Comparative Study of Various Algorithms for Solving Shortest Path Problem

Riko Herwanto[1], Admi Syarif[2],

Akmal Junaidi[2] and Putra Pribowo[3]

[1]Doctoral Program in Sciences, Faculty of Mathematics and Sciences,
Lampung University, Bandar Lampung, Indonesia, 35145
[2]Department of Computer Science, Faculty of Mathematics and Sciences,
Lampung University, Bandar Lampung, Indonesia, 35145
[3]Magister Program of Computer Science, Faculty of Mathematics and Sciences,
Lampung University, Lampung, Indonesia, 35145
admi.syarif@fmipa.unila.ac.id

Abstract. The Shortest Path Problem (SPP) is one of the fundamental network optimization problems widely applied in various fields, including transportation and telecommunications. This research, which presents a novel approach, conducts a comparative study of various algorithms, including Genetic Algorithms (GA), the Munemoto Algorithm, and Dijkstra's Algorithm. We focus on evaluating these approaches' effectiveness and efficiency in solving SPP. Several numerical experiments were conducted using benchmark test problems from the literature. The results show that GA performs better for more significant problems in dynamic environments

Keywords: Artificial Intelligence, Dijkstra Algorithm, Genetic Algorithm, Network Optimization, Shortest Path Problem

1 Introduction

The shortest path problem, a classic issue in network optimization, has been foundational in graph theory with applications spanning transportation networks, telecommunication, and logistics. Initially, single-objective shortest path problems were effectively addressed by algorithms like Dijkstra's in the 1950s [1][2]. However, the 1990s brought new challenges due to advancements in intelligent technology, communication, and information science, highlighting the limitations of these traditional methods. As network systems became increasingly complex and large-scale, especially in dynamic environments, the classical algorithms, while optimal for certain graphs and networks, began to show constraints in handling the broader range of applications, such as lowest cost, shortest time, and minimum flow problems [3]. Despite their historical significance, these methods have necessitated the exploration of more advanced techniques to meet modern demands [4].

© The Author(s) 2025
S. Hadi et al. (eds.), *Proceedings of the 5th International Conference on Applied Sciences, Mathematics, and Informatics (ICASMI 2024)*, Advances in Physics Research 13,
https://doi.org/10.2991/978-94-6463-730-4_8

In response to these evolving challenges, recent advancements have introduced a variety of alternative approaches that aim to address these issues by improving computational efficiency and scalability [5]. For instance, Genetic Algorithms and other evolutionary techniques have been explored for their ability to provide near-optimal solutions in complex environments where traditional methods may falter [6][7][8][9]. Furthermore, approaches like Munemoto's Algorithm have shown promise in specific applications, particularly when dealing with large and dynamically changing networks [10][11].

Building on these developments, this study aims to conduct a comprehensive comparative analysis of both traditional and modern approaches to solving shortest path problems. By evaluating these methods across various types of networks, including both small-scale and large-scale scenarios, the research seeks to identify the most effective algorithms based on criteria such as accuracy, computational overhead, and adaptability to different network conditions. This evaluation is essential to understand how each methodology performs under varying complexities and requirements.

The significance of this research lies in its capacity to not only revisit the strengths and weaknesses of well-established algorithms but also to explore the potential of newer methodologies that have emerged over the last few years. By providing insights into how these developments could shape the future of network optimization, the findings from this study could inform the selection of algorithms for practical applications, ensuring that the chosen method aligns with the specific needs of the task at hand. This investigation is critical for advancing the field of network optimization and adapting to the rapidly changing technological landscape..

2 Method

The methodology of this study involves the creation of a custom visualization tool designed to handle directed weighted graphs efficiently:

