Floyd-Warshall Algorithm for Delivery Cost Optimization in Online Transportation Services

Reysha Syafitri Mulya Ramadhan - 13524137
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: reysha4school@gmail.com , 13524137@std.stei.itb.ac.id

Abstract—Online transportation companies, specifically food delivery sector, often rely on promotions to attract customers. This practice raises a critical challenge: sustaining profitability, making operational efficiency paramount. In this paper, this study explores the application of the Floyd-Warshall algorithm to optimize delivery routes by modeling the network as a directed graph. A simulation using a six-node network demonstrates that the algorithm successfully identifies all-pairs shortest paths, often revealing non-obvious, indirect routes. The findings highlight the use of such algorithms in enhancing cost efficiency and help companies maintain profitability better.

***Keywords*—Floyd-Warshall Algorithm; Discreet Mathematics; Online Transportation Business**

I. Introduction

Online transportation services have experienced rapid growth in Indonesia, becoming an indispensable part of daily life for a significant portion of the productive-age population. The sector, pioneered by major brands since the early 2010s, has expanded to include numerous companies.

This leads to a central question: how can they sustain such offers without incurring significant financial losses? The underlying strategy often lies in rigorous cost control, where optimizing delivery routes is paramount. Shorter and more efficient routes then translate directly to reduced operational costs, enabling companies to offer attractive pricing and promotions, which in turn stimulates customer demand and loyalty.

While route optimization is not a new problem, existing models may not be perfectly suited for the complexities of modern online delivery. For instance, a notable study on the Mumbai Dabbawalas by Suraj Patro and Motahar Reza presents an efficient, large-scale delivery system. However, its methodology, which relies on a simple coding system and the local railway network, is not directly transferable to today's technology-driven platforms that must account for complex urban road networks [1].

Therefore, this study aims to demonstrate a more suitable approach for modern online transportation businesses. We investigate the application of the Floyd-Warshall algorithm to compute the all-pairs shortest paths within a delivery network, providing a complete and static distance matrix. Such matrix is a foundational tool that can empower these companies to enhance logistical efficiency, reduce operational costs, and attempts to create a viable model for customer satisfaction and revenue growth.

II. Literature Review

A. Graph Theory

A graph is a mathematical structure used to model the relationships between discrete objects. Formally, a graph G is defined as a pair of sets (V, E), where V is a non-empty set of vertices {v1, v2, ..., vn}, and E is a set of edges {e1, e2, ..., en} that represent connections between pairs of those vertices.

Based on the presence of loops or multiple edges, graphs can be classified into two types, 1) Simple Graph: A graph is considered a simple graph if it does not contain any loops (edges that connect a vertex to itself) or multiple edges (more than one edge connecting the same pair of vertices). 2) Unsimple Graph (or Multigraph): Conversely, a graph that contains at least one loop or one pair of multiple edges is known as an unsimple graph or, more commonly, a multigraph.

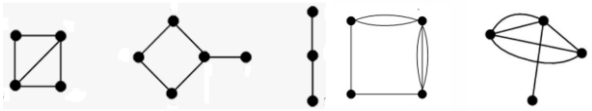

Figure 1. Illustration of Simple Graph (left) and Unsimple Graph (right) Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, accessed on 20/06/2025

Based on the orientation of their edges, graphs are classified into two main types. An undirected graph consists of edges that have no specific orientation. In contrast, a directed graph (or *digraph*) is a graph in which every edge is given a distinct orientation from one vertex to another.

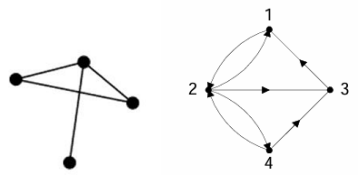

Figure 2. Illustration of Undirected Graph (left) and Directed Graph (right) Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, accessed on 20/06/2025

B. Floyd Warshall Algorithm

The following explanation of the Floyd-Warshall algorithm below is adapted from [2].

