CSCI 3302
DATA STRUCTURES AND ALGORITHMS II

Heuristic and Exact Approaches to Solving the Travelling Salesman Problem (TSP)

Group 404

Team

YUSUF MOHAMMAD YUNUS (2314467)

FARIS BIN SUHAIMI (2317561

EIMRAN HAKEEM BIN UDA ISKANDAR (2225533)

ABSTRACT

the Travelling Salesman Problem (TSP) is a well-known mathematical problem and classified as a non-deterministic polynomial (NP) hard problem, which tries to find the shortest route that passes through a set of points only once. the Travelling Salesman Problem (TSP) has also found widespread application in several scientific and technological domains. It is very hard to solve effectively and efficiently due to its NP-hard nature. there are also a multitude of optimization approaches that have been proposed and developed by scientists and researchers during the last several decades.

ABSTRACT

Nearest Neighbor is a TSP heuristic method. the main objective of this algorithm is to visit the nearest city (node). the time complexity of the Nearest Neighbor algorithm is O (n2). Based on the previous research, the Held-Karp algorithm is used to optimize the delivery route, considering efficient distance and delivery sequence. Held-Karp Algorithm produces better results than the Iterative Deepening Search algorithm in finding optimal delivery routes.

this study proposes a base case to determine NNA for solving the symmetric TSP. Travelling Salesman Problem (TSP) finds the shortest Hamiltonian cycle in a graph, and it is an improved NNA, where it starts with the shortest edge consisting of two cities, then it includes the closest city on the route repeatedly until an effective route is established.

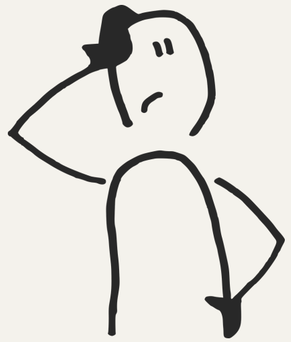
INTRODUCTION

the Travelling Salesman Problem (TSP) involves finding the shortest possible route that visits a set of cities exactly once and returns to the origin city. This problem is NP-hard and has numerous real-world applications in logistics, robotics, and delivery systems.

Due to its computational complexity, it is essential to evaluate both exact and heuristic approaches. TSP is used to find the shortest Hamiltonian cycle in a graph, with exponentially increasing complexity to solve large problems with guaranteed optimality. There are also several algorithms used to find optimal tours, but none lead to feasible solutions for large instances since they all grow exponentially.

the Nearest Neighbor is a TSP heuristic method where the algorithm is to visit the nearest city (node). the time complexity of the Nearest Neighbor is O(n2). the Held-Karp algorithm is used to optimize the delivery route, considering efficient distance and delivery sequence. Held-Karp Algorithm produces better results than the Iterative Deepening Search algorithm in finding optimal delivery routes.

PROBLEM STATEMENT



the Travelling Salesman Problem (TSP) presents a significant computational challenge in the field of combinatorial optimisation due to its NP-hard nature. As the number of cities increases, the complexity of finding an exact optimal solution grows exponentially, making brute-force and exhaustive search methods impractical for real-world applications.

Heuristic algorithms like the Nearest Neighbor Algorithm (NNA) offer a faster and more scalable alternative but often sacrifice accuracy due to their greedy and locally optimal decision-making process. the quality of the solution from NNA can vary significantly depending on the starting point and may not reflect the globally shortest path.

PROBLEM STATEMENT

In logistics and transportation, determining the most efficient route that minimizes distance and travel time is essential for reducing operational costs and improving delivery performance. While exact algorithms such as Held-Karp can provide optimal solutions, they become computationally expensive and infeasible for larger datasets.

This study addresses the challenge of solving the symmetric TSP using a baseline heuristic approach the Nearest Neighbor Algorithm and evaluates its performance on a dataset of Malaysian cities. The goal is to assess the effectiveness of this heuristic in producing an efficient travel route and compare its practicality against theoretical optimal methods in terms of distance, travel time, and computational efficiency.

