steps and it has the Bellman-Ford costly method. And if there is a larger graph then it takes a larger time. If the edges are high then the algorithm is not suitable. When we reweight, majorly mistaken happens in solving the graph.

Helps in futureWe can focus on negative cycles and they can detectand remove the error and they can also optimise for largegraphs, and they can also build a hybrid model with the help ofAI/ML techniques.

III. METHODOLOGY

This review is based on a systematic selection of research papers to compare different algorithms thoroughly. The focus was primarily on the original research papers in which the respective shortest path

algorithms were first introduced by their authors. This foundational work plays a very pivotal role in understanding the theoretical and core principles of each algorithm. The papers were chosen mainly based on how closely they relate to the topic, the clarity of and the significance of their contribution. In addition to these original contributions, a wide range of contemporary articles and papers have also been studied to provide a broader applications. Studies that lacked sufficient technical detail, were outdated without significant contribution, or were not directly related to shortest path problems were excluded from the review.

IV. COMPARISON

Analysis of Table 1: Technical Performance and Complexity

Table 1. comparison between all Five Algorithm for Shortest Path

Algorithm	Scope	Data Structure	Time Complexity	Edge Weights	Ideal Use Case
Dijkstra	Single Source	Priority Queue (Min-Heap)	O((V+E) \log V)	Only Positive	Real-time navigation, GPS, Network routing.
Bellman-Ford	Single Source	Array / List	$O(V \cdot E)$	Positive & Negative	Financial networks (Arbitrage), Negative cycle detection.
Floyd-Warshall	All-Pairs	2D Matrix	$O(V^3)$	Positive & Negative	Small networks, Dense graphs, Transitive closure.
Johnson's	All-Pairs	Fibonacci Heap + Bellman	$O(V^2 \log V + VE)$	Positive & Negative	Large Sparse graphs where Floyd-Warshall is too slow.
A (A-Star)*	Single Source	Heuristic + Priority Queue	O(E)	Only Positive	Game AI, Robotics, Pathfinding with target direction.

A high analysis of Table 1 tells some key points about the technical skills of the algorithms: Complexity vs. Performance: The complexity of Dijkstra's The algorithm isO((V+E) \log V), making it the fastest for single-source conditions. On the other hand, Bellman-Ford has a much higher complexity of O (V \cdot E), which makes it perform slower on large graphs, but its application is unequalled in handling negative weights. Role of Data Structure: We can see that the capability of an algorithm depends on its data structure. While A* and Dijkstra enhance the search space using a 'Priority Queue', Floyd-Warshall uses a '2D Matrix' which is necessary for finding the All-Pairs shortest path but is recall-intensive. Compatibility: It is clear from the table that Dijkstra and A* are ideal for real-time applications like GPS and Navigation, whereas Bellman-Ford is unequalled for financial models where negative cycles have to be detected.

Analysis of Table 2: Insights from Literature Survey

Table 2: Insights from Literature Survey Summarizing the work of different researchers [22]

Author and Year	Algorithm	Method
Oyola *et al.* [2]	Dijkstra-based algorithm	The suggested strategy makes the algorithm appropriate for use in dynamic settings where distinct types of sensors can change the accessibility status of inner areas.
Dinitz and Itzhak [3]	Bellman_Ford and Dijkstra algorithms	The new proposed Bellman_Ford algorithm can produce the shortest paths tree.
Singh and Tripathi [4]	Bellman_Ford and Dijkstra algorithms	The Bellman_Ford algorithm deals more effective with a small quantityof nodes, and the Dijkstra algorithm is greater efficient for a large wide of nodes.
Ryash and Tamimi [5]	Dijkstra, Bellman_Ford, Floyd_ Warshall, and Johnson algorithms	Dijkstra-based algorithm'scomplexity is faster than the Bellman_Ford algorithm with respect to time. The algorithm of Johnson is faster than Floyd_Warshall when the graph is sparse, but on the dense graph, the Floyd–Warshall algorithm is faster.
Lacorte and Chavez [6]	A* and Dijkstra algorithms	The algorithm of A* performs better than Dijkstra's algorithmas expected arrival time (ETA) is shorter on a small graph but Dijkstra's algorithm produce shorter ETA in both small and big graphs.
Permana *et al.* [7]	A*, Breadth FirstSearch and Dijkstra algorithms	A* can be considered as the good path finding algorithm, especially in the game/grids of Maze.
WANG [8]	Dijkstra, Bellman_Ford and Floyd_Warshall algorithms	Bellman-Ford algorithm is simple. Floyd-Warshall algorithm is very slow among the three algorithms. Dijkstra algorithm can only be useful in the shortest route issueof a single source.
Abbas et al [9]	Caption algorithm	The new proposed algorithm effectiveness and performance is better than the Dijkstra's algorithm on real-world examples such as discovering the shortest route between towns.

Summarising the work of different researchers presented in Table 2, the following observations appear: Performance in Different Networks: According to Ryash and Tamimi, Johnson's algorithm work best when the graph is sparse (has fewer edges), but Floyd-Warshall is more powerful in dense graphs. This shows that no single An algorithm is best for every type of network. Small vs. Large Graphs: Studies by Singh and Tripathi show that Bellman-Ford is more suitable for graphs with fewer nodes, but as the number of nodes increases, Dijkstra's efficiency becomes advanced. Intelligent Pathfinding: Permana et al. and Lacorte & Chavez focused on the A* algorithm. Their analysis shows that A* outperforms Dijkstra in grids and games environments because of its ability to maximise the "Estimated Arrival Time" (ETA). Dynamic and Real-world Adaptability: Oyola et al. and Abbas et al. Research points out that modern algorithms are not limited to static maps; they need to be modified to detector and real-world traffic data.

