Comparative Performance Analysis of Some Priority Queue Variants in Dijkstra's Algorithm

Idowu, Abel Iyanda[1] Olabiyisi, Stephen Olatunde.[2] Alo, Oluwaseun Olubisi [3] Adeleke, Israel Adewale[4] Jokotoye, Ayoade Alade[5] and **Omotade, Adedotun Lawrence[6]**

[1,2,3,6] Department of Computer Science, Ladoke Akintola University of Technology, Ogbomoso, Oyo State, Nigeria.

[4] Department of Data Science, Informatics and Computer Science, Emmanuel Alayande University of Education, Oyo. Oyo State, Nigeria

[5] Department of Computer Science, Bowen University, Iwo. Osun State, Nigeria.

DOI: https://doi.org/10.51244/IJRSI.2025.120800078

Received: 06 Aug 2025; Accepted: 13 Aug 2025; Published: 06 September 2025

ABSTRACT

Efficient shortest-path computation in weighted graphs is essential in domains like networking and logistics. Dijkstra's algorithm depends heavily on the choice of priority queue, and while theoretical complexities are well-documented, their real-world performance varies. This study compares three priority queue implementations-Binary Heap**,** Fibonacci Heap**,** and Binomial Heap- within Dijkstra's algorithm using road network data from Zenodo (https://doi.org/10.5281/zenodo.1290209). The dataset was preprocessed, normalized, and converted into a usable format using MATLAB (R2024b). Theoretical time complexities for core operations—*insert*, *decrease-key*, and *extract-min*—were analyzed. Experiments conducted on synthetically generated graphs showed Binary Heap achieved the fastest execution time (0.00126s) and highest throughput (3313 edges/sec), outperforming Fibonacci and Binomial Heaps. Results indicate that Binary Heap is the optimal choice for execution speed and throughput, especially for large or dense graphs. The findings provide practical guidance for selecting priority queues in real-world shortest-path applications and contribute to the empirical evaluation of data structures in algorithm design.

Keywords: Shortest path, Dijkstra algorithm, Priority queue, Binary heap, Binomial Heap and Fibonacci heap.

INTRODUCTION

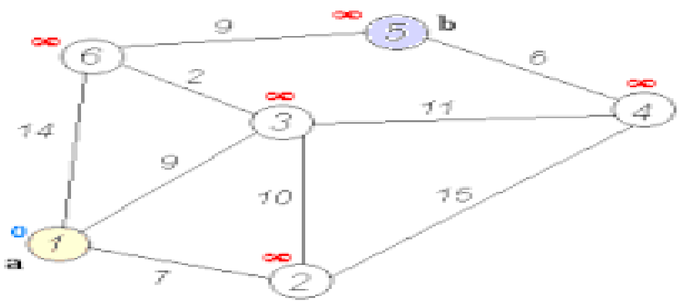
The aspect of algorithms is an integral part of the theoretical computer science that has been in existence since the early days of the information age. It gave birth to many brilliant ideas used in solving fundamental computational problems. Therefore, algorithms are the solution to computational problems. They define methods that are used in solving problems that require the formulation of an algorithm for the solution.

Dijkstra's algorithm (DA) is an efficient, exact method for finding the shortest paths between vertices in edge- and arc-weighted graphs. The Shortest Path Problem (SPP) approach involves finding the shortest path between two vertices (or nodes) in a graph such as sum of weight of its constituent edges when minimized. The shortest path problem is represented by a graph, G = (V - E), where V is the set of vertices or nodes and E is the set of edges or arcs. Sometimes, the graph is known as a network. It is particularly useful in transportation problems when we want to determine the shortest (or fastest) route between two geographic locations on a road network (Rillet, 2006). It is also applicable in areas such as telecommunication, social network analysis, arbitrage, and currency exchange (Lewis, 2023).

The algorithm exists in many variants such as Fibonacci heap, List, Indexed priority queue, Binary heap, D-way heap, Binomial heap and Array.

Priority queue variants in Dijkstra's algorithms

Priority queue variants in Dijkstra's algorithm is known as Heaps. The heap is one of most efficient implementation of an abstract data type called a priority queue. A priority queue is a fundamental data structure that allows efficient management of elements based on their priority. Priority queue variants in Dijkstra's algorithm used in this project include: Binary heap. Binomial heap and Fibonacci heap.



