ISSN 2664-4150 (Print) & ISSN 2664-794X (Online)
South Asian Research Journal of Engineering and Technology
Abbreviated Key Title: South Asian Res J Eng Tech

| Volume-7 | Issue-5 | Nov-Dec- 2025 | **DOI: https://doi.org/10.36346/sarjet.2025.v07i05.002**

Review Article

Comparative Analysis of Shortest Path Algorithms

Saja Saeed Abed[1]*, Saja Faeq Noaman[2]

[1]College of Electrical Engineering, University of Technology- Iraq, Baghdad, Iraq

***Corresponding Author:** Saja Saeed Abed
College of Electrical Engineering, University of Technology- Iraq, Baghdad, Iraq
Email: saja.s.abed@uotechnology.edu.iq

Article History
Received: 05.10.2025
Accepted: 26.11.2025
Published: 18.12.2025

Abstract: This paper is a comparative between classical, hybrid and heuristic shortest path algorithms. Shortest path algorithms are necessary in graph theory and has different applications in operations research, artificial intelligence, networking, and transportation. In dynamic or huge graphs, the deterministic correctness and limitations of classical algorithms such as Dijkstra's and Bellman-Ford have been investigated. The current research examined the flexibility and scalability of heuristic techniques like Ant Colony Optimization, A*, and Genetic Algorithms. Furthermore, examined the potential to improve and enhance the accuracy of hybrid algorithms that include characteristics from both categories has been studied. This work employs important evaluation standards, for instance, optimality, time complexity, and scalability, to inspect the theoretical underpinnings, practical applications, and performance compromises of each strategy.

Keywords: A*, Ant Colony Optimization, Bellman-Ford, Comparative Analysis, Dijkstra, Heuristic Algorithms, Hybrid Algorithms, Shortest Path.

INTRODUCTION

The shortest path problem encompasses determining the smallest route from a starting point to a specified endpoint. We use graphs to represent the shortest path problem. What's a graph? A graph is a mathematical abstract object that contains sets of vertices (or nodes) and edges. Edges link between vertices. We can move from one vertex to another through the edge that directly links them. Graphs contain directed graphs and undirected graphs to They are discerned from each other based on the ability to traverse the edges from both sides or only one side. Besides, the length of edges is usually referred to as weights, which are typically used for determining the shortest path between two vertices or points [1].

Graph theory is used in many different practical applications, such as social networks, transportation systems, communication infrastructures, computer science, engineering, and operations research extensively use these structures. These problems are crucial in real-world situations such as supply chain optimization, logistics, internet data routing, autonomous robot navigation and GPS navigation. Also this problem very important theoretically.

Academics have find many algorthim over time, to solve the shortest path problems under many constraints and network kinds. Algorithms include the Bellman-Ford algorithm, which can handle negative edge weights, Floyd-Warshall algorithm, between any pair of vertex we can find the shortest paths, the A* algorithm, which is general in artificial intelligence because of its heuristic based practice, and Dijkstra's algorithm, which is effective for graphs without negative weights. Sympathetic their differences and use cases is necessary for selecting the suitable method in practical applications because these algorithms, tradeoffs, proposals, and unique strengths have computational complexity, suitability, and efficiency for different kinds of applications and graph.

The main goal of this study is to methodically analyze the main shortest path algorithms used in graph theory. This work explains their limitations, working principles, advantages, time complexities, and highlights key application

areas. This paper is to help readers, especially researchers, students, develop a clearer sympathetic of how these algorithms work and when they are used [2, 3].

1 Literature Review

A necessary topic in graph theory, the shortest path problem (SPP) has lot applications in, computer networks, robotics, geographic information systems and transportation systems. Different algorithms have been presented throughout the years, starting from machine learning techniques and sophisticated heuristic to traditional exact methods. To maximize pathfinding efficiency and accuracy under a variety of constrictions.

Focusing on application domains, by many papers have offered thorough analyses of shortest path algorithms, theoretical foundations and performance traits. Classical algorithms of comparative analysis were provided by Magzhan and Jani [1], who emphasized trade-offs between optimality and computing complexity. Similar to this, Nosirov *et al*., [3], and Madkour *et al*., [2], examined a wide variety of algorithms, highlighting how algorithmic performance is impacted by graph size, density, and dynamic changes. These assessments were expanded to incorporate contemporary network routing settings and directed graph architectures, respectively, by more recent evaluations like Hadi and Ibrahim [4], and Sapundzhi *et al*., [5], which also identified performance hindrances and provided new standards.

Well-known algorithms like Floyd-Warshall [6], Bellman-Ford [7], and Dijkstra's [8], form the basis of short path problem solutions. These algorithms are still important today because of their deterministic nature and assured optimality. Their flexibility on real-world networks was examined in early performance evaluations by Zhan and Noon [9], and Dreyfus [10], who found that Bellman–Ford can handle negative edge weights at the expense of increased time complexity. At the same time, Dijkstra performs optimally in sparse graphs with non-negative weights. Recent studies [11-13], have reviewed these techniques in modern settings, comprehensive SDN systems, and smart transportation networks, evaluating their space to regulate to real-time constraints.

Finding a balance between computing efficiency and adaptation to dynamic, large-scale networks is still challenging, despite much exertion. Formal guarantees of optimality are lacking in many heuristic approaches, while AI-based solutions frequently need large amounts of training data and might not apply well to unknown network conditions. Furthermore, there hasn't been much effort put into scalable integration of real-time environmental elements (such as traffic and weather) into shortest path calculations.

1.1 Classical Algorithms:

The basis for pathfinding in networking applications and graph theory has been well-known by traditional shortest path methods, such as Dijkstra's and Bellman-Ford. Dijkstra's algorithm, which was first future in 1959, is famous for its effectiveness at determining the shortest path between a single source node (or vertex) and every other node in a graph with nonnegative edge weights. Though being less effective in terms of computational complexity, Bellman-Ford, which was first obtainable in 1958, can handle graphs with negative edge weights.

Contemporary research has looked into ways to improve and use these traditional techniques. The paper "An Efficient Routing Algorithm for SDN Using Bellman-Ford" [14], for example, highlights Bellman-Ford's suppleness to new networking models and shows how it may be used in software-defined networking environments. Furthermore, "Towards shortest path identification on large networks. "[15], is another outstanding work that addresses the ability problem of classical techniques when used in large-scale, real-world networks. The study highlights how important it is to time use and optimize memory, especially when working with high density graphics.

Classical algorithms differ in terms of performance; this is proved by Comparative research. The Similar assessments in "Comparative Analysis of Different Shortest Path Algorithms" and "Comparative Study on Bellman-Ford and Dijkstra" [11-16], make Bellman-Ford still valuable in more complex or dynamically weighted surroundings [17], even though Dijkstra is typically faster in networks with positive weights.

1.2 Comparative Studies of Shortest Path Algorithms:

Measured the performance of these algorithms over the years using a diversity of parameters, including accuracy, execution time, application in real-world networks, and memory utilization, to understand the advantages, disadvantages, and contextual applicability of shortest path algorithms. Needs a comparative study.

This paper, "Comparative Study on Bellman-Ford and Dijkstra"[16], for instance, looks at how effective both algorithms are on numerous types of graphs. It accomplishes that Bellman-Ford is more reliable in graphs where negative weights may exist, even if Dijkstra frequently works faster on graphs with nonnegative weights.