RESEARCH OBJECTIVES

1. Implement the **Nearest Neighbor Algorithm (NNA)** as a heuristic method for solving the symmetric Travelling Salesman Problem (TSP) using a real-world dataset of selected Malaysian cities.

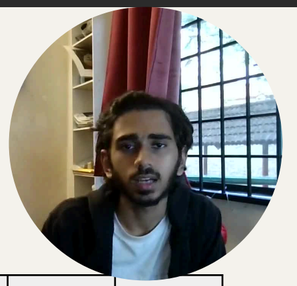
2. **Determine an efficient travel route** that minimizes the total travel distance and time between cities based on a given distance matrix.

3. **Evaluate the performance of the NNA** in terms of total distance, time travel, and computational speed, and identify its strengths and limitations when compared to optimal solutions such as the Held-Karp algorithm.

4. **Analyze the practical applicability of heuristic methods** like NNA in real-world logistics and route optimization scenarios where computational efficiency is crucial.

5. **Provide visualization and interpretation** of the computed travel path to support data-driven decision-making in transportation and planning tasks.

DATASETS



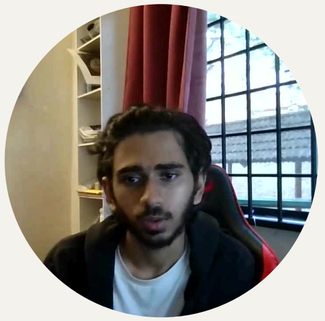
From / To	KL	Penang	Johor	Ipoh	Malacca	Kuantan	Kota Bharu	KT	Alor Setar	Seremban
KL	0	355	330	205	145	250	440	455	410	65
Penang	355	0	670	160	500	550	210	240	100	420
Johor	330	670	0	500	220	340	640	660	720	270
Ipoh	205	160	500	0	360	430	300	330	260	280
Malacca	145	500	220	360	0	280	550	570	580	90
Kuantan	250	550	340	430	280	0	370	240	610	180
Kota Bharu	440	210	640	300	550	370	0	165	290	510
Kuala Terenggan	455	240	660	330	570	240	165	0	320	530
Alor Setar	410	100	720	260	580	610	290	320	0	470
Seremban	65	420	270	280	90	180	510	530	470	0

Table 5.1: Distance Matrix Between Major Malaysian Cities (in Kilometers)

From / To	KL	Penang	Johor	Ipoh	Malacca	Kuantan	Kota Bharu	KT	Alor Setar	Seremban
KL	0	4.4	4.1	2.6	1.8	3.1	5.5	5.7	5.1	0.8
Penang	4.4	0	8.4	2	6.3	6.9	2.6	3	1.3	5.3
Johor	4.1	8.4	0	6.3	2.8	4.3	8	8.3	9	3.4
Ipoh	2.6	2	6.3	0	4.5	5.4	3.8	4.1	3.3	3.5
Malacca	1.8	6.3	2.8	4.5	0	3.5	6.9	7.1	7.3	1.1
Kuantan	3.1	6.9	4.3	5.4	3.5	0	4.6	3	7.6	2.3
Kota Bharu	5.5	2.6	8	3.8	6.9	4.6	0	2.1	3.6	6.4
Kuala Terengganu	5.7	3	8.3	4.1	7.1	3	2.1	0	4	6.6
Alor Setar	5.1	1.3	9	3.3	7.3	7.6	3.6	4	0	5.9
Seremban	0.8	5.3	3.4	3.5	1.1	2.3	6.4	6.6	5.9	0

Table 5.2: Travel Time Matrix Between Major Malaysian Cities (in Hours)

