Traveling Salesman Problem Application for ‘Jasur Errands Service’ Courier Travel Optimization in Purwakarta City

Shulha - 13522087
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*13522087@std.stei.itb.ac.id*

***Abstract*—Traveling Salesman Problem has been one of the most discussed and a difficult problem in computer science world. Many algorithms have been developed to solve the problem as the problem represents many real-life applications. One of its applications is to find the most optimal route for an errands service courier travel as courier may receive request to visit multiple locations in an order. This paper shows how Traveling Salesman Problem application may find the shortest route to visit all requested locations, specifically using the greedy approach Nearest Neighbor Heuristic algorithm.**

***Keywords*—Errands Service, Nearest Neighbor Algorithm, Traveling Salesman Problem, Travel Optimization**

I. Introduction

The increasing dependence on technology today has significantly transformed various aspects of our lives. From utilizing transport and delivery services to purchasing goods from supermarkets and utilizing on-demand service platforms, such as Gojek and Grab, the influence of technology is pervasive, particularly among the Millennial and Gen Z demographics. As we delve into the dynamics of this technological dependence and its impact on the transportation and logistics sector, it becomes evident that the interplay between technology and consumer behavior is reshaping traditional business models.

Growing up in the capital city of Jakarta, the author was surprised when she visited her hometown in the small city of Purwakarta. Amidst the presence of major startups such as Gojek and Grab, a modest enterprise stood out, recognized for its service locally known as ‘Jasa Suruh,’ commonly referred to as ‘Jasur’. ‘Jasur’ service provides assistance in tasks such as ordering food, purchasing necessities, buying medicine at the pharmacy, and delivering goods or meals.

Fig. 1. Jasur Logo
Source: Jasur Purwakarta Facebook Page

The concept behind 'Jasur' may appear as an attempt to compete with large transportation startups. However, 'Jasur' is essentially a traditional model of an errands service. Customers can request 'Jasur' services by messaging or calling the Jasur Official WhatsApp. Unlike single-service models where customers can only order a specific service (e.g., food delivery or motorcycle taxi service), 'Jasur' offers multiple services simultaneously. For instance, a customer can ask 'Jasur' not just to buy bakso but also to deliver cake orders to neighbors and then pick up their children from school—all in a single order and with the same courier. This unique approach distinguishes 'Jasur' from the likes of giant startups such as Gojek and Grab.

Most times, 'Jasur' couriers face the challenge of making multiple stops and reaching various locations to fulfill a customer's orders. This means that courier must form a Hamilton path. In the case of delivering something from the customer and getting back again to customer like the example given above, courier must form Hamilton circuit. For the sake of efficiency, couriers may want to get the best routes to get to all the locations, with the fastest time and shortest distance.

In this paper, the author will use the graph theory and explore the traveling salesman problem application with greedy approach the Nearest Neighbor algorithm to identify optimal routes for ‘Jasur’ courier.

II. Theoretical Framework

A. Graph Definition

A graph is determined as a mathematical structure that represents a particular function by connecting a set of points. It is used to create a pairwise relationship between objects.

A graph $G = (V, E)$ consists of a nonempty set of vertices (or nodes) $V = \{v_1, v_2, \dots, v_n\}$ and a set of edges $E = \{e_1, e2, \dots, e_n\}$. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoints.

The set of vertices V of a graph G may be infinite. A graph with an infinite vertex set or an infinite number of edges is called an infinite graph, and in comparison, a graph with a finite vertex set and a finite edge set is called a *finite graph*, which is commonly used.

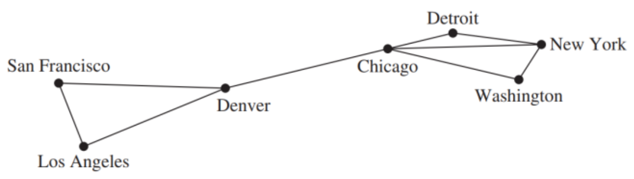


Fig. 2. Graph Example
Source: [2]

B. Graph Terminologies

To generalize understanding of graph theory, there are several terms used. The terminologies are as follows:

1. Adjacency

   Two vertices *u* and *v* in an undirected graph *G* are called adjacent (or neighbors) in *G* if *u* and *v* are endpoints of an edge *e* of *G*.

   When *(u, v)* is an edge of the graph *G* with directed edges, *u* is said to be adjacent to *v* and *v* is said to be adjacent from *u*. The vertex *u* is called the initial vertex of *(u, v)*, and *v* is called the terminal or end vertex of *(u, v)*. The initial vertex and terminal vertex of a loop are the same.
2. Incidence

   An edge *e* is called incident with the vertices *u* and *v* if *e* is said to connect *u* and *v*. Formally, edge $e = (v_j, v_k)$, it is said that *e* is incident to vertex v_j and vertex v_k.
3. Isolated Vertex

   A vertex *v* is said to be isolated if it does not have any edges incident to it.
4. Null Graph

   A graph whose set of edges is an empty set or consists solely of isolated vertices is said to be a *null graph* or *empty graph*.
5. Degree

   The degree of a vertex in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex *v* is denoted by *deg(v)*.

   In a graph with directed edges the in-degree of a vertex *v*, denoted by $deg^-(v)$, is the number of edges with *v* as their terminal vertex. The out-degree of *v*, denoted by $deg^+(v)$, is the number of edges with *v* as their initial vertex.
6. Path

   A path is a sequence of vertices in a graph where each adjacent pair of vertices is connected by an edge. A path of length n from the initial vertex v_0 to the destination vertex v_n in the graph *G* is a sequence alternating between vertices and edges in the form v_0, e_1, v_1, e_2, v_2, ..., v_{n-1}, e_n, v_n such that $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2)$, ..., $en = (vn_{-1}, vn)$ are the edges of the graph *G*.

