Application of Travelling Salesman Problem for Planning Ice Cream Material Distribution Routes to Mixue Branches in Bandung City

Benardo - 13522055[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522055@std.stei.itb.ac.id

***Abstract*—Mixue is a well-known Chinese brand, famous for its drinks and ice cream. It began as a small, shaved ice shop in Zhengzhou, Henan, and has since expanded significantly, now boasting 60 stores in Bandung. Mixue ensures fresh and high-quality products by centrally supplying ingredients to each store. To minimize delivery costs, it is crucial for the company to find the shortest and most efficient delivery routes. This paper explores methods to optimize ingredient delivery to the Bandung stores. The objective is to reduce expenses by identifying the quickest and most straightforward delivery paths. The study employs a strategy known as the Traveling Salesman Problem (TSP), which is solved using dynamic programming. This approach is used to determine the most effective delivery routes. Such meticulous planning is essential to ensure that all Mixue stores receive their supplies promptly and cost-effectively.**

***Keywords*—Graf, Mixue, Most Effective Delivery Routes, Travelling Salesman Problem**

I. INTRODUCTION

Mixue, one of the world's largest ice cream franchise companies, first established its roots in Zhengzhou, Henan, China on June 16, 1997. Founded by the Zhang brothers, Mixue initially focused on selling ice cream. However, the soaring popularity of their ice cream led to the decision to start a franchise business. Today, Mixue boasts a remarkable presence with 21,581 outlets operating across China and 12 other countries in the Asia-Pacific region.

In Indonesia, Mixue made its debut in 2020, opening its first store in Cihampelas Walk, Bandung. Since then, the company has expanded rapidly, now operating over a thousand outlets throughout Indonesia. Bandung alone is home to around 60 Mixue branches. To support this growth, Mixue has developed a central warehousing and logistics hub, aiming to minimize production chain costs. The company's primary focus has shifted from just selling ice cream to supplying raw materials, packaging, and processing machines to its franchises, essentially functioning as a supply chain company.

Ensuring the daily delivery of raw materials from the warehouse to various Mixue branches is a critical operation. With numerous branches in Bandung, distribution is organized among several trucks, each covering different delivery locations. The selection of delivery routes plays a vital role in this process, as choosing the shortest and most efficient routes minimizes distance, time, and fuel consumption.

In this paper, we will explore how Mixue can maximize its profits by minimizing distribution costs. This will involve optimizing delivery routes to determine the shortest possible paths for the trucks. We will model the distances between the warehouse and each Mixue branch as nodes in a weighted graph, connected by edges. The paper will focus on applying the Travelling Salesman Problem approach to optimize these delivery routes.

II. BASIC THEORY

A. Graph Definition

A graph is defined as a discrete structure consisting of a collection of vertices (nodes) connected through a set of edges. Vertices represent individual points or elements within the graph, and they may or may not be connected to each other. In graphical representation, vertices are often denoted by dots or circles. On the other hand, edges are the connections or links between two vertices and are represented as lines connecting the corresponding dots or circles. There are also edges that start and end at the same vertex, known as loops. Graph is represented in the form $G = (V, E)$, where G represents the graph, V is a non-empty set of vertices such as $v1, v2, \ldots, vn$, and E is a set of edges like $e1, e2, \ldots, en$, which connect pairs of vertices within the graph.

B. Types of Graphs

Based on the presence and presence and absence of multiple and loop edges connecting the same vertices, the graph has two types:

1. Simple graphs
   A simple graph is characterized by having precisely one edge connecting any pair of vertices, and it does not include multiple edges between the same pair of vertices or self-connections (loops).

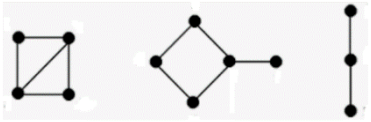

Fig. 1. Simple Graphs Example (Source:[1])

2. Unsimple graphs
   An unsimple graph refers to a type of graph in which loops, which are edges that connect a vertex to itself, or multiple edges connecting the same pair of vertices are present. Unsimple graphs can be categorized into two subcategories:
   a. Multi-graph
      A multigraph is a type of graph characterized by the presence of multiple edges that connect the same pair of vertices. In other words, a multigraph can have multiple edges associated with the same unordered pair of vertices $\{u, v\}$, and the number of these edges is known as the multiplicity of the edge $\{u, v\}$.
   b. Pseudo-graph
      A pseudograph is a type of graph that can include loops, which are edges connecting a vertex to itself, and it may also have multiple edges connecting the same pair of vertices or a vertex to itself.

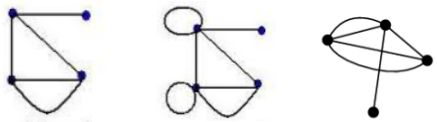

Fig. 2. Unsimple Graphs Example (Source : [1])

Based on the direction of the edges, graphs can be divided into two types:

1. Undirected graph
   Undirected graph is a graph that does not have any direction on its edges

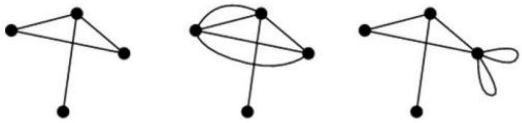

Fig. 3. Undirected Graphs Example (Source: [1])

2. Directed graph
   A directed graph is a collection of vertices connected by edges, where the edges have a specific directionality, indicating the path from one vertex to another. In a directed graph, each edge is characterized as an ordered pair of vertices. This means that each edge has a direction indicated by its starting and ending points. For example, an edge represented by the ordered pair $\{u, v\}$ is understood to be an edge that starts at vertex u and ends at vertex v.

