Multi-Objective GoFood Route Optimization: Hamiltonian Path Approach on Real-Time Traffic-Weighted Graphs in Bandung

Bryan Pratama Putra Hendra - 13524067
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10, Bandung 40132, Indonesia
bryanpratama0111@gmail.com, 13524067@std.stei.itb.ac.id

***Abstract*— In urban areas, food delivery platforms such as GoFood are essential for daily convenience. However, assigning multiple customers to a limited number of drivers in a fair and efficient manner remains a significant challenge, especially in cities with unpredictable traffic like Bandung. This paper presents a graph-based optimization approach that models the delivery network using weighted graphs, where edge weights represent real-time travel durations. By evaluating all possible customer groupings and solving small-scale Traveling Salesman Problems (TSP) for each, the method aims to minimize both the average delivery time and the time range across all drivers. This dual-objective strategy improves delivery fairness and efficiency.**

***Keywords*— Graph theory, Traveling Salesman Problem, hamiltonian graph, route optimization, GoFood, fairness, vehicle routing**

I. INTRODUCTION

Food delivery services have become an essential part of everyday life in today's digital era, especially in cities like Bandung. Users can buy food from local restaurants and have it delivered directly to their homes through platforms like Gojek's GoFood service. This service not only provides jobs for the community, but also makes it easier for people who want to order food without leaving their homes.

However, there is a big question that is asked behind the ease of use of food delivery software. How can a limited number of drivers be assigned to many customers in a way that ensures efficiency and fairness? It's not just about figuring out the shortest path. It involves figuring out how to group and route deliveries so that no driver is overburdened and no customer is left waiting too long.

To address this, the author modelled the delivery scenario using graph theory and combinatorial optimization. Each delivery task consists of three stages: a driver travels from their starting location to the restaurant (pickup point), and then proceeds to deliver food to one or more customers in sequence. This system will be represented as a weighted graph, where nodes are locations and edges are estimated travel times.

This paper presents a fairness-aware optimization approach inspired by the Vehicle Routing Problem (VRP). By evaluating all possible groupings of customers and solving a small-scale Traveling Salesman Problem (TSP) for each assignment, this program aims to minimize not just the average delivery time, but also the range—the difference between the longest and shortest delivery times among all drivers. This ensures a more balanced workload and consistent delivery experience for all parties.

Through this method, the author showed how theoretical concepts from discrete mathematics and graph algorithms can be applied to real-world systems like GoFood to improve efficiency, equity, and user satisfaction, even under the dynamic constraints of urban traffic.

II. THEORETICAL FOUNDATION

A. Graph Theory

A.1. Graph Definition

Graphs are used to represent discrete objects and the relationships between them. By definition, Graph G = (V, E), where V is a non-empty set of vertices v1,v2,. . . ,vn, while E is the set of edges connecting a pair of vertices e1,e2,. . . ,en.

Based on the presence or absence of loops or multiple edges in a graph, graphs are classified into two types, namely Simple Graphs and Non-Simple Graphs. Simple Graphs are graphs that do not contain loops or multiple edges, while Non-Simple Graphs are graphs that have multiple edges or loops.

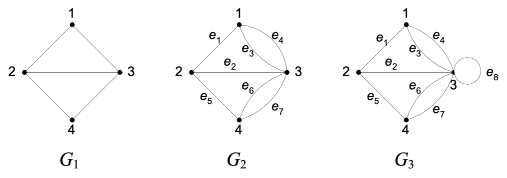


Fig. 1. Simple Graph (G1), Multigraph (G2) and Pseudograph (G3) Source: `https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf`

A.2. Graph Type

Based on the orientation of the direction on the side of a graph, graph is classified into two types, namely Undirected Graph and Directed Graph. An undirected graph is a graph whose edges do not have a direction orientation, while a directed graph is a graph whose edges are given a direction orientation.

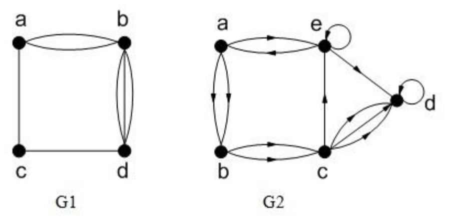


Fig. 2. Illustration of Undirected Graph (G1) and Directed Graph (G2)
Source: `https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf`

A.3. Weighted Graph

A graph which each of its edge is assigned a numerical value, or weight, that typically represents the cost, distance, or time required to travel between two connected vertices. In the context of route optimization, these weights are crucial because they allow the graph to model real-world factors such as traffic conditions, road lengths, or delivery time estimates.

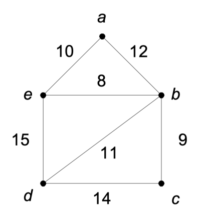
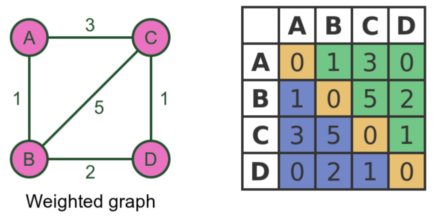


Fig. 3. Illustration of Weighted Graph
Source: `https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf`

A.4. Graph Representation

In graph theory, one common way to represent a graph is using an adjacency matrix. An adjacency matrix is a 2D array where the rows and columns represent the vertices of the graph. Each cell (i,j) contains a value that indicates whether there is an edge from vertex i to vertex j, and in the case of a weighted graph, the cell holds the weight of that edge. If there is no edge, the cell may contain a zero or a special value like infinity. This representation is especially useful for dense graphs, as it allows quick access to check the existence and weight of edges between any two vertices.



