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

Eimran Hakeem [1], Faris [2], Yusuf Mohammad Yunus[3]

[1,2,35] *Dept. Computer Science, International Islamic University Malaysia. Kuala Lumpur, Malaysia*

***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. The first problem was formulated in 1930 and is one of the most intensively studied problems in optimization. Among these algorithms, the heuristic approaches are deemed appropriate for addressing this significant issue. The simplest and easiest to implement heuristic algorithms for TSP are the nearest neighbor algorithm (NNA). The nearest neigbor algorithm solution suffers from its quality owing to randomness during the optimization process. 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. 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.**

Keywords—Keywords: Travelling Salesman Problem (TSP), Heuristic Algorithm, Nearest Neighbor Algorithm, Held-Karp Algorithm.

I. 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 uses to find the shortest Hamiltonian cycle in a graph, for exponentially complexity increases exponentially 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.

II. Problem Statement

The Travelling Salesman Problem (TSP) presents a significant computational challenge in the field of combinatorial optimization 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.

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.

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.

Given the cities represented as points in 2D space to find the shortest possible tour that visits each city once and returns to the starting city. Distances between cities are computed using Euclidean distance.

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.

III. Research Objectives

The primary aim of this study is to explore and evaluate heuristic approaches for solving the Travelling Salesman Problem (TSP), specifically focusing on the Nearest Neighbor Algorithm (NNA) for determining the shortest route among multiple cities. The specific objectives of this research are:

(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.

IV. Datasets

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

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 Terengganu	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

This table shows the estimated driving distances between ten major cities in Malaysia. The distances are based on common routes found on Google Maps in 2025. They represent the shortest practical driving routes using highways and main roads, rounded to the nearest 5–10 kilometres. This table can be used for route planning and solving problems like the Traveling Salesman Problem.

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

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

This table shows the estimated travel times between the same ten Malaysian cities. The times are calculated using an average driving speed of about 80 km/h and reflect typical travel durations in normal traffic. This information is useful for planning time-based routes and can be used together with the distance table for better analysis in transport and logistics.

Table 5.3: Walking Distance Matrix between Kulliyyahs at IIUM Gombak

From / To	AIKOL	KICT	KENMS	KOED	KOE	KAED	KIRKHS	CELPAD
AIKOL	0	1.300	0.150	0.840	0.930	0.820	0.420	0.440
KICT	1.300	0	1.600	0.900	0.450	0.750	1.150	1.170
KENMS	0.150	1.600	0	1.220	1.290	0.900	0.500	0.450
KOED	.840	0.900	1.220	0	90	0.450	0.160	0.220
KOE	0.930	0.450	1.290	0.090	0	0.170	0.450	0.450
KAED	0.820	0.750	0.900	0.450	0.170	0	0.600	0.600
KIRKHS	0.420	1.150	0.500	0.160	0.450	0.600	0	0.040
CELPAD	0.440	1.170	0.450	0.220	0.450	0.600	0.040	0

This table shows the measured walking distances (in meters) between various kulliyyahs within the IIUM Gombak campus.

Table 5.4: Walking Time Matrix between Kulliyyahs at IIUM Gombak

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

This table presents the walking time estimates (in minutes) between kulliyahs, based on actual campus measurements.

V. METHADOLOGY

(I) Instance Generation

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 the 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:

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

(II) Exact Approaches

- **Brute-force:** This method calculates the total distance for every possible permutation of city visits to find the optimal solution. While it guarantees the best result, its time complexity is O(n!), which makes it practical only for very small datasets (e.g., $n \leq 10$).

- **Held-Karp Algorithm:** The Held-Karp algorithm is a dynamic programming approach that significantly reduces computational complexity compared to brute-force methods. It has a time complexity of $O(n^2 \cdot 2^n)$, making it suitable for moderately sized problems (e.g., $n \leq 25$). It still guarantees the optimal solution but requires substantial memory and processing power as n increases.$n \leq 25$).