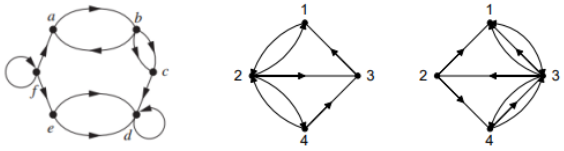

Fig. 4. Directed Graphs Example (Soruce: [2])

C. Figures and Tables

In graph theory, there is some terminologies that is used when analyzing graphs:

1. Adjacent
   In the realm of graph theory, adjacency in an undirected graph implies that two vertices are connected by an edge. Conversely, in a directed graph, adjacency is specifically defined: the vertex from where an edge originates is adjacent to the vertex where the edge terminates. For an edge (u, v) in a graph G, vertex u is known as the starting or initial vertex, while vertex v is referred to as the terminal or end vertex. Notably, in cases where an edge loops back to its origin, the initial and terminal vertices are identical.
2. Incidence
   In the context of edges and vertices, an edge (u, v) that links vertices u and v is said to be incident with these vertices. This term describes the relationship between vertices and the edges that connect them.
3. Degree
   In undirected graphs, the degree of a vertex is the total count of edges incident to it. When a vertex contains a loop, this loop is counted twice towards the vertex's degree. The degree is denoted as $deg(v)$. A vertex is termed isolated if no edges are incident to it. Furthermore, a graph comprised entirely of isolated vertices is classified as a null or empty graph.
4. Path
   A path within a graph is essentially a series of edges connecting a sequence of vertices, starting from one vertex and proceeding through others along the edges. A key concept here is that two vertices are deemed connected if a path exists between them. A graph is considered connected if there is a path linking every pair of vertices. In directed graphs, strong connectivity is the presence of directed paths both from u to v and from v to u for any pair of vertices u, v. Weak connectivity, in contrast, is a state where a directed graph becomes connected only after transforming all its directed edges to undirected ones.
5. Cycle or Circuit
   A cycle is defined as a sequence of vertices and edges that forms a closed loop, beginning and ending at the same vertex. This means a cycle is a path without a distinct start or end point but rather a continuous loop. The length of a cycle is determined by the number of edges it encompasses. It is possible for a graph to contain various cycles, each differing in length.
6. Subgraph

A subgraph is essentially a smaller portion of a graph, comprising a selection of vertices and edges from a larger graph. This smaller graph retains the properties and connections of the original, larger graph, effectively forming a graph within a graph.

7. Weighted graph
   In weighted graphs, a numerical value, or weight, is assigned to each edge. These weights may represent different quantities or values, such as distances, costs, or capacities. This concept of weighted graphs is applicable to both directed and undirected graphs, adding an extra layer of information to the graph's structure.
8. Completed graph
   A complete graph is a type of graph in which every vertex is connected to every other vertex by an edge. In this graph, there is a direct link between each pair of vertices, ensuring that no vertex is isolated. This means that if the graph has n vertices, there will be an edge connecting every possible pair of vertices, resulting in a highly interconnected structure. The defining characteristic of a complete graph is this thorough and comprehensive connectivity among all its vertices.

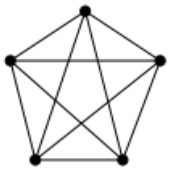

Fig. 5. Completed graph (Source: [1])

D. Graph Representation

There is some way to represent graph:

1. Adjacency matrix
   An adjacency matrix is a representation of an undirected graph with n vertices (nodes) using a square matrix M of size *n x n*. Each element in the matrix, denoted as M[i][j], represent the connection between i and j vertex. If M[i][j] is set to 1, it means there is an edge connecting vertex I to vertex j, but if it sets to 0, it means there is no edge connecting i and j vertex. For directed graph , if it sets to 1 , it means there is an edge from i vertex to j vertex , and 0 if no. For weighted graph, M[i][j] represents the edge weight from vertex i to vertex j.

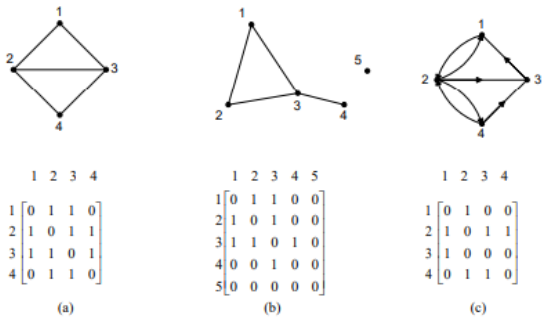

Fig. 6. Adjacency Matrix (Source: [2])

2. Incidence matrix
   The *incidence matrix A* of an *undirected* graph has a row for each vertex and a column for each edge of the graph. The element $A_{[i][j]}$ of A is 1 if the *i* vertex is a vertex of the j^{th} edge and 0 otherwise.

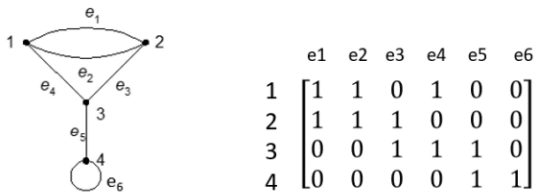

Fig. 7. Incidence Matrix (Source: [2])

E. Hamilton Trail and Circuit

A Hamiltonian path is a route in a graph that visits every vertex at least once. Conversely, a Hamiltonian circuit is a Hamiltonian path that returns to its starting point, meaning all vertices are visited with the starting vertex being visited twice. Graphs with only Hamiltonian paths are known as semi-Hamiltonian, while those with Hamiltonian circuits are termed Hamiltonian graphs. For a simple undirected graph to be Hamiltonian, a common condition is that each of its *n* vertices (for $n \geq 3$) should have a degree of at least $\frac{n}{2}$. However, a graph may still contain a Hamiltonian circuit even if it doesn't meet this criterion.