Binary heap: A binary heap is a complete binary tree which is used to stored data efficiently either to get the Max or Min element based on its structure. A binary heap is either Min heap or Max heap. For example, in Min. Binary heap, the key at the root must be minimum among all keys present in Binary heap. The same property must be recursively true for all nodes in binary tree. Max. Binary heap is like Min. heap. (Srivastava, 2020).

Binomial heap(BH): is an extension of binary heap that provides faster union of merge operations provided by binary heap. Binomial heap is a collection of binomial trees**.** A binomial tree of order has one node. A binomial tree of order K can be constructed by taking two binomial trees of order K-1 and making one the leftmost child of the other.

Fibonacci Heap: To speed up Dijkstra Algorithm for the single-source shortest path problem with nonnegative length edges, Freedman, and Tarjan in 1987 developed the Fibonacci heap.

The Fibonacci heap (FH) has similar properties to the binomial heap, such that every binomial heap is a Fibonacci heap, but not the other way around. The Fibonacci heap can contain multiple trees consisting of single nodes, while the binomial heap always consists of trees with 2n nodes, where each tree has a different n value.

Algorithm Techniques

Dijkstra algorithm: to find the shortest path between a. and b. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors shown in Figure1.1 Usually used with priority queue or heap for optimization.

The technique of finding shortest path (Fredman, 1987).

In the Pursuit to contribute to knowledge, this study aims to evaluate and compare three priority queue variant to classify them base on their predefined categories with the objectives to:

i) implement Dijkstra's algorithm with each priority queue variant (Binary Heap, Fibonacci Heap, and Binomial Heap);

ii) design experiments to measure runtime performance of each priority queue variant on different graph sizes;

iii) evaluate and compare the performance (execution time and throughput) of each variant under varying graph sizes.

a

RELATED WORK

Santoso (2010) analyzed the performance of Dijkstra, using A* and Ant algorithm for finding optimal path, A* and Ant algorithm when finding optimal path of certain route. The result shows that Ant algorithm is not suitable for the path finding algorithm because it is less stable and requires a long time to do a search while Dijkstra and A* algorithm provide optimal results in a quick time.

Buyue (2016) presented the Performance Evaluation (PE) of A* Algorithms, the used of A* algorithms, which over the years has had a lot of variations, the thesis find how Dijkstra algorithm, IDA*, Theta* and HPA* compare against A* based on the variable's computation time, number of opened nodes, path length as well as number of path nodes. A* and way they specifically improve A* an experiment was set up where the algorithms were implemented and evaluated over several maps with varying attributes of the result of the tested algorithms for computation time, number of opened nodes, path length and number of path nodes over several different maps as well as the average performance overall maps.

Enech and Arinze (2017) Carried out Comparative Analysis and Implementation of Dijkstra shortest path algorithm for emergency response and logistic planning Dijkstra algorithm implemented with double bucket dynamic data structure is selected for implementing the proposed route planning system as past research efforts has proven that it is the fastest with run-time improvements from 0(m+n/logc) to 0(m) respectively.

WANG (2018) compared three Algorithms in shortest path issue, aiming to find the shortest path between the two nodes in a network; the paper introduces three basic algorithms and compares them theoretically by analyzing the time and space complexity. Then the paper discusses their performance and summarizes the best application range.

Samah. (2020) Conducted Comparative Analysis between Dijkstra and Bellman Ford Algorithm in shortest paths optimization. In that article Dijkstra algorithm and Bellman Ford Algorithm are used to make a comparison between them based on complexity and performance in terms of shortest path optimization. The result show that Dijkstra algorithm is better than the Bellman inter of execution time and more efficient for solving shortest path issue but the algorithm of Dijkstra work with non-negative edge weight.

Yen and Bhandari (2023) proposed Algorithms for variants of the single-source single-target shortest path problem; Yen's algorithm is used to find the k shortest s-t-paths, where k is a user-defined parameter. The methods of Bhandari are used to produce a pair of shortest s-t-paths that are either edge-disjoint and/or vertex-disjoint.

