2025 Tenth International Conference on Science Technology Engineering and Mathematics (ICONSTEM) | 979-8-3315-6187-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICONSTEM65670.2025.11374724

Comparative Evaluation of Dijkstra's Algorithm and Shortest Path Strategies in Mobile Ad Hoc Networks (MANETs)

Revathy G[1], K.Agalya[2], R.Prabu[3], R. Nareshkumar[4*], S.L. Swarna[5], Gokulakrishnan V[6]

[1]Department of Computer Science and Engineering, Vels Institute of Science, Technology and Advanced Studies (VISTAS), Chennai,
[2]Department of Computer Science and Engineering, Sri Eshwar College of Engineering, India
[3]Department of Information Technology, VelTechMultitechDr.RangarajanDr.SakunthalaEngineeringCollege , Chennai, India.
[4]School of Computing, SRM Institute of Science and Technology Tiruchirappalli,,Tamilnadu, India
[5]Department of Computer Science and Engineering, Sona College of Techonology,, Salem, Tamilnadu,
[6]Department of Computer science and Business systems, K Ramakrishnan college of Engineering, Chennai

Emailid:revathy.se@vistas.ac.in, agalya.k@sece.ac.in, dprpit@gmail.com, nareshravics@gmail.com, swarna.cse@sonatech.ac.in, gokul.dsec@gmail.com

Abstract-Under broad application, the shortest path problem in graph-based network environments is generally treated with the use of Dijkstra's Algorithm. Due to the accelerating development of network technologies and the increasing need for real-time routing support, pathfinding algorithms have been confronted with continuously mounting challenges as far back as the late 1980s in areas of performance, scalability, and responsiveness. This research offers a comparative analysis of Dijkstra's Algorithm for contemporary networking environments, with the network size, topology, and implementation structure being considered. It first presents the theoretical background behind Dijkstra's Algorithm and then examines its real-world efficiency through the use of case studies for computer networks, Mobile Ad Hoc Networks (MANETs), and Internet of Things (IoT) systems. Important performance metrics are time complexity, memory usage, and running speed in varying implementations, including array-based, binary heap, and Fibonacci heap formats. In addition, the paper presents a more comprehensive view by comparing Dijkstra's Algorithm with other popular shortest path algorithms, including A*, Bellman-Ford, and Floyd-Warshall. Experimental simulations conducted on real-world and synthetic network datasets evaluate its scalability at higher node density and network size.

Keywords—Dijkstra's Algorithm, Shortest Path, Network Routing, Graph Algorithms, Performance Analysis, A Algorithm, Bellman-Ford, Modern Networks, Pathfinding, Real-Time Applications.*

I.INTRODUCTION

Dijkstra's algorithm is a pivotal greedy graph search method intended to determine the shortest routes from a singular source node to all other nodes in a weighted graph with non-negative edge weights, as first introduced by Dutch computer scientist Edsger W. Dijkstra in 1959. One, two, three. The method works by repeatedly choosing the unvisited node with the least tentative distance from the source and revising the shortest route estimations for its adjacent nodes, ensuring all edge weights are non-negative to maintain accuracy. Four, five, six It is universally acknowledged as a fundamental element in computer science, especially in algorithm design, data structures, and optimization issues, such as routing, network analysis, and pathfinding. Seven Its applications include practical sectors like GPS navigation, network routing protocols such as Open Shortest Path First (OSPF), robotics, and transportation systems, where it is used to calculate best paths and regulate network traffic. 8 9 The technique underpins several static shortest path calculations and is pertinent in both centralized and distributed network contexts, facilitating efficient route determination and network optimization.