Current algorithms belong to comparison owing to extension to this comparison, such as A* and its differences in "Comparative Analysis of Different Shortest Path Algorithms"[11]. The study confirms that A* does better when a heuristic is present, especially in real-time steering systems and network-based maps.

In this paper, "A Comparison in Different Types of Shortest Path Algorithms for Smart City Applications" [18], proposes a useful assessment of algorithms in city settings. It concludes that when real-time constraints are current, heuristic and hybrid-based algorithms do better than conventional [19].

Additionally, Tamimi examines many methods, using performance standards on directed and undirected graphs in "Comparison Studies for Different Shortest Path Algorithms" [20]. The findings explain that no single approach is always the best option; instead, the ideal option is determined by variables such as update frequency, pathfinding precision, and graph density.

In finally years, "A Performance Comparison of Shortest Path Algorithms in Directed Graphs"[5], provided a quantitative assessment of various traditional and modern algorithms in controlled simulation environments.

1.3 Modern and Heuristic-Based Algorithms:

Traditional algorithms for shortest path, repeatedly perform poorly in real time and in terms of scalability due to the increasing complexity and size of real world networks these algorithms, like Dijkstra and Bellman-Ford. As a result, machine learning, contemporary algorithms that use optimization, or heuristics to increase efficiency, have emerged.

A* algorithm is one of the most public heuristic based methods. Changes to the heuristic function can significantly reduce computation time without sacrificing accuracy, as Chen explains in "An Improved a* Search Algorithm for Road Networks" [11]. In settings such as civilian traffic systems, the algorithm performs particularly well. To improve reliability, my routing in crowded regions, Werner & Zeitz suggest a hybrid approach that blends time-dependent A* potential with real-time traffic data. This method explains the possibility of adaptive shortest route calculation by dynamically updating edge weights depending on traffic estimates.

Strasser & Zeitz [21], made another breakthrough by introducing a narrow heuristic that speeds up A* search without compromising accuracy. For large-scale maps, when normal techniques become computationally costly, their algorithm works especially well. Additionally, deep learning has joined the industry. In "Deep Heuristic Learning for Real-Time Civil Pathfinding" [16], Abo El-Ela & Fergany [22], neural networks are trained to forecast the most promising routes based on past data. This method enables quick path selection, which is especially helpful for smart city infrastructures. Quantum computing is also being investigated as a possible game-changer. Early-stage algorithms that use quantum entanglement and superposition to simultaneously investigate several pathways are presented by researchers in “Advances in Quantum Algorithms for the Shortest Path Problem” [23]. Additionally, this is a significant area for study, even though it is still theoretical.

Randomized and Greedy k-domination techniques are two additional heuristic developments that are perfect for emergency routing systems and sensor networks since they narrow down the search space by choosing high-impact nodes.

1.4 Practical Applications for Algorithms in Real Life:

Theoretical Evolution in shortest path algorithms is necessary; their practical value is often determined by how well they work in practice. Numerous contemporary studies have focused on measuring and realizing these algorithms in actual settings, such as traffic systems, road networks, and town planning.

For example, Dijkstra, Bellman-Ford's, and A*, on real world city maps, are assessed in "Shortest path algorithms: an evaluation using real road networks" [9]. Without former accuracy, when using an appropriate heuristic, it's obvious. The results show that A* consistently performs faster than others. "Shortest Path Routing Performance Evaluation over SDN Environment" [24], assesses the performance of traditional algorithms in the context of software defined networking (SDN) in dynamic network topologies. The study stresses the need for additional real time reactive algorithms while also highlighting Bellman-Ford's flexibility in changeable contexts.

To improve way prediction in systems for navigation, studies like "A Graph-Based Analysis with Path Learning Algorithms and Finding "[25], research the combination of shortest path calculations and machine learning. In unexpected conditions, such as, traffic, bad weather, or road closures, this hybrid approach grows path stability. "A New Exact Algorithm for the Shortest Path Problem" [26], offers a deterministic method specifically designed for delivery in transportation systems, and logistics services, maximizing cost and route efficiency.

Also particular applications contain car sharing programs, where an A* based heuristic is suggestion in "Heuristic Optimal Meeting Point for Car Sharing" [27], to determine the better meeting places based on traffic and user locations. Analyzing tour time uncertainty, "Study on the Shortest Reliable Path of Stochastic Time Dependent Networks" proves how traditional deterministic algorithms can produce less than ideal or untrustworthy routes in stochastic settings. "Path Planning of Scenic Spots Based on Improved A*" [28], offers a fresh take on preference-based path planning in the tourism industry by tailoring the A* algorithm to strike a balance between shortest time and scenic preferences.

Finally, a comprehensive experiential review utilizing a diversity of datasets is provided in "An Appraisal of Some Shortest Path Algorithms" [10], which settles that application-specific tailoring is frequently necessary for optimal method presentation.

1.5 Specialized and Technical Enhancements:

Modern research has extended the use of traditional shortest path algorithms (SPP) by uniting them with sophisticated theoretical, machine learning, hybrid approaches, and methodologies to address subjects like, real-time data, dynamic environments and scalability.

For adaptive routing in dynamic networks, a good example is the Hybrid Algorithm Combining Dijkstra, Bellman-Ford, and Machine Learning, which blends learned forecasts with deterministic algorithms. This integration improves performance when traffic patterns are continuously shifting. According to the theory, "Sublinear Time Shortest Path in Expander Graphs" [29], presents a revolutionary technique for very big graphs with strong connections that achieves sublinear complexity, which is perfect for distributed systems or huge-scale systems, such as social networks.

In the argumentative but thought-provoking study "New Algorithm Beats Dijkstra's Time", specific graph sparsity patterns enable faster than Dijkstra performance. It offers helpful information for the creation of future algorithms, even though it is not noble reviewed. "Heuristic Path Selection Algorithms in SDN" assesses different adaptive and static heuristics in SDN-specific situations, stressing the meaning of dormancy minimization and topological consciousness in software-defined networks.

"Efficient Shortest Path Counting on Large Road Networks" [30], presents methods for counting the number of unique shortest pathways in addition to determining the shortest path, which is a useful feature for joblessness planning and robust routing. A way time cost optimal A* variation that concurrently reduces travel time and expense is published by SciDirect (2024) and can be used in emergency response or logistics.

Using modified Dijkstra algorithms, "Optimal Routing in Urban Road Networks" [31], investigates multi-criteria optimization in urban routing, including trip time, fuel cost, and environmental influence.

Finally, a full classification of robust optimization models for shortest path problems that address doubt, risk, and delay risk is provided by "Robust and Distribution Ally Robust Shortest Path Problems: A Survey"[32]. This classification is especially obliging for network resilience and transportation planning.

1.6 Summary of the Literature Review:

After rudimentary techniques similar to Bellman-Ford and Dijkstra's, to more complex differences like A*, Johnson's, and Floyd-Warshall algorithms, the studied literature proves a thorough evolution in shortest path algorithms. The basis for numerous adaptations appropriate for a range of subject spaces was established by these traditional approaches.