B.1 Definition

The Floyd-Warshall algorithm operates by using a two-dimensional array to store the shortest distances between all pairs of nodes. At the beginning, this array is initialized with the direct edge weights between nodes. The algorithm then iteratively updates the array by checking whether shorter paths can be found through intermediate nodes.

It is applicable to both directed and undirected weighted graphs and can handle edges with positive or negative weights. However, it does not work correctly on graphs that contain negative cycles—cycles in which the total sum of edge weights is negative.

B.2 Concept behind the algorithm

Given a graph represented by a dist[][] matrix with V vertices labelled from 0 to V-1, the goal is to compute the shortest distances between every pair of vertices, where dist[i][j] holds the shortest distance from vertex i to vertex j.

The core idea of the Floyd-Warshall algorithm is to treat each vertex k (from 0 to V-1) as a potential intermediate point between all pairs of vertices (i, j). When processing vertex k, it is assumed that all shortest paths using only vertices 0 to k-1 have already been determined. This enables the algorithm to refine previous shortest paths and refine them by including vertex k.

B.3 Proof

The algorithm is based on the principle of *optimal substructure*, which states that: If the shortest path from vertex i to vertex j passes through an intermediate vertex k, then both the sub-paths i → k and k → j must themselves be shortest paths.

By iteratively considering each vertex as an intermediate node and using already computed shortest paths for smaller subsets of nodes, the algorithm guarantees globally optimal distances.

B.4 Step-by-Step Implementation

The implementation of the Floyd-Warshall algorithm is fundamentally an iterative process that updates a distance matrix. The procedure can be detailed as follows:

1. *Distance Matrix Initialization*: The first step is to initialize a V × V distance matrix, dist, where V is the number of vertices in the graph. For each pair of vertices (i, j), dist[i][j] is populated with the direct edge weight from i to j. If no direct edge exists, its value is set to infinity (∞) to indicate that no path is yet known. The distance from a vertex to itself, dist[i][i], is always initialized to 0.
2. *Iterative Processing*: The algorithm then iterates through all vertices in the graph, from k = 0 to V-1. In each iteration, vertex k is considered as a potential intermediate node for all possible pairs of vertices (i, j).
3. *Distance Update*: For every pair (i, j), the existing distance dist[i][j] is compared against the length of the path passing through k, which is dist[i][k] + dist[k][j]. If the path via k is shorter, dist[i][j] is updated with this lower value. This operation is formally expressed as:

Upon completion of all iterations, the dist matrix will contain the shortest path distances between every pair of vertices in the graph.

C. Comparison of Shortest Path Algorithms

In the field of graph theory, several algorithms exist for finding the shortest path, each with distinct characteristics and use cases. The selection of an appropriate algorithm is crucial as it depends on the specific requirements of the problem, such as graph structure and the desired output.

Dijkstra's algorithm is widely recognized for its efficiency in solving the single-source shortest path (SSSP) problem [5]. It excels at finding the shortest path from a single origin node to all other nodes in a graph. The algorithm operates by iteratively selecting the node with the smallest known distance that has not yet been visited, making it highly effective for graphs where all edge weights are non-negative [6]. However, its fundamental limitation is its inability to correctly process graphs containing negative edge weights, as this can lead to erroneous path calculations [5].

To address the limitation of negative weights, the Bellman-Ford algorithm provides a more robust solution [6]. It also solves the SSSP problem but can operate on graphs that include edges with negative values. Its iterative approach, which relaxes edges |V|-1 times, guarantees a correct shortest path calculation in the presence of negative weights and provides the added benefit of detecting negative-weight cycles—a condition where an infinitely short path could exist [7]. The trade-off for this versatility is a slower computational performance compared to Dijkstra's algorithm on graphs with exclusively non-negative weights [6].