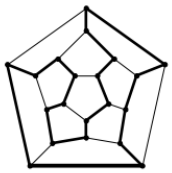

Fig. 8. Hamilton Graf (Source: [2])

F. Travelling Salesman Problem

The Traveling Salesman Problem (TSP) is an optimization challenge well recognized in computer science and operations research. It involves finding the shortest possible route that a salesman can take to visit each city in a given set exactly once and return to the starting city. This requires determining a Hamiltonian circuit in a complete graph where the total weight of all the edges is minimized.

In a typical TSP scenario, a number of cities are given, along with the distances between them. For example, in a graph representation, each city is a node, and the paths between cities are edges with weights representing the distances. The goal is to find the shortest path that covers all cities and returns to the origin, optimizing travel time and costs.

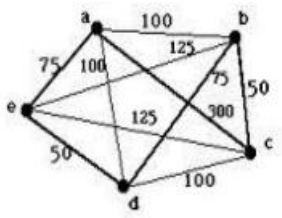

Fig. 9. Weighted graph for TSP example (Source: [2])

There are several approaches to solving the TSP. A brute-force method calculates the weight of every Hamiltonian circuit and chooses the one with the minimum weight. For instance, a graph with five nodes has $\frac{(5-1)!}{2}$ = 12 Hamiltonian circuits. This brute-force approach, however, becomes inefficient for many cities due to its time complexity of O(n!), where n is the number of nodes in the complete graph. For n that larger than 5, this approach not efficient. Therefore, in this paper using dynamic programming, because it offers a more efficient solution, especially for larger values of n, by reducing the time complexity from exponential to polynomial. It does this by storing solutions to subproblems and avoiding redundant calculations. Besides that, there is other approaches to solving TSP like nearest neighbor, branch and bound, and genetic algorithms.

G. Dynamic Programming TSP Algorithms

Dynamic programming is a method for solving a complex problem by breaking down the given problem into several sub problems and solving these sub problems once and storing the solution to these sub problems in a table. Generally, dynamic programming is applied to optimization problems. Dynamic programming is applied when there is an overlapping between sub problems of the same problem. In many computational problems, the brute-force approach, which evaluates all possible configurations to find a solution, can be extremely inefficient, especially with a growing number of elements. Dynamic programming addresses this inefficiency by storing the solutions to subproblems, thus avoiding redundant calculations. If applicable to some problem, it takes less time than naive methods. It can be used to solve problem in time polynomial time for which a naive approach would take exponential time.

The Traveling Salesman Problem (TSP) is a classic example where dynamic programming is particularly advantageous. In TSP, the goal is to find the shortest possible route that visits each city exactly once and returns to the starting point. This problem presents a significant number of overlapping sub-routes, especially as the number of cities increases. The brute-force method of evaluating every possible permutation of cities becomes impractical due to its factorial time complexity. For example, route A - B - C - D -A has the same sub-route with A - C - B - D - A. Instead of calculating the distance for every possible route, dynamic programming would solve smaller sub-routes and store these solutions. Dynamic programming, however, efficiently tackles this by solving each sub-route once and reusing the solution in the context of larger routes.

In this paper, we address the Traveling Salesman Problem (TSP) using the Held-Karp algorithm, a dynamic programming approach renowned for reducing the computational complexity from factorial to polynomial time, making it suitable for moderately sized TSP instances. The algorithm is implemented through the function *held_karp_tsp(matrix),* where matrix is a weighted adjacency matrix indicating distances between cities. It begins by initializing a dictionary C that stores the minimum cost of reaching each subset of cities, ending at a specific city. Utilizing binary representation for efficient management of city subsets, the algorithm determines the minimum cost path for each city within these subsets. Crucially, by storing and reusing results for each subset in C, the algorithm adheres to the dynamic programming principle of solving each subproblem only once, thereby avoiding redundant calculations and optimizing the route-finding process in TSP.

Here is the pseudocode for dynamic programming using bellman-Held-Karp algorithm:

```
Algorithm Bellmann-Held-Karp(G, c)
  foreach v ∈ V − s do
    OPT[{v}, v] = c(s, v)
  for j = 2 to n − 1 do
    foreach S ⊆ V − s with |S| = j do
      foreach v ∈ S do
        OPT[S, v] = min{ OPT[S − v, u] + c(u, v) | u ∈ S − v }
  return min{ OPT[V − s, v] + c(v, s) | v ∈ V − s }
```

Fig. 10. Pseudocode for solving TSP using Held-Karp algorithm (Source: [4])

III. METHODOLOGY

A. Limitations

In this paper, there is certain limitations encountered while applying the Traveling Salesman Problem (TSP) to determine the most efficient route for distributing ice cream materials to Mixue branches in Bandung. These limitations are outlined as follows.

1. The assumption that the shortest route is always the optimum route. This assumption overlooks other critical factors that impact the efficiency of the route, such as traffic conditions, road infrastructure, and the time required for delivery.
2. The analysis and solutions provided are applied to only 20 Mixue branches in Bandung, under the assumption that these branches are serviced by a single delivery truck.
3. The distances considered for each route are based on the minimum possible routes.