	A	B	C	D
A	0	1	3	0
B	1	0	5	2
C	3	5	0	1
D	0	2	1	0

Fig. 4. Illustration of Weighted Graph and Its Adjacency Matrix
Source: `https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf`

A.5. Graph Application

Graphs are widely used to model and solve real-world problems involving connections and relationships between objects. In logistics and delivery systems, graphs represent locations as vertices and roads or routes as edges, allowing for efficient route planning and navigation. Applications include shortest path algorithms for GPS navigation, network routing in telecommunications, and optimization problems like the Traveling Salesman Problem (TSP) and vehicle routing. For food delivery services like GoFood, graphs help simulate urban road networks and traffic conditions, enabling the system to compute optimal delivery routes that minimize travel time and cost.

B. Hamiltonian Path Theory

B.1. Definition of Hamiltonian Path

In graph theory, a path is a sequence of vertices in which each consecutive pair is connected by an edge. Paths can be either directed or undirected, depending on the type of graph. A path may or may not revisit the same vertices or edges, depending on the specific constraints of the problem. One important type of path is the Hamiltonian path, which is a path that visits each vertex in the graph exactly once, regardless of how many edges are used. Hamiltonian paths are commonly applied in real-world scenarios where each location must be visited once without repetition, such as in the Traveling Salesman Problem (TSP), logistics planning, and tour scheduling.

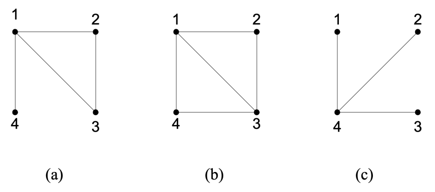


Fig. 5. Ilustration of Hamiltonian Path
Source: `https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf`

B.2. Algorithms

Since finding a Hamiltonian path or circuit is an NP-complete problem, exact algorithms can be computationally prohibitive for large graphs. For the purpose of optimizing multi-stop deliveries, where the number of stops (vertices) can be significant, the problem is often framed as a Traveling Salesman Problem (TSP). The goal in TSP is to find the shortest possible route that visits each city exactly once and returns to the origin city. While the pure Hamiltonian path problem doesn't require returning to the start, in delivery logistics, drivers typically start from a hub and ideally return there after completing all deliveries.

C. GoFood

GoFood is one of the core services within the Gojek ecosystem, a leading super app in Southeast Asia, particularly prominent in Indonesia. Go-Food was first launched in Indonesia in 2015 in Jakarta. This service focuses on the delivery of food and beverages from restaurants to consumers through a network of driver-partners. Since its launch, GoFood has become a key player in Indonesia's food delivery industry, significantly transforming how consumers order and receive food in urban areas.

Fig. 6. GoFood Logo
Source: Author's Archive

GoFood's relevance to this research lies in its operational nature, which necessitates multi-objective route optimization. GoFood drivers often have to pick up orders from one or multiple restaurants, then deliver them to various customer locations within a single delivery run. The primary challenge is to determine the most efficient sequence of pickups and deliveries, taking into account real-time travel times influenced by traffic congestion, and distance. The success of services like GoFood is measured not only by the number of orders completed but also by delivery speed, customer satisfaction, and driver operational efficiency, all of which are significantly impacted by the quality of the route optimization algorithms employed. Therefore, research into GoFood route optimization using dynamic traffic-weighted graph approaches is crucial for enhancing the performance and sustainability of food delivery services in dense urban environments.

D. Real-Time Weighted Graphs

In this study, the author model the GoFood delivery problem using a traffic-weighted directed graph, where each edge represents the estimated travel time between two locations. These weights are not static or theoretical—they are derived from real-time route data obtained via the OpenRouteService (ORS) API. This allows the graph to reflect current traffic conditions in urban environments such as Bandung at the moment of computation.

Unlike fully dynamic graph models that continuously update weights in response to live traffic feeds, our system adopts a snapshot-based approach. At the start of the optimization process, we retrieve up-to-date travel durations between all relevant nodes (drivers, restaurant, and customers), and use these values as fixed edge weights during route assignment and evaluation.

This preprocessed real-time weighting enables more accurate modeling of urban travel conditions without the overhead of constant API calls or re-optimization. By embedding real traffic data directly into the adjacency matrix of the graph, the system better approximates real-world constraints such as congestion, detours, and variable road speeds.

Using this snapshot model, each edge in the graph effectively encodes the latest known travel cost, forming a realistic basis for solving the Traveling Salesman Problem (TSP) and evaluating customer-driver assignments. While not dynamically reactive, this approach strikes a practical balance between realism and computational feasibility, especially for small-scale delivery batch planning.

