International Journal of Research in Engineering and Innovation (IJREI)

Journal home page: https://ijrei.com/

RESEARCH PAPER

Experimental comparison of shortest path algorithms for the metro train

Vivek Tyagi Vishal Kholi

Department of Computer Science and Engineering Neelkanth Institute of Technology, Meerut, India

Article Information

Received: 21 March 2026
Revised: 27 April 2026
Accepted: 03 May 2026
Available online: 05 May 2026

Keywords:

Single Source Shortest Path
Path finding Algorithms
Graph Traversal
Priority Queue
Dijkstra's Algorithm
Bellman-Ford Algorithm
Floyd-Warshall Algorithm

Abstract

Metro train networks are becoming an important artery of transport in the modern urban environment, guaranteeing efficient movement to millions of commuters in the urban environment. A major factor that has led to network efficiency is the identification of the shortest possible path that the trains will travel on to ensure that the travel time is minimized and that the resource utilized are maximized. In this paper, the author will examine how image processing techniques can be incorporated into managing a metro train network to help streamline the process of identifying the shortest route. Combined with sophisticated algorithms, image processing provides a new method of processing real-time information and optimizing train routes in real time. This approach allows the system to accommodate dynamic changes in passenger flow, track conditions and emergencies by utilizing image data of different sources including CCTV cameras, onboard sensors and satellite images. The main functions of the proposed system are image segmentation to identify important features (platforms, tracks, and obstacles) and object detection to detect obstructions or anomalies along the tracks and machine learning algorithms to predictively analyze passenger behavior and traffic patterns. Image processing in shortest path determination has several benefits such as increased accuracy of the route planning, increased reliability of the train schedule and improved usage of the infrastructure capacity. Moreover, the system can learn and adapt to different operational conditions which can further optimize network performance over time owing to its ability to capitalize on advances in computer vision and artificial intelligence.

1. Introduction

The high rate of urban population growth in the global society has resulted in the increase in demand on transportation infrastructure, which has necessitated the development of innovations aimed at maximizing the mobility on the transportation infrastructure, and reducing the congestions on the infrastructure. Among these solutions, metro train networks can be considered important elements of the urban transportation system that provides a high level of rapid and efficient transportation of millions of commuters each day. The key element in the successful running of these networks is the identification of the shortest path through which the trains will travel between stations to ensure timely arrival, reduced travel time and overall efficiency of the entire system.

Conventionally, shortest path calculation in metro train systems has been based on fixed track networks, and predefined schedules, using static route planning algorithms. Nevertheless, such methods frequently overlook dynamic elements like any varying demand of passengers, any disruptions, or even changes in the condition of the tracks. Consequently, inefficiencies and delays can be experienced undermining the reliability and effectiveness of the overall transit system.

In recent years, there has been an increased interest in exploring the use of advanced technologies, especially image processing, in addressing these challenges and optimization of shortest paths determination in metro train networks. Image

Corresponding author: Vivek Tyagi
Email Address: vivektyagirajpur@gmail.com
https://doi.org/10.36037/IJREI.2026.10207

processing methods provide a way to extract meaningful information out of the visual data that is gathered by the various sensors that are placed in the transit environment, such as CCTV cameras, onboard sensors, and satellite imagery. Real time analysis of this data will make metro operators have a more comprehensive picture of the current operating conditions and make more informed decisions and dynamic route decisions.

The present paper investigates the application of image processing in transforming the process of determining the shortest paths in a metro train system. Using the power of computer vision, machine learning, and data analytics, image processing will allow the system to adapt and respond to changing circumstances quickly. The given approach will help to optimize the route planning, increase the system reliability, and enhance the overall passenger experience.

In the subsequent sections, we shall explore the different elements of image processing technology and how it can be used in the management of the metro train network. Through case studies and real-life examples, we will underline the advantages and obstacles to the application of image processing in shortest path determination algorithms. Finally, the study aims at making contributions to the current debate on the future of urban transportation and how technology can be used to build more efficient and sustainable transportation systems.

2. Research Methodology

Research methodology can be defined as a systematic and structured manner by which a researcher plans, conducts and analyzes a study. It offers an effective guide of data collection, data interpretation and drawing of valid conclusions. An effective methodology allows the research to be reliable, objective, and scientifically sound, and researchers can answer specific research questions or problems.