B. Data Used

The data on Mixue branches in Bandung was obtained using the Google Maps API, a method that allowed for a thorough and precise compilation of information. This approach led to the identification of over 60 Mixue branches throughout Bandung. The dataset includes essential details such as the names and addresses of these branches, with each address providing the city and postal code.

	Name	Address	Street	Kelurahan	Kecamatan	Kota	Provinsi	Country
0	MIXUE Cihampelas	Jl. Cihampelas No.58 A, Lamansari, Kec. Bandun...	Jl. Cihampelas No.58 A	Lamansari	Kec. Bandung Wetan	Kota Bandung	Jawa Barat 40116	Indonesia
1	Mixue	Jl. Cihampelas No.160, Cipaganti, Kecamatan Co...	Jl. Cihampelas No.160	Cipaganti	Kecamatan Coblong	Kota Bandung	Jawa Barat 40131	Indonesia
2	Mixue BSC	Istana BSC LU SEVEN, Babakan Ciamis, Kec. Sum...	Istana BSC LU SEVEN	Babakan Ciamis	Kec. Sumur Bandung	Kota Bandung	Jawa Barat 40117	Indonesia
3	Mixue	Jl. Kebon Kawung No.30, Pasir Kaliki, Kec. Ci...	Jl. Kebon Kawung No.30	Pasir Kaliki	Kec. Cicendo	Kota Bandung	Jawa Barat 40171	Indonesia
4	Mixue paskal sukajadi	Jl. Pasir Kaliki No.215, Sukabungah, Kec. Suka...	Jl. Pasir Kaliki No.215	Sukabungah	Kec. Sukajadi	Kota Bandung	Jawa Barat 40162	Indonesia
5	Mixue Pajajaran	Jl. Pajajaran No.122B, Pajajaran, Kec. Cicendo...	Jl. Pajajaran No.122B	Pajajaran	Kec. Cicendo	Kota Bandung	Jawa Barat 40172	Indonesia
6	Mixue Otista	Jl. Otto Iskandar Dinata, Balonggede, Kec. And...	Jl. Otto Iskandar Dinata	Balonggede	Kec. Andir	Kota Bandung	Jawa Barat	Indonesia
7	Mixue Banda	Jl. Banda No.32, Citarum, Kec. Bandung Wetan, ...	Jl. Banda No.32	Citarum	Kec. Bandung Wetan	Kota Bandung	Jawa Barat 40115	Indonesia
8	MIXUE RINGA	Jl. Ringa No.57, Ringa, Kec. Sumur Bandung, Ko...	Jl. Ringa No.57	Ringa	Kec. Sumur Bandung	Kota Bandung	Jawa Barat 40111	Indonesia
9	Mixue	Jl. Cihapit No.25A, Cihapit, Kec. Bandung Weta...	Jl. Cihapit No.25A	Cihapit	Kec. Bandung Wetan	Kota Bandung	Jawa Barat 40114	Indonesia
10	Mixue	Jl. Surya Sumanti No.72b, Sukapalih, Kec. Suk...	Jl. Surya Sumanti No.72b	Sukapalih	Kec. Sukajadi	Kota Bandung	Jawa Barat 40161	Indonesia

Fig. 11. Snapshot of Mixue Branches Data in Bandung (Source: Google Maps API)

The starting point for all delivery is from Mixue warehouse that located on Jalan Raya Terusan Kopo no.611.

C. Problem Modeling

Step 1

Select n Mixue Branches in Bandung that delivered by one truck.

Below is the data of the chosen Mixue branches for one delivery shift:

Mixue Branches	Address
Mixue Tubagus Ismail	Jl. Tubagus Ismail No.27B, Sekeloa, Kecamatan Coblong, Kota Bandung, Jawa Barat 40134, Indonesia
Mixue Dipatiukur	Jl. Dipati Ukur No.72F, Lebakgede, Kecamatan Coblong, Kota Bandung, Jawa Barat 40132, Indonesia
Mixue Dago Pusat	Jl. Ir. H. Juanda No.314A, RT.1/RW.1, Dago, Kecamatan Coblong, Kota Bandung, Jawa Barat 40135, Indonesia
Mixue Cihampelas	Jl. Cihampelas No.160, Cipaganti, Kecamatan Coblong, Kota Bandung, Jawa Barat 40131, Indonesia
Mixue Ciumbuleuit	Jl. Ciumbuleuit No.91, Hegarmanah, Kec. Cidadap, Kota Bandung, Jawa Barat 40141, Indonesia
Mixue Sukamaju	Jl. Sukamaju No.6, Pasteur, Kec. Sukajadi, Kota Bandung, Jawa Barat 40161, Indonesia
Mixue Cigadung	Jl. Cikondang No.15, Sadang Serang, Kecamatan Coblong, Kota Bandung, Jawa Barat 40133, Indonesia
Mixue Banda	Jl. Banda No.32, Citarum, Kec. Bandung Wetan, Kota Bandung, Jawa Barat 40115, Indonesia
Mixue Pahlawan	Jl. Pahlawan No.41, Cihaur Geulis, Kec. Cibeunying Kaler, Kota Bandung, Jawa Barat 40122, Indonesia
Mixue Sumanti	Jl. Surya Sumantri No.72b, Sukagalih, Kec. Sukajadi, Kota Bandung, Jawa Barat 40164, Indonesia
Mixue Cihapit	Jl. Cihapit No.25A, Cihapit, Kec. Bandung Wetan, Kota Bandung, Jawa Barat 40114, Indonesia
Mixue SetiaBudi	Jl. Dr. Setiabudi No.170d, Hegarmanah, Kec. Cidadap, Kota Bandung, Jawa Barat 40141, Indonesia
Mixue paskal sukajadi	Jl. Pasir Kaliki No.215, Sukabungah, Kec. Sukajadi, Kota Bandung, Jawa Barat 40162, Indonesia
Mixue Cikutra	Jl. Cikutra No.150, Cikutra, Kec. Cibeunying Kidul, Kota Bandung, Jawa Barat 40124, Indonesia
Mixue Dago Atas	Jl. Ir. H. Juanda, Dago, Kecamatan Coblong, Kota Bandung, Jawa Barat 40135
Mixue BEC	Istana BEC LU S01/02, Babakan Ciamis, Kec. Sumur Bandung, Kota Bandung, Jawa Barat 40117, Indonesia
Mixue Kawung	Jl. Kebon Kawung No.30, Pasir Kaliki, Kec. Cicendo, Kota Bandung, Jawa Barat 40171, Indonesia
Mixue Padjajaran	Jl. Pajajaran No.122B, Pajajaran, Kec. Cicendo, Kota Bandung, Jawa Barat 40172, Indonesia
Mixue Cibadak	Jl. Cibadak No.125, Karanganyar, Kec. Astanaanyar, Kota Bandung, Jawa Barat 40241, Indonesia