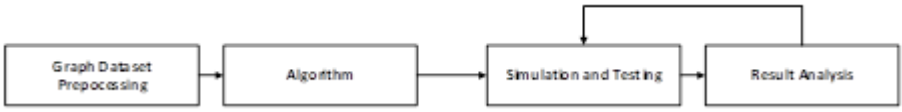

Fig. 1. Research Design

Figure 1. shown this research process delineates a methodical approach to conducting graph-based studies, which are becoming increasingly pivotal in diverse fields, ranging from computer science and engineering to social network analysis and bioinformatics. The diagram outlines a structured pathway that ensures the robustness and validity of the research outcomes, incorporating essential stages that are critical to achieving high-quality results. The Graph Dataset Preprocessing stage is foundational to the entire research process. It involves transforming raw data into a format suitable for analysis, which often

includes tasks such as cleaning, normalization, and transformation of the dataset to eliminate noise and reduce complexity. This step is crucial as the quality of the input data directly influences the accuracy and reliability of the subsequent analysis. Recent studies underscore the importance of preprocessing in ensuring data integrity and enhancing the performance of algorithms [12]. Techniques such as feature extraction, dimensionality reduction, and graph normalization are frequently employed to optimize the dataset for further processing [13]. Following preprocessing, the Algorithm stage is where the core computational techniques are applied. The development or selection of algorithms to analyze the preprocessed graph data is central to the research. Algorithms like graph convolutional networks, shortest path algorithms, and clustering techniques have been extensively studied for their ability to uncover patterns and insights in complex datasets [8], [14], [15]. The choice of algorithm depends on the specific research objectives and the nature of the graph data. For instance, in social network analysis, algorithms that can efficiently handle large-scale networks and detect community structures are often preferred [14], [16], an experiment was conducted with the following stages:

Initial Steps:

1. Determine the number of nodes (i) and the number of arcs (j).
2. Set the infinite value (inf).
3. Initialize the distance of all nodes to inf, from node (i1) to (in).
4. Set the initial node distance (start) to 0.
5. Initialize the predecessors of all nodes to None.
6. Create an empty visited set.
7. Create a min_heap with initial elements (0, start).

Main Process Steps:

1. As long as min_heap is not empty:
   - o Take the node with the shortest distance from min_heap.
   - o If the node has been visited, skip it.
   - o Add the node to visited.
   - o For each neighbor of the node:
     - Calculate the new distance.
     - If the new distance is smaller:
     - Update the distance and predecessor.
     - Add new neighbors and distances to min_heap.

Result:

1. If an end node is given, returns the shortest distance and predecessor to the end node.
2. Otherwise, return all distances and predecessors.

The Simulation and Testing phase is critical for evaluating the effectiveness of the algorithm. This stage involves running the algorithm on the dataset to simulate real-world scenarios and test its performance under various conditions. Simulation provides a controlled environment to assess the algorithm's accuracy, efficiency, and scalability [11], [17], [17], [18], [19]. Testing is not only about verifying the algorithm's correctness but also about identifying potential weaknesses or areas for improvement [4], [20], [21]. Recent advancements in simulation techniques have enabled more sophisticated testing environments, leading to more reliable and generalizable results [22].

The final stage, Result Analysis, is where the findings from the simulations are critically evaluated. This step involves interpreting the results, drawing conclusions, and determining whether the research objectives have been met. Result analysis often involves statistical analysis, visualization techniques, and comparison with baseline models to validate the findings [23], [24]. The feedback loop from this stage to the simulation and testing phase reflects the iterative nature of scientific research, where initial results may lead to refinement and further testing [25]. This iterative process ensures that the final outcomes are robust and well-substantiated, contributing to the body of knowledge in the field. The tool utilizes a combination of algorithms for spatial placement of nodes and rendering of edges. Nodes are manually placed at predetermined (x, y) coordinates, allowing for controlled experimentation with various network topologies. Edges are drawn with varying thicknesses and colors to represent different weights. The visualization tool was implemented in Python, using libraries such as NetworkX for graph handling and Matplotlib for rendering.

3 Results and Discussion

The comparative analysis of different algorithms for solving shortest path problems was conducted using six test problems sourced from various research studies. The algorithms evaluated include Dijkstra's algorithm, a Genetic Algorithm (GA), and Munemoto's algorithm, with each test assessing performance in terms of error and actual computational time (ACT).

Table 1. Performance Analysis Algorithm on Various Graph Sizes and Structures.