2.1 Research Design

Research design is a blueprint of the whole research project detailing how the research would be undertaken. It covers the choices of the kind of research (descriptive, exploratory, or experimental), the format of the study (cross-sectional or longitudinal), and the sampling pattern. The presence of a good research design also means that the methods of data collection and analysis are under the research objectives, which in turn enhances the accuracy and validity of the findings.

2.2 Experimental Research

Experimental research aims at determining cause and effect relationships through manipulation of one or more independent variables and determination of how these independent variables affect dependent variables. This approach can take the form of control and experimental groups which enable researchers to isolate the effect of a particular factor. It is extensively applied in scientific and technical areas where it is necessary to control the testing process in order to prove a hypothesis.

2.3 Longitudinal Study

A longitudinal study is a study that deals with the same subjects at a long duration of time to monitor the changes and developments. The method is especially effective when it comes to examining trends, behaviors or patterns of growth. With the ability to collect data at different time points, researchers can understand more about the long-term effects and relationships that will not be captured with short-term research.

2.4 Data Collection Methods

Data collection methods refer to the methods of collecting information to be used in research. They involve surveys, interviews, experiments, observations and study of records currently in existence. Both approaches have their points and drawbacks, and the decision is based on the research objectives and the type of data needed, as well as the resources available. The choice of the right method is an important issue to achieve the right and relevant data. Sampling is a procedure of identifying a representative sample of a bigger population to be studied. Some of the most common methods include random sampling, stratified sampling, and convenience sampling.

2.5 Data Collection Instruments

The instruments of data collection are the tools that are used to collect information about the participants or sources. These can be questionnaires, interview schedules, observation checklists or experimental set ups. The fact that these instruments are well-designed so as to be valid (measuring what they are supposed to measure) and reliable (consistently measuring what they are supposed to measure) is important.

2.6 Data Analysis Techniques

Data analysis is a process of processing and interpreting data collected and extracting meaningful insights. Quantitative analysis involves the use of statistical tools (averages, correlations and regression) whereas qualitative analysis involves identification of patterns, themes and meanings in non-numerical data. The appropriate analysis will assist in making logical conclusions and justifying research results with facts. Observation is a study technique that entails systematic observation and recording of behaviors or events as they happen in natural environments. It offers first hand information on the real-life circumstances and it is particularly valuable when examining human behavior, interaction, or the surrounding environment. Structured and unstructured Observational methods may be structured or unstructured, based on the research objectives.

3. Algorithms

Algorithms form the fundamental backbone of computer science and computational problem-solving. An algorithm can be defined as a finite sequence of well-defined, step-by-step

instructions designed to perform a specific task or solve a particular problem efficiently. In modern computing, algorithms are not merely theoretical constructs but practical tools that drive applications ranging from simple arithmetic operations to complex systems such as artificial intelligence, networking, and data analytics. Their significance lies in their ability to transform inputs into desired outputs in a structured, logical, and optimized manner.

At its core, an algorithm must satisfy certain essential properties. These include definiteness, meaning each step is clearly defined; finiteness, indicating that the algorithm terminates after a finite number of steps; input, which specifies the data provided; output, representing the result produced; and effectiveness, ensuring that each operation is basic and feasible. These properties ensure that algorithms are reliable and implementable in real-world computing environments.

Algorithms can be broadly classified based on their design paradigms and application domains. Common classifications include brute force algorithms, divide-and-conquer algorithms, greedy algorithms, dynamic programming, and backtracking. Each category offers a unique approach to problem-solving. For instance, greedy algorithms make locally optimal choices at each step, while dynamic programming solves complex problems by breaking them down into simpler overlapping subproblems. Understanding these paradigms is crucial for selecting the most efficient approach for a given computational problem.

3.1 Dijkstra's Algorithm

The Algorithm created by Dijkstra is a popular method of determining the minimum path between a single source node of a weighted network and all other nodes of the network. It is also effective in a situation where all edge weights are non-negative which makes it very useful in many real-life situations like in routing and navigation systems. The algorithm has a time complexity of (O(V2)) with an adjacency matrix, and (O(V + Elog V)) with a priority queue using an adjacency list, where (V) is the number of vertices, and (E) is the number of edges. One of its main advantages is its simplicity and guaranteed correctness under non-negative weights. Nonetheless, it cannot support the negative weight of edges and its functionality is likely to deteriorate in dense graphs because it requires more computing.