Datasets



From / To	AIKOL	KICT	KENMS	KOED	KOE	KAED	KIRKHS	CELPAD
AIKOL	0	1.3	0.15	0.84	0.93	0.82	0.42	0.44
KICT	1.3	0	1.6	0.9	0.45	0.75	1.15	1.17
KENMS	0.15	1.6	0	1.22	1.29	0.9	0.5	0.45
KOED	0.84	0.9	1.22	0	90	0.45	0.16	0.22
KOE	0.93	0.45	1.29	0.09	0	0.17	0.45	0.45
KAED	0.82	0.75	0.9	0.45	0.17	0	0.6	0.6
KIRKHS	0.42	1.15	0.5	0.16	0.45	0.6	0	0.04
CELPAD	0.44	1.17	0.45	0.22	0.45	0.6	0.04	0

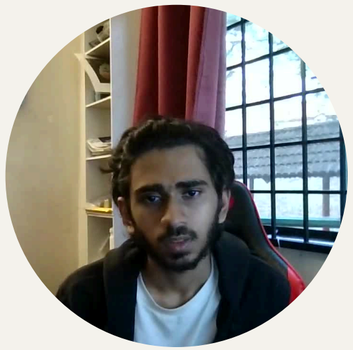
Table 5.3: Walking Distance Matrix between Kulliyyahs at IIUM Gombak (in kms)

From / To	AIKOL	KICT	KENMS	KOED	KOE	KAED	KIRKHS	CELPAD
AIKOL	0	18	2	11	12	13	5	5
KICT	18	0	21	12	5	9	15	16
KENMS	2	21	0	16	17	13	8	7
KOED	11	12	16	0	2	5	3	4
KOE	12	5	17	2	0	2	7	7
KAED	13	9	13	5	2	0	8	8
KIRKHS	5	15	8	3	7	8	0	1
CELPAD	5	16	7	4	7	8	1	0

Table 5.4: Walking Time Matrix between Kulliyyahs at IIUM Gombak (in Minutes)

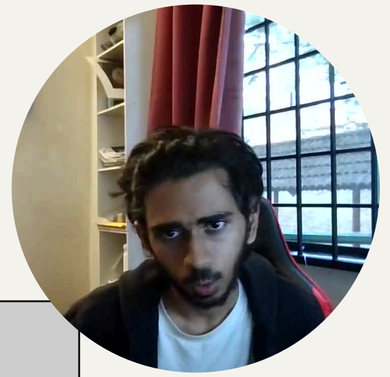
Methodology

this project uses two real-world datasets to generate TSP instances. the first includes 10 major Malaysian cities—such as Kuala Lumpur, Penang, and Johor—with driving distances and estimated travel times based on Google Maps data from 2025. the second dataset focuses on IIUM Gombak campus, covering walking distances and times between 8 kulliyyahs, measured and timed manually. these datasets offer realistic scenarios for solving the TSP at both national and campus scales, enabling meaningful comparisons of algorithm performance in terms of distance, time, and scalability. the distance matrix is generated using the Euclidean formula:



$$d(i,j) = \sqrt{\left(x_i - x_j\right)^2 + \quad + \left(y_i - y_j\right)^2}$$

Methodology



Algorithm	Max Cities	Avg. Time (s)	Avg. Tour Length	Complexity
Brute-force	≤ 10	Very High	Optimal	$O(n!)$
Held-Karp	≤ 25	Moderate	Optimal	$O(n^2 \cdot 2^n)$
Nearest Neighbor	≤ 1000	Very Low	Suboptimal	$O(n^2)$
Genetic Algorithm	≤ 1000	Low–Moderate	Near Optimal	Varies
Simulated Annealing	≤ 1000	Moderate	Near Optimal	Varies

Algorithm Used

7.1 Exact Approach: Held-Karp Algorithm (Dynamic Programming)

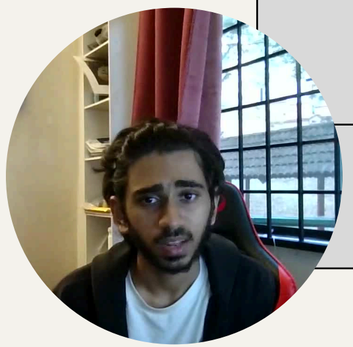
the Held-Karp algorithm is a classic exact algorithm based on dynamic programming for solving the TSP. Unlike the brute-force method which computes all n!n!n! possible tours, Held-Karp reduces the computational complexity to O(n2 2n) , which is significantly faster and more efficient for medium-sized instances (e.g. n ⩽ 25).