E. Route Optimization

E.1. Traveling Salesman Problem (TSP) for Subroutes and VRP Formulation

The Traveling Salesman Problem (TSP) is a classical graph optimization problem where the goal is to find the shortest possible route that visits each given node exactly once and returns to the starting point (optional in open TSP). In the context of this research, we apply TSP as a subroutine to determine the optimal visiting order within each driver's assigned group of customers, starting from the central restaurant.

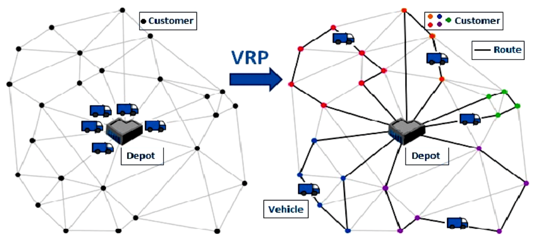


Fig. 7. Vehicle Routing Problem
Source:
https://www.researchgate.net/figure/Classical-Vehicle-Routing-Problem

This local optimization is part of a broader formulation resembling the Vehicle Routing Problem (VRP)—a generalization of TSP where multiple vehicles (drivers) are tasked with visiting subsets of customers. In our case, we solve a capacitated VRP without distance limits, where the objective is not just efficiency but also workload fairness among drivers.

Instead of solving a single large TSP across all customers, we divide the six customers into four non-overlapping groups—each assigned to a single driver—and solve a small-scale TSP for each group. This allows the system to evaluate every possible permutation within a group and determine the best visiting order to minimize delivery time.

E.2. TSP vs. Eulerian Path

It is important to distinguish between the TSP (a form of the Hamiltonian Path problem) and the Eulerian Path problem. While both are fundamental problems in graph theory, they target different goals:

- **Eulerian Path**: Seeks to traverse every *edge* in a graph exactly once. This is relevant for tasks like street sweeping, garbage collection, or utility inspection.
- **Hamiltonian Path / TSP**: Seeks to visit every *node* exactly once. This model is more appropriate for food

delivery, where each customer's location must be visited directly and efficiently.

In this delivery system, we adopt the TSP formulation within a VRP framework, as it aligns directly with the need to serve all customer nodes without redundancy or unnecessary detours.

E.3. Fairness as a Multi-Objective Constraint in VRP

While traditional VRP approaches prioritize minimizing the total distance or time, real-world urban logistics systems must also address the issue of fairness among delivery agents. In this study, fairness is formalized as the minimization of the *range* between the longest and shortest delivery times across all drivers:

$$\text{Range} = \max(\text{TotalTimedriver}) - \min(\text{TotalTimedriver}) \quad (1)$$

This fairness objective ensures that no single driver is disproportionately burdened with long or inefficient delivery routes. A more balanced workload distribution improves operational equity, increases driver satisfaction, and enhances consistency in customer wait times—especially critical in urban settings with variable traffic conditions.

E.4. Exhaustive Search in Small-Scale VRP vs. Heuristic Approaches

Given the small size of the problem in this paper, the author implement a full exhaustive search strategy. This involves:

- Generating all valid groupings (partitions) of customers into four non-empty subsets.
- Evaluating all permutations of driver-to-group assignments.
- Solving a TSP for each group to determine the optimal visiting order from the restaurant to the customers.
- Calculating the total delivery time per driver and computing the overall delivery range and mean time.

Although exhaustive search has exponential complexity, it guarantees globally optimal results in both fairness and efficiency for small instances. However, for larger-scale delivery systems—such as those involving dozens or hundreds of customers—this approach becomes infeasible.

In such cases, the fairness-aware VRP could be approximated using heuristic or metaheuristic techniques such as:

- Clustering-based pre-grouping (e.g., k-means, DBSCAN)
- Greedy or regret-based insertion heuristics
- Evolutionary algorithms (e.g., genetic algorithms, ant colony optimization)
- Reinforcement learning-based route planning

These methods can approximate fairness while maintaining computational feasibility, allowing the model to scale to real-world logistics operations.

III. IMPLEMENTATION

A. Graph Initialization

The author modeled a delivery area in Bandung as a weighted directed graph $G = (V, E)$, where each vertex $v \in V$ represents a real-world location such as a driver's starting point, a customer's home, or a restaurant. Each edge $e = (u, v) \in E$ represents a directed route between two points, weighted by the estimated travel time in minutes.

To visualize this graph, we plotted the spatial layout of drivers, customers, and the restaurant on a map of Bandung using their respective GPS coordinates. The nodes in the visualization are color-coded based on their role in the delivery system:

- **Red node** represents the restaurant,
- **Green nodes** represent the drivers,
- **Blue nodes** represent the customers.

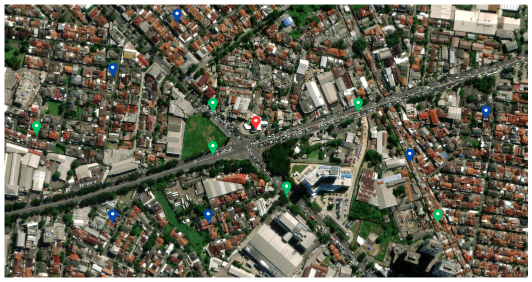


Fig. 8. Spatial Visualization of Drivers, Customers, and Restaurant
Source: Author's Archive

At the heart of every order you receive, there's a carefully planned journey designed to ensure everything goes smoothly. Think of it this way:

Each of the drivers (the green nodes) has a crucial mission. That is to deliver a meal to the customers (the blue nodes). But before your meal arrives at your door, the driver first needs to stop by the central restaurant (the red node) to pick up your order. So, the complete path for every delivery always follows a two-stage route: Driver $\rightarrow$ Restaurant $\rightarrow$ N Customer.