Table 1. Selected Mixue Branches

Step 2

From the data on the locations of Mixue branches requiring delivery, the distance between each branch and all other branches is determined. The distance between each branch and the Mixue warehouse is also calculated. This distance calculation is performed using the Google Maps Distance Matrix API, ensuring that the obtained data represents the shortest route passable by a truck.

The gathered data is then modeled as a weighted graph, with each location serving as a node, and each edge bearing a weight corresponding to the distance between locations. Below is the modeling of the problem as a weighted graph (weights

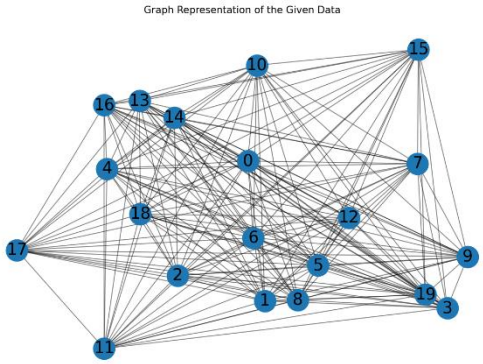

Fig. 12. Data modeling as weighted graph (Source: Primary)

The graph in Figure 12 is a directed and weighted graph. It is concerned that the distance from node-I to node-j is different from node-j to node-i. Below is the mapping of nodes in the graph to Mixue Branches:

Vertices	Mixue Brranches	Vertices	Mixue Branches
0	Mixue Warehouse	**10**	Mixue Sumatri
1	Mixue Tubagus Ismail	**11**	Mixue Cihapit
2	Mixue Dipatiukur	**12**	Mixue Setiabudi
3	Mixue Dago Pusat	**13**	Mixue Paskal Sukajadi
4	Mixue Cihampelas	**14**	Mixue Cikutra
5	Mixue Ciumbuleuit	**15**	Mixue BEC
6	Mixue	**16**	Mixue Dago

	Sukamaju		Atas
7	Mixue Cigadung	**17**	Mixue Kawung
8	Mixue Banda	**18**	Mixue Padjajaran
9	Mixue Pahlawan	**19**	Mixue Cibadak

Table 2. Vertices representation to Mixue Branches

The Adjacency matrix that represents the graph, which each cell represents the distance (in meter):

From/To	0	1	2	3	4
0	∞	31278	29741	31041	28908
1	29535	∞	1310	1324	3561
2	28704	1065	∞	1599	3459
3	29611	1054	1387	∞	3638
4	27146	4418	2981	4181	∞
5	29224	3342	2742	3105	2657
6	37194	12248	13231	14410	16063
7	30944	1977	3287	3180	5538
8	28866	4239	2824	4003	5655
9	29848	3100	3924	5103	6755
10	24917	7567	8142	7330	6566
11	29461	4155	3547	4725	6378
12	27432	4839	4239	4603	3838
13	27622	5504	4903	5267	2151
14	30864	3956	4940	6118	7771
15	27830	4673	3257	4436	4626
16	31783	3225	3558	2171	5809
17	20405	5986	4571	5750	3415
18	20314	6576	5160	6339	3967
19	18723	6952	5536	6715	4794

From/To	5	6	7	8	9
0	31707	29159	31249	23949	30979
1	3039	12246	1977	3613	3100
2	2936	12540	2759	2782	3226
3	3115	13300	3031	3690	4154
4	4847	13710	4666	2979	4396
5	∞	15812	5319	5045	6498
6	15540	∞	10499	12859	9563
7	5016	10498	∞	4158	1351
8	5133	12263	3375	∞	2713
9	6233	9569	1351	2938	∞
10	6336	18693	9650	8216	9380
11	5855	11566	3444	970	2346
12	3609	16947	6817	6216	7633
13	4273	14600	5557	3869	5286
14	7248	8505	2208	3904	1271
15	5566	13348	4776	1819	4505
16	5286	14155	3658	5861	5009
17	5537	14587	5544	3133	5273
18	6090	15177	6133	3723	5863
19	6916	16186	6922	3936	6652