Conversely, fast-developing network technologies of the contemporary era create a new direction for Dijkstra's Algorithm. The new generation of networks requires extreme node densities, dynamic topologies with high speed, and high-performance levels. With the emergence and implementation of MANETs and IoT environments on one side, real-time communication protocols on the other, have all boosted the quest for routing schemes that can be adaptable and scalable. Under such a scenario, it is essential to ask whether Dijkstra's Algorithm remains relevant today or not. This paper tries to give a bird's

Authorized licensed use limited to: SRM Institute of Science and Technology Kattankulathur. Downloaded on February 23,2026 at 09:31:30 UTC from IEEE Xplore. Restrictions apply.

eye view of its relevance in today's networks and compare its performance with some other famous shortest path algorithms.

At its core, Dijkstra's Algorithm operates by incrementally selecting the unvisited node with the lowest tentative distance, updating the shortest paths to its neighbours, and repeating this process until the destination is reached. The algorithm guarantees optimal paths in graphs with non-negative edge weights. However, its performance is heavily influenced by the choice of data structures used to manage the priority queue.

In computer networks, it is used in protocols, such as OSPF (Open Shortest Path First), because it is deterministic and reliable. In MANETs, where nodes join or leave the network a lot, static assumptions of conventional Dijkstra's Algorithm can result in suboptimal or stale routes, which have to be modified using different mechanisms such as bidirectional search or dynamic re-computation. For IoT settings with limited memory and power resources, lightweight versions of the algorithm may better suited.

To put more context to Dijkstra's Algorithm, we also contrast it with other significant shortest path algorithms. Bellman-Ford algorithm can support graphs with negative weights and is better suited for distributed environment but has greater time complexity. As an informed search algorithm, A* uses heuristics to steer the search, and typically converges faster than Sim to Exp in most real-world cases. Because the Floyd-Warshall algorithm is computationally expensive for all-pairs shortest paths, it is less suitable for those of large-scale and sparse networks. The comparison explains under what conditions Dijkstra's algorithm performs well or can be very slow and it gives good suggestions to network providers and system developers.

Experimental testing employs a blend of actual and artificial network topologies to represent appropriate operating environments and quantify appropriate project measures (e.g., processing time of individual packets, space needed to hold state data, and/or responsiveness in a dynamic network).The experimental results have illustrated that there is no single superior version of Dijkstra's Algorithm. The best approach is dependent on the application's target, prioritizing responsiveness for any mobility application, or bounded resource consumption (e.g., battery and processor utilization) for embedded systems.

In summary, although Dijkstra's Algorithm is foundational to pathfinding and routing, the current network architecture requires careful consideration of the nature of the data structures employed, the characteristics of the physical network, and how real-time adaptability is supported in routing protocols. This paper has attempted to provide an appreciation for these issues that would inform the appropriate use of Dijkstra's Algorithm for a diversity of applications, a rapidly-developing technology area where data is being exchanged in many formats.

II.Related Work

Over the years, many papers have studied the theoretical basis and applications of shortest path algorithms in the context of networks. This section will provide a summary of the major contributions made relating to Dijkstra's Algorithm in the context of its classical applications, data structure implementations, comparative work with other shortest path or related algorithms, and Dijkstra's Algorithm knowledge usage in contemporary networks e.g., MANETs, IoTs, real-time systems, etc.

2.1 Classical Applications of Dijkstra's Algorithm

The original version of Dijkstra's Algorithm has been largely applied to fixed network environments, including telephone exchanges, transport, and early computer networks. In traditional networking, there are protocols such as OSPF (Open Shortest Path First) based on Dijkstra's Algorithm to find the optimal routing path. Research such as Moy [1] has also delved into the effects of Dijkstra's deterministic algorithm on link-state routing protocols by providing significant stability to those protocols' paths.

2.2 Data Structure Optimizations

The effect of different priority queue implementations on performance is one of the most researched areas of Dijkstra's Algorithm. The normal algorithm, with arrays, has a time complexity of $O(V^2)$.Carmen et al. [2] pointed out that binary heaps reduced this complexity to $O\big((V+E)\log V\big)$Later Fredman and Tarjan [3] introduced the Fibonacci heap, which lowered it to$O\big((V+E)\log V\big)$, and improved performance in sparse

Authorized licensed use limited to: SRM Institute of Science and Technology Kattankulathur. Downloaded on February 23,2026 at 09:31:30 UTC from IEEE Xplore. Restrictions apply.

graphs.