Figure 1.4 Source: Held-Karp Algorithm

(3) Heuristic Approaches

- *Nearest Neighbour Algorithm*

  This is a simple greedy algorithm that builds a route by always visiting the closest unvisited city next. While fast and easy to implement, it often produces suboptimal solutions because it doesn't consider the overall tour cost.

- *Genetic Algorithm (GA)*

  A metaheuristic inspired by natural selection, the Genetic Algorithm maintains a population of candidate solutions (tours). Through processes such as crossover (combining parts of two tours) and mutation (random changes), the algorithm evolves better solutions over generations. It can find near-optimal results for large problem sizes but does not guarantee optimality.

- *Simulated Annealing (SA)*

  This probabilistic technique allows occasional acceptance of worse solutions in order to escape local optima and explore a wider solution space. Over time, the algorithm reduces the probability of accepting worse solutions (analogous to cooling in annealing metal), gradually converging to a near-optimal solution.

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–Modera te	Near Optimal	Varies
Simulated Annealing	≤ 1000	Moderate	Near Optimal	Varies

VI. Algorithm Used

(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 $O(n^2 \cdot 2^n)$, which is significantly faster and more efficient for medium-sized instances (e.g. n ≤ 25).

Algorithm Overview

The idea is to build solutions incrementally by remembering the shortest paths to subsets of cities. The algorithm uses memoization to store and reuse results of subproblems, avoiding redundant calculations.

We define:

- C(S,j): the minimum cost to reach city jjj from city 0, having visited all cities in subset S⊆{1,2,...,n−1} (excluding city 0), and ending at city j.

The recurrence relation is:

$$C(S,j) = \min_{i \in S,\ i \neq j} [C(s \setminus \{j\}, i) + d(i,j)]$$

Where:

- S is a subset of cities visited
- j is the current city
- d(i,j) is the Euclidean distance between city i and city j

The final solution is:

$$\min_{j \in \{1, \dots, n-1\}} [C(\{1, \dots, n-1\}, j) + d(j, 0)]$$

Figure 1: Held-Karp Subproblem Dependency Graph

Below is a conceptual diagram showing how Held-Karp breaks down the problem into smaller subproblems and reuses results:

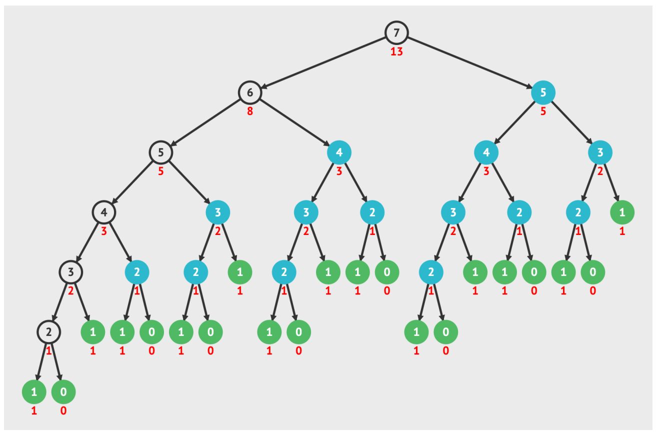


Figure 1: The problem is decomposed into smaller overlapping subproblems with cities grouped in subsets. Arrows represent recursive dependencies used in memoization.

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	$O(n!)$	$O(n^2 \cdot 2^n)$
Space Complexity	$O(1)$	$O(n \cdot 2^n)$
Optimality	✓ (Optimal)	✓ (Optimal)
Practical Input Size	$n \leq 10$	$n \leq 25$
Performance	Exponential and slow	Faster due to memoization

(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

Example

Given 5 cities with the following distance matrix:

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.)