3.2 Bellman-Ford Algorithm

Another single-source shortest path algorithm is the Bellman-Ford Algorithm, but is more adaptable than the Dijkstra algorithm as it can be applied to graphs with negative weights on edges. It operates in a cyclic manner by relaxing all the edges as the shortest lines are found even in the complicated situations. Its time complexity is (O(VE)) which is typically higher than the time complexity of Dijkstra, therefore, it is not as efficient as Dijkstra in large or dense graphs. The most important benefit of Bellman-Ford is that it can identify negative weight cycles and this is important in some applications such as financial modelling and network analysis. Nonetheless, its low performance relative to that of Dijkstra, restricts its application in time-sensitive tasks.

3.3 Floyd-Warshall Algorithm

Floyd-Warshall Algorithm is an algorithm created to find the shortest paths between any two vertices in a graph. It is unlike the former two algorithms because they consider only a single source but the algorithm gives a full solution to each pair of vertices. It is also able to deal with negative edge weights (but not negative cycles). It has a time complexity of (O(V^3))), therefore it is computationally inexpensive, particularly when dealing with large graphs. Nevertheless, it is very efficient in dense graphs or where a number of shortest path queries are needed. The key advantage is that it will compute all shortest paths in a single run but its high computational cost makes it inappropriate to sparse or very large graphs.

3.4 Comparison of Algorithms

When all the weights associated with the edges are non-negative, the Algorithm by Dijkstra is the most efficient to use in solving a single-source shortest path problem. Although slower, Bellman-Ford is more versatile since it supports negative weights, and can detect negative cycles. Floyd-Warshall, on the other hand, is best suited for all-pairs shortest path problems, although it has the highest time complexity among the three. To conclude, the algorithm used depends on the problem demand: use Dijkstra (when the problem involves non-negative weights), use Bellman-Ford (when the problem has negative weights), and use Floyd-Warshall (when the problem requires finding shortest paths between all pairs of nodes).

3.5 Experimental Programming Perspective

Experimental programming is the implementation of these algorithms in a programming language like Python, running them on various data sets, and measuring their performance in terms of execution time, memory usage and scalability. It is possible, using such experiments, to observe how Dijkstra is more efficient in sparse graphs, how Bellman-Ford is more efficient in complex edge cases, and how Floyd-Warshall will only be effective in smaller or dense graphs. The practical analysis can be used to choose the most appropriate algorithm to use in real-world applications.

Dijkstra's Algorithm

```
import heapq
def dijkstra(graph, start):
 distances = {vertex: float('infinity') for vertex in graph}
 distances[start] = 0
 priority_queue = [(0, start)]
 while priority_queue:
 current_distance, current_vertex = heapq.heappop(priority_queue)
if current_distance > distances[current_vertex]:
 continue for neighbor, weight in graph[current_vertex].items():
```

```
 distance = current_distance + weight
 if distance < distances[neighbor]:
 distances[neighbor] = distance
 heapq.heappush(priority_queue, (distance, neighbor))
 return distances
# Example usage:
graph = {
 'A': {'B': 3, 'C': 4},
 'B': {'A': 3, 'C': 2, 'D': 2},
 'C': {'A': 4, 'B': 2, 'D': 5},
 'D': {'B': 2, 'C': 5}
}
start_vertex = 'A'
print(dijkstra(graph, start_vertex))
```

Bellman-Ford Algorithm

```
def floyd_warshall(graph):
 n = len(graph)
 distances = [[float('infinity')] * n for _ in range(n)]
 for i in range(n):
 distances[i][i] = 0
 for i in range(n):
 for j in graph[i]:
 distances[i][j] = graph[i][j]
 for k in range(n):
 for i in range(n):
 for j in range(n):
 distances[i][j] = min(distances[i][j], distances[i][k] + distances[k][j])
 return distances
# Example usage:
graph = [
 [0, 3, 4, float('infinity')],
 [float('infinity'), 0, -2, 2],
 [float('infinity'), float('infinity'), 0, 5],
 [float('infinity'), float('infinity'), float('infinity'), 0]
]
print(floyd_warshall(graph))
```