The author's main goal is to make this delivery experience as fair and efficient as possible for everyone involved. The program is not just aiming for the fastest overall delivery, but it's deeply committed in making sure that no customer has to wait disproportionately longer than others. We strive to flatten out the delivery times, minimizing the difference between the quickest and slowest deliveries.

By doing this, we can ensure that all customers receive their orders within a relatively uniform time window. It's not just about speed; it's about fairness. Our drivers can work more smoothly without getting overwhelmed by bottlenecks, and ultimately, you, our valued customer, can enjoy your food without the stress of an unexpectedly long wait. This more consistent and predictable system is especially beneficial in a bustling urban environment like Bandung.

To construct the delivery graph, we begin by connecting each node—representing a driver or customer—to the central restaurant node. Then we will connect all the customer to each other. This results in a directed graph structure, as illustrated in Figure 13.

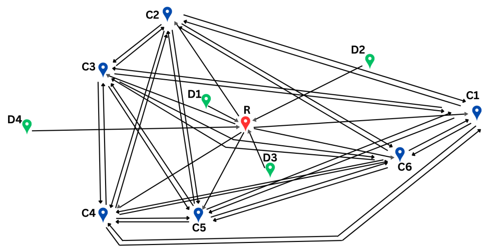
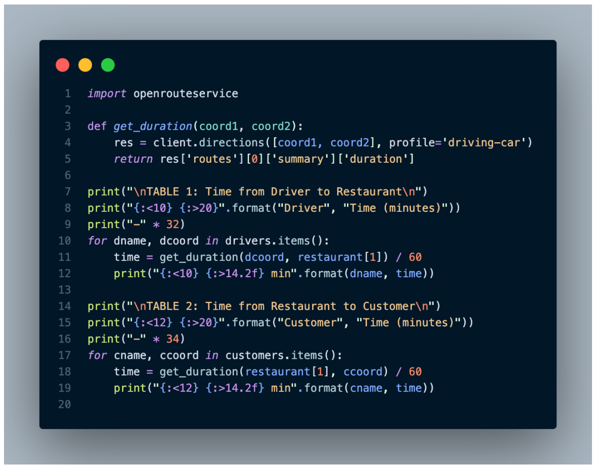


Fig. 9. Graph Representing Delivery Route
Source: Author's Archive

To assign weights to each edge, the author calculate the estimated travel time between two locations. This is done using real-time route data fetched from the OpenRouteService (ORS) API. The code snippet in Figure 10 demonstrates how the travel duration between two geographical coordinates can be retrieved.

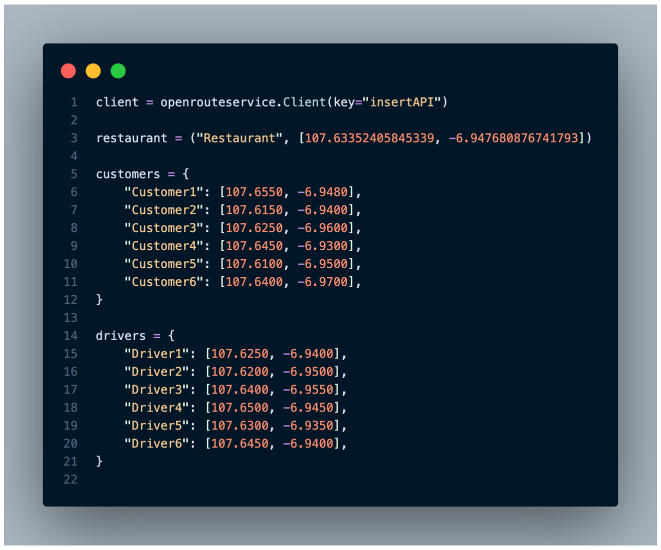


```python
import openrouteservice

def get_duration(coord1, coord2):
    res = client.directions([coord1, coord2], profile='driving-car')
    return res['routes'][0]['summary']['duration']

print("\nTABLE 1: Time from Driver to Restaurant\n")
print("{:<10} {:>20}".format("Driver", "Time (minutes)"))
print("-" * 32)
for dname, dcoord in drivers.items():
    time = get_duration(dcoord, restaurant[1]) / 60
    print("{:<10} {:>14.2f} min".format(dname, time))

print("\nTABLE 2: Time from Restaurant to Customer\n")
print("{:<12} {:>20}".format("Customer", "Time (minutes)"))
print("-" * 34)
for cname, ccoord in customers.items():
    time = get_duration(restaurant[1], ccoord) / 60
    print("{:<12} {:>14.2f} min".format(cname, time))
```

Fig. 10. Function to fetch travel time using OpenRouteService
Source: Author's Archive

Before executing the function, it is necessary to define the geographical coordinates of all relevant entities in the delivery system(the drivers, customers, and the restaurant). These coordinates are represented using latitude and longitude, which are numerical values that pinpoint a specific location on the Earth's surface.