While both Dijkstra's and Bellman-Ford's algorithms are designed to compute paths from a single source, the Floyd-Warshall algorithm addresses the all-pairs shortest path (APSP) problem [5]. Its purpose is to compute the shortest distance between every possible pair of vertices in a given graph simultaneously. To achieve the same result with an SSSP algorithm, one would need to execute it once for each vertex in

the graph, which is often less efficient [7]. The Floyd-Warshall algorithm is therefore exceptionally well-suited for applications where a complete, static matrix of all shortest routes is needed beforehand. In the context of a delivery service network, this allows for the pre-computation of all potential routes, enabling quick lookups for dynamic routing and logistical planning.

III. METHOD

This study utilizes a C++ implementation of the Floyd-Warshall algorithm to determine the shortest path distance between all pairs of nodes in a weighted, directed graph. The methodology is divided into the experimental setup and the graph construction for simulation.

A. Implementation and Pseudocode

The graph is represented by a V × V adjacency matrix, dist, where V is the total number of nodes. Each element dist[i][j] stores the weight of the directed edge from node i to node j. If no direct edge exists between two nodes, dist[i][j] is assigned a large integer value (e.g., 10^8) to represent infinity. The diagonal elements, dist[i][i], are initialized to 0. The implementation is designed to handle graphs with non-negative edge weights.

The core logic follows the standard Floyd-Warshall algorithm, as detailed in the pseudocode below:

Algorithm 1: Floyd-Warshall

Input: A V × V adjacency matrix dist Output: The matrix dist containing all-pairs shortest paths

```
1: for k ← 0 to V-1 do
2:    for i ← 0 to V-1 do
3:       for j ← 0 to V-1 do
4:          if dist[i][k] + dist[k][j] < dist[i][j] then
5:             dist[i][j] ← dist[i][k] + dist[k][j]
6:          end if
7:       end for
8:    end for
9: end for
10: return dist
```

B. Graph Construction for Simulation

To create a realistic test case, a weighted, directed graph of a small urban area was constructed using geographical data sourced from Google Earth. The network consists of six nodes, representing two points of origin (restaurants) and four delivery destinations (houses).

The initial edge weights, representing the travel distance between directly connected nodes, were measured using the 'Measure distance' tool available in Google Earth. Each measurement was traced along (assumed) plausible road networks to approximate the actual routes a delivery driver would take. The resulting network topology and initial distances are visualized in Fig. 3 and detailed in TABLE I.

Figure 3. Illustration of 6 nodes graph
Souce: https://earth.google.com, accessed on 19/06/2025

Several key assumptions were made during the construction of this model to maintain a clear scope:

- Static Environment: The model is static. All edge weights (distances) are considered constant and do not account for dynamic variables such as real-time traffic conditions, road closures, or traffic signals.
- Distance as Cost: The primary metric for optimization is travel distance. It serves as a direct proxy for operational costs (e.g., fuel) and delivery time. Factors such as tolls or driver compensation models are not included.
- Directed Paths: The graph is modeled as a directed graph to accurately reflect real-world urban road conditions, such as one-way streets, where the travel distance from node A to B may not be the same as from B to A.

TABLE I. Initial Matrix of 6 Nodes (h stands for home/houses, r stands for restaurants).

from/to	h0	h1	h2	h3	r1	r2
h0	0	156.68 m	485.39 m	∞	652.86 m	202.55 m
h1	∞	0	308.49 m	63.92 m	371.44 m	∞
h2	∞	∞	0	∞	130.51 m	∞
h3	∞	∞	∞	0	∞	53.5 m
r1	∞	∞	∞	∞	0	319.52 m
r2	202.55 m	∞	∞	∞	∞	0

IV. RESULTS AND ANALYSIS

The algorithm iterated six times (for k = 0 to 5), treating each node as an intermediate vertex to find potential shortcuts. Through this process, numerous path distances were updated as shorter routes were discovered. For instance, the path from node h0 to h3, which was initially non-existent (∞), was updated to 220.60 meters via the intermediate node h1. Similarly, the path from h2 to r2 was established through r1 with a final distance of 450.03 meters.