Key Advantages

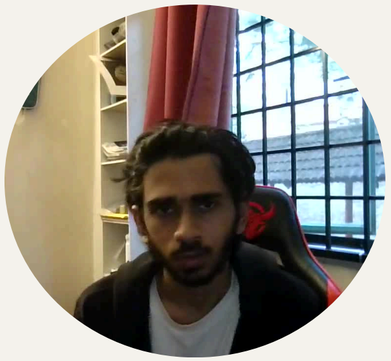
- **Efficiency:** Compared to brute-force, Held-Karp significantly reduces computation by storing and reusing solutions to overlapping subproblems.
- **Optimality:** Unlike heuristics, it guarantees finding the shortest possible route.
- **Scalability:** While still exponential, it is feasible for medium-sized datasets (up to 20–25 cities), making it useful for logistics problems requiring high accuracy.

Why Held-Karp is Better (Compared to Brute Force)

Metric	Brute Force	Held-Karp
Time Complexity	O(n!)	**O(n2 2n)**
Space Complexity	O(1)	**O(n2n)**
Optimality	✓ (Optimal)	✓ (Optimal)
Practical Input Size	n ≤ 10	n ≤ 25
Performance	Exponential and slow	Faster due to memoization



Algorithm Used



7.2 Heuristic Approach: Nearest Neighbour Algorithm

the Nearest Neighbour (NN) algorithm is one of the simplest and fastest heuristic methods for solving the travelling Salesman Problem (TSP). Unlike exact algorithms, NN does not guarantee an optimal solution, but it often provides a quick approximation that is acceptable for large datasets where exact methods become infeasible

Algorithm Overview

the Nearest Neighbour heuristic follows a greedy approach:

1. Start from a chosen city (e.g., city 0).
2. At each step, visit the nearest unvisited city (based on Euclidean or actual distance).
3. Repeat until all cities are visited.
4. Return to the starting city to complete the tour

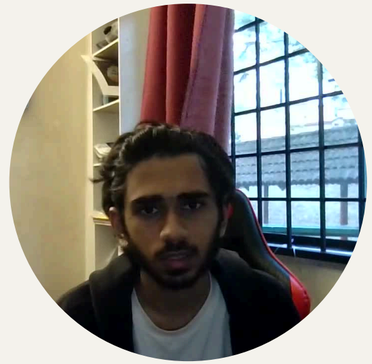
Algorithm Used

7.2 Heuristic Approach: Nearest Neighbour Algorithm

From / To	A	B	C	D	E
A	-	10	15	20	25
B	10	-	35	25	17
C	15	35	-	30	28
D	20	25	30	-	16
E	25	17	28	16	-

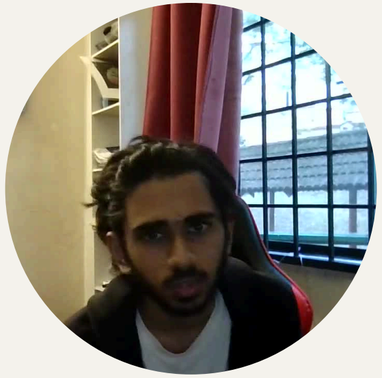
Starting from A, the algorithm selects the closest next city at each step:

- A → B (10)
- B → E (17)
- E → D (16)
- D → C (30)



Total Cost: 73 (this route is not guaranteed to be optimal, but is found in linear time.)