Relative research shows that Bellman-Ford is still useful for classifying negative cycles even though Dijkstra's method works well for graphs with nonnegative weights. Additionally, A* and its heuristics perform better in path planning with recognized terminus targets, whereas Floyd-Warshall is better for all-pairs pathfinding in dense graphs.

To address modern subjects including scalability, dynamic environments, and multi-criteria optimization, new developments combine heuristics, machine learning, real-time traffic prediction, and quantum computation. Specifically, time-dependent and hybrid models show prominent gains in routing efficiency in urban transportation systems and software-defined networks (SDNs).

Practical data is used in applications concerned with research to test these algorithms, to represent their usefulness in intelligent transportation, resource planning, and navigation systems. Furthermore, studies on stochastic and robust versions emphasize how crucial doubt management is for real-time applications.

Altogether belongings considered, the works offer a strong basis and highlight important research gaps and chances, especially in the areas of adaptive routing, interaction with intelligent systems, and scalable heuristics.

2. Comparative Analysis Framework:

In this section, we produce a systematic outline for analyzing and contrasting dissimilar graph theory shortest path algorithms. Highlighting each memory need, the algorithm's scalability, practicality is the goal, and computational efficiency. We achieve this by using a multicriteria method that blends experiential findings from earlier research with theoretical difficulty analysis.

Consistent metrics similar to space complexity and time, algorithmic design (dynamic programming, greedy, heuristic-based), and presentation on numerous graph types (e.g., weighted, unweighted, directed, undirected) are used to evaluate the algorithms in this comparison: Dijkstra's algorithm, Bellman-Ford, Floyd-Warshall, A*, and others. According to previous comparison research, these pointers were validated and developed.

We additionally consider domain-specific applications such as transportation networks, software-defined networks, and vehicular ad hoc networks. To evaluate algorithmic presentation in accurate environments [3, [2].

Through combining both theoretical and practical perspectives, this structure allows for a nuanced sympathy for each algorithm's strengths and limitations. For example, while Dijkstra's algorithm excels in nonnegative weighted graphs with comparatively low computational cost [8], Bellman-Ford is more suitable in situations involving negative edge weights, albeit with higher complexity (O(VE)) [7-33]. Floyd-Warshall, is limited in scalability due to its $O(n^3)$ complexity, although comprehensive for all pairs shortest paths, [6].

This regular comparison forms the basis for the following part, where algorithm tabulated analyses and specific evaluations will be presented in detail.

3. Comparative Analysis of Shortest Path Algorithms:

In this part of this paper we do complete comparison of the most famous shortest path algorithms in graph theory.

1. Efficiency of Computation (Complexity of Time and Space):

Shortest path algorithms different in their computational efficiency depending on the graph structure and the algorithm. Dijkstra's algorithm has a time complexity of $O(V^2)$ when perform with a simple array and $O((V + E) \log V)$ when a min heap or priority queue is used, beside a space complexity of $O(V)$. It is effective for graphs with nonnegative weights. Illustration 2 presents the calculation of the shortest path using Dijkstra's algorithm.

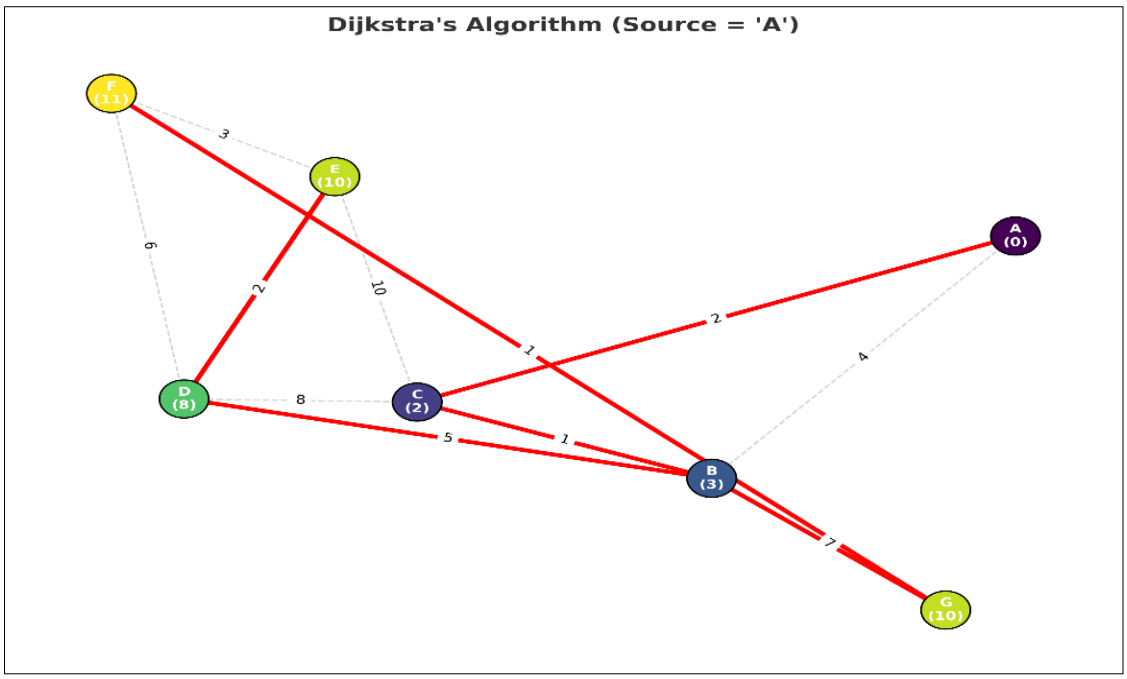

Illustration 1: calculating the shortest path in Dijkstra's algorithm

An another side, the Bellman Ford algorithm can handle negative edge weights while also detecting negative cycles. Has space complexity $O(V)$ and a time complexity of O(V × E). Illustration 2 presents the calculation of the shortest path using the Bellman Ford algorithm.