   The length of the path is the total number of edges in the path.
7. Circuit

   A *circuit* or a *cycle* is a path that starts and ends at the same vertex. It forms a closed loop and does not have a specific start or end vertex. The length of the circuit is the total number of edges in the circuit.
8. Connectivity

   Two vertices v_1 and v_2 are said to be connected if there is a path from v_1 to v_2. *G* is said to be a connected graph if for every pair of vertices v_i and v_j in the set *V*, there exists a path from v_i to v_j. Otherwise, *G* is called a disconnected graph.

   Two vertices, *u* and *v*, in a directed graph *G* are said to be strongly connected if there is a directed path from *u* to *v* and also a directed path from *v* to *u*. If *u* and *v* are not strongly connected but are connected in their underlying undirected graph, then *u* and *v* are said to be weakly connected.
9. Subgraph

   A subgraph is a subset of set of vertices end edges of a graph. Suppose $G = (V, E)$ is a graph. $G1 = (V_1, E_1)$ is a subgraph of *G* if $V_1 \subseteq V$ and $E_1 \subseteq E$.

   The *complement of the subgraph* G_1 with respect to the graph G is the graph $G_2 = (V_2, E_2)$ such that $E_2 = E - E_1$, and V_2 is the set of vertices that are incident to the edges in E_2.

   The *connected components* of a graph are the maximum number of connected subgraphs within the graph G.

   A subgraph $G_1 = (V_1, E_1)$ of $G = (V, E)$ is called a *spanning subgraph* if $V_1 = V$ (meaning G_1 contains all the vertices of G).
10. Weighted Graph

    A weighted graph is a graph in which each of its edges is assigned a value (weight). In a weighted graph, the emphasis is on quantifying the relationships between vertices, providing additional information beyond mere connectivity.

C. Types of Graphs

Based on the presence or absence of loops or multiple edges in a graph, graphs are classified into two types:

1. Simple graph

   A simple graph is a graph that does not contain loops or multiple edges. It means that no two edges connect the same pair of vertices.
2. Unsimple graph

   A graph that contains multiple edges or loops is called an unsimple graph. Unsimple graphs further divided to two categories:
   a. Multigraph

      A multigraph has multiple edges connected to the same vertices. When a single unordered pair of vertices *{u, v}* is connected by *m* distinct edges, then *{u, v}* is an edge with a multiplicity of *m*.
   b. Pseudograph

      A pseudograph has edges that connect a vertex to itself, these edges called loops.

Based on the orientation of edges, graphs are categorized into two types:

1. Undirected Graph

   An undirected graph is a graph in which edges do not have a specific direction

2. Directed Graph

   A directed graph or diagraph is a graph in which each edge is given a specific direction. A directed graph (V, E) consists of a nonempty set of vertices V and a set of directed edges E. Each directed edge is associated with an ordered pair of vertices. Edge with the ordered pair $\{u, v\}$ is an edge that starts at u and ends at v.

There are also some special types of graphs such as the following:

1. Complete Graph

   A complete graph on n vertices, denoted by K_n, is a simple graph that contains exactly one edge between each pair of distinct vertices. A simple graph for which there is at least one pair of distinct vertex not connected by an edge is called noncomplete. The number of edges in a complete graph consisting of n vertices is $n(n - 1)/2$.
2. Cycles (Circle Graph)

   A cycle C_n, $n \geq 3$, consists of n vertices $v_1, v_2, ..., v_n$ and edges $\{v_1, v_2\}$, $\{v_2, v_3\}$, ..., $\{v_{n-1}, v_n\}$, and $\{v_n, v_1\}$. Each vertex in a cycle has a degree of two.
3. Regular Graph

   A graph in which every vertex has the same degree is called a regular graph. If the degree of each vertex is denoted as r, then the graph is referred to as a regular graph of degree r. The total number of edges in a regular graph is $nr/2$.

D. Graph Representation

Graphs are often visualized with circles (nodes/vertices) and lines (edges), however a method is needed to store graphs in the memory of a computer. That is called graph representation. There are several ways to represent a graph:

1. Adjacency List

   One way to represent a graph without multiple edges is to list all the edges of this graph. Another way to represent a graph with no multiple edges is to use adjacency lists, which specify the vertices that are adjacent to each vertex of the graph. Representing a directed graph can be by listing the initial vertex and their terminal vertices.
2. Incidence Matrix

   Another common way to represent graphs is to use incidence matrices. Let $G = (V, E)$ be an undirected graph. Suppose that $v_1, v_2, ..., v_n$ are the vertices and $e_1, e_2, ..., e_m$ are the edges of G. Then the incidence matrix with respect to this ordering of V and E is the $n \times m$ matrix $M = [m_{ij}]$, where $m_{ij} = 1$ when edge e_j is incident with v_i, 0 otherwise.
3. Adjacency Matrix

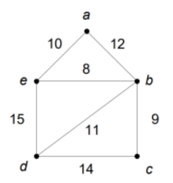
   Another representation of graph which also the author will use in this paper is adjacency matrix.

   The matrix for a directed graph $G = (V, E)$ has a 1 in its (i, j)th position if there is an edge from v_i to v_j, where $v_1, v_2, ..., v_n$ is an arbitrary listing of the vertices of the directed graph. In other words, if $A = [a_{ij}]$ is the adjacency matrix for the directed graph with respect to this listing of the vertices, then $a_{ij} = 1$ if (v_i, v_j) is an edge of G, 0 otherwise.