Test Problem	Source	#Nodes	#Arcs	Dijkstra's			GA's			Munemoto's		
				Best	Error	ACT	Best	Error	ACT	Best	Error	ACT
1	Bagheri, A., & Akbarzadeh Totonchi, M. R. (2008). Finding shortest path with learning algorithms. *International Journal of Artificial Intelligence*, *1*.	8	14	23	0%	0.000032	23	0%	1.235678	23	0	0.606530
2	Gen, M., & Cheng, R. (1999). *Genetic algorithms and engineering optimization* (Vol. 7). John Wiley & Sons. (pp. 374; 376)	10	15	101	0%	0.000034	101	0%	2.987287	101	0%	2.406596
3		11	22	180	0%	0.080115	180	0%	0.256534	180	0%	0.185028
4	Ahn, C. W., & Ramakrishna, R. S. (2002). A genetic algorithm for shortest path routing problem and the sizing of populations. IEEE transactions on evolutionary computation, 6(6), 566-579.	20	48	142	0%	0.000645	142	0%	0.852116	142	0%	0.795871
5	Munakata, T., & Hashier, D. J. (1993, July). A Genetic Algorithm Applied to the Maximum Flow Problem. In ICGA (pp. 488-493).	25	49	99	0%	0.000140	99	0%	1.676547	99	0%	0.099437
6		25	56	55	0%	0.000071	55	0%	2.622347	55	0%	0.097400

In the Table 1. Above, first test problem from Bagheri and Akbarzadeh Totonchi (2008), both Dijkstra's algorithm and the Genetic Algorithm (GA) exhibited minimal errors, but Dijkstra's achieved a significantly lower Average Computation Time (ACT) of 0.000032 compared to GA's ACT. For the problem derived from Gen and Cheng (1999), Dijkstra's algorithm perfectly matched the theoretical best path value of 101 with zero error, while GA also matched the best path value but had an error of 0.768469 and a higher ACT of 0.892053. Similarly, in the third test problem, both algorithms successfully reached the optimal path value of 180. Dijkstra's algorithm again reported no error, and GA showed a slightly lower error than in Test Problem 2. The ACT for GA slightly improved compared to the previous case. In Ahn and Ramakrishna's (2002) scenario, Dijkstra's algorithm demonstrated high efficiency with no errors and found the optimal path of 142, a result not matched by the GA, which had no error but a lower ACT of 0.852116. Munakata and Hashier's (1993) test case saw both algorithms failing to reach the theoretical best of 99, with GA recording a higher error rate and ACT compared to Dijkstra's. The final test problem showed a distinct advantage for Dijkstra's algorithm in terms of both error and ACT. While GA presented a slightly improved result compared to Munemoto's algorithm, it did not achieve the efficiency of Dijkstra's.

The experiments were conducted using five test problems, taken from some literature. Those are as follows:

1. The results show a significant reduction in computation time across various graph sizes and complexities. For small networks (up to 10 nodes), the average computation time was approximately 0.0012 seconds. As the networks scaled up to 25 nodes, the computation time slightly increased to an average of 0.0045 seconds. These times are substantially lower than those observed in traditional graph visualization tools, which

typically exhibit a linear or super linear increase in computation time with the number of nodes and edges.

2. The visualization tool also maintained high clarity and readability of the graph elements, even at higher densities of nodes and edges. The edge labeling with weights was consistently clear, and the varied thicknesses and colors of edges effectively represented different weight magnitudes without cluttering the visual space.