Despite significant advancement in performance evaluation of Dijkstra algorithm, several research gaps persist. The subjective nature of Dijkstra definition across different priority queue variants and operations poses challenges leading to inconsistencies in selecting an appropriate priority queue for specific real-world applications of Dijkstra's algorithm. Furthermore, variability in dataset quality and techniques introduces noise and biases that complicate running time of an algorithm changes. Many existing studies rely on limited and road network graph datasets, which do not adequately represent the full spectrum of Dijkstra and finds an optimal path but becomes inefficient for large-scale problems. However, utilizing road network graph datasets as a test bed and implemented in a MATLAB (R2024b) environment by selecting an appropriate priority queue such as; Fibonacci heap, Binary heap and Binomial heap for specific real-word applications of Dijkstra algorithm by bridging the gap between theoretical efficiency and practical performance, the study aim to revealed the most efficient variant of Dijkstra's algorithm to be adopted when solving optimization problem.

METHODOLOGY

In the process of comparing of some priority queue variants in Dijkstra algorithms and evaluate their performance, the study involved. Data collection, Data pre-processing, Priority queue implementation, Development of platform and Evaluation of priority queue variants

a

i). Data Collection:The road network graph dataset utilized in this research was obtained from *Zenodo* *https://doi.org/10.5281/zenodo.1290209* (dataset). It consists the Czech Republic (CZE) and Entire Portuguese (PT) road network graph with total nodes (intersections): 1,285,250 (CZE: 1,043,139 + PT: 242,111) and total edges (roads): 3,254,122 (CZE: 2,673,011 + PT: 581,111). 345 MB. While the PT consists of road networks from Lisbon, Porto, and entire Portuguese road network. Lisbon Nodes (intersections): 22,107 Edges (roads): 53,811 2.5 MB. Porto Nodes (intersections): 13,444, Edges (roads): 32,313, 1.5 MB. Portuguese Road Network: Nodes (intersections): 242,111 Edges (roads): 581,111, 25 MB data generated was first subjected to data cleaning to remove irrelevant, noisy or incomplete records and ensured to contain numeric value and non-negative that could be negatively impact the algorithm's performance. The data was also mapped to represent the edges for Dijkstra algorithm particularly in cases where road network are weighted by the segments in meters, length of the road making them suitable for shortest path calculation. Finally, the cleaned and formatted data was converted into data frames making it easier to manipulate, analyzed and feed into the Dijkstra algorithms.

ii.) Data Pre-processing**:** Data pre-processing was a critical step in preparing for priority queue, ensuring it was structured and ready for algorithm and testing. Data was converting the dataset into a data frame was a pivotal step in the data pre-processing pipeline, enabling efficient organization and manipulation of the data. A data frame is a data structure constructed with row and columns, similar to a database or Excel Spreadsheet, ideal for handling large and multidimensional datasets. Table 1 Summarizes some samples edges in the preprocessed Zenodo dataset. Figure 1 and Figure 2 display the preprocessed dataset in visualization form.

Table 1: Dataset composition table

Source	Target	Weight
616	111	55
523	80	68
93	246	44
912	888	1
282	932	63
667	153	65
300	294	6
326	282	20
376	51	51
394	610	63
915	74	94
379	907	80
959	521	17
601	440	64
122	464	21