Key Advantages

- **Speed:** Nearest Neighbor has linear time complexity, making it extremely fast even for large datasets with thousands of cities.
- **Simplicity:** Easy to implement and understand, requiring minimal computational resources and memory.
- **Scalability:** Can handle very large problem instances where exact algorithms become computationally infeasible.
- **Real-time Application:** Suitable for scenarios requiring quick, approximate solutions rather than optimal ones.

Why Nearest Neighbor is Better (Compared to Exact Algorithms)

Metric	Exact Algorithms (Held-Karp)	Nearest Neighbor
Time Complexity	$O(n^2 \cdot 2^n)$	$O(n^2)$
Space Complexity	$O(n \cdot 2^n)$	$O(n)$
Optimality	✓ (Optimal)	✗ (Approximate)
Practical Input Size	$n \leq 25$	**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

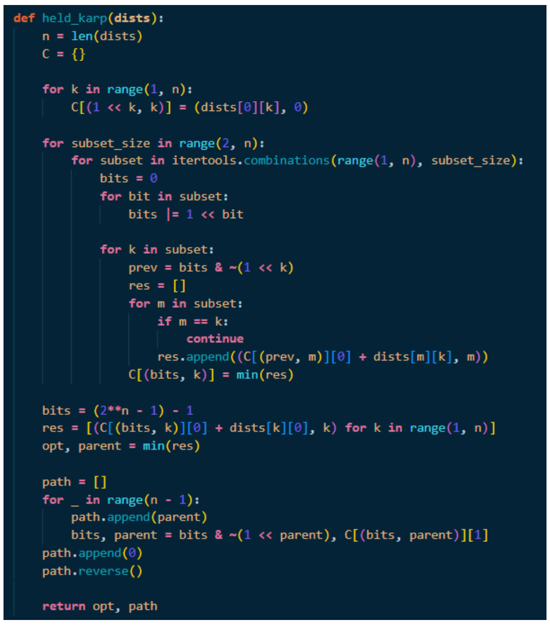
- **Starting Point Sensitivity:** Results can vary significantly depending on the chosen starting city

VII. Discussion

In this project, two different algorithms were implemented to solve the Travelling Salesman Problem (TSP): the Held-Karp algorithm and the Nearest Neighbor algorithm. Both methods aim to find a route that visits all cities in Malaysia and also the Kulliyyah in IIUM with the shortest possible distance, but they differ in approach, accuracy, and efficiency.

(1) Held-Karp Algorithm

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 cities. This method guarantees finding the optimal (shortest possible) route, but its computational cost grows exponentially with the number of nodes.



```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
```

How it works:

1. It uses **bit masking** to represent subsets of cities.
2. The dictionary `C` stores the minimum cost to reach a subset of cities ending at city `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 cities and returns to the starting city.
5. The path is reconstructed in reverse by tracking the parent city that led to each state.

Advantages:

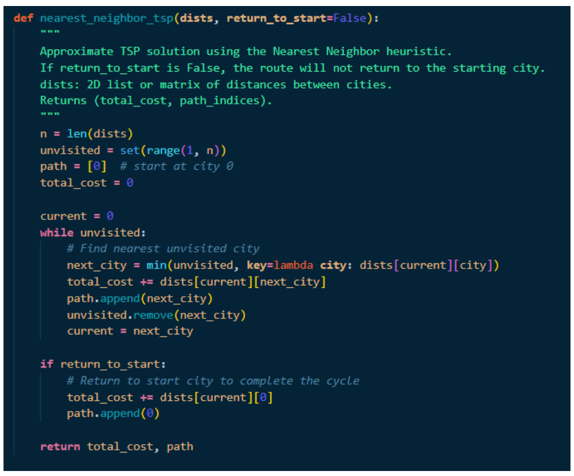
- Produces the exact shortest route.
- Suitable for small datasets.

Disadvantages:

- Not practical for large datasets due to high time and space complexity.

(2) Nearest Neighbour Algorithm

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.



```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
```

How it works:

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

Advantages:

- Very fast and easy to implement.
- Suitable for large datasets where exact methods like Held-Karp are too slow.

