Optimizing Supermarket Layouts with Graph Theory to Enhance Consumer Experience

Alvin Christopher Santausa - 13523033[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*alvinchrisantausa@gmail.com*, *13523033@std.stei.itb.ac.id*

***Abstract*—This paper explores the application of graph theory to optimize supermarkets layouts to enhance the consumer experience. By representing supermarket layouts as graphs, where nodes represent product categories and edges represent distances, the shortest through all points (categories) while visiting each exactly once can be determined using the Hamiltonian circuit/path. This can be solved by using several algorithms, including Held-Karp algorithm. This approach aims to provide consumers with a more convenient shopping experience by ensuring efficient navigation through all categories, thereby maximizing product exposure.**

***Keywords*—Graph, Hamiltonian Circuit, Hamiltonian Path, Travelling Salesman Problem, Held-Karp, Shortest Path.**

I. Introduction

Supermarket layouts with numerous categories and aisles often make navigation confusing and overwhelming for consumers. Consumers/shoppers may find it difficult to locate specific items or determine an efficient route through the store. It can not only detract the shopping experience but also result in missed opportunities for product exposure.

Stores like IKEA solve this problem by using a "racetrack layout" to providing guidance for customers through a predefined path, increasing exposure to products. Supermarkets could implement a similar solution by using graph theory to find an efficient route. Optimal routes from the entrance area to the checkout area, while still making sure all categories are visited by consumers can be identified using the Hamiltonian path or Hamiltonian circuit, depending on the layout

- If the entrance and checkout are at the same location, then we can use the Hamiltonian circuit, otherwise
- If the entrance and checkout are at different locations, we can use the Hamiltonian path.

This problem is analogous to the Travelling Salesman Problem (TSP) where the goal is to find the shortest path from node to a node while making sure other nodes are visited exactly once. Supermarkets layouts can be represented as a graph, where

- Nodes represent categories or sections
- Edges represent paths between categories
- Weights represent distances

With this representation, we can find the shortest route we need by determining the Hamiltonian path/circuit using the Held-Karp algorithm. However, if there is no Hamiltonian path/circuit (it is impossible to visit all nodes exactly one time from start to finish), then we can not use this approach and have to use another approach.

Once the shortest route is identified, supermarkets can also rearrange category locations based on the route order. For instance, the most frequently purchased category can be placed at the first node of the path/route, second most purchased category at the second node, and so on. This will enhance consumers' convenience by not only providing a shortest route to go to all categories but also allowing them to find items from the most popular category quickly, improving their overall shopping experience.

II. Theoretical Foundation

A. Graph

Graph is an object that consists of a vertex set (V) and an edge set (E). Vertex set (V) is a set of points often called vertices or nodes, which are the fundamental units of the graph. Edge set (E) is a set of connections often called edges or links that represent relationships between the vertices. A graph G is represented as

$$G = (V, E)$$

Where:

- V is a non-empty finite set of vertices
- E is a set of pairs of elements of V

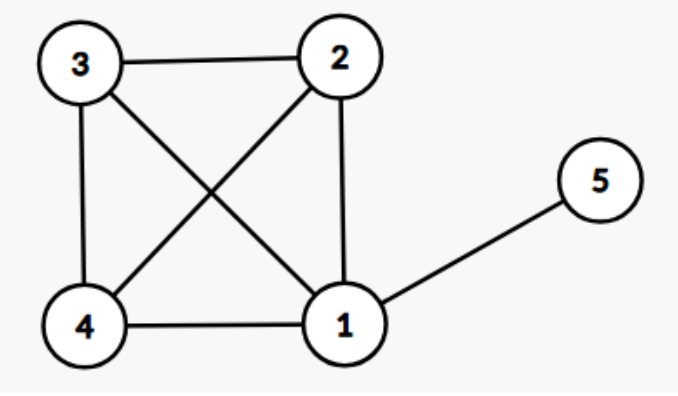

Figure 1 A Graph
Source: https://github.com/Incheon21/MatdisMakalah

Graph can be classified based on the nature of their vertices and their vertices. There are many types of graphs including simple graph, multigraph, directed graph, undirected graph,

weighted graph, unweighted graph, and many more.