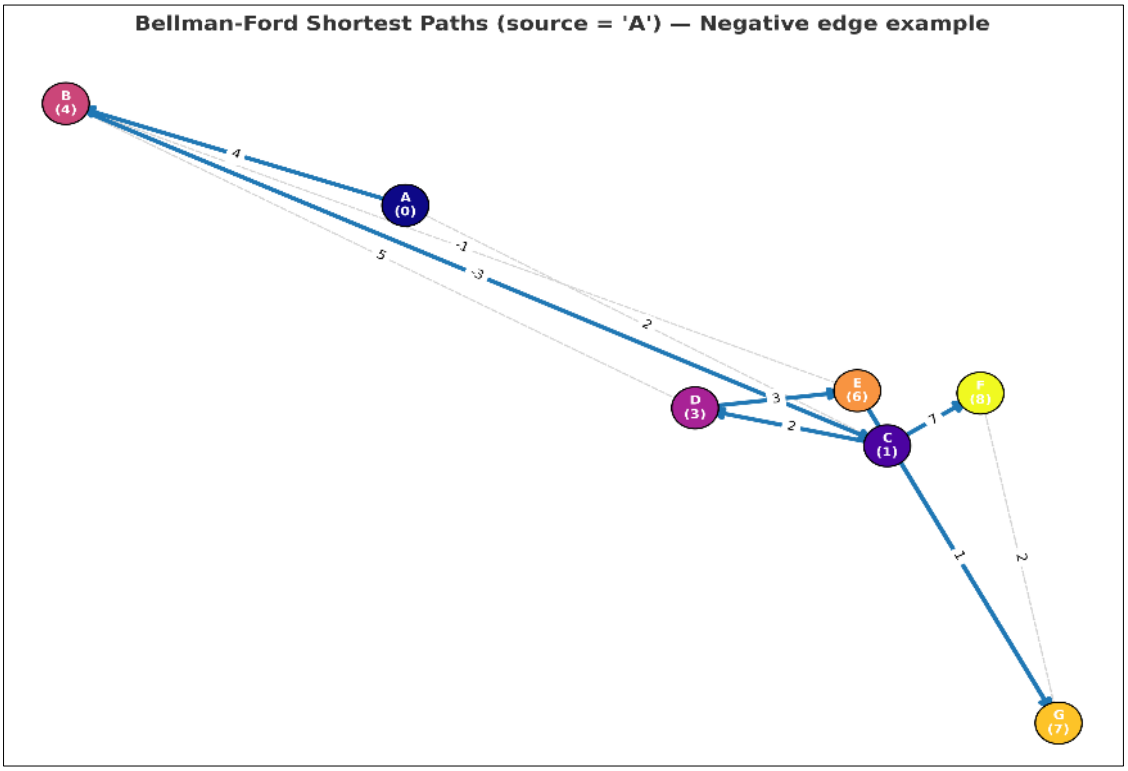

Illustration 2: calculating the shortest path in the Bellman Ford algorithm.

Floyd Warshall algorithm is used to compute all-pairs shortest paths in thick graphs, but it is considered less efficient for large, dispersed graphs. Its space complexity is $O(V^2)$ and time complexity is O(V³). Illustration 3 presents the calculation of the shortest path using the Floyd Warshall algorithm.

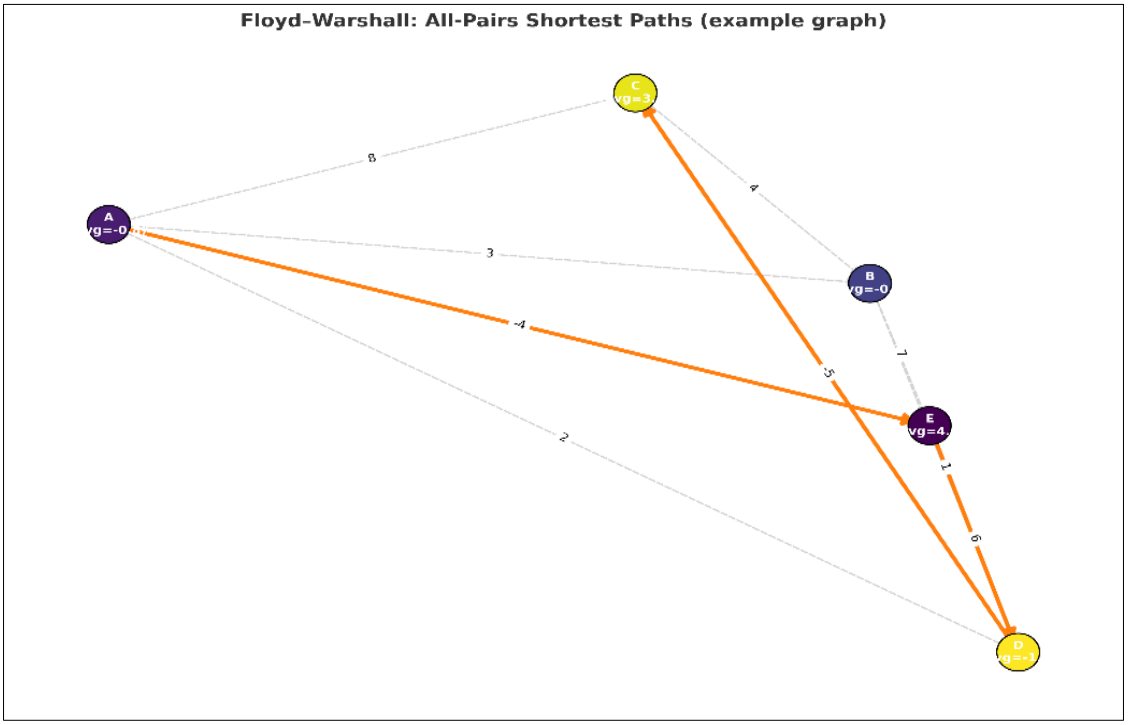

Illustration 3: calculating the shortest path in the Floyd–Warshall algorithm.

A*'s performance depends on the heuristic function. In the best case, it can arrive at $O(E)$ while in the worst case, it may take $O(b^d)$ where d the search depth and b is the branching factor, its space complexity is $O(V)$ and it ensures optimality if the heuristic is consistent and admissible. Recently, Bellman–Ford and Dijkstra this two algorithms are combined , in Johnson's algorithm to compute all pairs shortest paths for dispersed graphs with possible negative weights, attaining Space Complexity $O(V + E)$ and Time Complexity O(V^2 log V + VE) Each of these algorithms offers different tradeoffs between space efficiency and time, depending on specific application requirements, graph density and edge weights. As illustrated in Table 1 and Figure 1.

Table 1: Efficiency of Computation (Complexity of Time and Space

Algorithm	Time Complexity	Space Complexity	Efficiency Notes
Dijkstra	$O((V + E) log V)$	$O(V)$	efficient with priority queue; best with Fibonacci heap
Bellman-Ford	$O(V \times E)$	$O(V)$	Slower than Dijkstra; linear passes over edges
Floyd-Warshall	$O(V^3)$	$O(V^2)$	High overhead for large graphs
A*	$O(E)$ $heuristic - bound$	$O(V)$	Dependent on heuristic; efficient if h(n) is admissible
Johnson's	$O(V^2 log V + VE)$	$O(V + E)$	faster than Floyd-Warshall for sparse graphs
Hint: - where (O): indicates the algorithm's time complexity growth rate, (V) number of vertices, and (E): number of edges.			

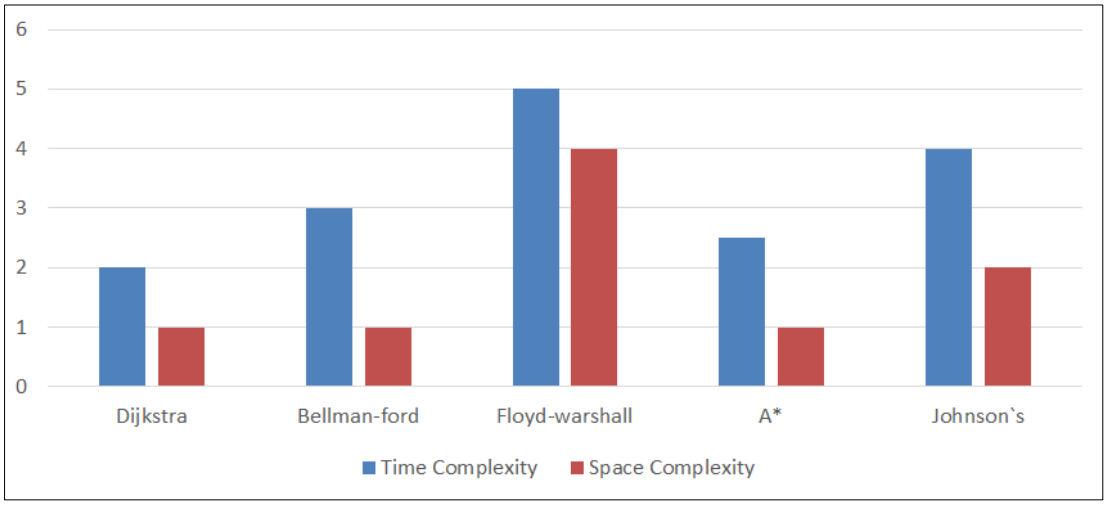

Figure 1: Complexity of Time and Space

It can be seen that Floyd–Warshall is the highest in time complexity and space complexity, while Dijkstra is relatively lower.