   The adjacency matrix for a directed graph does not have to be symmetric, because there may not be an edge from v_j to v_i when there is an edge from v_i to v_j.

   Adjacency matrices can also be used to represent directed multigraphs. Again, such matrices are not zero–one matrices when there are multiple edges in the same direction connecting two vertices. In the adjacency matrix for a directed multigraph, aij equals the number of edges that are associated to (v_i, v_j).

   Adjacency matrices can also be used to represent weighted graphs. For adjacency matrix A, then $a_{ij} = x$ if (v_i, v_j) is an edge of G, and x is the corresponding weight of (v_i, v_j).



$$\begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & \infty & 12 & \infty & \infty & 10 \\ b & 12 & \infty & 9 & 11 & 8 \\ c & \infty & 9 & \infty & 14 & \infty \\ d & \infty & 11 & 14 & \infty & 15 \\ e & 10 & 8 & \infty & 15 & \infty \end{array}$$

Fig. 3. Adjacency Matrix Representation for Weighted Graph
Source: [2]

E. Hamilton Path and Circuit

This terminology comes from a game, called the Icosian puzzle, invented in 1857 by the Irish mathematician Sir William Rowan Hamilton.

A simple path in a graph G that passes through every vertex exactly once is called a Hamilton path, and a simple circuit in a graph G that passes through every vertex exactly once is called a Hamilton circuit. That is, the simple path $x_0, x_1, ..., x_{n-1}, x_n$ in the graph $G = (V, E)$ is a Hamilton path if $V = \{x_0, x_1, ..., x_{n-1}, x_n\}$ and $x_i \neq x_j$ for $0 \leq i \leq j \leq n$, and the simple circuit $x_0, x_1, ..., x_{n-1}, x_n$ is a Hamilton circuit if $x_0, x_1, ..., x_{n-1}, x_n$ is a Hamilton path.

If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least $n/2$, then G has a Hamilton circuit.

If G is a simple graph with n vertices with $n \geq 3$ such that $deg(u) + deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G, then G has a Hamilton circuit.

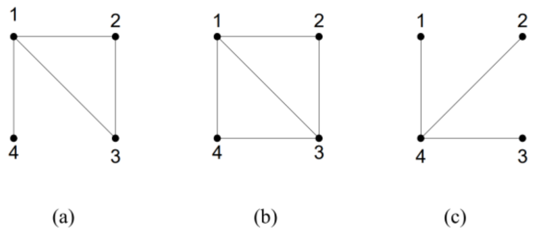


Fig. 4. (a) A graph has Hamilton path (b) A graph has Hamilton path (c) A graph with no Hamilton Path
Source: [5]

F. Traveling Salesman Problem

The traveling salesman problem or TSP is one of the famous and difficult computer science problems. This was first formulated in1930 by Karl Menger and since then it became one ofthe most studied problems in optimization. The TSP asks for the shortest route a traveling salesperson should take to visit a set of cities. This problem reduces to finding a Hamilton circuit in a complete graph such that the total weight of its edges is as small as possible. However, the TSP application is not limited to complete graph, and can be implemented to other form of graph as long as it has Hamilton circuit. The TSP has many applications such as logistics, vehicle routing, and circuit design.

The graph in TSP is represented as a weighted graph. In a complete graph consisting of *n* vertices, there are $(n-1)!/2$ Hamiltonian circuits, which means there are $(n-1)!/2$ different routes to visit each city exactly once and return to the starting city. Therefore, it is almost impossible to list down all different Hamiltonian circuits manually for many vertices.

Computer scientists developed many approaches and algorithms to solve this problem. While none of these are proven to be the optimal and perfect solution to the problem, they are a *good* heuristic approach to the solve the TSP problem. Some of the algorithms are explained below.

1. Nearest Neighbor

   This algorithm is a straightforward heuristic approach to address TSP. It initiates by randomly selecting a city, designating it as the starting point (*n0*). Subsequently, it identifies the nearest unvisited city and moves to that location, marking the current city as visited. This process repeats until all cities have been visited. Once all cities are visited, the algorithm returns to the starting city, completing the route. The solution involves $O(N^2 \log(N))$ iterations, where N represents the number of cities to be visited. Notably, the nearest neighbor heuristic ensures that the solution remains within 25% of the Held-Karp lower bound.
2. Genetic Algorithm

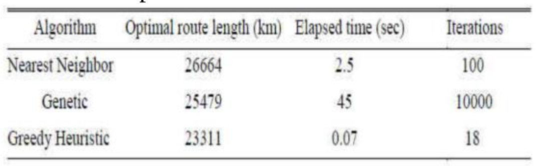
   The algorithm calculates the fitness function measuring quality of tour by using total distance for each member (tour) of the population. Then it creates new individuals in the population. It uses mutation to add randomization to the process. Finally, it selects the individual (fittest tour solution) with the higher fitness function. Applying genetic algorithm to TSP requires certain limitations. Each city should not be repeated, and only valid routes are considered. This algorithm can handle large number of vertices and explore various possible routes, however applying this algorithm is often considered expensive and may not always find the optimal solution.
3. Greedy Heuristic Algorithm

   Belonging to the category of heuristic algorithms, this algorithm seeks local optima to optimize the best local solution for finding global optima. The process involves sorting all edges and selecting the edge with the minimum cost, continually choosing the best next options without forming loops. The computational complexity is $O(N^2 \log(N))$, and while there is no guarantee of a global optimum solution, the greedy algorithm terminates in a reasonable number of steps, maintaining the solution within 15-20% of the Held-Karp lower bound.
4. Branch and Bound Algorithm

   The branch-and-bound algorithm for the traveling salesman problem uses a branch-and-bound tree. Every node in the branch-and-bound tree has a node number, a label (representing the decision made at that node either to take or not to take a specific link from one city to another), a bound (giving a lower limit on the possible lengths of circuits below that node in the tree), an incoming matrix, and an opportunity matrix.

