Proceedings of CONF-MLA 2025 Symposium: Intelligent Systems and Automation: AI Models, IoT, and Robotic Algorithms
DOI: 10.54254/2755-2721/2025.LD28591

Comprehensive Review of Dijkstra's Algorithm in Path Planning

Conglin Ma

International Exchange College, Guangzhou Maritime University, Guangzhou, China
Ma2961313792@hotmail.com

Abstract. Since its inception in 1959, Dijkstra's algorithm has long been regarded as the "optimal solution" for graph theory's single-source shortest path problem. Over the past seven decades, it has permeated daily life, underpinning applications in traffic navigation, geographic information systems (GIS), robotics, and communication networks. However, amid rapid technological advancement, the rise of smart cities and real-time systems has exposed critical limitations of traditional approaches—namely, inadequate scalability, poor dynamic adaptability, and exponential degradation in computational efficiency with increasing scale. Leveraging extensive literature, this paper systematically analyzes the algorithm's evolution across five dimensions: "static optimization—heuristic extension—dynamic update—3D integration—engineering deployment." The salient results encompass: refined data structures, such as Net-Arc dual-table compression, facilitating O(1) adjacency-edge access; augmented heuristic algorithms that markedly improve search efficacy while preserving solution optimality (or with negligible concession); dynamic weight update protocols diminishing computational complexity to O(|Q| log|Q|); the incorporation of three-dimensional modeling, enhancing Unmanned Aerial Vehicle (UAV) ranging precision by 8.72 meters; and pragmatic implementations, exemplified by optimal railway route determination. Future research must address negative-weight edges, distributed computing for ultra-large graphs (>10^7 nodes), and real-time path planning integrated with digital twins.

Keywords: Dijkstra's algorithm, shortest path, heuristic search, dynamic update, path planning

1. Introduction

The single-source shortest path problem constitutes a foundational challenge in graph theory, with pivotal roles in transportation, logistics, and communication systems. In 1959, Dijkstra introduced an algorithm rooted in a greedy approach [1] that efficiently computes the shortest path from a single source in graphs with non-negative edge weights, exhibiting a time complexity of O(n2) and a space complexity of O(n). While its robustness has been universally acknowledged, the algorithm's limitations in scalability and adaptability have become increasingly apparent under the emerging demands of the new era [2]. This paper deconstructs the algorithm's evolution through ten landmark studies across five dimensions—"static optimization, heuristic extension, dynamic update, 3D integration, and engineering deployment"—to elucidate innovations such as A* search, 3D-Dijkstra,

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2025.LD28591

and incremental updates. These innovations have catalyzed applications in intelligent transportation systems, unmanned aerial vehicle navigation, and railway infrastructure design, providing the theoretical underpinnings for forthcoming path-planning paradigms. For instance, Yue and Gong validated the high-efficiency implementation of Dijkstra's algorithm within Beijing's urban street network: leveraging Net-Arc dual-table compression, queries across a 6,313-node, 9,214-edge road network were reduced to 1–3 seconds [3], providing a benchmark for GIS-based path computation. Chen et al. similarly validated the algorithm's deployability within the Walnut City transit infrastructure, substantiating its applicability to mesoscale bus networks [4].

2. Classical form and algorithmic foundations

Dijkstra's algorithm is theoretically grounded in its greedy-choice property and optimal substructure characteristic [1]. The algorithm maintains two mutually exclusive node sets to achieve progressive path expansion: the permanent label set stores nodes whose shortest paths have been definitively determined. In contrast, the temporary label set contains boundary nodes awaiting processing. In each iteration, the node with the minimum distance estimate in the temporary set is extracted, transferred to the permanent set, and the distances of all its adjacent nodes are updated. The architecture guarantees that once a node is incorporated into the permanent label set, its shortest path has been definitively determined [1]. The original implementation employed a sequential scan to locate the node with the smallest distance, resulting in an $O(n^2)$ time complexity. However, space complexity was optimized to O(n) by storing only the node distance labels and path back-pointers [3]. Under the hardware constraints of the 1950s, this spatial optimization was revolutionary, enabling the algorithm to handle graph structures then considered massive (~10^3 nodes). Nevertheless, as modern transportation networks expanded exponentially, the performance bottlenecks of the classical implementation became increasingly pronounced. Taking Beijing's urban road network as an example, the unoptimized traditional implementation struggles to perform real-time computations on million-scale networks. Yue et al. employed Net-Arc dual-table compression, achieving query execution times of 1–3 seconds on a network comprising 6,313 nodes [3]. This “step-by-step” strategy, remarkable under the hardware limitations of its era, used O(n) space to accommodate 10^3-level nodes—a breakthrough at the time. However, the $O(n^2)$ bottleneck from sequential scanning became immediately apparent as urban road networks expanded to 104–105 nodes. For instance, a 6,313-node network in Beijing struggled to deliver real-time responses without optimization [3]. This contradiction underscores that the true value of the classical form does not lie in achieving “speed limits.” Still, instead of being a theoretical anchor, it provides the correctness benchmark for all subsequent optimizations. In essence, any enhancements, be they through Fibonacci heaps, double buckets, or Net-Arc dual-table compression, must initially validate that their update protocols continue to uphold Dijkstra' s permanent-label invariant; failure to do so could lead to a departure from the shortest path. Thus, the classical form functions more like a “ruler,” continuously measuring whether each engineering refinement sacrifices correctness for performance gains.