- *Undirected Graph*

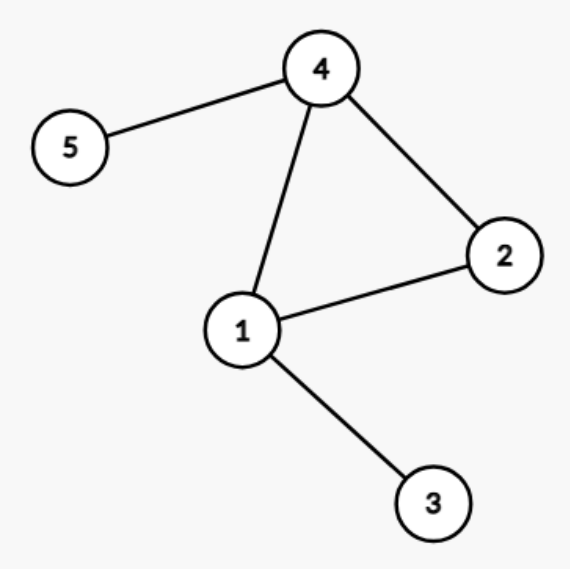

Figure 2 Undirected Graph
Source: https://github.com/Incheon21/MatdisMakalah

Undirected graph is a type of graph where the edges have no direction. This means that if there is an edge between two vertices u and v, it can be traversed in either direction

- *Weighted Graph*

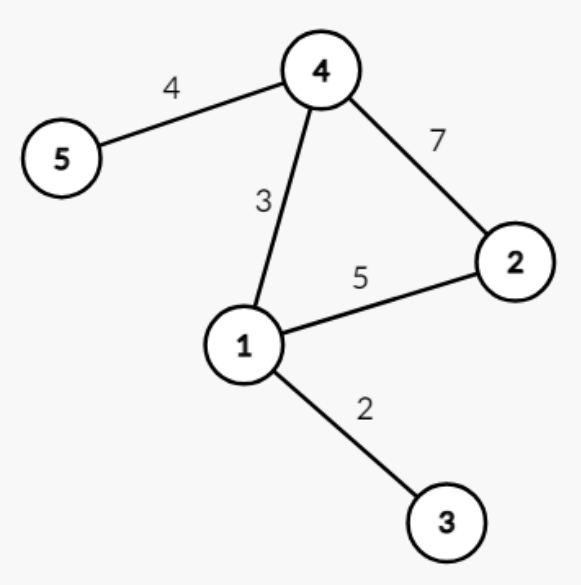

Figure 3 Weighted Graph
Source: https://github.com/Incheon21/MatdisMakalah

Weighted Graph is a type of graph where each edge is assigned to a numerical value called weight. These weights can represent various properties such as distance, cost, time, or capacity on the application.

Graphs are versatile structures used to model relationships, interactions, and networks across a wide range of disciplines. Their flexibility and applicability make them fundamental in many real-world scenarios. They are commonly represented using mathematical structures like matrices or lists to simplify analyzing and working with graphs computationally. The choice of representation depends on the graph's characteristics and the computational problem being solved. Graphs can be represented as an adjacency matrix, incidence matrix, adjacency list, and many more.

1. *Adjacency Matrix*

Nodes	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	0
3	1	0	0	0	0
4	1	1	0	0	1
5	0	0	0	1	0

Adjacency matrix is a square matrix A of size $|V| \times |V|$, where $|V|$ is the number of vertices in the graph [3]. Each element a_{ij} represents the relationship between vertices i *and* j. In the undirected graph, the matrix is symmetric, $a_{ij} = a_{ij}$, but in the directed graph, the matrix may not be symmetric, as $a_{ij} \neq a_{ij}$. In the weighted graph, the matrix entries a_{ij} store the weight of the edge. If there is no edge, the value is typically 0 or ∞. In unweighted graph, the matrix entries a_{ij} are 1 (edge exists) or 0 (no edge).