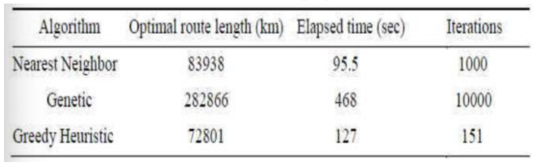
   Every matrix (both incoming matrices and opportunity matrices) has an associated “L-value,” which could be called the “deduction total.” As the algorithm progresses, distances in the matrix are decreased, and the L-value keeps track of the total amount we have subtracted from distances since the original distance matrix.

Reference [3] shows the comparison of the heuristic algorithms in performing TSP. In this case, although the Greedy Heuristic consumes more iterations to solve the TSP, its result is the closest to the optimum solution.



Algorithm	Optimal route length (km)	Elapsed time (sec)	Iterations
Nearest Neighbor	26664	2.5	100
Genetic	25479	45	10000
Greedy Heuristic	23311	0.07	18

Fig. 5. TSP Algorithm Comparison of 100 Cities
Source: [6]



Algorithm	Optimal route length (km)	Elapsed time (sec)	Iterations
Nearest Neighbor	83938	95.5	1000
Genetic	282866	468	10000
Greedy Heuristic	72801	127	151

Fig. 6. TSP Algorithm Comparison of 1000 Cities
Source: [6]

III. Discussion

A. Scope of Discussion

In this paper, the author uses the map of the center of Purwakarta City as reference to list down some of the locations that are commonly visited in the city. In this paper, we will use ten locations listed down in the table below.

Table I
Location Data

Vertex	**Location**
1	Purwakarta Station
2	‘Mie Gacoan’
3	Yogya Department Store
4	Pasar Rebo Market
5	‘Baso Cepot’
6	STS Sadang Square
7	Al Ghazali Primary School
8	Sambal Hejo SHSD Restaurant
9	Sadang Sari Housing Complex
10	R.E. Martadinata Road (House)

By performing TSP application, author will analyze the best route for ‘Jasur’ courier to visit all the locations needed. In this paper, author will implement Nearest Neighbor algorithm to solve the traveling salesman problem.

B. Limitations

In performing TSP application to the problem, there are very few factors. In this paper, the author set some limitations to make application simpler. Limitations of the discussions are listed below:

1. Weight of the route of two locations only calculated by distance of those two locations (distance from Google Maps), set aside other factors like traffic. Assuming that the shortest route is the quickest route.
2. Assume all ‘Jasur’ couriers use motorcycles.
3. Assume all customers use the services in a logical manner.
4. Assume ‘Jasur’ courier doesn’t have to perform a task before another. It means courier has no obligation to visit one location before other locations and can freely choose what task the courier wants to perform first.
5. Assume the courier is in a certain location when get the order, and it is set to be the starting location that needs to be chosen.
6. If the final location needs to be selected, it will be fixed point and is not much considered in TSP calculation.

C. Graph Representation

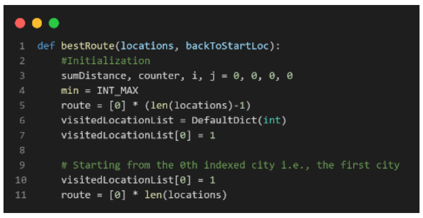
The author uses adjacency matrix to represent the graph of Purwakarta city locations. The distance (in kilometers) will be the weight of the edge connecting two vertices (locations). The graph is represented in Fig. 7 below.

$$\begin{bmatrix}
\infty & 1.8 & 0.5 & 1.4 & 1.2 & 6.1 & 1.6 & 2.9 & 6 & 0.85 \\
2.1 & \infty & 1.7 & 3.4 & 3.3 & 4.8 & 1 & 4.3 & 8.2 & 2.1 \\
0.45 & 1.3 & \infty & 1.7 & 1.6 & 5.6 & 1.1 & 3 & 5.5 & 1 \\
1.5 & 2.8 & 1.5 & \infty & 0.6 & 7.1 & 2.7 & 2.2 & 7.1 & 0.6 \\
1.3 & 3 & 1.8 & 0.2 & \infty & 7.4 & 2.9 & 2.5 & 7.3 & 0.85 \\
6.4 & 4.8 & 5.9 & 7.6 & 7.6 & \infty & 5.3 & 8.6 & 0.35 & 6.9 \\
1.6 & 1 & 1.2 & 2.9 & 2.8 & 5.3 & \infty & 3.4 & 5.2 & 2.1 \\
3.3 & 3.9 & 3 & 2.8 & 2.7 & 8.3 & 3.4 & \infty & 8.2 & 2.1 \\
6 & 8.2 & 5.6 & 7.3 & 7.2 & 0.35 & 5.3 & 8.2 & \infty & 6.5 \\
0.85 & 2.1 & 0.95 & 1.1 & 1 & 6.5 & 2 & 2.1 & 6.4 & \infty
\end{bmatrix}$$