a

946	776	61
438	494	9
742	403	62
783	994	67

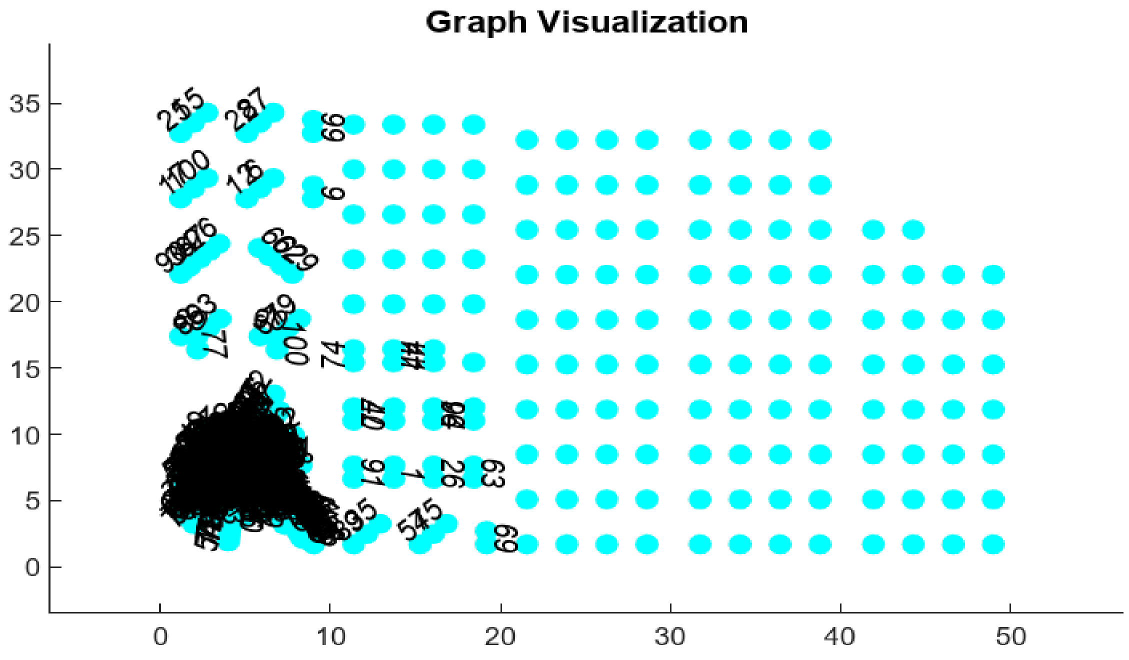


Figure 1 Dataset Visualization without zoom

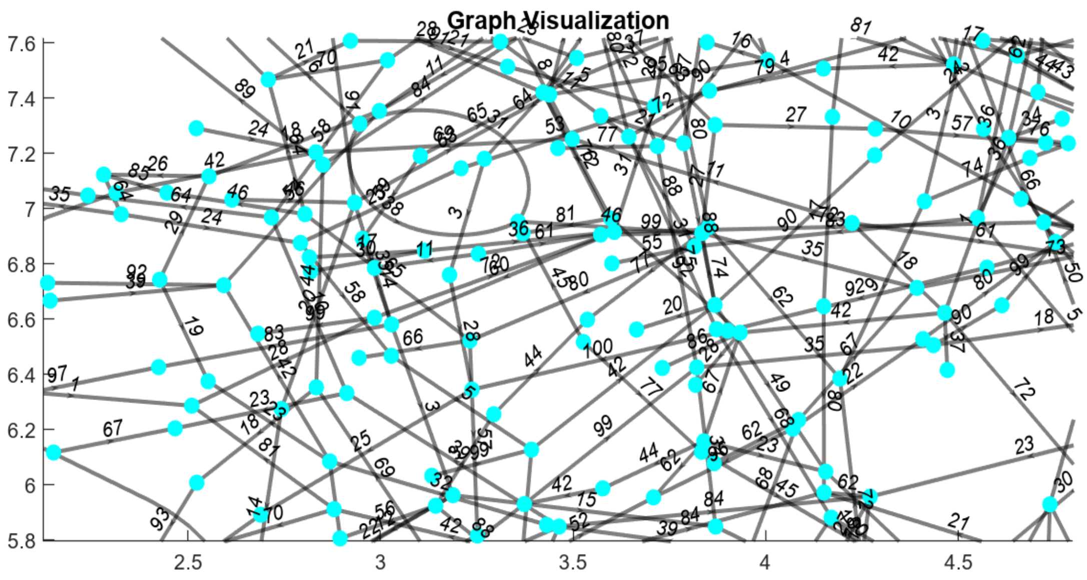


Figure 2: Dataset Visualization after zooming

To adopt the dataset for this study, several preprocessing steps were performed:

Column Selection: Only the id_1, id_2, and dist columns were retained and transformed to source and destination column respectively. These columns were mapped to represent the Source, Target, and Weight of the edges for Dijkstra's algorithm as illustrated in Figure 1 above.

Data Cleaning The data was checked for inconsistencies and ensured to contain only numeric values.

Sub-setting: A specific subset of the data (e.g., Lisbon or Porto) was selected to maintain computational feasibility while ensuring a realistic representation of road network complexity.

Validation: The graph's structure was verified as directed and weighted, which aligns with the requirements of Dijkstra's algorithm. The preprocessing and adaptation steps for this research were performed independently to align with the study objectives.