B. Hamiltonian Circuit/Path

In graph theory, a Hamilton circuit is a circuit that visits every vertex exactly once, without repeating any vertex. Similarly, a Hamilton path also visits every vertex once with no repeats, but have different start and end vertex [4]. These concepts are crucial in solving optimization problems where traversing all points efficiently is necessary, such as in logistics, network design, or supermarket layout optimization.

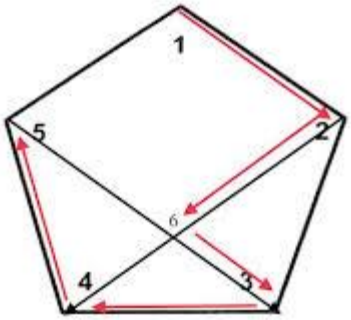

Figure 4 A Supermarket
Source: https://study.com/academy/lesson/hamilton-circuits-and-hamilton-paths.html

C. Travelling Salesman problem (TSP)

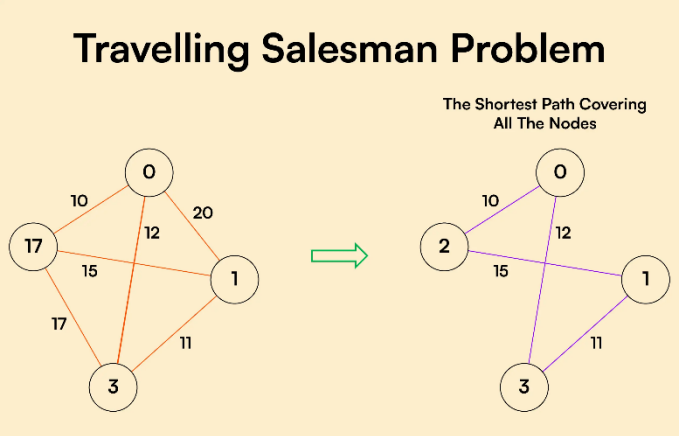

Figure 5 Travelling Salesman Problem
Source: https://www.lystloc.com/blog/what-is-a-travelling-salesman-problem-tsp/

Travelling Salesman Problem (TSP) is a classic optimization problem in mathematics and computer science that formulated in 1930 by Karl Menger. It involves finding the shortest possible route for a salesman to visit a given set of cities exactly once and return to the starting point (Hamiltonian cycle). TSP has practical applications in various fields, far beyond its humble beginnings. It applied in logistics (optimizing delivery routes), manufacturing (tool path optimizations), DNA sequencing, and many more. TSP is classified as an NP-hard problem, meaning that as the number of cities increases, the complexity of finding an optimal solution grows exponentially, making it computationally challenging to solve for large datasets. There are two approaches to solve TSP.

1. Exact Algorithms
   the brute-force approach can be used to evaluate all possible permutations to find the optimal route
2. Heuristic and Approximation Algorithms
   It is useful for larger instances because it provides approximate solutions within a reasonable timeframe but does not guarantee the optimal solution.

D. Held-Karp Algorithm

Held-Karp algorithm or Bellman-Held-Karp algorithm is a dynamic programming solution that found by Bellman, Held, and Karp to solve the Travelling Salesman Problem (TSP) exactly. This algorithm accepts a distance matrix of a cities set and then it will find significantly reduces the time complexity compared to brute-force methods by storing and reusing intermediate results (subproblems). This algorithm has better time complexity $O(2^n \cdot n)$ rather than the time complexity $O(n!)$ of brute-force approach. The Held-Karp algorithm builds the solution incrementally by breaking the problem into smaller subproblems, using dynamic programming to avoid redundant computations, and exploring all possible subsets of cities to find the shortest path to each subset, and finally combining these results.