A. Step-by-Step Iteration Analysis

To demonstrate how the algorithm progressively finds shorter paths, the state of the distance matrix after the first two iterations is examined.

1. *Iteration k = 0 (Intermediate Node: h0)*: The algorithm first considers h0 as an intermediate node. It checks if traveling through h0 can create new paths or shorten existing ones. The most significant changes occur for paths originating from node r2, as it has a direct connection to h0. For instance, the path from r2 to h1 was previously unknown (infinity). By routing through h0, a path is established with a distance of dist[r2][h0] + dist[h0][h1] = 202.55 + 156.68 = 359.23 meters. Similar updates were made for other pairs, as shown in TABLE II.

TABLE II. First Iteration Matrix with k = 0

from/to	h0	h1	h2	h3	r1	r2
h0	0.00	156.68	485.39	inf	652.86	202.55
h1	inf	0.00	308.49	63.92	371.44	inf
h2	inf	inf	0.00	inf	130.51	inf
h3	inf	inf	inf	0.00	inf	53.50
r1	inf	inf	inf	inf	0.00	319.52
r2	202.55	359.23	687.94	inf	855.41	0.00

2. *Iteration k = 1 (Intermediate Node: h1)*: In the next iteration, h1 is used as the intermediate node. This step yields several critical path optimizations. A key discovery is the path from h0 to h3. Previously infinite, the algorithm now finds a route via h1 (h0 → h1 → h3), calculating the distance as dist[h0][h1] + dist[h1][h3] = 156.68 + 63.92 = 220.6 meters. Furthermore, an existing path, such as from h0 to h2, is improved. The initial direct distance of 485.39 meters is replaced by the shorter path through h1, which is 156.68 + 308.49 = 465.17 meters. The matrix after these updates is shown in TABLE III.

TABLE III. Second Iteration Matrix with k = 1

from/to	h0	h1	h2	h3	r1	r2
h0	0.00	156.68	465.17	220.6	528.12	202.55
h1	inf	0.00	308.49	63.92	371.44	inf
h2	inf	inf	0.00	inf	130.51	inf
h3	inf	inf	inf	0.00	inf	53.50
r1	inf	inf	inf	inf	0.00	319.52
r2	202.55	359.23	667.72	423.15	730.67	0.00

B. Final Distance Matrix and Insights

The final matrix, containing the shortest path distances between all pairs of nodes, is shown in TABLE IV.

TABLE IV. Final Result Matrix

from/to	h0	h1	h2	h3	r1	r2
h0	0.00	156.68	465.17	220.60	528.12	202.55
h1	319.97	0.00	308.49	63.92	371.44	117.42
h2	652.58	809.26	0.00	873.18	130.51	450.03
h3	256.05	412.73	721.22	0.00	784.17	53.50
r1	522.07	678.75	987.24	742.67	0.00	319.52
r2	202.55	359.23	667.72	423.15	730.67	0.00

The final distance matrix reveals several key insights. A primary finding is the importance of path directionality in route optimization. For example, the shortest path from h1 to r1 is 371.44 meters, whereas the return path from r1 to h1 is significantly longer at 678.75 meters. This asymmetry, common in urban environments with one-way streets, highlights the necessity of using a directed graph model for accurate cost calculation.

Furthermore, the results imply that pre-computing an all-pairs shortest path matrix allows a system to dynamically select the most efficient routes. With optimal paths pre-computed, online transportation businesses can reduce total travel distance and time, which in turn lowers operational expenses for fuel and driver compensation. Over time, these efficiencies can significantly improve profitability and service speed.

V. LIMITATIONS

While this study successfully demonstrates the applicability of the Floyd-Warshall algorithm for delivery route optimization, it is important to acknowledge the boundaries and assumptions of the model to contextualize the findings. The primary limitations of this research are as follows:

- Static Environment Model: The simulation is based on a static graph where edge weights represent fixed distances derived from Google Earth. This model does not account for the dynamic nature of a real-world urban environment. Variables such as real-time traffic congestion, road closures, weather conditions, and time-of-day fluctuations (e.g., rush hour) are not considered, yet they critically affect optimal route selection in practice.