Fig. 7. Adjacency Matrix Representation of Locations Graph
Source: Primary

D. Nearest Neighbor Greedy Approach TSP Algorithm

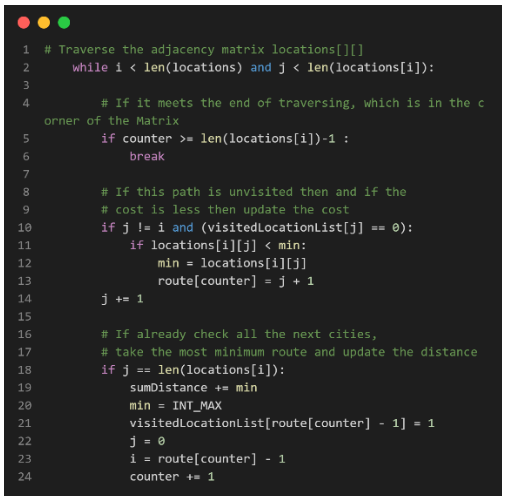
The algorithm used in this paper to find the best route for ‘Jasur’ courier is adapted from the greedy approach, Nearest Neighbor Algorithm to solve TSP. In addressing the 'Jasur' problem, there are two functions designed to manage orders with a predetermined final vertex and those without. Despite this distinction, the overall algorithmic flow remains remarkably similar.



```
def bestRoute(locations, backToStartLoc):
    #Initialization
    sumDistance, counter, i, j = 0, 0, 0, 0
    min = INT_MAX
    route = [0] * (len(locations)-1)
    visitedLocationList = DefaultDict(int)
    visitedLocationList[0] = 1

    # Starting from the 0th indexed city i.e., the first city
    visitedLocationList[0] = 1
    route = [0] * len(locations)
```

Fig. 8. Initialization of TSP Algorithm
Source: Primary

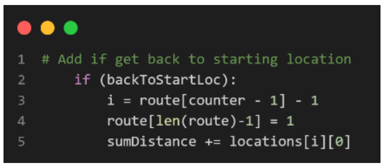
The algorithm takes a graph of *locations* as a parameter, along with a boolean variable, *backToStartLoc*, indicating whether the courier needs to return to the starting location. It utilizes two lists: visitedLocationList to track already visited locations and route to enumerate the steps indicating the sequence of locations to be visited.



```
# Traverse the adjacency matrix locations[][]
    while i < len(locations) and j < len(locations[i]):

        # If it meets the end of traversing, which is in the c
orner of the Matrix
        if counter >= len(locations[i])-1 :
            break

        # If this path is unvisited then and if the
        # cost is less then update the cost
        if j != i and (visitedLocationList[j] == 0):
            if locations[i][j] < min:
                min = locations[i][j]
                route[counter] = j + 1
        j += 1

        # If already check all the next cities,
        # take the most minimum route and update the distance
        if j == len(locations[i]):
            sumDistance += min
            min = INT_MAX
            visitedLocationList[route[counter] - 1] = 1
            j = 0
            i = route[counter] - 1
            counter += 1
```

Fig. 9. Nearest Neighbor TSP Main Algorithm
Source: Primary

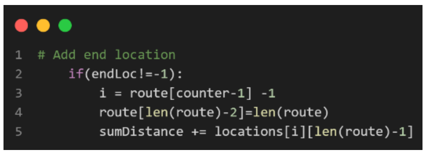
The algorithm selects the nearest location from the starting point by traversing the list of *locations* in the graph. Upon identifying the nearest location, the distance is added to the *sumDistance*, and that location is then set as the current location to initiate the search for the next nearest location. This loop process continues until all locations have been visited.



```
# Add if get back to starting location
    if (backToStartLoc):
        i = route[counter - 1] - 1
        route[len(route)-1] = 1
        sumDistance += locations[i][0]
```

Fig. 10. Go Back to Start Location Option Algorithm
Source: Primary

If the courier needs to return to the starting location, the algorithm will designate the last location in the *route* as the starting location.



```
# Add end location
    if(endLoc!=-1):
        i = route[counter-1] -1
        route[len(route)-2]=len(route)
        sumDistance += locations[i][len(route)-1]
```

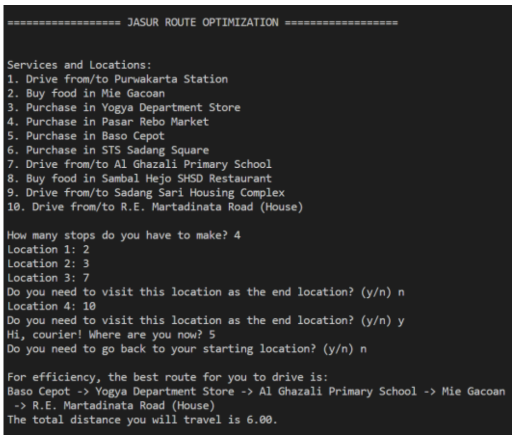
Fig. 11. Final Location Option Algorithm
Source: Primary

In cases where the courier needs to reach a particular location at the end of the travel, the algorithm will assign the second-to-last location in the route to this specific destination.

E. Greedy Approach TSP Experiment in 'Jasur' Route Optimization

Let's apply our algorithm to evaluate two 'Jasur' route optimization scenarios.