The key idea of this algorithm is to represent the problem using subsets of cities. It defines a dynamic programming table $dp[S][i]$, where S is a subset of cities, and i is the last city visited in the subset. The value of $dp[S][i]$ stores the shortest path that visits all cities in the subset S and ends at city i. The algorithm builds the solution incrementally with the recursive relation:
$dp[S][i] = \min\big(dp[S-\{i\}][i] + dist(j,i)\big)$ for all $j \in S, j \neq i$
Where $dist(j,i)$ is the distance between cities j and I, and $S - \{i\}$ denotes the subset of cities excluding city i.

```
function algorithm TSP(G, n)
    for k := 2 to n do
        C({i, k}, k) := d_{1,k}
    end for
    for s = 3 to n do
        for all S ⊆ {1, 2, ..., n} ||S|| = s do
            for all k ∈ S do
                {C(S, k) = min_{m≠k, m∈S}[C(S − {k}, m) + d_{m,k}]}
                opt := min_{k≠1}[C({1, 2, 3, ..., n}, k) + d_{1,k}
            end for
        end for
    end for;
    return (opt)
end
```

Figure 6 Pseudocode for Held-Karp Algorithm
Source:
https://web.archive.org/web/20150208031521/http://www.cs.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf

E. Supermarket

Supermarket is a large-scale retail establishment also known as 'Combination Store' that primarily sells a wide variety of food and grocery items, along with household products, typically organized into category sections or departments. It offers an extensive range of product categories to fulfil diverse consumer needs and preferences. Supermarket operated on a self-service basis where they do not employ salesmen and the customers have to pick up the good from different racks or bins.

Figure 7 A Supermarket
Source: https://www.britannica.com/money/supermarket

III. Implementation

The demonstration of the Held-Karp algorithm to find the path to help supermarkets determine the shortest path will be done with two scenarios:

A. Simple Layout

For example, given a very simple supermarket layout with 3 categories like the picture below, where the entrance and checkout locations is same at node 0.

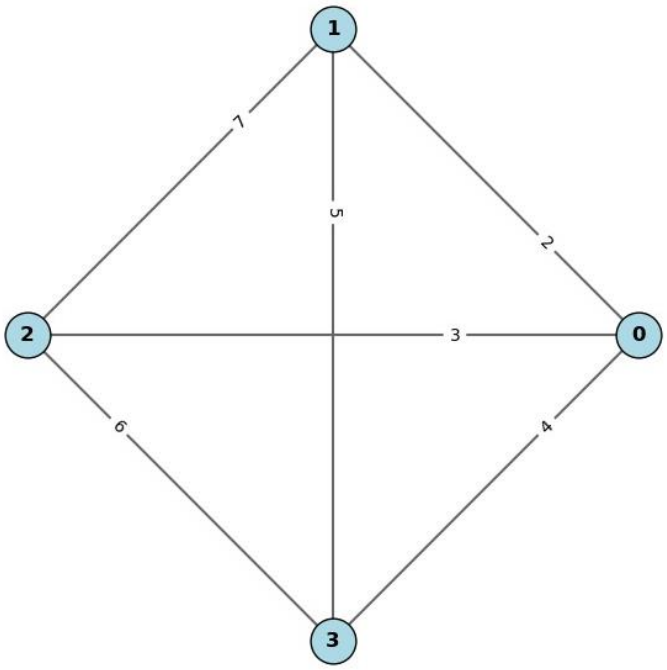

Figure 8 Simple Layout's Visualized Graph
Source: https://github.com/Incheon21/MatdisMakalah

It can be represented as an adjacency matrix below,

nodes	**0**	**1**	**2**	**3**
0	0	2	3	4
1	2	0	7	5
2	5	7	0	6
3	4	5	6	0

Then the Held-Karp algorithm can be used to recursively find the shortest possible path from node 0 back to node 0 again with the notation below

$$dp[S][i] = \min\big(dp[S - \{i\}][i] + dist(j, i)\big)$$

Then, do an initialization

$$dp[\{0\}, 0] = 0$$

Calculation for subset with 2 nodes

- Node 1
  $dp[\{0,1\}, 1] = dp[\{0\}][0] + dist(0,1)$
  $= 0 + 2$
  $= 2$
- Node 2
  $dp[\{0,2\}, 2] = dp[\{0\}][0] + dist(0,2)$
  $= 0 + 3$
  $= 3$
- Node 3
  $dp[\{0,3\}, 3] = dp[\{0\}][0] + dist(0,3)$
  $= 0 + 4$
  $= 4$