2. Capabilities for Handling Weights (Negative and Dynamic):

The ability of shortest-path algorithms to handle different types of edge weights different significantly. Dijkstra's algorithm is unsuitable for negative edges cannot correctly process them, but it is limited to nonnegative edges, making it unsuitable for graphs with negative weights. Bellman Ford, whereas, is able of handling negative edge weights, and can also detect the existence of negative cycles, which makes it more flexible for graphs where negative weights happen. Floyd–Warshall can handle negative edge weights also, provided there are no negative cycles in the graph, and it computes shortest paths for all pairs of nodes, making it appropriate for dense graphs with mixed weights. Johnson's algorithm extends this ability by first reweighting the edges using Bellman–Ford to remove negative weights and then applying Dijkstra for each vertex. However, it can efficiently compute all-pairs shortest paths even in graphs with negative edge weights, but it assumes that negative cycles are missing. Concerning dynamic weights, most of these classical algorithms are designed for stationary graphs. However, progressive or real-time variants of these algorithms exist to adapt to changes in edge weights, although they need additional modifications. As illustrated in Table 2 and Figure 2.

Table 2: Capabilities for Handling Weights (Negative and Dynamic)

Algorithm	Negative Weights	Negative Cycles	Notes
Dijkstra	Not allowed	Not allowed	Can fail or give incorrect paths
Bellman-Ford	Supported	Detectable	Preferred in uncertain or loss networks
Floyd-Warshall	Supported	Detectable	All-pairs plus cycle detection
Johnson's	Supported	Detectable	Uses Bellman-Ford as a preprocessing step

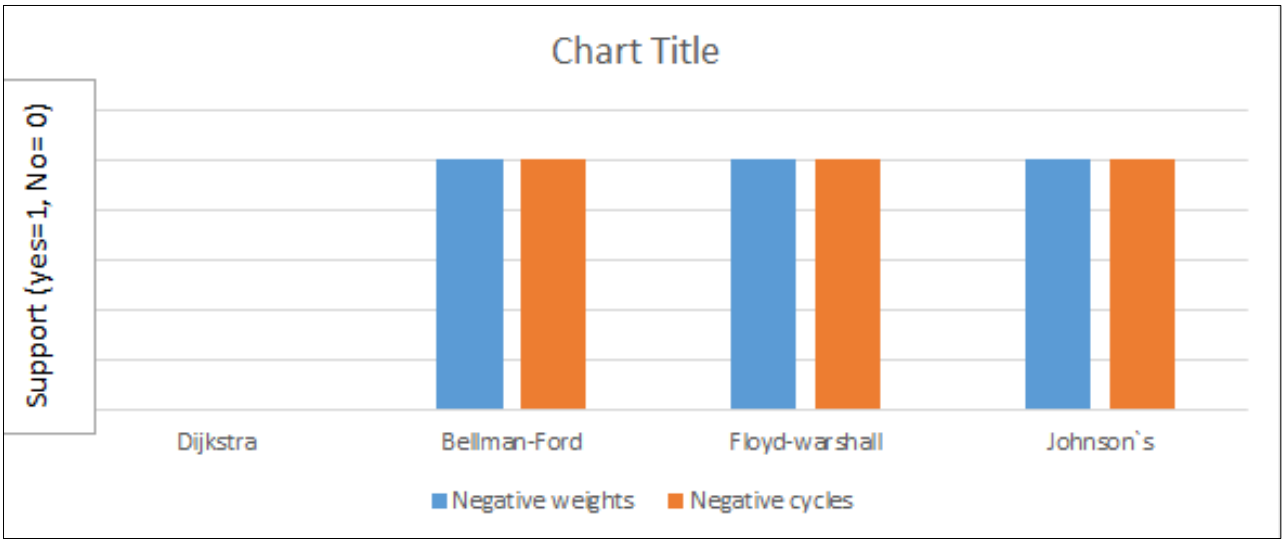

Figure 2: Negative and Dynamic

3. Appropriateness and Scalability for Big Graphs:

The adequacy and scalability of shortest-path algorithms for extensive graphs depend on their design and computational complexity. Dijkstra's algorithm is highly efficient for sparse, large graphs, especially whsen applied with a priority queue or min-heap. It is suitable for real-world networks with non-negative edge weights. Bellman–Ford, while more versatile due to its ability to handle negative weights, has higher time complexity $O(V(E))$, which limits its ability for big graphs. Floyd–Warshall computes all-pairs shortest paths with $O(V^3)$ time complexity, making it less suitable for big graphs, but it executes well for dense graphs of moderate size. A* performs well in pathfinding for massive graphs when an effective heuristic is available, as it can considerably decrease the search space and computation time compared to uninformed algorithms, making it highly scalable for real-world applications like navigation systems. Finally, Johnson's algorithm is particularly suitable for sparse, large graphs that may contain negative edge weights, as it combines Bellman–Ford and Dijkstra efficiently; however, its performance may degrade for extremely dense graphs due to the initial reweighting step. As illustrated in Table 3 and Figure 3.

Table 3: Appropriateness and Scalability for Big Graphs

Algorithm	Scalability Rating	Suitable Graph Size	Notes
Dijkstra	High (with binary heap)	Medium–Large	Works best on sparse graphs
Bellman-Ford	Medium	Small–Medium	Slows down with dense graphs
Floyd-Warshall	Low	Small	Impractical on large graphs
A*	High	Medium–Large	Depends on the quality of the heuristic
Johnson's	Very High	Large (Sparse)	Designed for all-pairs shortest path