In the first case, a courier receives orders to purchase 'Mie Gacoan', buy household supplies from Yogya Department Store, and deliver them to a customer's house on R. E. Martadinata Road. Additionally, the courier is tasked with picking up a customer's child from Al Ghazali Primary School and bringing them home. The courier is initially at 'Baso Cepot' after previous work, and he doesn't need to go back to 'Baso Cepot' again.



```
================= JASUR ROUTE OPTIMIZATION =================


Services and Locations:
1. Drive from/to Purwakarta Station
2. Buy food in Mie Gacoan
3. Purchase in Yogya Department Store
4. Purchase in Pasar Rebo Market
5. Purchase in Baso Cepot
6. Purchase in STS Sadang Square
7. Drive from/to Al Ghazali Primary School
8. Buy food in Sambal Hejo SHSD Restaurant
9. Drive from/to Sadang Sari Housing Complex
10. Drive from/to R.E. Martadinata Road (House)

How many stops do you have to make? 4
Location 1: 2
Location 2: 3
Location 3: 7
Do you need to visit this location as the end location? (y/n) n
Location 4: 10
Do you need to visit this location as the end location? (y/n) y
Hi, courier! Where are you now? 5
Do you need to go back to your starting location? (y/n) n

For efficiency, the best route for you to drive is:
Baso Cepot -> Yogya Department Store -> Al Ghazali Primary School -> Mie Gacoan
 -> R.E. Martadinata Road (House)
The total distance you will travel is 6.00.
```

Fig. 12. First Case of 'Jasur' Route Optimization
Source: Primary

For optimal efficiency, the algorithm recommends the courier to first purchase supplies at Yogya Department Store, followed by picking up the child from school, then buying 'Mie Gacoan,' and finally delivering all items to the house on R. E. Martadinata Road. The total distance traveled is 6 km.

$$\begin{array}{c} 5 \\ 2 \\ 3 \\ 7 \\ 10 \end{array}\begin{bmatrix} \infty & 3 & 1.8 & 2.9 & 0.85 \\ 3.3 & \infty & 1.71 & 1 & 2.1 \\ 1.6 & 1.3 & \infty & 1.1 & 1 \\ 2.8 & 1 & 1.2 & \infty & 2.1 \\ 1 & 2.1 & 0.95 & 2 & \infty \end{bmatrix}$$

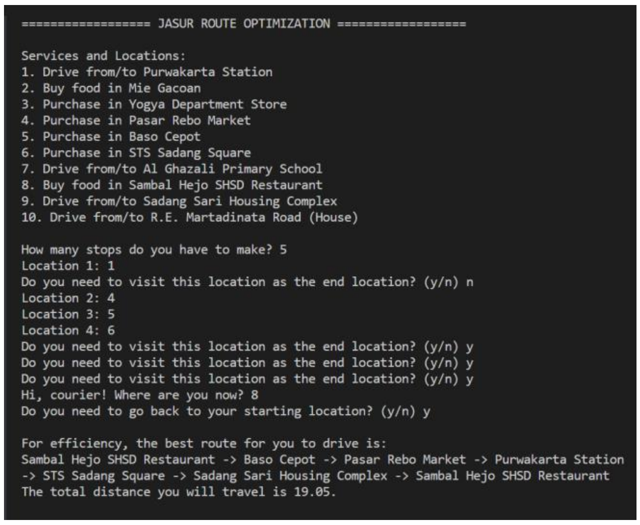
Fig. 0. Filtered Locations Graph of Forst Case
Source: Primary

If we were to employ a brute-force approach to identify the shortest Hamiltonian path, we would need to explore 3! = 6 possible routes. The author reordered the locations manually, and the resulting data is presented in Table II. The comparison reveals the algorithm's accuracy when contrasted with the manually generated brute-force ordering.

Table II
First Case of 'Jasur' Route Possibility

Route	**Distance**
5 – 2 – 3 – 7 – 10	7.9 km
5 – 2 – 7 – 3 – 10	6.2 km
5 – 3 – 2 – 7 – 10	6.1 km
5 – 3 – 7 – 2 – 10	*6.0 km*
5 – 7 – 2 – 3 – 10	6.6 km
5 – 7 – 3 – 2 – 10	7.5 km

In the second case, a courier receives orders while he is in front of SHSD Sambal Hejo Sambal Dadak Restaurant in Ciganea. He needs to pick up the customer in Purwakarta Station and take him to his house in Sadang Sari Housing Complex. The customer wants to make a surprise home so he wants to buy many things before getting home. He needs to buy some food in STS Sadang Square, 'Baso Cepot', and buy some new cloth in Pasar Rebo Market before taking the customer home. He needs to go back to SHSD Restaurant because he has work there after doing errands.



```
================= JASUR ROUTE OPTIMIZATION =================

Services and Locations:
1. Drive from/to Purwakarta Station
2. Buy food in Mie Gacoan
3. Purchase in Yogya Department Store
4. Purchase in Pasar Rebo Market
5. Purchase in Baso Cepot
6. Purchase in STS Sadang Square
7. Drive from/to Al Ghazali Primary School
8. Buy food in Sambal Hejo SHSD Restaurant
9. Drive from/to Sadang Sari Housing Complex
10. Drive from/to R.E. Martadinata Road (House)

How many stops do you have to make? 5
Location 1: 1
Do you need to visit this location as the end location? (y/n) n
Location 2: 4
Location 3: 5
Location 4: 6
Do you need to visit this location as the end location? (y/n) y
Do you need to visit this location as the end location? (y/n) y
Do you need to visit this location as the end location? (y/n) y
Hi, courier! Where are you now? 8
Do you need to go back to your starting location? (y/n) y

For efficiency, the best route for you to drive is:
Sambal Hejo SHSD Restaurant -> Baso Cepot -> Pasar Rebo Market -> Purwakarta Station
-> STS Sadang Square -> Sadang Sari Housing Complex -> Sambal Hejo SHSD Restaurant
The total distance you will travel is 19.05.
```

Fig. 13. Second Case of 'Jasur' Route Optimization
Source: Primary

For optimal efficiency, the algorithm recommends the courier to first buy 'Baso Cepot', followed by purchasing new cloth in Pasar Rebo Market. The courier then picks up the customer from Purwakarta Station, taking him to his house in Sadang Sari Housing Complex while making one stop in STS Sadang Square to buy some food. Finally, he needs to get back to SHSD Restaurant. The total distance traveled is 19.05 km.