Calculation for subsets with 3 nodes

- $\{0,1,2\}$ subset
  $dp[\{0,1,2\}, 2] = dp[\{0,1\}][1] + dist(1,2))$
  $= 2 + 7$
  $= 9$
- $\{0,1,3)$ subset
  $dp[\{0,1,3\}, 3] = dp[\{0,1\}][1] + dist(1,3))$
  $= 2 + 5$
  $= 7$
- $\{0,2,1\}$ subset
  $dp[\{0,2,1\}, 1] = dp[\{0,2\}][2] + dist(2,1))$
  $= 3 + 7$
  $= 10$
- $\{0,2,3\}$ subset
  $dp[\{0,2,3\}, 3] = dp[\{0,2\}][2] + dist(2,3))$
  $= 3 + 6$
  $= 9$
- $\{0,3,1\}$ subset
  $dp[\{0,3,1\}, 1] = dp[\{0,3\}][3] + dist(3,1))$
  $= 4 + 5$
  $= 9$
- $\{0,3,2\}$ subset
  $dp[\{0,3,2\}, 2] = dp[\{0,3\}][3] + dist(3,2))$
  $= 4 + 6$
  $= 10$

Calculation for subsets with all nodes

- Ends in node 1
  $dp[\{0,1,2,3\}, 1] = \min(dp[\{0,2,3\}][3] + dist(3,1), dp[\{0,3,2\}][2] + dist(2,1)$
  $= \min(9 + 5, 10 + 7)$
  $= 14$
- Ends in node 2
  $dp[\{0,1,2,3\}, 2] = \min(dp[\{0,1,3\}][3] + dist(3,2), dp[\{0,3,1\}][1] + dist(1,2)$
  $= \min(7 + 6, 9 + 7)$
  $= 13$
- Ends in node 3
  $dp[\{0,1,2,3\}, 3] = \min(dp[\{0,1,2\}][2] + dist(2,3), dp[\{0,2,1\}][1] + dist(1,3)$
  $= \min(9 + 6, 10 + 5)$
  $= 15$

Finally, the shortest path that starts can be determined by choosing the minimum distance from 3 available choices of path.

Total distance $= \min(dp[\{0,1,2,3\}][1] + dist(1,0), dp[\{0,1,2,3\}][2] + dist(2,0), dp[\{0,1,2,3\}][3] + dist(3,0)$
$= \min(14 + 2, 13 + 3, 15 + 4$
$= 16$

This means the shortest path from 0 to 0 again while making sure all nodes are visited is {0,2,3,1,0} with 16 meters total distances.

The same algorithm can also be used to find the shortest path from point a to point b where a ≠ b (Hamiltonian path not circuit). For example, the shortest path from node 0 to node 2 while making sure all nodes are visited is {0, 1, 3, 2} with 13 meters total distance. These paths are visualized below.

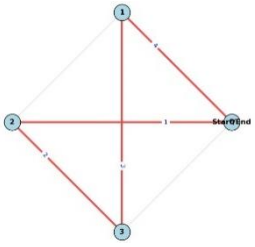

Figure 9 0 to 0 path

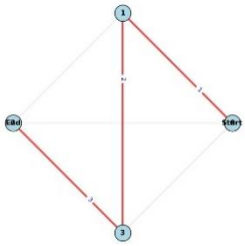

Figure 10 0 to 2 path

Source: https://github.com/Incheon21/MatdisMakalah

B. More Complex Layout

In the more complex layout, the supermarket layout will be more complex with 5 categories like the picture below, where the entrance and checkout location is different now. The entrance is at node 0 and the checkout is at node 6.