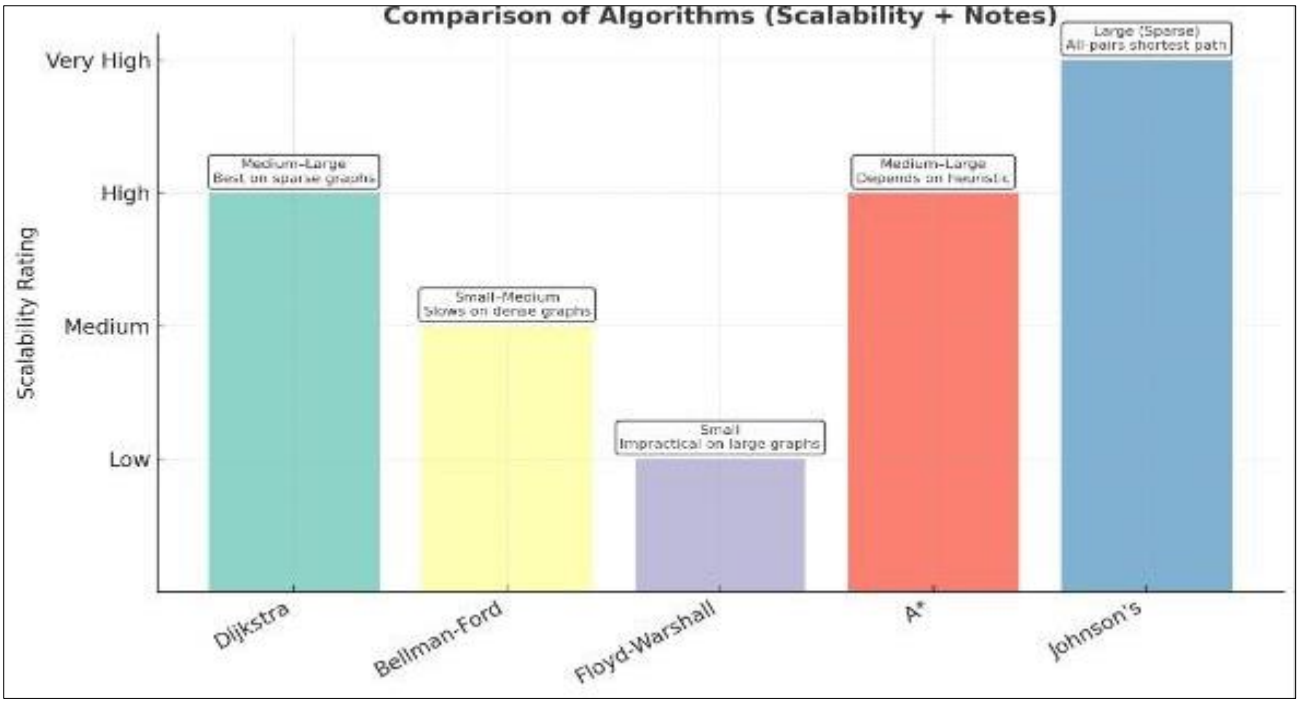

Figure 3: Scalability for Big Graphs

4. Application Environments and Use Cases:

The application environments and use cases of shortest-path algorithms are contingent on their efficiency and properties. Dijkstra's algorithm is extensively used in networks with non-negative edge weights, such as GPS navigation systems, transportation networks, and routing in communication networks, caused by to its efficiency in finding the shortest paths from a single source. Bellman–Ford is suitable for graphs containing negative edge weights, making it useful in arbitrage detection and financial network scenarios where edge costs can be negative or adjusted dynamically. Floyd–Warshall is mainly applied in scenarios requiring all-pairs shortest paths, such as urban traffic planning, dense graph computations, and network latency analysis in operations research. A* is commonly applied in robotics for pathfinding in large search spaces, artificial intelligence, autonomous vehicles, robotic navigation, and games, benefiting greatly from heuristic functions to guide the search efficiently. Johnson's algorithm is ideal for sparse, large graphs with negative weights, making it useful in network optimization, advanced transportation planning, and computational problems where all-pairs shortest paths need to be computed efficiently without the restriction of non-negative edges. As illustrated in Table 4 and Figure 4.

Table 4: Application Environments and Use Cases

Algorithm	Best for	Example Applications
Dijkstra	Road networks, routing with known costs	GPS, network routing
Bellman-Ford	Communication networks with variable loss	Distributed routing protocols
Floyd-Warshall	Dense small networks, simulations	Traffic simulations, classroom examples
A*	AI navigation, robotics	Game AI, drone flight paths
Johnson's	All-pairs routing in sparse large networks	Airline route planning, internet graphs

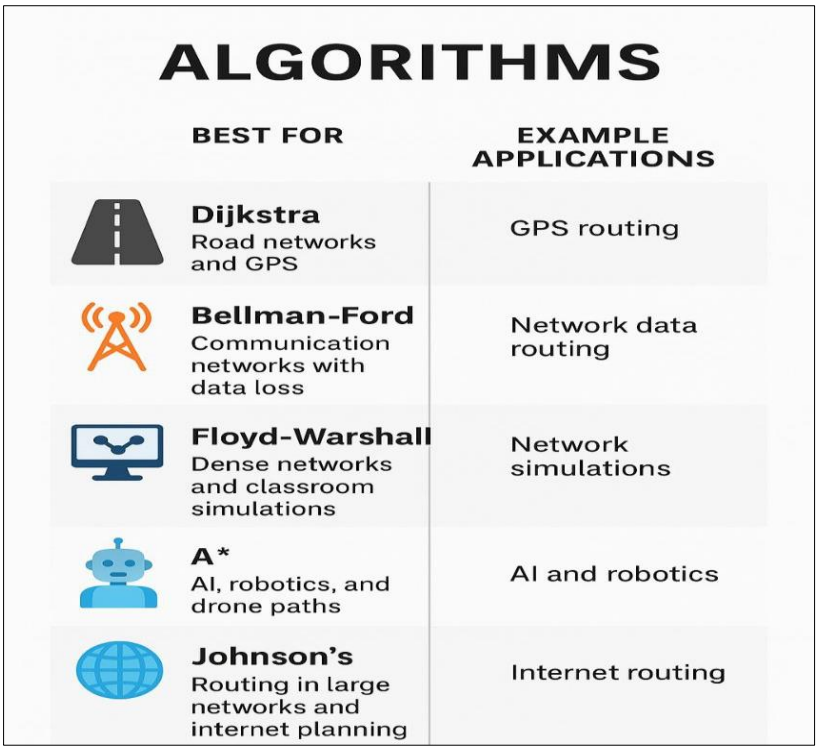

Figure 4: Application Environments and Use Cases

3.1 Summary of Comparative Findings:

Each algorithm has advantages and disadvantages. Heuristic and machine learning based approaches do well in dynamic, uncertain contexts, whereas classical algorithms such as Bellman-Ford and Dijkstra provide simplicity and reliability. The problem context, update frequency, including network size, and computational constraints, determine the top option.

3.2 Comparative Analysis of Classical, Heuristic, and Hybrid Shortest Path Algorithms:

Shortest-path algorithms can be widely classified into classical, heuristic, and hybrid approaches, each with distinct characteristics, limitations, and advantages. As illustrated in Table 5.