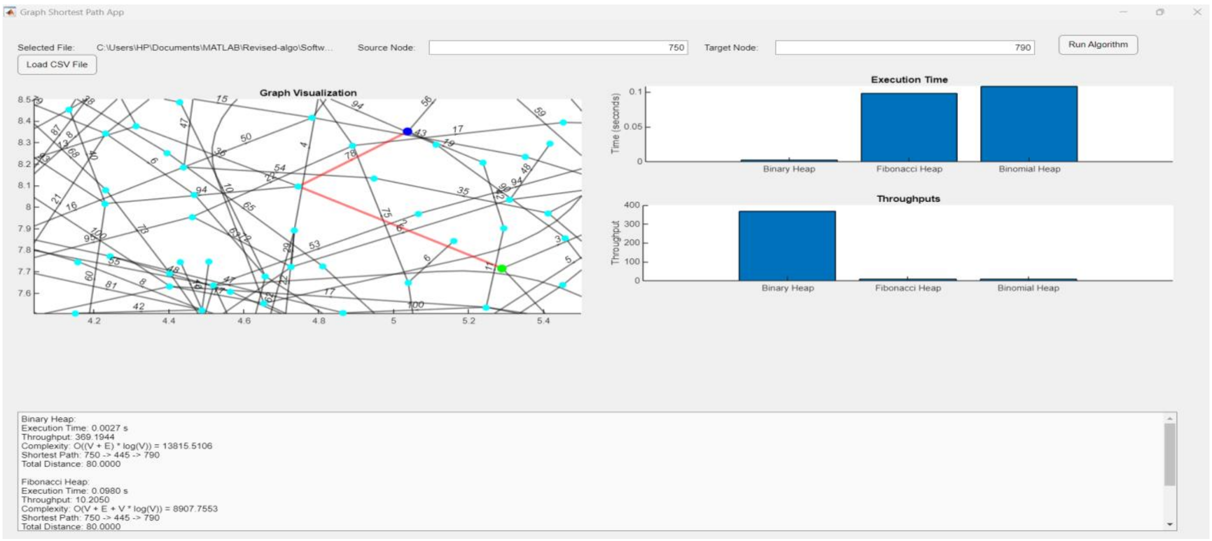
iii**).** Priority queue implementation: Implementing Dijkstra's algorithm with each priority queue variant and evaluating performance of Dijkstra algorithm involved a structured approach that integrated data pre-processing, algorithms implementation, development of a platform and evaluation. The process began with dataset preparation, which include data selection, data cleaning, normalization and validation the graph structure was verified as directed and weighted which aligns with the requirement of Dijkstra algorithm.The three priority queue variants of Dijkstra algorithms implemented using MATLAB programming Language (MATLAB R2024b). The experiment performed on the intel® core™ i5-8365u with windows 11 pro version 23Hz, 64 bits operating system, CPU@ 1.60GHZ 1.90GHz Central Processing Unit (CPU), 16.0GB Random Access Memory (RAM) and 500GB hard disk drive. During the implementation different Dijkstra algorithms applied using the same population size on the Zenodo dataset.

iv.) Development of a platform: A platform for experiments to measure execution time and throughput on synthetically generated graphs of varying sizes was developed using MATLAB (R2024b) programming Language. When the file was executed, the Graphical User Interface (GUI) window showed up as the file name. The quantity of data related to shortest path that need to be determined were input as source and target nodes. The Interface was made explicit with the help of some created button such as Execution time, Throughput, Graph visualization, User CSV file, Source node, target node, Result, and Run Algorithm and select Dataset buttons to input data for source node and target node, the algorithm to be used and table to display results and summary result.

v.) Evaluation of Priority queue variants: Comparative analysis by performance metrics of the obtained experimental result carried out for the purpose of evaluating the performances of each metric to the success of the selected priority queue variants in Dijkstra algorithm and validation of most important metric.

RESULTS AND DISCUSSIONS

A comparative result with three priority queue variants of Dijkstra among binary heap, Fibonacci heap and binomial heap using Portuguese Road network graphs for between centrality algorithm dataset were analyzed and reported. The metrics used to compare the algorithms are execution time and throughput showed in figure 3 below.



Selected File: C:\Users\HP\Documents\MATLAB\Revised-algo\Softw...
Source Node: 750
Target Node: 790

```
Binary Heap:
Execution Time: 0.0027 s
Throughput: 369.1944
Complexity: O((V + E) * log(V)) = 13815.5106
Shortest Path: 750 -> 445 -> 790
Total Distance: 80.0000

Fibonacci Heap:
Execution Time: 0.0980 s
Throughput: 10.2050
Complexity: O(V + E + V * log(V)) = 8907.7553
Shortest Path: 750 -> 445 -> 790
Total Distance: 80.0000
```

Figure 3 Application GUI and result after running for Source Node: 750 and Target Node 790

a

In Table 1, it was shown that, Fibonacci heap has the longest execution time (0.0459) with lower throughput (21.7709), Binomial heap has longer execution time (0,0243) and throughput (41.1909), while Binary heap has shortest execution time (0.0006) with higher throughput (1812.2508).