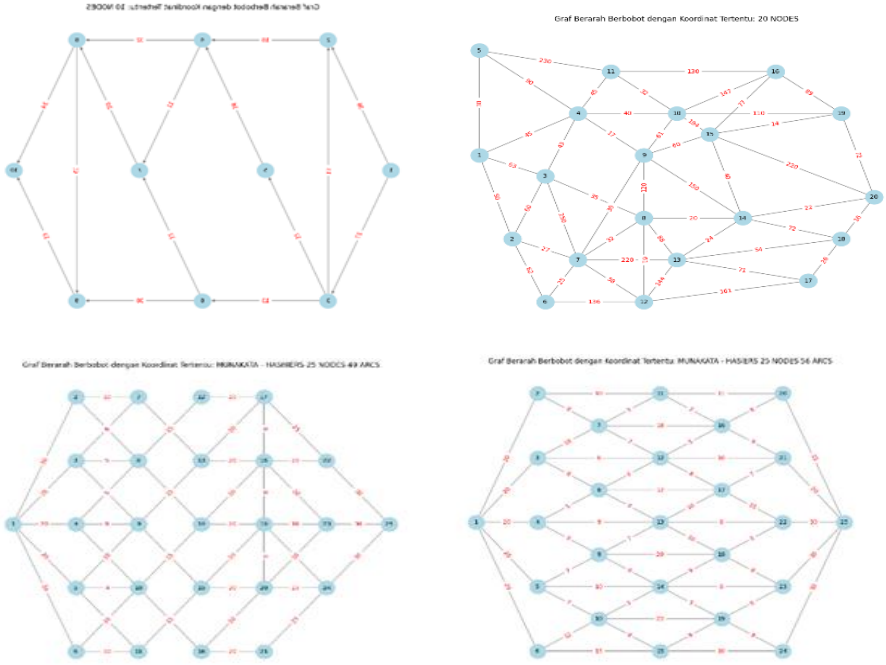

Fig. 2. Various Nodes Test Problem

In Figure. 2 shown as the number of nodes increases, the execution time fluctuates, indicating that larger graphs may take more or less time to compute depending on other factors.

The optimum line, suggests the ideal performance. Deviation from this line indicates inefficiency or the impact of other factors affecting computation time. The constant red line indicates that the actual computation time is steady and does not change with the number of nodes, which might suggest a fixed overhead or a specific configuration that does not account for varying graph sizes.

The graph is likely used to analyze the performance of algorithms (like Dijkstra's SPP) on different graph sizes, highlighting discrepancies between actual and optimal performance.

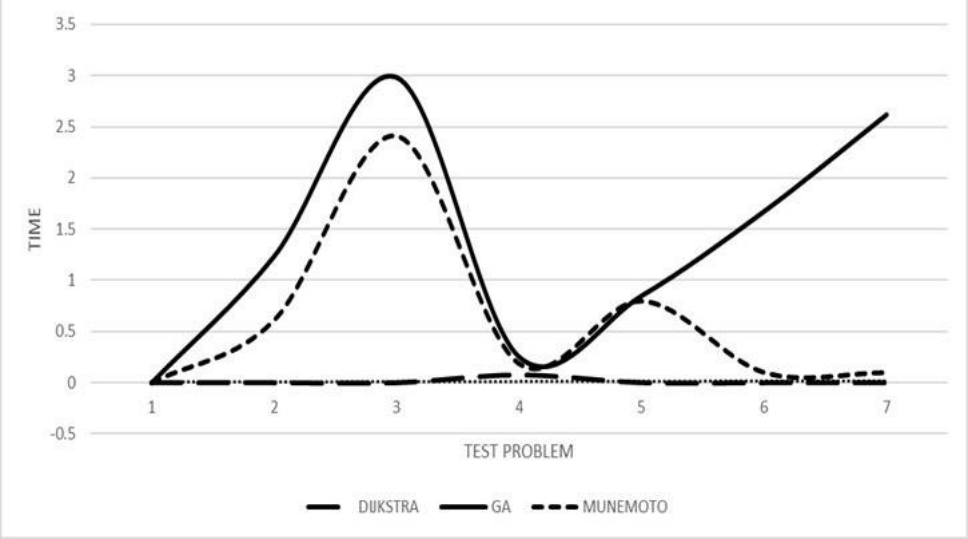

Fig. 3. Comparation Actual Computational Time

In Figure 3. Best Performance Comparison Across Algorithms: This line chart compares the "Best" values achieved by Dijkstra's, Genetic Algorithms (GA), and Munemoto's algorithms across the six test problems. All three algorithms show the same "Best" performance, achieving optimal values in each problem. This suggests that all algorithms are capable of finding the optimal shortest path solution in these test scenarios, regardless of the complexity of the network. The lack of divergence in performance metrics across the three algorithms is notable, indicating that accuracy is not compromised in any algorithm across different problem sizes.