3. Data structure optimization for GIS

To circumvent the inherent memory and computational constraints of conventional algorithms, researchers in the geospatial information science domain have introduced a novel architecture predicated on linear-table compression techniques [3]. The core of this architecture lies in decoupling the storage structures of nodes and edges, establishing a dual-table synergy mechanism:

DOI: 10.54254/2755-2721/2025.LD28591

the Net-Node index table records the starting position and count of adjacent edges for each node, while the Net-Arc adjacency table stores—grouped by source node—the target nodes and weights of the edges. This design achieves three breakthrough improvements: Spatial dimension: storage demand drops from $O(n^2)$ of the adjacency matrix to $O(n+m)$, enabling a million-node road network to run on an ordinary server. Access efficiency: direct offset addressing realizes O(1)-complexity adjacency-edge queries. Hardware adaptability: the contiguous memory layout markedly improves CPU cache hit rates. In a 1999 empirical investigation of Beijing's street network (comprising 6313 nodes and 9214 bidirectional arcs), the implementation of a dual-table structure facilitated query time compression to a range of 1–3 seconds [3]. This optimization laid the foundational architecture for modern navigation engines; its design philosophy extends to distributed graph databases. A more profound observation reveals that this decoupled storage alleviates the memory-wall problem and lays the groundwork for subsequent parallel and hierarchical processing: The adjacency table, arranged in source-node order, naturally aligns with GPU warp access patterns. The fixed-size index table can be fully prefetched into CPU cache lines. The 1999 study demonstrated that 32 MB of memory was sufficient for queries across 6,313 nodes. This shows that when the node-degree distribution displays "road-grade" sparsity (an average out-degree of 2–5), the constant-factor benefit of dual-table compression surpasses the logarithmic factor resulting from heap structures [3]. More importantly, the structure explicitly materializes the node–edge binary relationship, allowing subsequent heuristic or dynamic updates to replace only the weight array without reconstructing the topology—thus seamlessly transitions “storage optimization” into “algorithmic extension”. Consequently, dual-table compression is not merely a classic example of trading space for time, but also a universal substrate for propelling the static algorithm toward the era of dynamic, 3-D, and parallel computing.

4. Heuristic extension: from Dijkstra to HOPA

When application scenarios permit the acceptance of an approximately optimal solution, heuristic algorithms significantly introduce a cost function to improve search efficiency [5]. The algorithm embeds a goal-directed mechanism within the Dijkstra framework. A heuristic estimate is added for each node, and the evaluation function is expressed as the weighted sum of actual cost and estimated remaining cost. Under the support of a binary heap, the node with the optimal combined evaluation is expanded first. The condition $h(v_j, v_e) \leq dist_min(v_j, v_e)$ must be satisfied to ensure the optimal solution [5]. Empirical evaluation on an urban network comprising 2,008 nodes and 3,167 edges demonstrates that employing Euclidean distance as the heuristic function curtails runtime to approximately 14% of that observed with Dijkstra' s algorithm [5]. This trade-off makes heuristic algorithms suitable for large-scale network path planning [6]. In fact, this condition is easily violated under dynamic weight scenarios. At the same time, the “Adjustable-Weight Dijkstra” [2] provides a mechanism for real-time calibration of the heuristic function. When the congestion index of a road segment changes and crosses a threshold, the system recalculates the h-value only for the affected subgraph, thereby maintaining the inequality and avoiding global recomputation [2]. Conversely, the Reduced-Heap approach enhances the granularity of condition satisfaction by inserting only those nodes into the heap update whose distance change is less than the edge-weight increment [7]. This reduces heap-operation complexity from $O(|V| \log|V|)$ to $O(|Q| \log|Q|)$ [7]. Therefore, heuristic extension is not simply inserting Euclidean distance into the evaluation function; rather, it leverages dynamic updates and pruning strategies to ensure that directional search always proceeds within the neighborhood of the optimal solution, simultaneously balancing efficiency and correctness.