Similarly, in Table 2, Fibonacci heap has the longest execution time (0.0698) with lower throughput (15.06216), Binomial heap has longer execution time (0,05968) and throughput (19.92956) while Binary heap has shortest execution time (0.00126) with higher throughput (3313.74744), It was observed that Binary heap is one of most efficient priority queue variants of Dijkstra algorithm when using execution time and throughput.

This implies that, Fibonacci heap, Binomial heap will have consumed large amount of execution time of a system with lower throughput, while Binary heap consumed lesser amount of execution time more efficient due to its shortest amount of execution time with the highest throughput.

Table 1: Shows the summary result for the running (base on source node 750, target node 790).

Heap Type	Execution Time (s)	Throughput (edges/s)	Shortest Path	Total Distance
Binary Heap	0.0006	1812.2508	750 -> 445 -> 790	80.0000
Fibonacci Heap	0.0459	21.7709	750 -> 445 -> 790	80.0000
Binomial Heap	0.0243	41.1909	750 -> 445 -> 790	80.0000

Table 2: Shows the summary result for the running up to five (5) different batches (source node – target node) and compute the average performance to determine our final observation

Heap Type	**Execution Time (s)**	**Throughput (edges/s)**	**Shortest Path**	**Average of total Distance**
Binary Heap	0.00126	3313.74744	750 -> 445 -> 790, 804 -> 105 -> 312, 722 -> 624 -> 238 -> 724 -> 56 -> 312,648 -> 5 -> 820	89.2000
Fibonacci Heap	0.0698	15.06216	750 -> 445 -> 790, 804 -> 105 -> 312, 722 -> 624 -> 238 -> 724 -> 56 -> 312, 648 -> 5 -> 820	89.2000
Binomial Heap	0.05968	19.92956	750 -> 445 -> 790, 804 -> 105 -> 312, 722 -> 624 -> 238 -> 724 -> 56 -> 312, 724 -> 56, 648 ->	89.2000

Table.3: Comparison Summary

a

Heap Type	Execution Time (s)	Throughput
Binary Heap	Fast	Very high
Fibonacci Heap	Slow	Low
Binomial Heap	Medium	High

CONCLUSION

This work has described the shortest path problem and shown how the performance of Dijkstra’s algorithm can be affected by the choice of data structure used for its priority queue. This research presented a comparative performance analysis of some priority queue variants in Dijkstra algorithm among Binary heap, Fibonacci heap, and Binomial heap. The algorithms were implemented with help of Matrix Laboratory (MATLAB R2024b) software tested over a large range of problem instances. Implementation operates by loading the entire graph with labeled vertices in to RAM before execution algorithms and its performance evaluation in comparison with the algorithms used based on some parameters, the dataset was processed without any alteration.

Based on experimentation, the results provide insights into the strengths and weaknesses of each priority queue. Binary Heap demonstrates practical efficiency for sparse graphs and moderate workloads due to its simplicity. Fibonacci Heap, while offering the best theoretical performance for dense graphs with frequent updates, presents higher implementation complexity and overhead. Binomial Heap, known for efficient merging operations, proves beneficial in specialized scenarios but generally underperforms compared to the other two. This research concludes with recommendations for selecting an appropriate priority queue for specific real-world applications of Dijkstra's algorithm. By bridging the gap between theoretical efficiency and practical performance, the findings aim to guide practitioners and researchers in optimizing shortest-path computations.