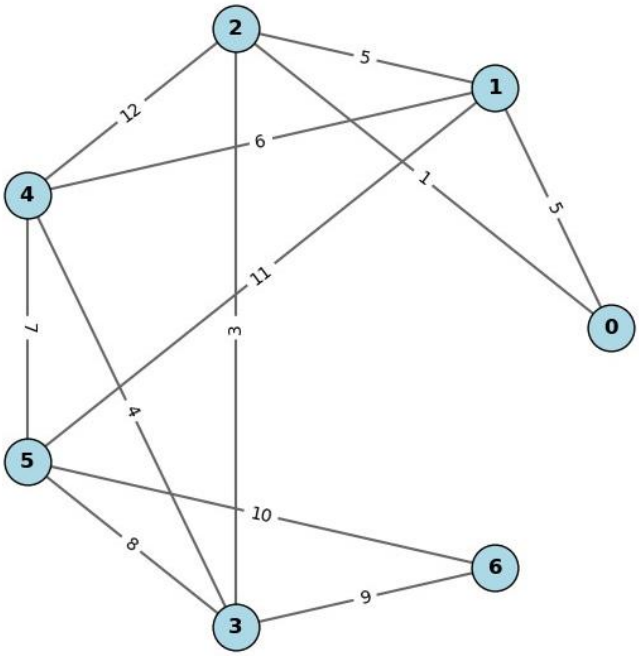

Figure 11 More Complex Layout's Visualized Graph

Source: https://github.com/Incheon21/MatdisMakalah

It can be represented as an adjacency matrix below,

nodes	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	∞	5	1	∞	∞	∞	∞
1	5	∞	5	∞	6	11	∞
2	1	5	∞	3	12	∞	∞
3	∞	∞	3	∞	4	8	9
4	∞	6	12	4	∞	7	∞
5	∞	∞	∞	8	7	∞	10
6	∞	∞	∞	9	∞	10	∞

For this more complex layout, we will use a program that has been made using Held-Karp algorithm to calculate the shortest path (Hamiltonian path).

`held_karp(graph, start, end)` function using dynamic programming approach to find the shortest path. It has 3 parameters,

- graph
  2D list representing the adjacency matrix of the supermarket layout graph
- start
  Starting node of the path (entrance area)
- end
  Ending node of the path (checkout area)

```
def held_karp(graph, start, end):
  n = len(graph)
  dp = [[float('inf')] * n for _ in range(1 <<
n)]  # table for storing shortest weights
  parent = [[-1] * n for _ in range(1 << n)]  #
Table to reconstruct the path
  dp[1 << start][start] = 0

  for mask in range(1 << n):
    for u in range(n):
      if mask & (1 << u):
        for v in range(n):
          if not (mask & (1 << v)) and
graph[u][v] != float('inf'):
            new_weight = dp[mask][u] +
graph[u][v]
            if new_weight < dp[mask | (1 <<
v)][v]:
              dp[mask | (1 << v)][v] = new_weight
              parent[mask | (1 << v)][v] = u
  # If start == end, we must return to the start
node
  if start == end:
    min_path_weight = float('inf')
    last_node = -1
    for u in range(n):
      if dp[(1 << n) - 1][u] != float('inf') and
graph[u][start] != float('inf'):
        new_weight = dp[(1 << n) - 1][u] +
graph[u][start]
        if new_weight < min_path_weight:
          min_path_weight = new_weight
          last_node = u
    # path recronstruction
    if min_path_weight != float('inf'):
      path = reconstruct_path(parent, last_node,
(1 << n) - 1, start, end)
      return path, min_path_weight
  # If start != end, find the shortest path to
the end node
  else:
    min_path_weight = float('inf')
    last_node = -1
    for u in range(n):
      if dp[(1 << n) - 1][u] != float('inf') and
graph[u][end] != float('inf'):
        new_weight = dp[(1 << n) - 1][u] +
graph[u][end]
        if new_weight < min_path_weight:
          min_path_weight = new_weight
          last_node = u
    # Path recronstruction
    if min_path_weight != float('inf'):
      path = reconstruct_path(parent, last_node,
(1 << n) - 1, start, end)
      return path, min_path_weight

  return None, float('inf')
```