These spatial coordinates serve as the input for computing real-world travel distances and durations between locations. To retrieve accurate routing information, the system integrates with the OpenRouteService (ORS) API, which requires a valid API key for authentication. The API uses the input coordinates to calculate the optimal road-based paths between nodes, taking into account the actual road network and optionally, real-time traffic data. Figure 11 shows how these locations are initialized in the code, laying the groundwork for route computation.



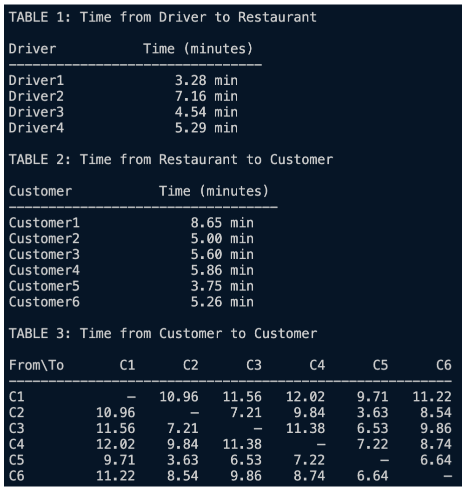
```python
client = openrouteservice.Client(key="insertAPI")

restaurant = ("Restaurant", [107.63352405845339, -6.947680876741793])

customers = {
    "Customer1": [107.6550, -6.9480],
    "Customer2": [107.6150, -6.9400],
    "Customer3": [107.6250, -6.9600],
    "Customer4": [107.6450, -6.9300],
    "Customer5": [107.6100, -6.9500],
    "Customer6": [107.6400, -6.9700],
}

drivers = {
    "Driver1": [107.6250, -6.9400],
    "Driver2": [107.6200, -6.9500],
    "Driver3": [107.6400, -6.9550],
    "Driver4": [107.6500, -6.9450],
    "Driver5": [107.6300, -6.9350],
    "Driver6": [107.6450, -6.9400],
}
```

Fig. 11. Initialization of coordinates and ORS client
Source: Author's Archive

Running the function will help generate three tables displaying the travel times from drivers to the restaurant, from restaurant to customers, and from customer to customer.

- **First Table** Time from Driver to Restaurant,
- **Second Table** Time from Restaurant to Customer,
- **Third Table** Time from Customer to Customer.

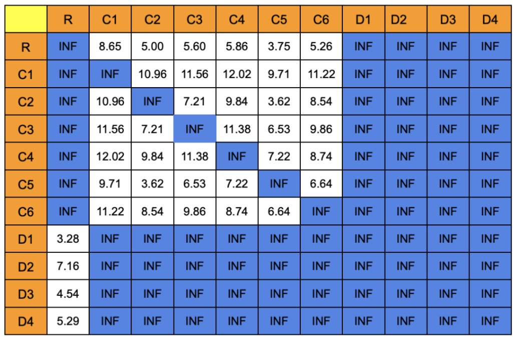
The product is as shown in Figure 12.



```
TABLE 1: Time from Driver to Restaurant

Driver           Time (minutes)
--------------------------------
Driver1              3.28 min
Driver2              7.16 min
Driver3              4.54 min
Driver4              5.29 min

TABLE 2: Time from Restaurant to Customer

Customer             Time (minutes)
----------------------------------
Customer1                8.65 min
Customer2                5.00 min
Customer3                5.60 min
Customer4                5.86 min
Customer5                3.75 min
Customer6                5.26 min

TABLE 3: Time from Customer to Customer

From\To       C1      C2      C3      C4      C5      C6
--------------------------------------------------------
C1             -   10.96   11.56   12.02    9.71   11.22
C2         10.96       -    7.21    9.84    3.63    8.54
C3         11.56    7.21       -   11.38    6.53    9.86
C4         12.02    9.84   11.38       -    7.22    8.74
C5          9.71    3.63    6.53    7.22       -    6.64
C6         11.22    8.54    9.86    8.74    6.64       -
```

Fig. 12. Output of travel times from each node to the restaurant

Using these values as edge weights, we can now construct the full weighted directed graph that forms the basis for route optimization in the next stage. For the representation we will use an adjacency matrix like below.



	R	C1	C2	C3	C4	C5	C6	D1	D2	D3	D4
R	INF	8.65	5.00	5.60	5.86	3.75	5.26	INF	INF	INF	INF
C1	INF	INF	10.96	11.56	12.02	9.71	11.22	INF	INF	INF	INF
C2	INF	10.96	INF	7.21	9.84	3.62	8.54	INF	INF	INF	INF
C3	INF	11.56	7.21	INF	11.38	6.53	9.86	INF	INF	INF	INF
C4	INF	12.02	9.84	11.38	INF	7.22	8.74	INF	INF	INF	INF
C5	INF	9.71	3.62	6.53	7.22	INF	6.64	INF	INF	INF	INF
C6	INF	11.22	8.54	9.86	8.74	6.64	INF	INF	INF	INF	INF
D1	3.28	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
D2	7.16	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
D3	4.54	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
D4	5.29	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF

Fig. 13. Simplified graph connecting all nodes to the restaurant
Source: Author's Archive

B. Route Assignment and Optimization

Once the weighted directed graph is constructed, the next step is to assign all customers to the available drivers in a way that minimizes disparities in delivery durations. Unlike a strict one-to-one mapping (one driver to one customer), the author allow each driver to serve multiple customers, as long as all deliveries start from the restaurant. This results in a delivery route of the form:

$$\text{Driver} \to \text{Restaurant} \to \text{Customer}_1 \to \cdots \to \text{Customer}_n$$

This problem can be modeled as a variant of the Vehicle Routing Problem (VRP), with the following constraints:

- Each driver must start by visiting the central restaurant (the pickup point).
- Every customer must be visited exactly once by exactly one driver.
- The route from the restaurant to the customers follows the most efficient visiting order (minimizing route cost).

The total delivery cost per driver is computed as:

$$\text{TotalTime}_{\text{driver}} = \text{Time}_{D\to R} + \text{TSP}_{R\to C_1\to\cdots\to C_n} \quad (2)$$

Unlike traditional VRP objectives that focus on minimizing the overall cost, our goal emphasizes fairness. We introduce a fairness metric called the *delivery range*, defined as:

$$\text{Range} = \max(\text{TotalTime}_{\text{driver}}) - \min(\text{TotalTime}_{\text{driver}}) \quad (3)$$

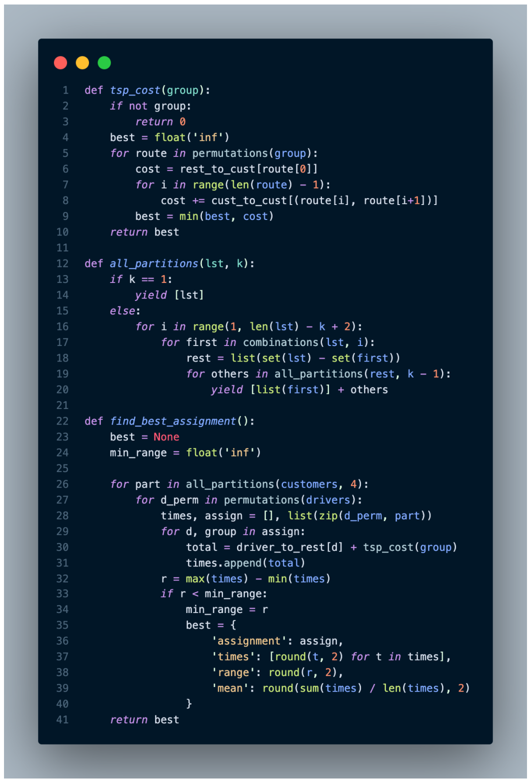
The lower this range, the more evenly distributed the delivery workload and waiting times are across all participants. This leads to a more balanced and equitable system:

- No single driver is disproportionately burdened with long delivery routes.
- All customers receive their orders within a similar time frame.

To find the optimal assignment, we exhaustively evaluated all valid partitions of the six customers into four non-empty groups (matching the number of drivers). For each partition:

1) All permutations of driver-to-group assignments are considered.
2) Each group's delivery route is optimized using a brute-force solution to the Travelling Salesman Problem (TSP).
3) The total delivery time for each driver is calculated using Equation (1).
4) The fairness of the configuration is assessed using Equation (2).

Among all configurations, the one with the lowest delivery range is selected as the most balanced. This method, although exhaustive and computationally expensive, guarantees optimality for our fairness-based objective within a small instance. In urban environments like Bandung, this approach helps mitigate traffic-induced inequalities in delivery time and enhances the predictability of the overall system.



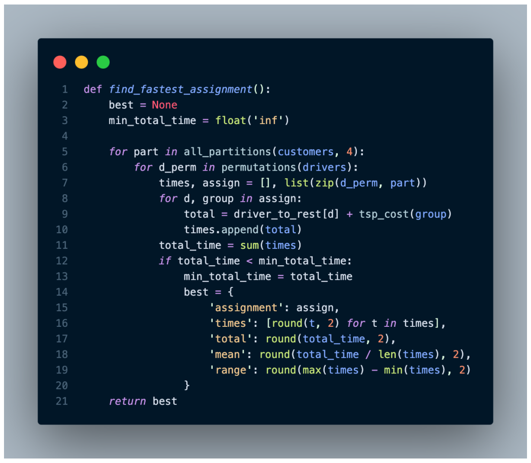
```python
def tsp_cost(group):
    if not group:
        return 0
    best = float('inf')
    for route in permutations(group):
        cost = rest_to_cust[route[0]]
        for i in range(len(route) - 1):
            cost += cust_to_cust[(route[i], route[i+1])]
        best = min(best, cost)
    return best

def all_partitions(lst, k):
    if k == 1:
        yield [lst]
    else:
        for i in range(1, len(lst) - k + 2):
            for first in combinations(lst, i):
                rest = list(set(lst) - set(first))
                for others in all_partitions(rest, k - 1):
                    yield [list(first)] + others

def find_best_assignment():
    best = None
    min_range = float('inf')

    for part in all_partitions(customers, 4):
        for d_perm in permutations(drivers):
            times, assign = [], list(zip(d_perm, part))
            for d, group in assign:
                total = driver_to_rest[d] + tsp_cost(group)
                times.append(total)
            r = max(times) - min(times)
            if r < min_range:
                min_range = r
                best = {
                    'assignment': assign,
                    'times': [round(t, 2) for t in times],
                    'range': round(r, 2),
                    'mean': round(sum(times) / len(times), 2)
                }
    return best
```