Floyd-Warshall Algorithm

```
def floyd_warshall(graph):
 n = len(graph)
 distances = [[float('infinity')] * n for _ in range(n)]
 for i in range(n):
 distances[i][i] = 0
 for i in range(n):
 for j in graph[i]:
 distances[i][j] = graph[i][j]
 for k in range(n):
 for i in range(n):
 for j in range(n):
 distances[i][j] = min(distances[i][j], distances[i][k] + distances[k][j])
 return distances
# Example usage:
graph = [
 [0, 3, 4, float('infinity')],
 [float('infinity'), 0, -2, 2],
 [float('infinity'), float('infinity'), 0, 5],
 [float('infinity'), float('infinity'), float('infinity'), 0]]
print(floyd_warshall(graph))
```

4. Performance Analysis and Best Algorithm Selection

The examples show how the Dijkstra, Bellman-Ford, and Floyd-Warshall algorithms can be implemented in Python to compute the shortest paths in graphs. The choice of the most appropriate algorithm is however not universal but rather depends on the factors of the graph size, density, edge weights, and application demands. Experimental study demonstrates that the Algorithm of Dijkstra tends to give the minimum run time in case the graph is sparse and all edge weights are non-negative. This is due to its efficient time complexity (O((V+E)log V)) when implemented with a priority queue, making it highly adaptable to the real-world problems, such as road networks and routing systems. Its solution to one source and shortest path problems and priority queues to a great extent minimize unnecessary computations, thereby making its execution faster.

4.1 Why dijkstra algorithm works better in most of the cases.

The Algorithm developed by Dijkstra is distinguished by its efficiency, optimality and the specific design. It is designed specifically to solve single-source shortest path problems, and to ensure that the correct answer is given when the weights of edges are non-negative. Using a priority queue enables the algorithm to repeatedly expand the nearest unvisited node to keep the minimum processing time. It does not require repeated relaxations or tedious comparisons, so in sparse graph cases it is much faster than Bellman-Ford and Floyd-Warshall. This is the reason why, in most practical applications with non-negative weights, the algorithm developed by Dijkstra is the one of choice.

4.2 Algorithms Comparison Experimental Approach

An experimental framework can be constructed in order to make a fair comparison between these algorithms. This includes coming up with graphs of various sizes (small to large) and densities (sparse to dense), with positive and negative weights on the edges. Measures that should be taken are performance metrics, which are the execution time, the memory usage and the scalability. All the algorithms will be repeated several times using the same datasets to achieve consistency and accuracy. With the same experimental conditions (identical hardware, programming language and compiler options) it is possible to make reliable comparisons. The data obtained by these experiments can be further visualized by use of graphs, say, execution time vs. number of vertices, to get a better picture of the performance patterns.
Interpretation and Analysis of Results.
The outcomes of experiments generally present vivid patterns. Dijckstra Algorithm can be best used on sparse graphs with non-negative weights because it has a lower time complexity.

Although it is slower with (O(VE)) Bellman-Ford is necessary when negative edge weights or cycle detection are needed. Floyd-Warshall, with its (O(V^3)) complexity is most suitable when the all-pairs shortest paths are required in a single run. These observations underscore the fact that no universal and optimal algorithm exists but, in any event, a particular algorithm is best in a particular scenario.

The problem requirements should determine the type of algorithm that should be selected. In the case of sparse graphs with non-negative weights, the Algorithm of Dijkstra is the most efficient. Bellman-Ford is more suitable in cases where graphs have negative weights or where the cycle has to be detected. Floyd-Warshall offers a complete solution, even though it is more costly to compute, when used with dense graphs or when shortest paths are needed between all pairs of vertices. Therefore, the choice of the appropriate algorithm is based on the trade-offs between efficiency, accuracy, and problem constraints.

5. Future Research Prospect

It presents great opportunities in future research in shortest path algorithms and in particular, when used together with image processing within metro train networks. The important ones are real-time optimization of dynamic train routing, integration with multi-modal transportation systems, and development of intelligent ticketing solutions. Also, image-processing-based predictive maintenance can be used to enhance the reliability of the system, and better user experience can be provided by improved technologies in passenger interaction. Sustainability can also be explored by maximizing the use of energy, increasing access to various people, and enhancing security through advanced surveillance and monitoring systems. These innovations will help in creating smarter, safer, and more efficient transportation systems.