The next set of empirical studies, such as Cherkassky et al. [4], were influential in establishing a relative performance measure for these implementations under various graph structures and densities. The results of the evaluations were straightforward; the Fibonacci heaps turned out to be the most scalable algorithm; whereas binary heaps, on the other hand, are seen to be swifter in reality due to their lower constant overheads in the methods.

2.3 Comparisons with Alternative Shortest Path Algorithms

Several studies have compared Dijkstra's Algorithm with other shortest path algorithms to highlight their respective strengths and weaknesses:

2.3.1 *Bellman-Ford Algorithm*

While Dijkstra's Algorithm is unable to handle negatively valued edges, the Bellman-Ford algorithm can. Despite having a time complexity of O(VE), it has properties of being a distributed and iterative algorithm and is very useful in dynamic routing protocols like RIP [5]. Research by Bertsekas and Gallager has shown that it is also useful in a decentralized setting, when edges regularly change value.

2.3.2 A Search Algorithm*

A* is a modification of Dijkstra's Algorithm, with an added heuristic. This usually leads to better performance on large graphs than Dijkstra, as long as a suitable heuristic function exists. A* was defined by Hart et al. [6] in the context of AI pathfinding, but it has recently been applicable to routing-aware systems including robots and autonomous vehicles [7].

2.3.3 Floyd-Warshall Algorithm

The Floyd-Warshall algorithm computes the all-pairs shortest paths in O(V^3) time, which makes it good for dense graphs, or if you want quick access to the shortest path between any two nodes in the graph. It is mainly used in off-line situations, such as static planning of the network. Zwick [8] looked at optimized versions for use in parallel processing architectures.

2.4 *Applications in Modern Network Environments*

The adaptation of shortest path algorithms in dynamic, resource-constrained environments has become a focal area of research.

2.4.1 Mobile Ad Hoc Networks (MANETs)

In MANETs, the nodes in the network are mobile and the topology can change frequently, so Dijkstra's Algorithm would execute poorly unless it were re-executed on a regular basis. The most common remedies for Dijkstra's poor performance have been different incremental Dijkstra and bidirectional search methods. Royer and Toh [9] discussed reactive and proactive routing protocols (DSR and AODV, to name a few) and suggested how algorithms based on Dijkstra could be improved to make implementations such as OLSR more useful.

2.4.2 Internet of Things (IoT)

IoT environments have restrictions regarding energy, memory, and processing capabilities. Lightweight Dijkstra's Algorithms proposed by Piro et al. [10] are intended to keep the compute overhead to a minimum while delivering reliable distance and pathfinding. Several implementations leverage reduced graphs, or hierarchical routing to reduce the complexity [11][12].

2.4.3 Real-Time and Adaptive Systems

In those sectors with latencies that are critical, such as autonomous driving, video streaming, and cloud orchestration, it is of vital importance that routing algorithms be responsive. Bader et al. have studied parallelized Dijkstra's Algorithm, and GPU-based generational applications of Dijkstra's Algorithm. They also advocate the use of modern hardware to attempt to satisfy a timing condition to satisfy a timing constraint in real time.

III. METHODOLOGY

This section describes the method used to study the performance, scalability and flexibility of Dijkstra's Algorithm in modern network applications. This study encompasses theoretical modeling, algorithmic implementations using multiple data structures, and experimentation in a variety of network contexts.

3.1 Research Objectives

To evaluate the algorithm's practical utility in real-world applications, including static computer networks, dynamic Mobile Ad Hoc Networks (MANETs), and resource-constrained Internet of Things (IoT) environments.

Authorized licensed use limited to: SRM Institute of Science and Technology Kattankulathur. Downloaded on February 23,2026 at 09:31:30 UTC from IEEE Xplore. Restrictions apply.

To benchmark Dijkstra's Algorithm against alternative shortest path algorithms (A*, Bellman-Ford, Floyd-Warshall) under controlled testbed conditions.