Algorithm Used



7.2 Heuristic Approach: Nearest Neighbour Algorithm

Metric	Exact Algorithms (Held-Karp)	Nearest Neighbor
Time Complexity	**O(n^2 . 2^n)**	**O(n^2)**
Space Complexity	**O(n . 2^n)**	**O(n)**
Optimality	✓ (Optimal)	✗ (Approximate)
Practical Input Size	n ≤ 25	**n ≤ 10,000+**
Performance	Exponential, memory intensive	**Linear, very fast**
Solution Quality	Perfect	25-50% above optimal
Implementation	Complex	**Simple**

trade-offs:

- **Accuracy vs Speed:** Sacrifices optimal solutions for dramatically faster computation
- **Use Cases:** Ideal for large-scale logistics, route planning, and situations where "good enough" solutions are acceptable
- **Starting Point Sensitivity:** Results can vary significantly depending on the chosen starting c

HELD-KARP ALGORITHM

```python
def held_karp(dists):
    n = len(dists)
    C = {}

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

    for subset_size in range(2, n):
        for subset in itertools.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 == k:
                        continue
                    res.append((C[(prev, m)][0] + dists[m][k], m))
                C[(bits, k)] = min(res)

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

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

    return opt, path
```

DISCUSSION

the Held-Karp algorithm is an exact algorithm used to solve the Travelling Salesman Problem (TSP) with a time complexity of $O(n^2 * 2^n)$, where n is the number of place. This method guarantees finding the optimal (shortest possible) route, but its computational cost grows exponentially with the number of nodes.

How it works:

1. It uses bit masking to represent subsets of places.
2. the dictionary C stores the minimum cost to reach a subset of places ending at place k.
3. It iteratively builds solutions for subsets of increasing sizes until the complete set is covered.
4. After considering all possible subsets and paths, it finds the minimum path that visits all places and returns to the starting place.
5. the path is reconstructed in reverse by tracking the parent place that led to each place.

NEAREST NEIGHBOUR ALGORITHM

```python
def nearest_neighbor_tsp(dists, return_to_start=False):
    """
    Approximate TSP solution using the Nearest Neighbor heuristic.
    If return_to_start is False, the route will not return to the starting city.
    dists: 2D list or matrix of distances between cities.
    Returns (total_cost, path_indices).
    """
    n = len(dists)
    unvisited = set(range(1, n))
    path = [0]  # start at city 0
    total_cost = 0

    current = 0
    while unvisited:
        # Find nearest unvisited city
        next_city = min(unvisited, key=lambda city: dists[current][city])
        total_cost += dists[current][next_city]
        path.append(next_city)
        unvisited.remove(next_city)
        current = next_city

    if return_to_start:
        # Return to start city to complete the cycle
        total_cost += dists[current][0]
        path.append(0)

    return total_cost, path
```

DISCUSSION

the Nearest Neighbor algorithm is a greedy heuristic for TSP with a time complexity of $O(n^2)$. It does not guarantee an optimal solution but offers a fast and reasonably good approximation, especially useful for large datasets.

How it works:

1. Start at an initial place (node 0).
2. At each step, move to the closest unvisited place.
3. Continue this process until all places are visited.
4. Optionally return to the starting place to form a complete cycle (depending on the return_to_start flag).