Fig. 14. Code Implementation(Fairness Time approach)
Source: Author's Archive

The code shown in Figure 14 implements the fairness-based route assignment algorithm. The `tsp_cost()` function exhaustively evaluates all permutations of a customer group to find the minimum delivery route from the restaurant. The `all_partitions()` function recursively generates all possible ways to divide the six customers into four non-empty groups. Finally, `find_best_assignment()` iterates through every valid partition and driver-to-group permutation, calculates the total time for each driver, and selects the configuration with the smallest delivery range.

This implementation guarantees optimality by evaluating every possible configuration, making it suitable for small-scale delivery problems where fairness is a critical factor. The total computational complexity of the algorithm is $\mathcal{O}(P \cdot D \cdot T)$, where P is the number of valid customer partitions into four non-empty groups, $D = 4!$ is the number of driver-to-group permutations, and T is the cost of solving the Traveling

Salesman Problem (TSP) for each group, which is factorial in the group size ($\mathcal{O}(n!)$ in the worst case).



```python
def find_fastest_assignment():
    best = None
    min_total_time = float('inf')

    for part in all_partitions(customers, 4):
        for d_perm in permutations(drivers):
            times, assign = [], list(zip(d_perm, part))
            for d, group in assign:
                total = driver_to_rest[d] + tsp_cost(group)
                times.append(total)
            total_time = sum(times)
            if total_time < min_total_time:
                min_total_time = total_time
                best = {
                    'assignment': assign,
                    'times': [round(t, 2) for t in times],
                    'total': round(total_time, 2),
                    'mean': round(total_time / len(times), 2),
                    'range': round(max(times) - min(times), 2)
                }
    return best
```

Fig. 15. Code Implementation(Fastest Time approach)

The code in Figure 17 implements the fastest-time-based assignment strategy, where the primary objective is to *minimize the total cumulative delivery time* across all drivers, regardless of fairness. Unlike the fairness-oriented method shown earlier, this approach selects the configuration with the lowest sum of delivery durations, even if it results in some drivers having significantly longer routes than others.

The algorithm structure reuses the same core components: `tsp_cost()` for solving the shortest delivery sequence within each group, and `all_partitions()` for generating all valid customer groupings. However, the main function, `find_fastest_assignment()`, prioritizes *global efficiency* by selecting the configuration with the minimal total time:

$$\text{TotalTime}_{\text{all drivers}} = \sum_{i=1}^{4} \text{TotalTime}_{\text{driver}_i} \qquad (4)$$

This method is particularly suitable for scenarios where speed is critical—such as time-sensitive deliveries or peak-hour dispatching. However, it may lead to unbalanced workload distribution among drivers since it does not explicitly minimize the delivery time range. As such, it represents a trade-off between overall efficiency and equity.

IV. RESULTS AND DISCUSSION

The author evaluated two optimization strategies for assigning six customers to four drivers: one minimizing the delivery **range** (fairness) and the other minimizing the **total delivery time** (efficiency). Each solution computes the best permutation of customer groupings and their internal routing using a brute-force Traveling Salesman Problem (TSP) approach.

Table I presents the fairness-optimized configuration. Each driver starts from their initial location, proceeds to the restaurant, and follows a route visiting one or more customers.

TABLE I
FAIRNESS-OPTIMIZED ASSIGNMENT

Driver	Assigned Customers	Total Time (minutes)
D1	[C2, C3]	15.49
D2	[C4]	13.02
D3	[C6, C5]	14.93
D4	[C1]	13.94

The configuration shown in Table I represents the most balanced assignment based on fairness criteria. Each driver is assigned one or more customers, with routing optimized via the Traveling Salesman Problem (TSP) to minimize delivery time within each group. This approach achieves a **mean delivery time** of **14.35 minutes** and a remarkably low **range of 2.47 minutes**, ensuring equitable workload distribution and consistent delivery experiences across all drivers.

In contrast, Table II presents the assignment optimized solely for **total delivery time**, without regard to fairness:

TABLE II
FASTEST-TIME ASSIGNMENT

Driver	Assigned Customers	Total Time (minutes)
D1	[C1]	11.93
D2	[C4]	13.02
D3	[C6]	9.80
D4	[C2, C5, C3]	19.87

This configuration yields a lower **mean delivery time** of **13.66 minutes**, indicating higher overall efficiency. However, the **range increases significantly to 10.07 minutes**, highlighting a substantial imbalance in route durations. In this scenario, one driver (D4) carries the heaviest burden, while others complete relatively short trips.