From/To	**10**	**11**	**12**	**13**	**14**
0	24674	15765	29457	26758	32002
1	6246	4464	5707	4193	3956
2	5415	3626	5605	3362	4250
3	6323	4541	5784	4270	5010
4	3857	3890	4355	1791	5420
5	5935	5993	4803	3869	7522
6	15867	12203	17347	13814	8488
7	7655	3503	7684	5603	2208
8	5578	1036	7058	3525	3736
9	6560	2282	8040	4507	1279
10	∞	9780	4784	5158	10403
11	6172	∞	7652	4119	3321
12	4169	7127	∞	4614	8657
13	4333	4780	3533	∞	6310
14	7576	3249	9056	5523	∞
15	4542	2114	6008	2475	5529
16	8494	6712	7955	6441	5865
17	3307	4045	4798	1253	6297
18	3860	4634	5350	1805	6887
19	4686	4090	6176	2632	7676

From/To	15	16	17	18	19
0	23125	33014	13570	20226	12665
1	3749	3668	5483	6453	6079
2	2918	3572	4652	5622	5248
3	3826	2651	5560	6529	6156
4	2155	6154	3873	4069	5040
5	4257	5078	5976	6189	7143
6	14160	14157	16423	16074	15264
7	5948	3658	7666	7862	7486
8	2303	5975	4309	5784	3668
9	4852	5009	6571	6766	6136
10	7102	9303	6495	7569	8612
11	3594	6698	5050	6379	3930
12	5392	6576	7110	6934	8277
13	3045	7240	4763	4587	5930
14	5868	5865	7587	7782	7102
15	∞	6409	2452	3954	3264
16	5997	∞	7731	8701	8327
17	2309	7722	∞	2225	3265
18	2899	8312	2551	∞	3855

19	3274	8688	1422	3604	∞

Table 3. Graph representation with weighted adjacency matrix.

D. The implementation of Dynamic Programming for the Traveling Salesman Problem (TSP) in Python

In this paper, the author applies an implementation of the Traveling Salesman Problem (TSP) solution algorithm through a dynamic programming approach, employing the Held-Karp algorithm in Python. This implementation can be seen in the function held_karp_tsp(matrix). However, the implementation has been modified so that it not only returns the shortest distance but also the nodes it passes through.

Firstly, the author performs the initialization of the data variable distance_matrix to hold the weighted adjacency matrix as can be seen in Table 3.

```
distance_matrix = [[0, 31278, 29741, 31041, 28908, 31707, 29159, 31249, 23949, 30979, 24674, 15765, 29457, 26758, 32002, 23125, 33014, 13570, 20226, 12665],
                   [29535, 0, 1310, 1324, 3561, 3039, 12246, 1977, 3613, 3100, 6246, 4464, 5707, 4193, 3956, 3749, 3668, 5483, 6453, 6079],
                   [28704, 1065, 0, 1599, 3459, 2936, 12540, 2759, 2782, 3226, 5415, 3626, 5605, 3362, 4250, 2918, 3572, 4652, 5622, 5248],
                   [29611, 1054, 1387, 0, 3638, 3115, 13300, 3031, 3690, 4154, 6323, 4541, 5784, 4270, 5010, 3826, 2651, 5560, 6529, 6156],
                   [27146, 4418, 2981, 4181, 0, 4847, 13710, 4666, 2979, 4396, 3857, 3890, 4355, 1791, 5420, 2155, 6154, 3873, 4069, 5048],
                   [29224, 3342, 2742, 3105, 2657, 0, 15812, 5319, 5045, 6498, 5935, 5993, 4803, 3869, 7522, 4257, 5078, 5976, 6189, 7143],
                   [37194, 12248, 13231, 14410, 16863, 15540, 0, 10499, 12859, 9563, 15867, 12203, 17347, 13814, 8488, 14160, 14157, 16423, 16074, 15264],
                   [30944, 1977, 3287, 3180, 5538, 5016, 10498, 0, 4158, 1351, 7655, 3503, 7684, 5603, 2208, 5948, 3658, 7666, 7862, 7486],
                   [28866, 4239, 2824, 4003, 5655, 5133, 12263, 3375, 0, 2713, 5578, 1036, 7058, 3525, 3736, 2383, 5975, 4309, 5784, 3668],
                   [29848, 3100, 3924, 5103, 6755, 6233, 9569, 1351, 2938, 0, 6560, 2282, 8040, 4507, 1279, 4852, 5009, 6571, 6766, 6136],
                   [24917, 7567, 8142, 7330, 6566, 6336, 18693, 9650, 8216, 9380, 0, 9780, 4784, 5158, 10403, 7102, 9303, 6495, 7569, 8612],
                   [29461, 4155, 3547, 4725, 6378, 5855, 11566, 3444, 970, 2346, 6172, 0, 7652, 4119, 3321, 3594, 6698, 5050, 6379, 3930],
                   [27432, 4839, 4239, 4603, 3838, 3609, 16947, 6817, 6216, 7633, 4169, 7127, 0, 4614, 8657, 5392, 6576, 7110, 6934, 8277],
                   [27622, 5504, 4903, 5267, 2151, 4273, 14600, 5557, 3869, 5286, 4333, 4780, 3533, 0, 6310, 3045, 7240, 4763, 4587, 5930],
                   [30864, 3956, 4940, 6118, 7771, 7248, 8505, 2208, 3904, 1271, 7576, 3249, 9056, 5523, 0, 5868, 5865, 7587, 7782, 7102],
                   [27838, 4673, 3257, 4436, 4626, 5566, 13348, 4776, 1819, 4505, 4542, 2114, 6008, 2475, 5529, 0, 6409, 2452, 3954, 3264],
                   [31783, 3225, 3558, 2171, 5809, 5286, 14155, 3658, 5861, 5009, 8494, 6712, 7955, 6441, 5865, 5997, 0, 7731, 8701, 8327],
                   [20405, 5986, 4571, 5750, 3415, 5537, 14587, 5544, 3133, 5273, 3307, 4045, 4798, 1253, 6297, 2309, 7722, 0, 2225, 3265],
                   [20314, 6576, 5160, 6339, 3967, 6090, 15177, 6133, 3723, 5863, 3860, 4634, 5350, 1805, 6887, 2899, 8312, 2551, 0, 3855],
                   [18723, 6952, 5536, 6715, 4794, 6916, 16186, 6922, 3936, 6652, 4686, 4090, 6176, 2632, 7676, 3274, 8680, 1422, 3604, 0]]
```