- Simplified Cost Function: The optimization in this study uses travel distance as the sole proxy for operational cost. A more comprehensive cost model for a transportation business would also incorporate other significant factors, including variable fuel consumption, vehicle maintenance schedules, driver wages, potential road tolls, and the implicit cost of meeting customer delivery time windows.

- Absence of Operational Constraints: The model does not factor in practical operational constraints faced by delivery services. These include vehicle capacity (the maximum number of orders a driver can carry), order batching strategies (grouping multiple orders for efficiency), or the dynamic assignment of incoming orders to available drivers. The analysis assumes an unconstrained, one-to-one delivery scenario for each calculated path.

Acknowledging these limitations highlights that while the Floyd-Warshall algorithm provides a powerful foundational tool for a static network, its direct implementation must be

enhanced for a fully dynamic, real-world logistics system. These points serve as a direct basis for the future work proposed in the conclusion.

VI. Practical Implications and Future Work

While the core analysis has demonstrated the algorithm's effectiveness, it is valuable to discuss how these findings translate into real-world operational strategies and what future research directions this opens.

A. Practical Implications for Businesses

The pre-computed all-pairs shortest path matrix, generated by the Floyd-Warshall algorithm, is not merely a theoretical outcome; it is a powerful foundational tool for enhancing logistical operations in several ways:

1. High-Speed Route Lookups: In a live system, this static matrix can be stored in a high-speed, in-memory database. When the dispatch system needs to assign a driver, it does not need to compute a route from scratch. Instead, it can perform a near-instantaneous lookup of the distance between any restaurant and customer. This drastically reduces server load and response time, which is critical during peak hours.
2. Foundation for Order Batching: Although the model in this paper does not cover order batching, the generated matrix is a prerequisite for it. To determine the most efficient route for a driver delivering multiple orders (e.g., from Restaurant A to Customer B then Customer C), the system can use the matrix to quickly calculate and compare the total distance of all possible permutations (e.g., $A \rightarrow B \rightarrow C$ vs. $A \rightarrow C \rightarrow B$). This enables the development of sophisticated, multi-stop optimization features.
3. Strategic Network Analysis: Beyond real-time dispatching, the distance matrix is a valuable asset for strategic business planning. By analyzing the matrix, a company can identify which areas are "costly" to serve, which routes are most efficient, and where to strategically place driver waiting hubs or "cloud kitchens" to minimize average delivery distance across the entire network.

B. Future Work

The limitations identified in the previous section naturally point toward several avenues for future research to create a more robust and dynamic model:

1. Integration with Real-Time Data: A key next step is transforming the current static model into a dynamic one. This would involve integrating real-time data from APIs (e.g., Google Maps API) to update edge weights based on current traffic congestion. Such a system might use a hybrid approach: using the Floyd-Warshall matrix as a baseline, but running a faster single-source algorithm like A* or Dijkstra for real-time adjustments on routes affected by traffic anomalies.
2. Advanced Cost Modeling: Future models could move beyond distance as a simple cost proxy. A multi-objective cost function could be developed, incorporating variables like estimated fuel consumption (which varies with speed and traffic), road tolls, and time-based driver compensation. This would provide a more accurate reflection of true operational costs.
3. Solving the Vehicle Routing Problem (VRP): To fully address operational constraints like vehicle capacity and delivery time windows, future work should frame the problem as a Vehicle Routing Problem (VRP). VRP is a well-known class of problems in operations research that aims to find the optimal set of routes for a fleet of vehicles to serve a number of customers. This would represent a significant step up in model complexity and real-world accuracy.

VII. Conclusion

This study successfully demonstrated the application of the Floyd-Warshall algorithm for computing all-pairs shortest paths in a simulated urban delivery network. By utilizing real-world geographical data to model the graph, the algorithm effectively identified optimized routes, many of which were non-obvious, indirect paths established through intermediate nodes.