`reconstruct_path(parent, last_node, mask, start, end)` function to reconstruct the shortest path or circuit from the

parent table created during the Held-Karp algorithm. It has 5 parameters,

- parent
  The table that stores the previous node for each subset of visited nodes
- last_node
  The last nodes of the path
- mask
  The subset of visited nodes
- start
  Starting node
- end
  Ending node

```
def reconstruct_path(parent, last_node, mask,
start, end):
  """Reconstructs the path from the DP parent
table."""
  path = []
  current = last_node

  while current != -1:
    path.append(current)
    next_mask = mask ^ (1 << current)
    current = parent[mask][current]
    mask = next_mask
  path.reverse()
    # Reverse to get the correct order
  if start == end:  # If it's a circuit, append
the start node to form a loop
    path.append(start)

  return path
```

Main script where the graph is represented into an adjacency matrix and the Held-Karp algorithm is run.

```
graph = [
  [0, 5, 1, float('inf'), float('inf'),
float('inf'), float('inf')],  # Node 0
  [5, 0, 5, float('inf'), 6, 11,
float('inf')],                        # Node 1
  [1, 5, 0, 3, 12, float('inf'),
float('inf')],                        # Node 2
  [float('inf'), float('inf'), 3, 0, 4, 8,
9],                          # Node 3
  [float('inf'), 6, 12, 4, 0, 7,
float('inf')],                        # Node 4
  [float('inf'), float('inf'), float('inf'), 8,
7, 0, 10],             # Node 5
  [float('inf'), float('inf'), float('inf'), 9,
float('inf'), 10, 0]  # Node 6
]

# Input Start and End Node
start_node = 0
end_node = 6

# Run Held-Karp Algorithm
shortest_path, shortest_path_weight =
held_karp(graph, start_node, end_node)
```

If the result is printed, it will be like the picture below

```
Shortest Path (Held-Karp): [0, 2, 1, 4, 3, 5, 6]
Total Weight: 34
```

Figure 12 Node 0 to 6 Result
Source: https://github.com/Incheon21/MatdisMakalah

The same program can also be used to find the shortest path from node 0, back to node 0 again, if the supermarkets move the checkout area to the same location as the entrance area (Hamiltonian circuit). The result for that case is shown in the picture below.

```
Shortest Path (Held-Karp): [0, 2, 3, 6, 5, 4, 1, 0]
Total Weight: 41
```

Figure 13 Node 0 to 0 Result
Source: https://github.com/Incheon21/MatdisMakalah

These 2 paths/routes is visualized below

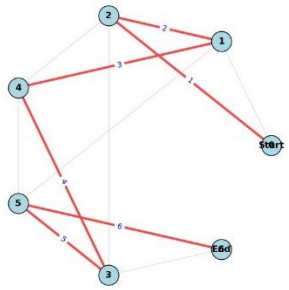

Figure 14 0 to 6 Path
Source: https://github.com/Incheon21/MatdisMakalah

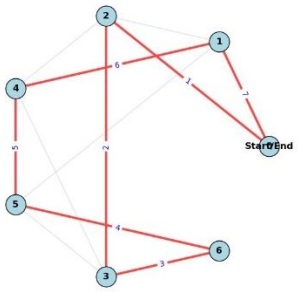

Figure 15 0 to 0 Path
Source: https://github.com/Incheon21/MatdisMakalah

IV. DISCUSSION

By applying the Held-Karp algorithm to determine the shortest path, supermarkets can use this to create a predefined path that can guide customers efficiently through all product categories. The route/path will not only enhance the shopping experience by minimizing navigation time but also improve convenience for consumers. Supermarkets can also use this shortest path and integrate it with other data/information, like category popularity data, to strategically relocate category placements along the route. For instance, more popular categories can be placed closer to the starting point, allowing consumers to access high-demand category's items more quickly.