Fig. 13. Store Matrix to variable (Source: Primary)

Then, the function held_karp_tsp(matrix) is called, where matrix is the weighted adjacency matrix, to obtain the shortest route and the total shortest distance.

```
def held_karp_tsp(distance_matrix):
    n = len(distance_matrix)
    C = {}

    for k in range(1, n):
        C[(1 << k, k)] = (distance_matrix[0][k], 0)

    for subset_size in range(2, n):
        for subset in combinations(range(1, n), subset_size):
            bits = 0
            for bit in subset:
                bits |= 1 << bit
            for k in subset:
                prev = bits & ~(1 << k)
                res = []
                for m in subset:
                    if m == 0 or m == k:
                        continue
                    res.append((C[(prev, m)][0] + distance_matrix[m][k], m))
                C[(bits, k)] = min(res)

    bits = (2**n - 1) - 1
    res = []
    for k in range(1, n):
        res.append((C[(bits, k)][0] + distance_matrix[k][0], k))
    opt, parent = min(res)

    path = [0]
    for i in range(n - 1):
        path.append(parent)
        new_bits = bits & ~(1 << parent)
        _, parent = C[(bits, parent)]
        bits = new_bits
    path.append(0)
    return list(reversed(path)), opt
```

Fig. 14. held_karp_tsp(matrix) function. (Source: Primary)

Inside the function, the function initially stores the distances from node 0 to other nodes, indicating that node 0 is designated as the starting point.

```
    for k in range(1, n):
        C[(1 << k, k)] = (distance_matrix[0][k], 0)
```

Fig. 15. Set vertices 0 as starting point (Source: Primary)

Then, the function searches for the shortest total distance Hamiltonian circuit by using the subset distance values that have been recorded in the dictionary C.

```
bits = (2**n - 1) - 1
res = []
for k in range(1, n):
    res.append((C[(bits, k)][0] + distance_matrix[k][0], k))
opt, parent = min(res)

path = [0]
for i in range(n - 1):
    path.append(parent)
    new_bits = bits & ~(1 << parent)
    _, parent = C[(bits, parent)]
    bits = new_bits
path.append(0)
```

Fig. 16. Code to set shortest route. (Source: Primary)

Subsequently, the results returned by the function are stored in the variables shortest route and total distance, and then output to the terminal.

```
shortest_route, total_distance = held_karp_tsp(distance_matrix)
print("Shortest route = ",end="")
print(shortest_route)
print("Total distance = ",end="")
print(total_distance,end=" ")
print("m")
```

Fig. 17. Output result. (Source: Primary)

IV. ANALYSIS AND DISCUSSION

A. Analysis of the Most Optimal Route: Results of the Program Execution for Solving TSP with Dynamic Programming

The result of the program execution implementing Travelling Salesman Problem (TSP) with dynamic programming to find the most optimum route for material distribution to Mixue Branches in Bandung:

```
Shortest route = [0, 17, 18, 13, 4, 10, 12, 5, 2, 1, 3, 16, 7, 6, 14, 9, 11, 8, 15, 19, 0]
Total distance = 91240 m
```

Fig. 18. The Result of program execution. (Source: Primary)

The most optimum route for delivering material to 20 Mixue Branches is:

1. Node 0 – Mixue Warehouse
2. Node 17 – Mixue Kawung
3. Node 18 – Mixue Padjajaran
4. Node 13 – Mixue Paskal Sukajadi
5. Node 4 – Mixue Cihampelas

6. Node 10 – Mixue Sumatri
7. Node 12 – Mixue Setiabudi
8. Node 5 – Mixue Ciumbuleuit
9. Node 2 – Mixue Dipatiukur
10. Node 1 – Mixue Tubagus Ismail
11. Node 3 – Mixue Dago Pusat
12. Node 16 – Mixue Dago Atas
13. Node 7 – Mixue Cigadung
14. Node 6 – Mixue Sukamaju
15. Node 14 – Mixue Cikutra
16. Node 9 – Mixue Pahlawan
17. Node 11 – Mixue Cihapit
18. Node 8 – Mixue Banda
19. Node 15 – Mixue BEC
20. Node 19 – Mixue Cibadak
21. Node 0 – Mixue Warehouse

The total distance for one time delivery from Mixue warehouse to 20 Mixue branches, then back to Mixue warehouse is 91240 meter or 91,240 kilometers.