V. CONCLUSION AND FUTURE SCOPE

Conclusion

In this research paper, we have done a detailed study of shortest path algorithms, covering from classic, that no single algorithm is "best" for every situation; Rather, the selection of an algorithm always depends on the nature of the graph and the conditions of the application. Dijkstra's algorithm is still the selected choice in navigation systems due to its speed, while Bellman-Ford is essential in complex scenarios where there is a risk of negative weights and cycles. Whereas Floyd-Warshall for All-Pairs shortest The path is suitable in dense graphs, but in large and sparse graphs networks the study of Johnson's algorithm surpasses it. The A* (A-Star)

algorithm has proven that through heuristic- Based on an "informed search" we can drastically reduce the computational load, which is a revolutionary step for modern AI and robotics.

Future Scope

Shortest path optimisation is a field that is always evolving. In the future, we can do more work in the following areas: Machine Learning Integration: Deep learning models can be used to make shortcut functions (h(n)) more accurate, permitting faster pathfinding in unknown environments. Quantum Computing: As quantum computing develops, Shortest path algorithms can be optimised with quantum algorithms (such as Grover's Search) to achieve super-fast results. Dynamic Real-time Graphs: By including real-time traffic data and weather conditions as "live parameters" into the algorithm, responsive routing systems can be created that can change paths every microsecond.

REFERENCES

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
2. E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, vol. 1, no. 1, pp. 269–271, Dec. 1959.
3. R. Bellman, "On a routing problem," Quarterly of Applied Mathematics, vol. 16, no. 1, pp. 87–90, 1958.
4. R. W. Floyd, "Algorithm 97: Shortest path," Communications of the ACM, vol. 5, no. 6, p. 345, June 1962.
5. D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," Journal of the ACM (JACM), vol. 24, no. 1, pp. 1–13, Jan. 1977.
6. P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, July 1968.
7. F. B. Zhan and C. E. Noon, "Shortest path algorithms: An evaluation using real road networks," Transportation Science, vol. 32, no. 1, pp. 65–73, 1998.
8. L. Fu, D. Sun, and L. R. Rilett, "Heuristic shortest path algorithms for transportation applications: State of the art," Computers & Operations Research, vol. 33, no. 11, pp. 3324–3343, 2006.
9. A. Javaid, "Understanding Dijkstra's algorithm," SSRN Electronic Journal, vol. 2340905, 2013.
10. A. Numawan and R. Wayahdi, "Comparison of Shortest Path Search Algorithms: Dijkstra, Bellman-Ford, Floyd-Warshall and A*," Jurnal Informatika dan Teknik Elektro (JITE), vol. 1, no. 1, pp. 45–52, 2023.
11. B. Alryash and A. Al-Tamimi, "Comparison Studies for Different Shortest Path Algorithms," International Journal of Computer Science and Information Security (IJCSIS), vol. 13, no. 12, pp. 104–109, Dec. 2015.
12. E. Dhulkefl, A. Durdu, and H. Terzioğlu, "Dijkstra algorithm using UAV path planning," Konya Journal of Engineering Sciences, vol. 8, pp. 92–105, 2020.
13. J. Bang-Jensen and G. Gutin, Digraphs: Theory, Algorithms and Applications, 2nd ed. London, UK: Springer-Verlag, 2008.
14. J. Uddin, I. K. Jeong, M. Kang, C. H. Kim, and J. Kim, "Accelerating IP routing algorithm using graphics processing unit for high speed multimedia communication," Multimedia Tools and Applications, vol. 75, pp. 2013-2014, 2014.
15. D. Bertsekas and R. Gallager, Data Networks, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 1992.
16. F. Sapundzhi, K. Danev, A. Ivanova, M. Popstoilov, and S. Georgiev, "A performance comparison of shortest path algorithms in directed graphs," Engineering Proceedings, vol. 100, no. 1, p. 31, 2025.
17. S. Habib, A. Majeed, M. Akram, and M. Al-shamiri, "Floyd-Warshall Algorithm Based on Picture Fuzzy Information," Computer Modeling in Engineering and Sciences, vol. 136, pp. 2873–2894, 2023.
18. S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2010.
19. Z. Zhang, J. Wu, J. Dai, and C. He, "Optimal path planning with modified A-Star algorithm for stealth unmanned aerial vehicles in 3D network

radar environment," Proc. Inst. Mech. Eng. G J. Aerosp. Eng., vol. 236, 2021.
20. S. H. Barkund, A. Sharma, and H. R. Bhapkar, "Survey of Shortest Path Algorithms," Int. J. Adv. Res. Comput. Commun. Eng. (IJARCCE), vol. 11, no. 5, pp. 118–123, May 2022.
21. P. Banerjee et al., "OptiDJS+: A Next-Generation Enhanced Dynamic Johnson Sequencing Algorithm for Efficient Resource Scheduling in Distributed Overloading within Cloud Computing Environment," Electronics, vol. 12, p. 4123, 2023.
22. S. Abu Salim, R. S. Bakry, and N. M. Abd El-Kader, "A Comparative Study of Shortest Path Algorithms," IOP Conference Series: Materials Science and Engineering, vol. 917, no. 1, p. 012077, 2020.