REFERENCES

1. Abhinau Jauhri (2013): Binomial and Fibonacci heaps in rocket (rkt heaps), abhinaujauhri@gmail.com.
2. Akhilesh Kumar Srivastava (2020}: A Practical approach to Data Structure and Algorithm with Programming C: https://searchwork.stanford.edu.
3. Buyue, W. (2016). Performance Evaluation of Path finding Algorithms. University Thesis.
4. David Epstein (1998). Finding the K shortest path. SIAM Journal on computing – Vol. 28 iss.2 (1998). https://doi.org/10.1137/5009753 9795.290 477.
5. Daniel Dominic Sleator and Robert Endrre Tanjan (1986); Self balancing heaps. SIAM J. compt. 15(1); 52-69, Feb. 1986.
6. Donald B. Johnson (1977). Efficient algorithm for shortest paths in sparse networks, J. ACM, 24 (1); 1-13
7. Edsger W. Dijkstra: A note on two problems in connection with graphs Numerical mathematics 1:269-271.1959.14,72.
8. Elwira Johansson (2020); Practical complexity of the Fibonacci heaps in simulation and modeling framework. Department of Information Technology IT 20084 Uppsata University, post address Box 536, 75121 Uppsala. https://www.teknat.uu.se/ student.
9. Fredman, Michael Lawrence; Tarjan, Robert E. (1987). "Fibonacci heaps and their uses in improved network optimization algorithms". Journal of the Association for Computing Machinery. 34 (3): 596–615. doi:10.1145/28869.28874. S2CID 7904683.
10. Gerth Stolting et--al (2012). Strict Fibonacci heaps; In processing of the 44th symposium on theory of computing STOC 12, pages 1177 – 1184, New York NY USA, ACM. 23
11. James B. Orlin (2010). A faster algorithm for the single source shortest path problem with few distinct positive lengths J. of Discrete Algorithms, 8(2): 189- 198

12. James R, Driscoll et-al (1988); Relaxed heaps; an alternative for Fibonacci heaps with applications to parallel computation commu. ACM 31(11); 1343 1354, Nov. 1988.23; MIT Open Access: https://dx.doi.org/10.1016/j.jda.2009.03.001.
13. Jiri Vyskocil, Radek Marik (2013) Advance algorithms; binary heap, d-ary heap, binomial heap, Amortized analysis, Fibonacci heap
14. Kahneman, D. (2011). Thinking, Fast and Slow. Farrar, Straus and Giroux.
15. Ritesh Bhat et-al (2023); Comparative analysis of Bellman-Ford ana Dijkstra's algorithms for optical evaluation route planning in multi-floor buildings. Article; https://doi.org/10.1080/23311916.2024.2319394
16. Martell, E., & Sandberg, J. (2016). Evaluation of A* and its variants in path finding. Computational Intelligence Journal, 14(6), 301–320.
17. Mikkel Thorup (2003). Integer priority queues with decrease key in constant time and the sole source shortest paths problem. In preceding of the thirty-fifth annual ACM symposium on theory of complexity. STOC '03 pages 149 158 New York, NY, USA
18. Natick M. (2024): Matrix Laboratory (R2024b) programming language. https://www.mathworks,com.
19. Rhyd Lewis (2023): A Comparison of Dijkstra's Algorithm using Fibonacci Heap, Binary Heap, and self-balancing binary trees. Using C++ Implementations of these algorithms Variants School of mathematics, Cardiff University, Cardiff, Wales. Lewis R9 @cf/ac.uk http://www:rhyd.lewis.eu or https://doi.org/10.48550/arX.v.2303.10034
20. Rillet, S. (2006). Shortest paths in edge-weighted graphs. Journal of Applied Mathematics, 45(3), 254–267.
21. Samah, H. (2020). Comparative analysis of Dijkstra and Bellman Ford algorithms in shortest path optimization. International Journal of Computational Theory and Engineering, 12(4), 100–108.
22. Santoso, P. (2010). Comparative performance of Dijkstra, A*, and ant algorithms. International Journal of Computer Science and Applications, 12(2), 125–134.
23. Siddhartha Sen Bernhard Haeuppler and Robert E. Tarjan. On Rank- pairing Heap. SIAM J. Compute. 40 96); 146314485, 2011.23
24. Sneha Sawlani (2017) "Explaining the performance of Bi-directional Dijkstra and A* on a road networks. Electronic theses and Dissertation.1303. University of Denver. https://digitalcommons.du.edu/etd/1303.
25. Sunita Deepak Garg (2018) Dynamiting Dijkstra: A solution to Dynamic shortest path problem through Retroactive Priority Queue; Journal of King Saud University; Computer and Information Science 33(4)
26. Zenodo. (2018). Road network graphs for betweenness centrality algorithm. [Dataset]. Zenodo. https://doi.org/10.5281/zenodo.1290209
27. Zhan, F. Benjamin; Noon, Charles E. (February 1998). "Shortest Path Algorithms: An Evaluation Using Real Road Networks". Transportation Science. **32** (1): 65–73. doi:10.1287/trsc.32.1.65. S2CID 14986297