Table 5:Comparative Analysis of Classical, Heuristic, and Hybrid Shortest Path Algorithms

Hybrid Algorithms ML + A*, Reinforcement +Dijkstra	Heuristic Algorithms A*, D*, Genetic Algorithms	Classical Algorithms: Dijkstra, Bellman-Ford, Floyd-Warshall	Criteria
Combine classical + heuristic or intelligent learning mechanisms	Use heuristics to guide search (e.g., Euclidean distance)	Deterministic, based on graph traversal and relaxation techniques	Basic Principle
Variable; often better in practice, but with training overhead	Depends on heuristic quality; A*: best-case O(E)	Depends on the algorithm: Dijkstra: O((V+E) log), Floyd: $O(V^3)$	Time Complexity
May require additional memory for models or heuristics	Moderate	Generally moderate to high	Space Complexity
Possible with proper integration (e.g., hybrid Bellman-A*)	Not designed for negative weights	Bellman-Ford Except Dijkstra Floyd-Warshall	Handling Negative Weights
Excellent, especially with online learning	Yes, many are adaptive like D*, TDA*	Mostly static (except dynamic variants)	Adaptability to Dynamic Graphs
Often optimal or better in dynamic environments	If heuristic is admissible (A*), otherwise near-optimal	Guaranteed optimal paths	Accuracy of Solution

High, especially in sparse or real-time routing systems	High scalability, especially in sparse graphs	Limited for large or dense graphs (especially Floyd-Warshall)	Scalability
High (due to integration of ML or multiple techniques)	Moderate to complex	Simple to moderate	Complexity of Implementation
Very effective in smart transportation, autonomous systems	Used in robotics, games, and GPS	Not Limited	Use in Real-Time Systems
Reinforcement Learning + Dijkstra, Deep Q-Learning Routing	A*, D*, Time-Dependent A*, Ant Colony Optimization	Dijkstra, Bellman-Ford, Floyd-Warshall, Johnson's	Examples of Algorithms
mart cities, self-driving vehicles, adaptive internet routing	AI navigation, robotics, traffic estimation	Traditional routing, network planning, and static maps	Common Application Areas
Yes (needs data and training phase)	Mostly no (except GA, ACO)	Not	Learning/Training Required

Numerous important proposals about the choice and use of shortest path algorithms can be completed in light of the comparative study that was compiled in the preceding table:

1. ***Context-Driven Selection:*** The top choice mostly relies on the particular use case; there is no one method that is always better. Floyd-Warshall and Dijkstra are examples of classical algorithms that are effective and reliable in static and predictable situations (such as routing in stable computer networks). Hybrid algorithms or heuristics, on the other hand, are more suited for dynamic situations like smart transportation systems or mobile ad hoc networks.
2. ***Heuristics for Real-Time Systems:*** Heuristic algorithms, like A*, Ant Colony Optimization, and D*, provide a good equilibrium between speed in time-sensitive applications and accuracy, where speed is more important than complete optimality. They work particularly well in robotics, games, and spatial decision-making systems.
3. ***Hybrid Approaches for Complex Scenarios:*** In circumstances when there is indecision, a lot of data, or changing network states, hybrid algorithms must be used. They are more suitable for new applications like intelligent traffic control, SDN routing, and IoT, because they are able to learn, integrate historical, and adapt or real time data.
4. ***Future Integration and Scalability:*** Scalability becomes crucial as systems and networks become more complicated. Though they may result in higher application difficulty, and processing overhead, hybrid approaches that use machine or AI learning provide charming alternatives. So, it is necessary to take into account a balance between efficiency and practicality.
5. ***Resource Tradeoffs and Constraints:*** Once choosing between these algorithm types, decision makers should consider tradeoffs, for example, memory practice, preprocessing time, and algorithm convergence rates. Heuristic approaches are faster but rarely imprecise, hybrid approaches are more adaptable, but demand more resources, and classical approaches are easy to use but may have problem scaling.

4. DISCUSSION

Shortest path algorithms have advanced meaning. The capacity of shortest path algorithms to maintain a balance between computing efficiency and adaptability while dealing with complex network routing problems is among their most attractive features. While Dijkstra's method is particularly commended for its efficacy in graphs with nonnegative weights, Bellman Ford extends its application to include scenarios with negative edge weights. Their disadvantage, however, become evident in dynamic networks where real time, flexibility is preventing by the demand for preprocessing. It is believed that, while classical algorithms are very beneficial for evidently specified, static problems, they are not adaptable enough for modern, dynamic systems. Through providing heuristic-driven efficiency and flexibility, heuristic algorithms like A* and Ant Colony Optimization (ACO), however, offer creative solutions. A* is perfect for applications such as navigation and robotics because it integrates actual costs with heuristic forecasts to guarantee optimal solutions. On the other hand, ACO used biological inspiration to improve pathways in large-scale, flexible networks dynamically. Even though these algorithms perform extraordinarily well in dynamic contexts, their applicability may be constrained by their dependence on heuristic quality for A* and computing complexity for ACO for dynamic and extensive systems. It is believed that heuristic algorithms offer a significant advance over conventional approaches, however, there are still numerous subjects and unmet research needs that stop them from being fully applied in complicated and dynamic environments such as Real-time Restraints, in real-time systems where quick reactions are crucial, like vehicular ad hoc networks and emergency routing systems, classical algorithms, similar to Bellman-Ford and Dijkstra's, frequently fail, and Scalability Limits, some algorithms find it hard to keep performance and efficiency as graph sizes grow in large-scale networks, like transportation systems or social graphs. Dynamic Graph Treatment, while many algorithms are based on static graphs, real-world situations frequently involve dynamic changes, including traffic differences or node failures, which are not sufficiently addressed by current techniques. Finally, Application Exact Optimization, Performance differs greatly based on graph structure, objective metrics (e.g., distance vs. time), and system constraints; there isn't one size fits

altogether approach. These problems show that shortest path computation requires more flexible contextually sensitive and flexible methods.

5. CONCLUSIONS

- This review presents a comprehensive comparative analysis of classical, heuristic, and hybrid shortest path algorithms in graph theory, emphasizing that no single algorithm is universally optimal across all problem settings.
- Classical algorithms (e.g., Dijkstra and Bellman–Ford) remain foundational due to their deterministic behavior, simplicity, and reliability, particularly for graphs with well-structured topology or nonnegative edge weights.
- Despite their strengths, classical methods exhibit limitations in large-scale, dynamic, or complex networks, where high computational cost and limited adaptability reduce their effectiveness.
- Heuristic algorithms (e.g., A*, Ant Colony Optimization, and Genetic Algorithms) offer greater flexibility, scalability, and robustness, making them suitable for real-world applications such as GPS navigation, robotics, and adaptive routing.
- However, heuristic approaches are often **s**tochastic in nature, leading to variability in solution quality and a strong dependence on carefully designed heuristic functions or parameter tuning.
- Hybrid algorithms, which integrate the optimality and correctness of classical methods with the adaptability of heuristic techniques, demonstrate promising performance in terms of both accuracy and execution time.
- Examples such as *A–Dijkstra hybrids and Genetic–ACO models** illustrate the potential of hybrid strategies to address large-scale and computationally challenging graph problems more effectively.
- Although hybrid approaches are still evolving, they represent a promising direction for future research, particularly for applications requiring efficient, accurate, and scalable shortest path solutions in complex graph environments.