$$\begin{array}{c} 8 \\ 1 \\ 4 \\ 5 \\ 6 \\ 9 \end{array}\begin{bmatrix} \infty & 3.3 & 2.8 & 2.7 & 8.3 & 8.2 \\ 2.9 & \infty & 1.4 & 1.2 & 6.1 & 6 \\ 2.2 & 1.5 & \infty & 0.6 & 7.1 & 7.1 \\ 2.5 & 1.3 & 0.2 & \infty & 7.4 & 7.3 \\ 8.6 & 6.4 & 7.6 & 7.6 & \infty & 0.35 \\ 8.2 & 6 & 7.3 & 7.2 & 0.35 & \infty \end{bmatrix}$$

Fig. 14. Filtered Locations Graph of Second Case
Source: Primary

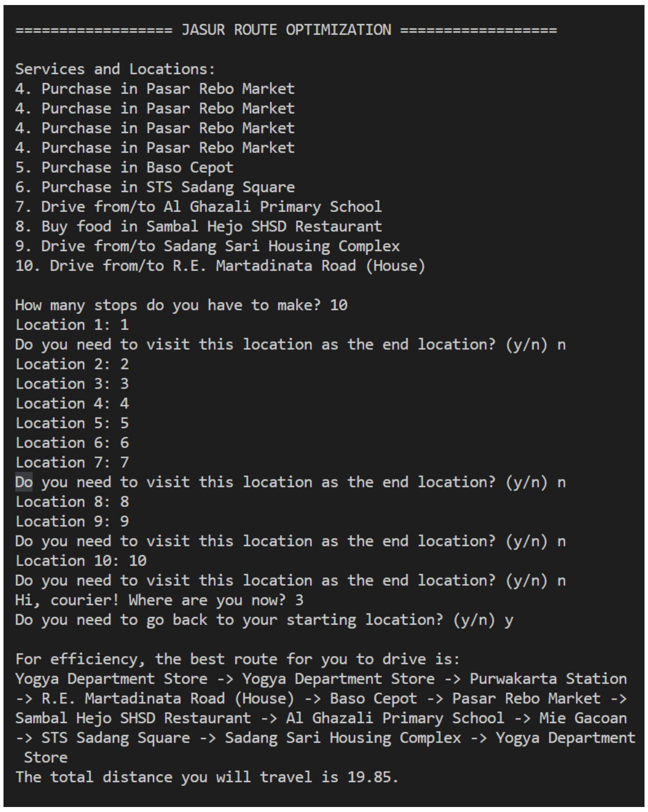
If we use brute force to find the shortest Hamilton path, we will have to find 4! = 24 possible routes. By reordering the location manually, the author gets the data in Table III. It shows our algorithm result match the result from manual ordering.

Table III
Second Case of 'Jasur' Route Possibility

Route	**Distance**
8 – 1 – 4 – 5 – 6 – 9– 8	21.25 km
8 – 1 – 4 – 6 – 5 – 9– 8	34.90 km
8 – 1 – 5 – 4 – 6 – 9– 8	20.38 km
8 – 1 – 5– 6 – 4 – 9– 8	34.80 km
8 – 1 – 6 – 4 – 5 – 9– 8	33.10 km
8 – 1 – 6 – 5 – 4 – 9– 8	32.50 km

8 – 4 – 1 – 5 – 6 – 9– 8	21.45 km
8 – 4 – 1 – 6 – 5 – 9– 8	33.50 km
8 – 4 – 5 – 1 – 6 – 9– 8	19.35 km
8 – 4 – 5 – 6 – 1 – 9– 8	31.40 km
8 – 4 – 6 – 1 – 5 – 9– 8	33.00 km
8 – 4 – 6 – 5 – 1 – 9– 8	33.00 km
8 – 5 – 1 – 4 – 6 – 9– 8	21.05 km
8 – 5 – 1 – 6 – 4 – 9– 8	33.00 km
8 – 5 – 4 – 1 – 6 – 9– 8	*19.05 km*
8 – 5 – 4 – 6 – 1 – 9– 8	30.60 km
8 – 5 – 6 – 1 – 4 – 9– 8	33.20 km
8 – 5 – 6 – 4 – 1 – 9– 8	33.40 km
8 – 6 – 1 – 4 – 5 – 9– 8	28.70 km
8 – 6 – 1 – 5 – 4 – 9– 8	27.90 km
8 – 6 – 4 – 1 – 5 – 9– 8	34.10 km
8 – 6 – 4 – 5 – 1 – 9– 8	32.00 km
8 – 6 – 5 – 1 – 4 – 9– 8	33.90 km
8 – 6 – 5 – 4 – 1 – 9– 8	31.80 km