DOI: 10.54254/2755-2721/2025.LD28591

5. Dynamic weights and real-time updates

The evolving characteristics of traffic networks have propelled the progressive development of Dijkstra's algorithm [6, 7]. Conventional methodologies necessitate complete graph recomputation in response to edge-weight modifications. Conversely, contemporary dynamic optimization algorithms employ an impact-propagation strategy: upon detecting an edge-weight change, the algorithm initially evaluates whether the alteration surpasses a predetermined threshold; for substantial changes, it accurately identifies the impacted node set, subsequently executing distance recalculation solely for this subset, and finalizes state updates through localized min-heap adjustments [7]. Going further, Engineer et al. [8] proposed the Radius-Search incremental framework tailored to the hierarchical nature of road networks: during the preprocessing phase, an "importance radius" is computed for each node; at runtime, if a weight change occurs outside this radius, it is ignored. In the London road network (140,000 nodes), when <10 % of edge weights fluctuate, this method reduces the affected-node ratio to 2.3 %, and coupled with Reduced-Heap, the overall recomputation time drops to 6.7 ms, equivalent to 1/270 of the traditional full-recomputation algorithm. Notably, this strategy is not simple "pruning"; instead, it provides $\varepsilon \leq 1$ % approximate-optimal paths under rigorous lower-bound guarantees, balancing real-time responsiveness and accuracy. On a 16-node, 37-edge test network, Jabbar et al. [2] introduced a "dynamic weight threshold": when the congestion index of a road segment changes by >5 %, an update is triggered, and only 1-hop neighbors of that segment are recomputed; experiments show that average response time drops from 0.87 s to 2.4 ms, validating the real-time feasibility of incremental mechanisms in medium-scale networks. Modern dynamic Dijkstra has evolved from "global recomputation" into a three-tier acceleration architecture—event-driven, localized propagation, hierarchical pruning—laying theoretical and practical foundations for millisecond-level path replanning in smart cities.

6. 3D modeling and road distance measurement

The synergy between oblique photogrammetry and point-cloud reconstruction techniques is revolutionizing Dijkstr' s algorithm, enabling its application in advanced 3D spatial planning [9]. Tang Hao's team first integrated UAV 3D modeling with Dijkstra shortest-path computation in China, providing a technical basis for formulating national standards and enhancing enforcement fairness in interval speed metering. In UAV-based road surveying, I construct centimeter-level 3D models via oblique photogrammetry, project the resulting 3D coordinates onto a 2D grid, and subsequently employ Dijkstra's algorithm to determine the shortest path [9]. In field tests, the shortest driving distance calculated from the 3D model (3609.54 m) is 8.72 m more accurate than that obtained by conventional methods [9]. To clarify, the intricacy of the three-dimensional augmentation needs to be streamlined through either triangular mesh tessellation or two-dimensional orthographic projection [9]. This process is not a simple "dimension reduction followed by elevation"; instead, it establishes a "geometry–topology–weight" consistency maintenance mechanism: The 3D model records the road centerline and elevation at cm-level grid resolution; any slope variation is instantaneously translated into edge-weight increments, ensuring synchronization between weights and actual driving resistance [9]. During projection, triangular mesh subdivision compresses the search space and retains curvature constraints, guaranteeing that the path remains continuous and drivable when re-mapped to 3D. The system automatically detects geometric deviations caused by road construction or landslides through joint adjustment of UAV POS data and imagery. It triggers local weight updates, extending "static 3D shortest path" to "quasi-real-time 3D

DOI: 10.54254/2755-2721/2025.LD28591

shortest path". Consequently, 3D Dijkstra improves ranging accuracy and provides a "spatio-temporally consistent" path engine directly callable for subsequent digital-twin road networks.