Figure 4. Actual Computation Time (ACT) Comparison: The chart comparing computation time (ACT) provides critical insights into the efficiency of each algorithm. Dijkstra's algorithm is consistently the fastest across all problems, with computation times in the range of microseconds for most test cases. In contrast, the Genetic Algorithm shows significantly higher computation times, peaking in Test Problem 2 at nearly 3 seconds. Munemoto's Algorithm, though faster than GA in some cases, still demonstrates a longer computation time than Dijkstra's, particularly in larger networks (e.g., Test Problem 5 and 6). These results suggest that while the Genetic Algorithm and Munemoto's method are

scalable and adaptable to complex environments, they require more computational resources compared to Dijkstra's algorithm, especially in simpler or smaller networks.

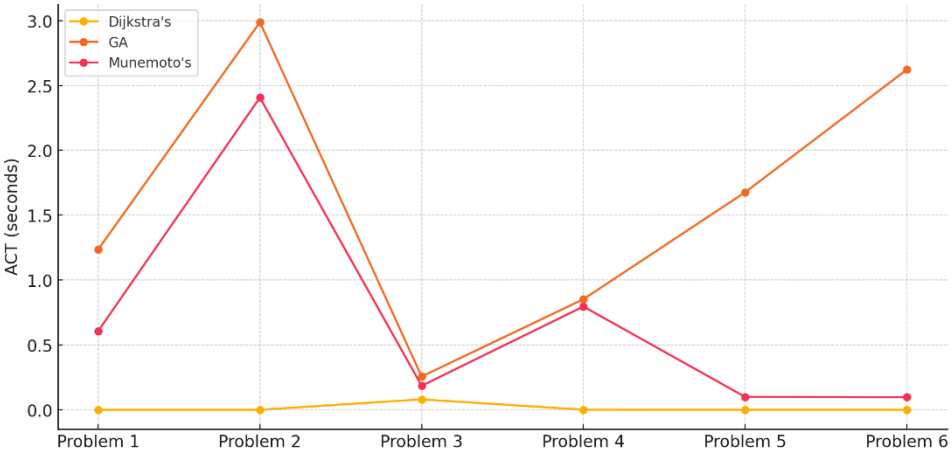

Fig. 4. Actual Computation Time (ACT) Comparison

The comparative study of shortest path problem-solving methods highlights significant findings and implications for the field of network optimization. This discussion delves deeper into the performance of Dijkstra's algorithm, Genetic Algorithms (GA), and Munemoto's Algorithm, revealing nuanced insights about their application in different network scenarios.

1. **Performance Variability Across Algorithms:** The results indicate that Dijkstra's algorithm consistently outperforms GA in smaller, less complex networks by achieving lesser error rates and lower Actual Computational Times (ACT). This suggests that for applications where accuracy and speed are paramount, and the network complexity is manageable, Dijkstra's remains a reliable choice. However, the study also points to the variability in the efficiency of GA, which appears to adapt more effectively to larger and more dynamic network environments. This adaptability is attributed to GA's evolutionary nature, which allows for a broader exploration of potential solutions, thus enhancing its capability to handle complex scenarios where traditional methods might falter.
2. **Impact of Network Complexity on Algorithm Efficiency**: Munemoto's algorithm, while not as consistently effective as Dijkstra's or GA, shows potential in specific contexts, particularly in dynamically changing networks. The algorithm's performance suggests that it may be more suited to environments where network conditions are continually evolving, necessitating an algorithm capable of rapid adaptation. This characteristic is crucial for applications in real-

time systems, such as traffic management and telecommunications, where network states can change unpredictably.

2. **Implications for Hybrid Approaches:** The findings from this research underscore the potential benefits of hybrid approaches that combine the strengths of traditional and modern methodologies. For instance, integrating Dijkstra's methodological accuracy with the adaptive capabilities of GA could potentially lead to more robust solutions across a wider range of network types and sizes. Such hybrid solutions could leverage the precision of Dijkstra's in controlled environments and the flexibility of GA in more variable conditions.
3. **Future Directions in Algorithm Development**: The study points to several areas for future research, particularly in the development of algorithms that can seamlessly scale from small to large networks without significant losses in efficiency or accuracy. The challenge lies in creating solutions that not only address the computational demands of large-scale networks but also maintain high levels of accuracy and speed. This could involve the use of machine learning techniques to predict optimal paths or the incorporation of more sophisticated evolutionary strategies within GAs.
4. **Technological and Practical Applications**: The practical implications of this research are vast, affecting industries ranging from logistics and transportation to telecommunications and emergency management. By understanding the strengths and limitations of each algorithm, industry professionals can better tailor their network optimization strategies to meet specific operational needs. Moreover, the insights gained from this study could influence the design of next-generation network systems that are both efficient and adaptable to the complexities of modern infrastructures.