REFERENCES

1. Magzhan, K., & Jani, H. M. (2013). A review and evaluations of shortest path algorithms. Int. J. Sci. Technol. Res, 2(6), 99-104.
2. Madkour, H., El-Sayed, A., & Khafagy, M. (2017). Survey of Shortest Path Algorithms. International Journal of Computer Applications.
3. Nosirov, S., Azimov, D., & Adilov, A. (2022). A Review of the Shortest Path Problem in Graph Theory. International Journal of Engineering Science and Computing.
4. Hadi, H. M., & Ibrahim, I. M. (2025). A Comprehensive Review of Shortest Path Algorithms for Network Routing. Asian Journal of Research in Computer Science, 18(3), 152-175.
5. Sapundzhi, F., Danev, K., Ivanova, A., Popstoilov, M., & Georgiev, S. (2025). A Performance Comparison of Shortest Path Algorithms in Directed Graphs. Engineering Proceedings, 100(1), 31.
6. Floyd, R. W. (1962). Algorithm 97: shortest path. Communications of the ACM, 5(6), 345-345.
7. Bellman, R. (1958). On a routing problem. Quarterly of applied mathematics, 16(1), 87-90.
8. Dijkstra, E. W. (2022). A note on two problems in connexion with graphs. In Edsger Wybe Dijkstra: his life, work, and legacy (pp. 287-290).
9. Zhan, F. B., & Noon, C. E. (1998). Shortest path algorithms: an evaluation using real road networks. Transportation science, 32(1), 65-73.
10. Dreyfus, S. E. (1969). An appraisal of some shortest-path algorithms. Operations research, 17(3), 395-413.
11. Chen, K. Y. (2022). An improved a* search algorithm for road networks using new heuristic estimation. arXiv preprint arXiv:2208.00312.
12. Nayak, U., Pandey, R., & Chourasia, U. (2018). A comparative study on analysis of various shortest path algorithms on GPU using OPENCL. International Journal of Computer Applications, 179(40), 1–5. https://doi.org/10.5120/ijca2018916948.
13. Wang, H. Y., Ji, C. J., & Ji, X. J. (2013). Analysis on improved Dijkstra algorithm in intelligent transportation system. Applied Mechanics and Materials, 385–386, 877–881. https://www.researchgate.net/publication/272619130_Analysis_on_Improved_Dijkstra_Algorithm_in_Intelligent_Transportation_System.
14. Kumar Rao, K. P., & Murugan, T. S. (2019). An Efficient Routing Algorithm for Software Defined Networking Using Bellman-Ford Algorithm. International Journal of Online & Biomedical Engineering, 15(14).
15. Selim, H., & Zhan, J. (2016). Towards shortest path identification on large networks. Journal of Big data, 3(1), 10.
16. Comparative Analysis of Different Shortest-Path Algorithms. (2022). International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 8(6), 652–657. https://ijsrcseit.com/CSEITCN228652.
17. Martin, D. P. (2017). Dynamic shortest path and transitive closure algorithms: A survey. arXiv preprint arXiv:1709.00553.

18. Abhisek Singh, & Singh, V. (2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm, their Implementation in Shortest Route network system. International Journal of Economic Perspectives, 12(1), 120–136. Retrieved from https://ijeponline.org/index.php/journal/article/view/266.
19. Zhao, D., Ni, L., Zhou, K., Lv, Z., Qu, G., Gao, Y., ... & Zhang, Q. (2024). A Study of the Improved A* Algorithm Incorporating Road Factors for Path Planning in Off-Road Emergency Rescue Scenarios. Sensors, 24(17), 5643.
20. Abu-Ryash, H., & Tamimi, A. (2015). Comparison studies for different shortest path algorithms. International Journal of Computers and Applications, 14(8), 5979-5986.
21. Strasser, B., & Zeitz, T. (2019). A fast and tight heuristic for A* in road networks. arXiv preprint arXiv:1910.12526.
22. Abo El-Ela, A. A., & Fergany, H. A. (2024). Deep reinforcement learning for route optimization in smart cities. International Journal of Artificial Intelligence and Smart Systems. https://arxiv.org/abs/2411.05044
23. Wesołowski, A., & Piddock, S. (2024). Advances in quantum algorithms for the shortest path problem. arXiv preprint arXiv:2408.10427.
24. Ghimire, R., & Basnet, R. K. (2024). Shortest Path Routing Performance Evaluation over SDN Environment. Journal of Electronics and Informatics, 5(4), 405-422.
25. Muthuvel, P., Pandiyan, G., Manickam, S., & Rajesh, C. (2025). Optimizing Road Networks: A Graph-Based Analysis with Path-finding and Learning Algorithms. Int. J. Intell. Transp. Syst. Res., 23(1), 315-329.
26. Yuan, H., Hu, J., Song, Y., Li, Y., & Du, J. (2021). A new exact algorithm for the shortest path problem: An optimized shortest distance matrix. Computers & Industrial Engineering, 158, 107407.
27. Baudru, J., & Bersini, H. (2024, September). Heuristic Optimal Meeting Point Algorithm for Car-Sharing in Large Multimodal Road Networks. In VEHITS (pp. 427-436).
28. Wang, X., Zhang, H., Liu, S., Wang, J., Wang, Y., & Shangguan, D. (2022). Path planning of scenic spots based on improved A* algorithm. Scientific reports, 12(1), 1320.
29. Alon, N., Grønlund, A., Jørgensen, S. F., & Larsen, K. G. (2023). Sublinear time shortest path in expander graphs. arXiv preprint arXiv:2307.06113.
30. Qiu, Y. X., Wen, D., Qin, L., Li, W., & Li, R. H. (2022). Efficient shortest path counting on large road networks. Proceedings of the VLDB Endowment.
31. Grujic, Z., & Grujic, B. (2025). Optimal Routing in Urban Road Networks: A Graph-Based Approach Using Dijkstra's Algorithm. Applied Sciences, 15(8), 4162.
32. Filippi, C., Maggioni, F., & Speranza, M. G. (2025). Robust and distributionally robust Shortest Path Problems: A survey. Computers & Operations Research, 107096.
33. Ford, L. R. (1956). Network flow theory. Santa Monica, CA: RAND Corporation.
34. Abdulrahman, A. A., Abed, S. S., Muhi-Aldeen, H. M., Tahir, F. S., & Khlaponin, Y. (2022). Deep learning algorithm for detecting skin diseases with new DWT technology. EUREKA: Physics and Engineering, (3), 211–220. https://doi.org/10.21303/2461-4262.2022.002284.
35. Alrasheedi, N. D. N., Masseran, N., & Mohd Tajuddin, R. R. (2025). Estimation of Generalized Extreme Value Parameters for Predicting Extreme Air Pollution Events in Klang Valley: A Comparative Study of Optimization Algorithms. Malaysian Journal of Mathematical Sciences, 19(2).
36. Ambreen, N. H. (2013). Wireless sensor network through shortest path route. International Journal of Emerging Technology and Advanced Engineering, 3(2), 158-161.
37. Cerny, J., et al. (2022). Hybrid routing with machine learning in graphs. Applied Sciences, 12(3), 1342. https://www.mdpi.com/2076-3417/12/3/1342.
38. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill.
39. Halim, M. A., et al. (2021). Shortest path problem in transportation. International Journal of Scientific Research in Science and Technology (IJSRST), 8(1), 239–245. https://ijsrst.com/?utm_source=chatgpt.com.
40. Martin, D. P. (2017). Dynamic shortest path and transitive closure algorithms: A survey. arXiv preprint arXiv:1709.00553.
41. Mukhlif, F., & Saif, A. (2020, February). Comparative study on Bellman-Ford and Dijkstra algorithms. In Int. Conf. Comm. Electric Comp. Net.
42. Parakkat, A. D., & Joseph, P. (2016). Heuristic Algorithms for Finding Area Constrained Non-Convex kkk-gons. Malaysian Journal of Mathematical Sciences, 10, 131-142.
43. Strasser, B., & Zeitz, T. (2019). A fast and tight heuristic for A* in road networks. arXiv preprint arXiv:1910.12526.