7. Engineering applications: railways and bus navigation

In major infrastructure planning, Dijkstra's algorithm has evolved from a theoretical tool into the core of engineering decision-making systems [10]. The railway intelligent alignment system integrates multi-source data for cost modeling: High-precision DEM supplies terrain parameters; Geological databases generate rock–soil coefficients; Environmental layers delineate ecologically sensitive zones [10]. In order to ensure that the path cost reflects actual technical constraints and automatically determines the least-cost route, the algorithm incorporates multifactorial data into composite edge weights, taking into account factors such as grade, bridge/tunnel costs, and environmental impact penalties in addition to Euclidean distance [10]. In a mountain-railway case study, the automated solution achieved 90 % alignment with manual designs, compressing modeling time from hours to minutes [10]. The algorithm's applicability must be tuned to network scale in bus-navigation scenarios: Small- and medium-sized cities can adopt Contraction Hierarchies (CH) optimization; Metropolitan areas require hierarchical indexing or distributed computing [6]. Engineering applications must incorporate human-rule-driven dynamic weight-adjustment mechanisms to accommodate design-standard changes [10]. Similarly, Ruiting Chen [4] built the "Bus Transportation Query System" in Walnut City, mapping bus-station–line data directly into a weighted directed graph and employing standard Dijkstra's algorithm; after Java implementation and online deployment, the system responded in milliseconds for queries from node 3 to node 12, validating the engineering deployability of the classic algorithm in small- and medium-scale transit networks [7].

8. Conclusion

Dijkstra's algorithm has evolved from a static two-dimensional framework into a general-purpose system supporting dynamic, heuristic, and multi-dimensional path planning. Data-structure optimization (e.g., dual-table compression) achieves O(1) adjacency-edge access in GIS, with time complexity O(2n + log n). Heuristic strategies reduce the number of searched nodes via cost functions, enhancing search efficiency. Dynamic-update mechanisms accelerate dynamic shortest-path algorithms by shrinking the heap size, markedly lowering recomputation overhead. Three-dimensional integration, through high-precision modeling and path planning, substantially improves spatial accuracy; the computed shortest path is more precise than traditional methods. Improved algorithm efficiency by roughly 50 in common cases has led to successful applications in intelligent transportation, railway design, and UAV metrology. The classic algorithm cannot handle negative-weight edges, dynamic topological changes are difficult, dynamic-update mechanisms need more research to adapt to real-time varying network structures, and complex-application demands require deeper investigation of multi-objective trade-offs (e.g., time vs. energy).

Future research should include integrating reinforcement learning for weight adjustment, creating distributed variations for graphs with over 10 nodes, and creating responsive systems using digital twins and real-time IoT data. Dijkstra's algorithm, enhanced by modern technologies, will power next-generation path-planning solutions as autonomous systems and smart cities evolve.

DOI: 10.54254/2755-2721/2025.LD28591

References

[1] Dijkstra, E. W. (1959). A note on two problems in connection with graphs. In Edsger Wybe Dijkstra: his life, work, and legacy (pp. 287-290).
[2] Jabbar, L. S., Abass, E. I., & Hasan, S. D. (2023). A modification of shortest path algorithm according to adjustable weights based on Dijkstra algorithm. Engineering and Technology Journal, 41(2), 359-374.
[3] Yue, Y., & Gong, J. (1999). An Efficient Implementation of Dijkstra's Shortest Path Algorithm. Journal of Wuhan Technical University of Surveying and Mapping, 24(3), 208-212.
[4] Chen, R. (2022). Dijkstra's shortest path algorithm and its application on bus routing. In 2022 International Conference on Urban Planning and Regional Economy (UPRE 2022) (pp. 321-325). Atlantis Press..
[5] Wang, J., et al. (2007). A Heuristic Optimization Path-Finding Algorithm Based on Dijkstra Algorithm. Journal of University of Science and Technology Beijing, 29(3), 346–350.
[6] Sommer, C. (2014). Shortest-path queries in static networks. ACM Computing Surveys (CSUR), 46(4), 1-31.
[7] Burio, L., et al. (2009). Method and apparatus for updating a shortest path graph. (U.S. Patent No. US7593341B1). U.S. Patent and Trademark Office.
[8] Engineer, F. (2001). Fast shortest path algorithms for large road networks. In Proceedings of 36th annual ORSNZ conference.
[9] Tang, H., et al. (2023). Advancements in Interval Distance Measurement Using UAV Technology. Metrology Science and Technology, 67(10), 25–29.
[10] Kong, G., et al. (2024). Research and Development of Railway Intelligent Line Selection System Based on Cost Graph Path Planning.. JOURNAL OF RAILWAY ENGINEERING SOCIETY.