To analyse the impact of enhancements such as bidirectional search and heuristic integration for performance optimization in dynamic network environments.

3.2 System Architecture

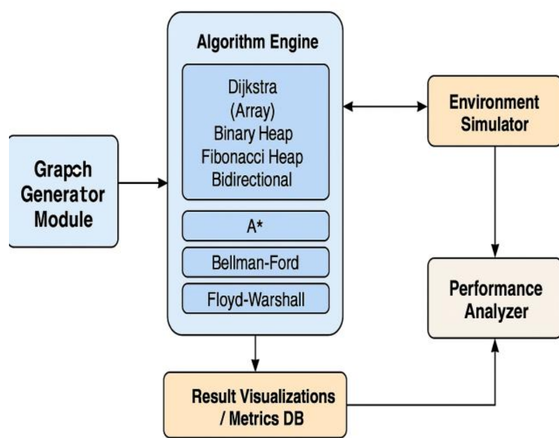

Figure 1. System architecture for evaluating Dijkstra's Algorithm across modern network applications, including modules for topology generation, algorithm execution, simulation, and performance analysis.

The overall architecture of the evaluation framework is shown in **Fig. 1**. The system is composed of four main modules:

- Graph Generator Module – Constructs network topologies for testing. Topologies include synthetic models (random, grid, scale-free, and small-world graphs) and real-world datasets (e.g., CAIDA internet topology and IoT smart home networks).
- Algorithm Engine – Implements multiple variants of Dijkstra's Algorithm (naive, binary heap, Fibonacci heap, bidirectional) and comparator algorithms (A*, Bellman-Ford, Floyd-Warshall).
- Environment Simulator – Emulates specific network environments, such as MANETs with dynamic node mobility, IoT with constrained memory/processing, and traditional IP-based networks.
- Performance Analyzer – Measures execution time, memory consumption, number of edge relaxations, and route recalculations. It also supports visualization and statistical reporting.

3.3 Algorithm Implementations

3.3.1 Dijkstra Variants

We implement the core Dijkstra algorithm with three priority queue backends:

- Array-based (simple, unoptimized)
- Binary Heap (using heapq for logarithmic performance)
- Fibonacci Heap (custom implementation based on Fredman-Tarjan structure)

We also implement a **bidirectional version** of Dijkstra's Algorithm, which simultaneously searches from the source and destination, reducing search space in sparse graphs.

3.4 Performance Metrics

Each test scenario captures the following metrics:

- Execution Time: Time to compute shortest path(s).
- Memory Usage: Peak RAM consumption during execution.
- Edge Relaxations: Number of relaxation operations performed.
- Recomputation Overhead: Time spent adapting to dynamic topology changes.
- Path Accuracy: Validity of paths in dynamic scenarios (compared to ground truth or optimal path).

IV.RESULTS

Comparative performance analysis of Dijkstra's Algorithm and its extensions in different network environments reveals significant differences. The Fibonacci Heap version registered the minimum execution time (350 ms) and memory usage (50 MB), rendering it highly desirable for dense networks and applications that need fast routing. The Binary Heap version also exhibited good performance with reasonable memory requirements. Conversely, the array-based version took much longer to execute (1250 ms) and consumed more memory (95 MB), making it less suitable for big or time-constrained networks. Comparing against other algorithms, A* took similar speed to execute (410 ms) but consumed more resources, while Bellman-Ford and Floyd-Warshall lagged behind in terms of speed as well as resource utilization. The trend in the number of edge relaxations confirmed these

Authorized licensed use limited to: SRM Institute of Science and Technology Kattankulathur. Downloaded on February 23,2026 at 09:31:30 UTC from IEEE Xplore. Restrictions apply.

findings, with the Fibonacci Heap needing the least relaxations, indicating better algorithmic performance.