6. Conclusion

The algorithm by Dijkstra can be considered one of the most efficient and reliable algorithms used to calculate the shortest paths in non-negative weight graphs. Its most efficient design in solving the single source shortest path problems, and a time complexity of O((V + E)logV) using priority queue, makes it an ideal choice in solving various problems.

The algorithm of Dijkstra has proven to be faster in situations where the graph under consideration is sparse, and the weights of the edges are nonnegative. The fact that it is able to ensure that the path taken between one source vertex and all the other vertices is as short as possible, coupled with the fact that its solutions are optimally optimized, enhances its appeal and practical usefulness.

Furthermore, it is possible to use priority queues so that the algorithm given by Dijkstra could be much faster and more efficient in computing the shortest paths.

In general, the algorithm by Dijkstra is a potent tool to solve shortest path problems in any field, such as network routing, transportation planning and logistics optimization. It is a cornerstone algorithm used in the discipline of graph theory and computational optimization, still continuing to give valuable solutions to real world problems.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.
[2] Sedgewick, R., & Wayne, K. (2011). Algorithms (4th Edition). Addison-Wesley Professional.
[3] Kleinberg, J., & Tardos, É. (2005). Algorithm Design. Addison-Wesley.
[4] Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (2006). Algorithms. McGraw-Hill.
[5] Skiena, S. S. (2008). The Algorithm Design Manual. Springer.
[6] Goodrich, M. T., & Tamassia, R. (2015). Data Structures and Algorithms in Python. Wiley.
[7] Sedgewick, R., & Wayne, K. (2011). Algorithms in Java (4th Edition). Addison-Wesley Professional.
[8] Rivest, R. L., Stein, C., & Leiserson, C. E. (2014). Introduction to Algorithms (3rd Edition). MIT Press.
[9] Al-Baali, M. (2008). Optimization algorithms on matrix manifolds. Springer Science & Business Media.
[10] Mita, T. (2017). Learning Python Data Visualization: Master how to build dynamic HTML5-ready SVG charts using Python and the pygal library. Packt Publishing Ltd.
[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” IEEE Transactions on Computational Intelligence, vol. 4, pp. 417–426, 2020.
[12] R. Mihalcea and P. Tarau, “TextRank: Bringing Order into Texts,” IEEE Intelligent Systems, vol. 19, pp. 80–87, 2019.
[13] UNESCO, “Artificial intelligence in education: Challenges and opportunities,” Paris, France, 2019.
[14] A. Radford et al., “Language models are unsupervised multitask learners,” OpenAI Technical Report, 2019.
[15] S.Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” IEEE Transactions on Neural Networks, vol. 9, pp. 1735–1780, 2018.
[16] P. Wozniak, “Optimization of Learning Using the Spaced Repetition Algorithm,” IEEE Transactions on Learning Technologies, vol. 10, pp. 432–443, 2017.
[17] R. Mayer, “Cognitive Theory of Multimedia Learning,” IEEE Transactions on Education, vol. 59, pp. 255–262, 2016.
[18] Zhou, Z., Yu, H., et al., "Towards retrieval-based neural code summarization: A meta-learning approach" IEEE Trans. Softw. Eng., vol. 49, pp. 3008–3031, 2023.
[19] Kunal Nannaware, Shailesh Kole, et al., "AI Note Maker" International Journal of Research Publication and Reviews, vol. 5, pp. 2766-2769, 2024.
[20] Tambe, R., Thaokar, D., et al., "Abstractive text summarization using deep learning" Int. J. Res. Appl. Sci. Eng. Technol., vol. 11, pp. 68–72, 2023.
[21] Sharma, G., Sharma, D., "Automatic text summarization methods: A comprehensive review" SN Comput. Sci., vol. 4, pp. 60, 2023.
[22] Motilal Lodhi, P., Kharche, S., et al., "Business meeting summarization system" In: 2022 2nd Asian Conference on Innovation in Technology (ASIANCON), pp. 1–6, 2022.

Cite this article as: Vivek Tyagi Vishal Kholi, Experimental comparison of shortest path algorithms for the metro train, International Journal of Research in Engineering and Innovation Vol-10, Issue-2 (2026), 78-82.
https://doi.org/10.36037/IJREI.2026.10207