This combination of predefined path and data-driven category placement will further enhance the shopping experience, offering consumers a streamlined and intuitive journey through the store. Moreover, increased exposure to all products from different categories along the routes can lead to higher sales and boost customer satisfaction, providing supermarkets with increased profitability.

V. SOME COMMON MISTAKES

When applying graph theory, specifically Held-Karp algorithm to optimize supermarket layouts, several common mistakes can arise, potentially reduce the effectiveness of the solution. One frequent issue is attempting to use the Held-Karp algorithm on a graph that lacks a Hamiltonian path or circuit. Since this algorithm relies on visiting each node exactly once, its application to such graphs is invalid and results with no path available. It is important to ensure the graph structure is first analyzed and validated to confirm the existence of a Hamiltonian path or circuit.

Another common mistake is the incorrect representation of the

graph. Mislabeling nodes or edges can lead to inaccurate results and make the optimization meaningless. Attention to detail when creating adjacency matrices for the layout representation is essential to avoid such errors.

Additionally, failing to map practical constraints, such as one-way path or restricted access areas, can produce routes that are theoretically optimal but impractical for real-world use. These constraints must be adjusted and implemented into the graph model to ensure the solution can be used for the real layout.

By recognizing and addressing these common mistakes, the graph theory can be used to optimize supermarket layouts, providing efficient and practical solutions to enhance the shopping experience.

VI. Conclusion

The application of graph theory for optimizing supermarket layout using the Held-Karp algorithm showed great potential. Both simple and complex layouts can be analyzed to determine the shortest paths through all product categories. Implementing these routes as predefined paths will enhance the shopping experience by guiding consumers efficiently and improving convenience. Furthermore, it will also increase products exposure that can lead to higher sales and better customer satisfaction, providing supermarkets with more profits.

However, we have to mind some cases when the graph representing the supermarket layout does not have a Hamiltonian path or circuit. In such cases, the Held-Karp algorithm cannot be used, and it becomes impossible to determine a shortest path that visits each node exactly once. Addressing this limitation requires alternative approaches, such as modifying the layout or exploring other algorithms that will not be discussed in this paper.

VII. Appendix

1. Github Repository: https://github.com/Incheon21/MatdisMakalah

VIII. Acknowledgment

The author expresses heartfelt gratitude to Almighty God for the blessings and guidance during the writing of this paper. Special thanks are extended to Dr. Ir. Rinaldi Munir, M.T., for the role as lecturer in the IF1220 Discrete Mathematics course and for making the lecture materials available on the course website, which supported the research process. The author also wishes to acknowledge the unwavering support from family and friends that were invaluable in completing this paper.

References

[1] Munir, Rinaldi. 2023. "Graf (Bag 3)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/22-Graf-Bagian3-2024.pdf

[2] Balakrishnan, V. K. (1997). Graph Theory (1st ed.). McGraw Hill. ISBN 978-0-07-005489-9

[3] Chaddha, Manvi. 2025. *Implementation of Graph in Python*. *https://www.codingninjas.com/codestudio/library/implementation-of-graph-in-python*

[4] *Euler and Hamiltonian Paths and Circuits | Mathematics for the Liberal Arts*. (n.d.). https://courses.lumenlearning.com/wmopen-mathforliberalarts/chapter/introduction-euler-paths/

[5] Wright, G. (2024, December 17). *What is traveling salesman problem (TSP)?* WhatIs. https://www.techtarget.com/whatis/definition/traveling-salesman-problem? Accessed: January 7th, 2025]

[6] *Wayback machine*. (n.d.). https://web.archive.org/web/20150208031521/http://www.cs.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf [Accessed: January 7th, 2025]

[7] *Britannica money*. (2025, January 4). https://www.britannica.com/money/supermarket

Statement of Originality

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

Bandung, 8th January 2025

Alvin Christopher Santausa
13523033