RESULT + ANALYSIS

0	355	330	205	145	250	440	455	410	65
355	0	670	160	500	550	210	240	100	420
330	670	0	500	220	340	640	660	720	270
205	160	500	0	360	430	300	330	260	280
145	500	220	360	0	280	550	570	580	90
250	550	340	430	280	0	370	240	610	180
440	210	640	300	550	370	0	165	290	510
455	240	660	330	570	240	165	0	320	530
410	100	720	260	580	610	290	320	0	470
65	420	270	280	90	180	510	530	470	0

the distance matrix pairwise road distance (in KM) between all ten cities in Malaysia

0.0	1.3	0.15	0.84	0.93	0.92	0.42	0.44
1.3	0.0	1.6	0.9	0.45	0.75	1.15	1.17
0.15	1.6	0.0	1.22	1.29	1.29	0.5	0.45
0.84	0.9	1.22	0.0	0.09	0.45	0.16	0.4
0.93	0.45	1.29	0.09	0.45	0.17	0.45	0.45
0.82	0.75	0.9	0.45	0.17	0.0	0.6	0.04
0.42	1.15	0.4	0.16	0.45	0.6	0.04	0.04
0.44	1.17	0.45	0.22	0.45	0.6	0.04	0.0
0.44	1.47	0.45	0.22	0.45	0.04	0.6	0.0

the distance matrix pairwise road distance (in KM) between all eight kulliyyah in IIUM

RESULT + ANALYSIS

(Total Distance: 1875 KM)
Route: Kuala Lumpur → Seremban → Malacca → Johor Bahru → Kuantan → Kuala Terengganu → Kota Bharu → Alor Setar → Penang → Ipoh

← HELD-KARP ALGORITHM for Malaysia Cities

```
(Total Distance: 1690.0 KM)
Route: Kuala Lumpur → Seremban → Malacca → Johor Bahru → Kuantan

→ Kuala Terengganu → Kota Bharu → Penang → Alor Setar → Ipoh
```

← NEAREST NEIGHBOUR ALGORITHM for Malaysia Cities

```
(Total Distance: 2.91 KM)
Route: AIKOL → KENMS → CELPAD → KIRKHS → KOED → KOE → KICT → KAED
```

← HELD-KARP ALGORITHM for Kulliyyah buildings in IIUM Gombak

```
(Total Distance: 1.81 KM)
Route: AIKOL → KENMS → CELPAD → KIRKHS → KOED → KOE → KAED → KICT
```

← NEAREST NEIGHBOUR ALGORITHM for Kulliyyah buildings in IIUM Gombak

RESULT + ANALYSIS

Computation Time Comparison

Algorithm	Dataset	Computation Time (seconds)
Held-Karp	Malaysia Cities	0.0045
Nearest Neighbour	Malaysia Cities	0.0000
Held-Karp	IIUM Kulliyyah Buildings	0.0000
Nearest Neighbour	IIUM Kulliyyah Buildings	0.0000

HELD-KARP ALGORITHM

Travel time between consecutive cities (in hours for cities/in minutes for kulliyyah):
Kuala Lumpur -> Seremban: 0.8
Seremban -> Malacca: 1.1
Malacca -> Johor Bahru: 2.8
Johor Bahru -> Kuantan: 4.3
Kuantan -> Kuala Terengganu: 3.0
Kuala Terengganu -> Kota Bharu: 2.1
Kota Bharu -> Alor Setar: 3.6
Alor Setar -> Penang: 1.3
Penang -> Ipoh: 2.0

Total travel time: 21.0

Travel time between consecutive cities (in hours for cities/in minutes for kulliyyah):
AIKOL -> KENMS: 2.0
KENMS -> CELPAD: 7.0
CELPAD -> KIRKHS: 1.0
KIRKHS -> KOED: 3.0
KOED -> KOE: 2.0
KOE -> KICT: 5.0
KICT -> KAED: 9.0

Total travel time: 29.0

NEAREST NEIGHBOUR ALGORITHM

Travel time between consecutive cities (in hours for cities/in minutes for kulliyyah):
Kuala Lumpur -> Seremban: 0.8
Seremban -> Malacca: 1.1
Malacca -> Johor Bahru: 2.8
Johor Bahru -> Kuantan: 4.3
Kuantan -> Kuala Terengganu: 3.0
Kuala Terengganu -> Kota Bharu: 2.1
Kota Bharu -> Penang: 2.6
Penang -> Alor Setar: 1.3
Alor Setar -> Ipoh: 3.3