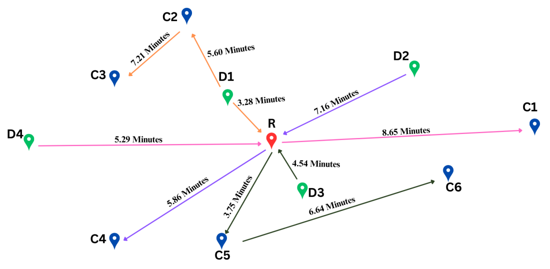


Fig. 16. Optimized Directed Delivery Graph (Fairness-Based)
Source: Author's Archive

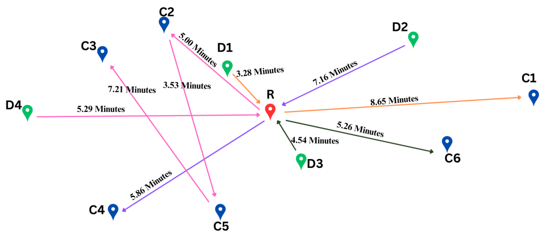


Fig. 17. Optimized Directed Delivery Graph (Fastest Time-Based)
Source: Author's Archive

From a graph-theoretic perspective, each driver's path forms a directed subgraph beginning at the restaurant and traversing one or more customers. The sequence within each subroute is determined using a brute-force TSP solver to ensure intra-group efficiency. While this approach has factorial complexity, it remains computationally feasible for small-scale problems involving a limited number of nodes.

In practical urban logistics scenarios such as those in Bandung, where traffic unpredictability can heavily influence delivery durations, fairness becomes a critical factor. A fairness-optimized model reduces the likelihood of overburdening specific drivers and helps maintain uniform delivery standards. Conversely, while the efficiency-focused model yields faster average completion times, it risks driver fatigue and service disparity due to uneven load allocation.

V. CONCLUSION

In multi-customer food delivery, it is essential to adopt routing strategies that balance speed and fairness. This paper successfully introduces a fairness-driven optimization model that assigns delivery tasks to drivers by exhaustively evaluating all valid customer groupings and delivery sequences, using combinatorial search and Traveling Salesman Problem (TSP)–based route optimization.

The proposed method demonstrates that equitable task distribution can be achieved without significantly sacrificing overall efficiency. By minimizing the delivery time *range*. The system ensures that no single driver is overloaded, and all customers receive their food with relatively consistent service times. This contributes to operational sustainability, reduces driver fatigue, and enhances user satisfaction.

Although the approach relies on exhaustive computation, which limits its scalability to small delivery batches, it serves as a reliable baseline for fairness-aware routing. Future enhancements may include real-time traffic integration, customer location clustering, or heuristic and machine learning methods to scale the approach while retaining its fairness objective.

Ultimately, this paper underscores the importance of equity in last-mile logistics and offers a practical foundation for designing routing systems that are not only fast, but also fair—especially in resource-constrained or traffic-sensitive settings like urban Indonesia.

VI. APPENDIX

The complete source code that is used in this paper will be available on my Github Repository

There will be a short video explaining this paper on my youtube channel, My YouTube Channel

VII. ACKNOWLEDGMENT

First and foremost, the author is deeply grateful to God Almighty for the strength, guidance, and perseverance granted throughout the completion of this work. Sincere gratitude is also given to Dr. Ir. Rinaldi Munir, M.T., for his invaluable guidance and inspiring lectures in the Discrete Mathematics course. Appreciation is also extended to fellow classmates for their encouragement, feedback, and teamwork throughout the learning journey. Finally, heartfelt thanks go to the author's family for their continuous support, patience, and understanding during the entire process

VIII. REFERENCES

REFERENCES

[1] R. Munir, "Graf (Bagian 1)," *Matematika Diskrit – Informatika ITB*, 2024. [Online]. Available: `https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf`
[2] H. Li, S. Liu, and J. Huang, "Optimization of food delivery route considering vehicle capacity and time windows," in *2018 IEEE 3rd Int. Conf. on Cloud Computing and Big Data Analytics (ICCCBDA)*, pp. 53–58, IEEE, 2018.
[3] A. Nugraha, M. Hatta, and D. R. Astuti, "Analysis of Factors Influencing GoFood Online Food Delivery Service User Loyalty in Surabaya," *J. of Management and Marketing Review*, vol. 5, no. 1, pp. 17–26, 2020.
[4] P. Toth and D. Vigo, *Vehicle Routing: Problems, Methods, and Applications*, 2nd ed., Society for Industrial and Applied Mathematics (SIAM), 2014.
[5] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, 1985.
[6] D. Bertsimas and J. N. Tsitsiklis, "A unified framework for deterministic and stochastic vehicle routing problems," *Operations Research*, vol. 39, no. 4, pp. 553–564, 1991.
[7] X. Yang, Y. Tang, and H. Yang, "Multi-objective optimization for last-mile delivery routing considering fairness and efficiency," *IEEE Access*, vol. 8, pp. 103513–103525, 2020.
[8] OpenRouteService, "OpenRouteService API Documentation," Heidelberg Institute for Geoinformation Technology, 2024. [Online]. Available: `https://openrouteservice.org/dev/#/`

STATEMENT

Hereby, I declare that this paper I have written is my own work, not an adaptation or translation of someone else's paper, and not a product of plagiarism.

Bandung, 20 July 2025

[illegible]

Bryan Pratama Putra Hendra
13524067