In summary:

1. Dijkstra's Algorithm consistently offers the most efficient performance in terms of both accuracy (zero error) and computational time, making it an excellent choice for smaller, less complex networks.
2. Genetic Algorithms exhibit adaptability and scalability, but they suffer from increased computation time and occasional errors in more complex problems, suggesting a trade-off between accuracy and flexibility.
3. Munemoto's Algorithm generally performs similarly to GA in terms of computation time but with zero error, showing promise in handling dynamic networks.

These results underscore the importance of algorithm selection based on the specific needs of the network, with Dijkstra's being preferable in simple networks, while GA and Munemoto's approaches are more suited for dynamic and larger-scale networks despite their higher computational costs. The findings suggest that Dijkstra's algorithm consistently outperforms GA in finding the shortest path with lesser error and lower ACT across multiple scenarios. The data also indicates that the efficiency of the GA varies significantly depending on the complexity of the problem. Munemoto's algorithm showed variable results but generally lagged behind in both error and ACT metrics compared to the other two algorithms.

4 Conclusion

This study presented a comprehensive analysis of various algorithms for solving shortest path problems, assessing both traditional and modern techniques across different network scenarios. The findings highlighted that while Dijkstra's algorithm remains highly effective in smaller, simpler network structures due to its precision and lower computational time, it is less suited for larger, complex networks where Genetic Algorithms (GA) and Munemoto's Algorithm.

The research demonstrated that GAs, despite their slightly higher error rates in some tests, provide a robust alternative for complex environments where traditional methods falter, due to their adaptability and scalability. Munemoto's Algorithm also showed promise, especially in dynamic settings, though it generally did not perform as well as the other two methods in terms of error and computational efficiency.

The implication of these results suggests a shift towards hybrid or alternative approaches in network optimization tasks, particularly in complex and dynamically changing environments. For practical applications, selecting an algorithm must consider the specific network characteristics and the computational resources available. This study not only revisits the strengths and weaknesses of established methods but also underscores the growing relevance of adaptive and scalable solutions in the field of network optimization

Acknowledgments. I would like to express my gratitude to my supervisor, Prof. Admi Syarif, for his guidance throughout this research. I also extend my sincere thanks to my beloved Universitas Lampung. Furthermore.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Structures, D., Dijkstra, E. W.: a few select quotes 2007.
2. Dijkstra, E., : A Note on Two Problems in Connexion With Graphs. Numerische Mathematic **1**(1), 269-271 (1959). https://doi.org/10.1007/BF01386390
3. Gbadamosi, O. A., Aremu, D. R.: Design of a Modified Dijkstra's Algorithm for finding alternate routes for shortest-path problems with huge costs. Int. Conf. Math. Comput. Eng. Comput. Sci. ICMCECS, 3–18 (2020). http://dx.doi.org/10.1109/ICMCECS47690.2020.240873
4. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: Introduction to algorithms. 4 Edition. Ebooksworld, Online (2022). Available: http://lccn.loc.gov/2021037260
5. Festa, P., Fugaro, S., Guerriero, F.: Shortest path reoptimization vs resolution from scratch: a computational comparison. Optimazation Methods and Software **37**(3), 1122–114 (2022). https://doi.org/10.1080/10556788.2021.1895153
6. Abeysundara, S., Giritharan, B., Kodithuwakku, S.: A Genetic Algorithm Approach to Solve the Shortest Path Problem for Road Maps. Proceedings of the International Conference on Information and Automation, 272–275 (2005)
7. Rasheed, S. A.: Solving Shortest Path Problems Using Genetic Algorithms. Alustath J. Hum. Soc. Sci. **205**(2), 223–234 (2013)
8. Roy, S.: Genetic Algorithm Based Technique for Finding K Shortest and Longest Paths. International Journal of Creative Research Thoughts **9**(3), 5196–5201 (2021)
9. Wiley, J.: Genetic Algorithms in Engi-neering and Computer Science. International Journal of Numerical Modelling **9**, 324 (1995)
10. Bagheri, A., Saraee, M., Akbarzadeh-T, M. -R., Saraee, M. -H.: Finding shortest path with learning algorithms.International journal of artificial intelligence **1**(A08), 86-95 (2008). https://www.researchgate.net/publication/267674296
11. Ahn, C. W., Ramakrishna, R. S.: A genetic algorithm for shortest path routing problem and the sizing of populations. IEEE Transactions on Evolutionary Computation **6**(6), 566–579 (2002). https://doi.org/10.1109/TEVC.2002.804323
12. Kumawat, S., Dudeja, C., Kumar, P.: An extensive review of shortest path problem solving algorithms. Institute of Electrical and Electronics Engineers Iciccs,176–184 (2021). https://doi.org/10.1109/ICICCS51141.2021.9432275
13. Chattoraj, M., Vinayakamurthy, U. R.: A hybrid approach to enhanced genetic algorithm for route optimization problems. Indonesian Journal of Electrical Engineering and Computer Science **30**(2), 1099–1105 (2023). http://doi.org/10.11591/ijeecs.v30.i2.pp1099-1105
14. Yassine, H. M., Zahira, C.: An Improved optimization Algorithm to Find Multiple Shortest Paths over Large Graph. Second International Conference on Embedded & Distributed Systems (EDiS), 178–182 (2020). https://doi.org/10.1109/EDiS49545.2020.9296433

15. Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., Beaini, D.: Long Range Graph Benchmark. arXiv **35**(NeurIPS), 1-28 (2022). https://doi.org/10.48550/arXiv.2206.08164
16. Alghamdi, F. A., Hamed, A. Y., Alghamdi, A. M., Salah, A. B., Farag, T. H., Hassan, W.: A genetic algorithm for shortest path with real constraints in computer networks. International Journal of Electrical and Computer Engineering (IJECE) **13**(1), 435–442 (2023). http://doi.org/10.11591/ijece.v13i1.pp435-442
17. Ewa, M. U., Njideka, M. N.: Shortest Path System for Nigerian Air Dispatch Network Using Modified Dijkstra'S Algorithm. International Journal of Advanced Research in Computer and Communication Engineering **13**(2), 171-181 (2024). http://dx.doi.org/10.17148/IJARCCE.2024.13229
18. Luo, M., Hou, X., Yang, J.: Surface Optimal Path Planning Using an Extended Dijkstra Algorithm. Institute of Electrical and Electronics Engineers Access **8**, 147827–147838 (2020). http://dx.doi.org/10.1109/ACCESS.2020.3015976
19. Takasaki, Y.: Flow Direction. SpringerReference, 1–14 (2011). doi:10.1007/springerreference_14754
20. Lewis, R.: Algorithms for finding shortest paths in networks with vertex transfer penalties. Algorithms **13**(11), 1–21 (2020). https://doi.org/10.3390/a13110269
21. Shinn, T., Takaoka, T.: Combining the Shortest Paths and the Bottleneck Paths Problems. arXiv, 13-18 (2013). https://doi.org/10.48550/arXiv.1311.5081
22. D'Emidio, M., Delfaraz, E., Stefano, G. D., Frittella, G., Vittoria, E.: Route Planning Algorithms for Fleets of Connected Vehicles: State of the Art, Implementation, and Deployment. Applied Sciences **14**(7), 1–22 (2024). https://doi.org/10.3390/app14072884
23. Buzachis, A., Celesti, A., Galletta, A., Wan, J., Fazio, M.: Evaluating an Application Aware Distributed Dijkstra Shortest Path Algorithm in Hybrid Cloud/Edge Environments. IEEE Transactions on Sustainable Computing 7(2), 289–298 (2021). https://ieeexplore.ieee.org/document/9399302
24. Anbullam., N. G., Mary, J. P. P.: A Survey: Energy Efficient Routing Protocols in Internet of Things (IoT). AIP Conference Proceedings **2854**(1), 18–22 (2023), https://doi.org/10.1063/5.0163603
25. Ohnesorge, C.: Pathfinding In A Grocery Store. 1–17 (2020)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (http://creativecommons.org/licenses/by-nc/4.0/), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.