The key findings confirm the algorithm's suitability for this application. First, the results underscore the critical importance of path directionality, as asymmetrical travel distances (e.g., the path from h1 to r1 versus r1 to h1) significantly affect cost and efficiency. Second, the systematic, iterative process of the algorithm guarantees that all possible routes are evaluated, ensuring the final distance matrix is globally optimal.

Ultimately, the pre-computed matrix of shortest paths provides a powerful tool for online transportation businesses. It enables the development of more efficient routing logic, which can lead to a direct reduction in operational costs, faster delivery times, and improved service reliability. Such optimization supports sustainable business operations in a competitive environment. Future work could extend this model to incorporate dynamic variables such as real-time traffic conditions.

Acknowledgment

The author wishes to express the deepest gratitude to God Almighty for His grace and blessings, which made the completion of this work possible.

Sincere appreciation is extended to Dr. Ir. Rinaldi Munir, M.T., the lecturer for the IF2120 Discrete Mathematics course at the Institut Teknologi Bandung. His insightful guidance, clear explanations of complex concepts such as graph theory and dynamic programming, and unwavering support throughout the semester were instrumental in shaping the direction and quality of this paper.

The author is also grateful to the Informatics Program and the School of Electrical Engineering and Informatics at the Institut Teknologi Bandung for providing an academic environment that is conducive to learning and research.

Special thanks are also due to colleagues and classmates for the engaging discussions, valuable feedback, and collaborative atmosphere that enriched the learning process. Finally, the author would like to convey thanks to family and friends for their constant encouragement, patience, and unwavering support throughout this endeavour.

The author would also like to acknowledge the use of generative artificial intelligence tools for assistance in language refinement and content structuring for this paper.

REFERENCES

[1] S. Patro and M. Reza, "Optimization of a Large-Scale Lunch Box Delivery System - A Simple, Serializable, Synchronous, Parallel Simulated Annealing, and Floyd-Warshall Approach,"[1] *EasyChair Preprint no. 1623*, 2019.

[2] GeeksforGeeks, "Floyd-Warshall Algorithm," *GeeksforGeeks*, 2024. [Online]. Available: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/. Accessed: Jun. 19, 2025.

[3] L. M. Wastupranata, "Strategi Rute UAV untuk Pengantaran Obat-obatan bagi Penderita Covid-19 saat Isolasi Mandiri dengan Algoritma Cheapest Link & Sirkuit Hamilton"[2] [UAV Route Strategy for Delivering Medicines for Covid-19 Patients during Self-Isolation with the Cheapest Link Algorithm & Hamiltonian Circuit], student paper, Dept. of Informatics, Institut Teknologi Bandung, Bandung, Indonesia, 2021.

[4] R. Munir, Matematika Diskrit edisi ketiga. Bandung, 2009.

[5] C. L. Tondo, "Dijkstra's Algorithm: The Shortest Path Algorithm," *Medium*, 2023. [Online]. Available: https://medium.com/@catherineleninatondo/dijkstras-algorithm-the-shortest-path-algorithm-244da622c489. Accessed: Jun. 20, 2025.

[6] S. Jang, "Dijkstra vs. Bellman-Ford," *Medium*, 2023. [Online]. Available: https://medium.com/@sejang/dijkstra-vs-bellman-ford-290c8846a36a. Accessed: Jun. 20, 2025.

[7] Programiz, "Bellman-Ford Algorithm," *Programiz.com*, 2024. [Online]. Available: https://www.programiz.com/dsa/bellman-ford-algorithm. Accessed: Jun. 20, 2025. [4] B. A. Forouzan and D. F. Mosher, *Foundations of Computer Science*, 4th ed. Cengage Learning, 2017, pp. 493-495.

DECLARATION

I hereby declare that this paper is my own work, not an adaptation or translation of another's work, and is not an act of plagiarism.

Bandung, 19 June 2025

Reysha Syafitri M.R 13524137