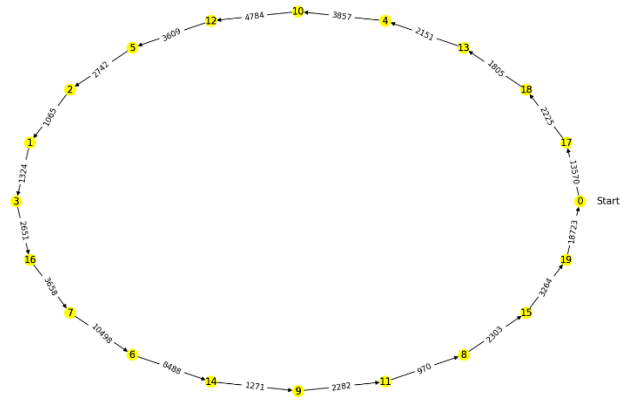

Fig. 19. Optimum Route Visualization. (Source: Primary)

In figure 19, it shows the Hamilton circuit with minimum weight (in meter) to deliver material to 20 Mixue branches in Bandung.

B. Algorithm and Method Analysis

The researchers utilized a dynamic programming approach incorporating the Held-Karp algorithm to address the Travelling Salesman Problem (TSP). This approach significantly reduces computational redundancy compared to brute-force methods, yet it still entails examining all possible routes to determine the shortest distance. The algorithm demonstrates considerable time complexity, particularly when applied to scenarios involving more than 20 nodes, with a complexity order of O (n^2 * 2^n). The initialization itself is O(n), then the loop runs for every subset size from 2 to n-1. The number of the subset is 2^n .Lastly, the algorithm iterates over *n* elements to find the minimum cost path. This is markedly more efficient than the brute-force approach, which has a complexity order of O(n!). However, the algorithm's efficiency diminishes for problems with more than 20 nodes, leading the researchers to limit their analysis to 20 Mixue branches.

The researchers concluded that employing the TSP approach was optimal for identifying the shortest route for material delivery to Mixue branches. The primary focus of the research was on minimizing the distance covered to visit all locations. However, they acknowledged that this method has limitations, as the most optimal route is not solely determined by distance. Factors such as traffic conditions, time, and road infrastructure also play significant roles. Despite these considerations, the researchers maintained that prioritizing shortest distance is the most effective strategy in optimizing delivery routes.

Applying this methodology, Mixue can potentially lower delivery costs to its branches, thereby enhancing its profitability.

V. CONCLUSION

This paper shows that the Traveling Salesman Problem (TSP) algorithm works well for figuring out the best way to deliver goods to Mixue shops in Bandung city. This algorithm helps find the shortest path to visit all the Mixue branches, which can help the Mixue company save money on deliveries. Researcher used a special way of solving problems, called dynamic programming, to make the TSP algorithm work. This method is good because it looks at every possible route and picks the shortest one. This makes sure it doesn't miss any better routes.

From this research, researcher found the best way to deliver materials to 20 Mixue branches. The route starts at Mixue Warehouse → Mixue Kawung → Mixue Padjajaran → Mixue Paskal Sukajadi → Mixue Cihampelas → Mixue Sumatri → Mixue Setiabudi → Mixue Ciumbuleuit → Mixue Dipatiukur → Mixue Tubagus Ismail → Mixue Dago Pusat → Mixue Dago Atas → Mixue Cigadung → Mixue Sukamaju → Mixue Cikutra → Mixue Pahlawan → Mixue Cihapit → Mixue Banda → Mixue BEC → Mixue Cibadak → Mixue Warehouse. The total distance of this route is approximately 91.24 kilometers. This methodology can be similarly applied to determine the optimal routes for other Mixue branches. By consistently applying this approach, the Mixue company can ensure the efficiency of its delivery operations across various locations, thereby optimizing operational costs and enhancing overall logistical efficiency.

VI. APPENDIX

The completed Traveling Salesman Problem algorithm cab be found below.
https://github.com/Benardo07/TSP-Mixue-Branches

VII. ACKNOWLEDGMENT

First and foremost, I extend my deepest gratitude to God, whose blessings and guidance have been my constant source of strength and inspiration throughout this research journey. Then, researcher would like to express his sincere thanks to his Discrete Mathematics instructor, Dr. Nur Ulfa Maulidevi, S.T., M.Sc., lecturer of class 01. Her knowledge and the resources provided were invaluable in the completion of this paper. Additionally, my appreciation extends to all the lecturers of IF2120 who have contributed to my understanding of Discrete Mathematics, making this research possible.

The researchers would also thank to Rachel Gabriela Chen, for helping researcher by giving a guide to using google Maps API, that is used to collect data for this research.

Lastly, the researcher would like to thank his parents, all his friends that always supporting him and gives motivation to learn and develop together.

REFERENCES

[1] R. Munir, "Graf Bagian 1," *IF2120 Matematika Diskrit*. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf [Accessed 9 December2023].

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

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

[4] S. Chaplick & A. Wolff, "Lecture 1. Introduction & Held-Karp-algorithm for TSP," [Online]. Available: https://wuecampus.uni-wuerzburg.de/moodle/pluginfile.php/1928459/mod_resource/content/2/advalg-ws19-vl01-intro%2Btsp.pdf

[5] "Travelling Salesman Problem using Dynamic Programming," *GeeksforGeeks, 19-Apr-2023.* [Online]. Available: https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/ [Accessed 10 December 2023].

[6] "Travelling Salesman Problem: Its definition and Implementation," *Bhumi Varta technology*, 25-Jan-2023. [Online]. Available: https://bvarta.com/travelling-salesman-problem-its-definition-and-implementation/ [Accessed 9 December 2023].

PERNYATAAN

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, 11 Desember 2023

Benardo 13522055