Although it will most likely not to be a case for ‘Jasur’ route optimization problem, the author tries to solve the original traveling salesman problem where salesman need to visit all location in the city. Applying the algorithm to all ten locations in Purwakarta City, the author gets that one can travel to all ten locations only with 19.85 km distance by starting at Yogya Department Store.



```
================= JASUR ROUTE OPTIMIZATION =================

Services and Locations:
4. Purchase in Pasar Rebo Market
4. Purchase in Pasar Rebo Market
4. Purchase in Pasar Rebo Market
4. Purchase in Pasar Rebo Market
5. Purchase in Baso Cepot
6. Purchase in STS Sadang Square
7. Drive from/to Al Ghazali Primary School
8. Buy food in Sambal Hejo SHSD Restaurant
9. Drive from/to Sadang Sari Housing Complex
10. Drive from/to R.E. Martadinata Road (House)

How many stops do you have to make? 10
Location 1: 1
Do you need to visit this location as the end location? (y/n) n
Location 2: 2
Location 3: 3
Location 4: 4
Location 5: 5
Location 6: 6
Location 7: 7
Do you need to visit this location as the end location? (y/n) n
Location 8: 8
Location 9: 9
Do you need to visit this location as the end location? (y/n) n
Location 10: 10
Do you need to visit this location as the end location? (y/n) n
Hi, courier! Where are you now? 3
Do you need to go back to your starting location? (y/n) y

For efficiency, the best route for you to drive is:
Yogya Department Store -> Yogya Department Store -> Purwakarta Station
-> R.E. Martadinata Road (House) -> Baso Cepot -> Pasar Rebo Market ->
Sambal Hejo SHSD Restaurant -> Al Ghazali Primary School -> Mie Gacoan
-> STS Sadang Square -> Sadang Sari Housing Complex -> Yogya Department
 Store
The total distance you will travel is 19.85.
```

Fig. 14. Finding the Shortest Hamilton Circuit
Source: Primary

\

Table IV
Finding Shortest Hamilton Circuit

Starting Location	**Distance**
Purwakarta Station	20.45 km
‘Mie Gacoan’	20.25 km
Yogya Department Store	*19.05 km*
Pasar Rebo Market	23.35 km
‘Baso Cepot’	24.70 km
STS Sadang Square	21.35 km
Al Ghazali Primary School	21.25 km
Sambal Hejo SHSD Restaurant	24.70 km
Sadang Sari Housing Complex	20.25 km
R.E. Martadinata Road (House)	24.60 km

Fortunately, the outcomes of both cases align with the manually ordered result. However, it's noteworthy that the algorithmic results may not always be the most optimal, given that the Nearest Neighbor Algorithm operates as a greedy approach for solving the traveling salesman problem. This approach focuses on selecting the best option at each step without considering potential superior route possibilities in the subsequent steps.

As the number of locations visited increases, manual ordering becomes impractical for humans. Even computers, when tasked with manual ordering, face the challenge of dealing with O(n!) complexity to obtain the most optimal routes. Hence, algorithms like this are developed to provide efficient solutions to such problems.

V. Conclusion

Based on the discussion and experimentation with the provided cases, it can be concluded that the Traveling Salesman Problem algorithm proves beneficial in determining the most optimal route for the 'Jasur' errands service with multiple stops. This optimization aids couriers in fulfilling customer requests more swiftly and cost-effectively.

Given the constraints of limited time, capabilities, and resources, this paper has inherent limitations and a narrow scope, leaving ample room for enhancements and further development. Exploring alternative TSP approaches, such as the Greedy Heuristic Algorithm, Branch and Bound Algorithm, or other modern techniques, may yield more optimal route suggestions for a larger number of locations or vertices. Additionally, addressing the problem in real-time with considerations for factors like traffic, market crowd, or congestion could enhance the overall solution.

VI. Appendix

The algorithm used in this paper can be accessed through https://github.com/shulhajws/JasurRouteOptimization.git

VII. Acknowledgment

The author expresses gratitude to Allah SWT for His blessings, enabling the completion of this paper. The author would also like to especially thank the lecturer of the IF2120 Discrete Mathematics course, Mrs. Fariska Zakhralativa

Ruskanda S.T., M.T, for her inspiring and dedicated teaching. Special thanks to Dr. Ir. Rinaldi Munir, M.T., and all lecturers of IF2120 for providing plenty sources on Discrete Mathematics courses as this paper would not have been possible without them.

The author acknowledges the contributions of all those involved in the creation of this paper, especially to senior students and the owners of the referenced materials mentioned, as their assistance was crucial for the completion of this paper. The author acknowledges the presence of imperfections in this work and welcomes suggestions and critiques for future improvement.

REFERENCES

[1] B. Kell, "Branch-and-bound algorithm for the traveling salesman problem, " *21-257 Models and Methods For Optimization Carnegie Mellon University*, 2014. [Online]. Available: https://www.math.cmu.edu/~bkell/21257-2014f/tsp.pdf. Accessed on December 9, 2023.

[2] K. H. Rosen, Discrete Mathematics and Its Applications Seventh Edition. New York, America: McGraw-Hill, 2017.

[3] R. Munir, "Graf Bagian 1," *IF2120 Matematika Diskrit*, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf. Accessed on December 9, 2023.

[4] R. Munir, "Graf Bagian 2," *IF2120 Matematika Diskrit*, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/20-Graf-Bagian2-2023.pdf. Accessed on December 9, 2023.

[5] R. Munir, "Graf Bagian 3 *IF2120 Matematika Diskrit*, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/21-Graf-Bagian3-2023.pdf. Accessed on December 9, 2023.

[6] H. Abdulkarim and I. F. Alshammari, "Comparison of Algorithms for Solving Traveling Salesman Problem," *International Journal of Engineering and Advanced Technology*, August 2015, vol. 4 no. 6. Available: https://www.researchgate.net/publication/280597707_Comparison_of_Algorithms_for_Solving_Traveling_Salesman_Problem. Accessed on December 10, 2023.

[7] "Travelling Salesman Problem (Greedy Approach)," *GeeksforGeeks*, January 2022. [Online]. Available: https://www.geeksforgeeks.org/travelling-salesman-problem-greedy-approach/. Accessed on December 10, 2023.

STATEMENT

I hereby declare that the paper I have written is my own work, not a reproduction or translation of someone else's paper, and is not plagiarized.

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, December 11th, 2023

[illegible]

13522087 Shulha