Total travel time: 21.3

Travel time between consecutive cities (in hours for cities/in minutes for kulliyyah):
AIKOL -> KENMS: 2.0
KENMS -> CELPAD: 7.0
CELPAD -> KIRKHS: 1.0
KIRKHS -> KOED: 3.0
KOED -> KOE: 2.0
KOE -> KAED: 2.0
KAED -> KICT: 9.0

Total travel time: 26.0

Analysis Summary

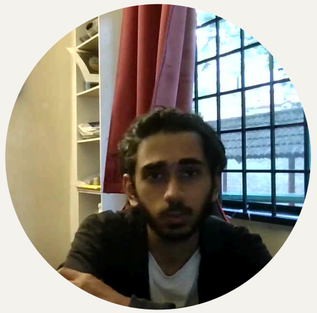
Held-Karp Algorithm produced the shortest possible route (optimal) for both datasets but required slightly more computation time for the Malaysia Cities dataset.

Nearest Neighbour Algorithm produced a route faster (zero computation time noticeable for small datasets) but the total distance and travel time were slightly higher compared to Held-Karp.

For the IIUM Kulliyyah dataset, both algorithms showed negligible computation time, but Held-Karp again provided a better total distance and travel time.

This confirms that Held-Karp is suitable for small-sized problems where optimality is required, whereas Nearest Neighbour is suitable for faster, approximate solutions with larger datasets.

Application-Specific Algorithm Recommendations



Logistics & Supply Chain

Hybrid Strategy Recommended

- Held-Karp: Small routes (≤15 locations), cost-critical operations
- Nearest Neighbor: Large-scale distribution, real-time fleet management
- Impact: 1-2% route optimization = thousands in annual fuel savings

Robotics & Autonomous Systems

Choose Based on Environment

- Held-Karp: Static environments, battery optimization
- Nearest Neighbor: Dynamic obstacles, multi-robot coordination
- Key factor: Speed vs. precision requirements

Delivery Route Planning

Match Method to Delivery Type

- Nearest Neighbor: Same-day delivery, emergency services
- Held-Karp: Planned deliveries, route optimization priority
- Hybrid: Small groups (exact) + neighborhood connections (fast)

Campus & Urban Navigation

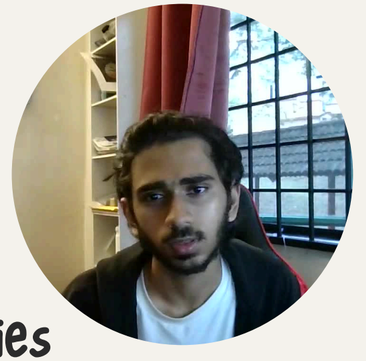
Size Determines Strategy

- Held-Karp: Small campuses (≤20 locations), tourist routes
- Nearest Neighbor: Large universities, public transit systems
- Factor: Area size and schedule flexibility

Manufacturing & Production

Quality vs Speed Trade-off

- Held-Karp: Precision work (circuit boards), maintenance schedules
- Nearest Neighbor: Warehouse picking, high-volume operations
- Decision point: Accuracy savings vs. processing time



Conclusion

this study compared two different methods for finding the shortest routes between cities using real data from Malaysia. the Held-Karp method always finds the best possible route but takes much longer to calculate, especially with more cities. the Nearest Neighbor method finds good routes very quickly but they might not be the absolute shortest.

the results show that choosing the right method depends on what you need most. use Held-Karp when you need the perfect route and can wait for the answer, like when saving fuel costs is very important. use Nearest Neighbor when you need fast results or have many locations to visit.

Although this study only tested small numbers of cities, it shows how these methods work in real life situations. this research helps people choose the right method for planning routes and can guide future studies with larger datasets.

THANK YOU FOR LISTENING!