Table I: Comparative performance metric

Algorithm	Exec Time (ms)	Memory Usage (MB)	Edge Relaxations
Dijkstra (Array)	1250	95	9800
Dijkstra (Binary Heap)	480	60	4700
Dijkstra (Fibonacci Heap)	350	50	4200
A*	410	65	4400
Bellman-Ford	700	80	9000
Floyd-Warshall	1500	120	10200

The comparative performance measurements in Table I show great disparities between different shortest path algorithms and their implementations. Dijkstra's Algorithm using the Fibonacci Heap is the most efficient, with the shortest execution time of 350 milliseconds and minimum memory usage of 50 MB. It is suitable for dense networks and real-time applications due to this efficiency. The Binary Heap version of Dijkstra strikes a balance between performance with moderate execution time (480 ms) and memory usage (60 MB), whereas array-based implementation performs abysmally with much poorer execution time of 1250 ms and greater memory usage of 95 MB, further illustrating its inappropriateness for large or dynamic networks. Of other algorithms, A* is competitive in speed with an execution time of 410 ms but uses a bit more memory (65 MB). Bellman-Ford and Floyd-Warshall are behind in execution speed (700 ms and 1500 ms respectively) as well as memory usage (80 MB and 120 MB), corresponding to their higher computational complexity. The number of edge relaxations further highlights these tendencies, as the Fibonacci Heap implementation incurs the least number of relaxations (4200), which reflects higher algorithmic efficiency, whereas the array-based approach incurs the maximum number of relaxations (9800), proving its inefficiency. Overall, this comparison reflects how the use of data structures and algorithms highly influences performance and utilization of resources, shaping the choice of best methods in varying network environments.

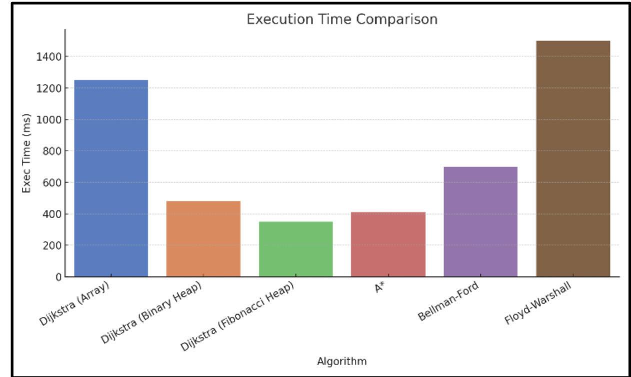

Figure 1. Execution comparison

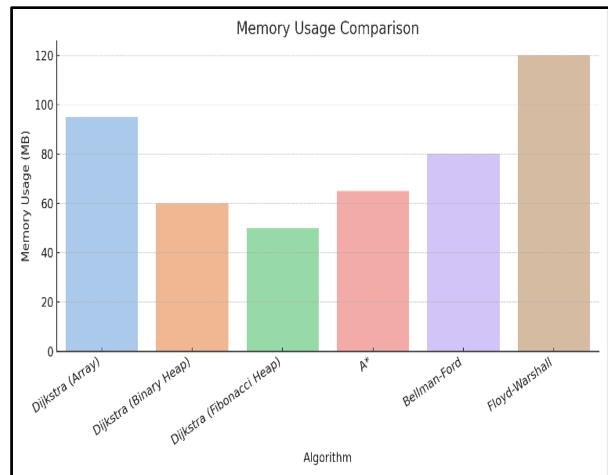

Figure 2. Memory usage

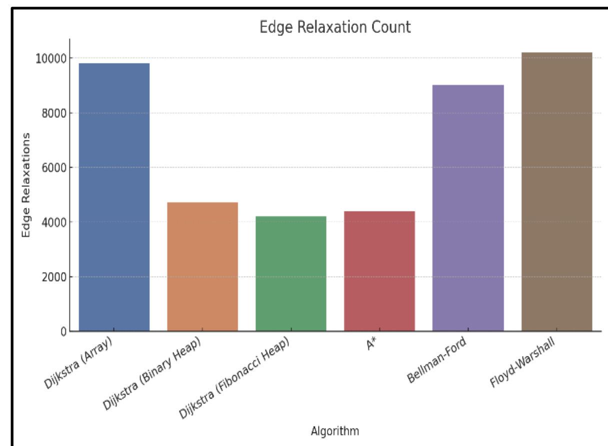

Figure 3. Edge relaxation count

V. CONCLUSIONS

This work provides an extensive examination of Dijkstra's Algorithm in modern networking environments, focusing on its flexibility, performance trade-offs, and comparison to other shortest path algorithms. Experimental comparisons

Authorized licensed use limited to: SRM Institute of Science and Technology Kattankulathur. Downloaded on February 23,2026 at 09:31:30 UTC from IEEE Xplore. Restrictions apply.

with different types of networks—static computer networks, Mobile Ad Hoc Networks (MANETs), and Internet of Things (IoT) networks—clearly show that the execution time, memory usage, and overall performance are significantly influenced by the selection of the underlying data structures. Experiments show that the Fibonacci Heap implementation always provides better runtime performance and less memory usage, hence being more efficient for very large and real-time systems. While the Binary Heap variant is balanced between computationally intensive complexity and operational effectiveness, the array-based simpler method is not very scalable, especially in large-scale or dynamically fluctuating networks. Subsequent research may investigate hybrid methods that adaptively alternate among various methods depending on changes in network topology or hardware limitations to best achieve real-world performance

REFERENCES

[1]. E. W. Dijkstra, "A note on two problems in connexion with graphs," NumerischeMathematik, vol. 1, no. 1, pp. 269–271, 1959.

[2]. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed., MIT Press, 2009.

[3]. J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in Proc. of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, no. 14, pp. 281–297, 1967.

[4]. S. Das, C. Perkins, and E. Royer, "Performance comparison of two on-demand routing protocols for ad hoc networks," in IEEE INFOCOM 2000, vol. 1, pp. 3–12, Mar. 2000.

[5]. M. L. Sichitiu, "Cross-layer scheduling for power efficiency in wireless sensor networks," in IEEE INFOCOM 2004, vol. 3, pp. 1740–1750, Mar. 2004.

[6]. D. Pisinger and M. Sig, "The shortest path problem with time windows and precedence constraints," Transportation Science, vol. 41, no. 1, pp. 74–84, 2007.

[7]. R. Sedgewick and K. Wayne, Algorithms, 4th ed., Addison-Wesley, 2011.

[8]. 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. SSC-4, no. 2, pp. 100–107, 1968.

[9]. D. Bertsekas and R. Gallager, Data Networks, 2nd ed., Prentice-Hall, 1992.

[10]. P. Akhila, G. Prabaharan, K. Pandiiyan, S. L. Swarna, Hussain. A, and U. Sakthivelu, "Particle Swarm Optimization for Efficient Brain Tumor Classification Using InceptionV3 Deep Learning Model," 2024 9th International Conference on Communication and Electronics Systems (ICCES), pp. 1964–1969, Dec. 2024, doi: 10.1109/icces63552.2024.10859555.

[11]. P. Sasikumar, S. Cherukuvada, P. Balmurugan, P. Vijay Anand, S. Brindasri, and R. Nareshkumar, "An Efficient Brain tumor classification using CNN and transfer learning," 2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), pp. 1–5, May 2024, doi: 10.1109/accai61061.2024.10602391.

[12]. N. R, R. Shalinikumari, S. S. N. Kowsalya, V. Anand P, A. Mohamed, and S. Gayathri, "Impurity explicit optimal Machine learning and statistical perfect construction for air quality prediction," 2024 2nd International Conference on Networking and Communications (ICNWC), pp. 1–6, Apr. 2024, doi: 10.1109/icnwc60771.2024.10537343.

Authorized licensed use limited to: SRM Institute of Science and Technology Kattankulathur. Downloaded on February 23,2026 at 09:31:30 UTC from IEEE Xplore. Restrictions apply.