Disadvantages:

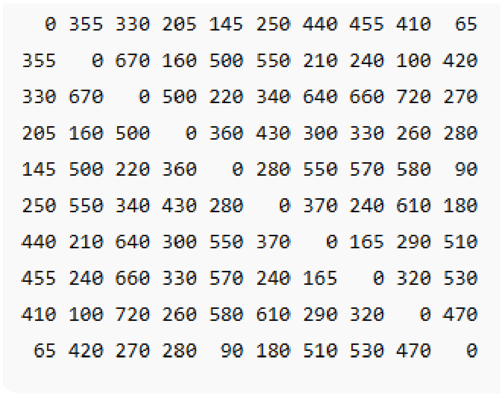
- May result in a **suboptimal solution** (longer total distance than optimal path).
- Can easily get "trapped" in local minima, especially in certain city distributions.

VI. Results & Analysis

In this project, we implemented the Travelling Salesman Problem (TSP) algorithm to determine the most efficient travel route across Malaysian cities based on a given distance matrix (in kilometres) and also the Kulliyyah buildings in IIUM Gombak. The cities included in the analysis were Kuala Lumpur, Seremban, Malacca, Johor Bahru, Kuantan, Kuala Terengganu, Kota Bharu, Alor Setar, Penang, and Ipoh, while as for kulliyyah buildings, there were AIKOL, KENMS, CELPAD, KIRKHS, KOED, KOE, KAED and KICT.

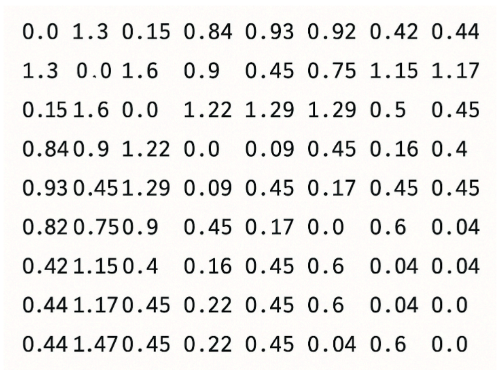
(1) Distance Matrix

The distance matrix represents the pairwise road distances (in KM) between all ten cities. Each cell [i] [j] indicates the shortest road distance from city i to city j. Below is the summary matrix:



```
  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
```

Image 8.1: The distance matrix of pairwise road distances (in KM) between all ten cities in Malaysia

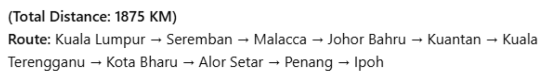


```
0.0  1.3  0.15 0.84 0.93 0.92 0.42 0.44
1.3  0.0  1.6  0.9  0.45 0.75 1.15 1.17
0.15 1.6  0.0  1.22 1.29 1.29 0.5  0.45
0.84 0.9  1.22 0.0  0.09 0.45 0.16 0.4
0.93 0.45 1.29 0.09 0.45 0.17 0.45 0.45
0.82 0.75 0.9  0.45 0.17 0.0  0.6  0.04
0.42 1.15 0.4  0.16 0.45 0.6  0.04 0.04
0.44 1.17 0.45 0.22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

Image 8.2: The distance matrix of pairwise road distances (in KM) between all eight Kulliyyah in IIUM

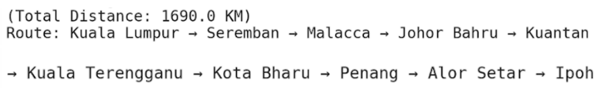
(2) Optimal Travel Path & Route

The method we used for the TSP optimization algorithm, the most cost-effective path as follows:



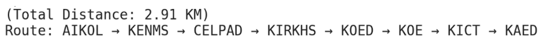
(Total Distance: 1875 KM)
Route: Kuala Lumpur → Seremban → Malacca → Johor Bahru → Kuantan → Kuala Terengganu → Kota Bharu → Alor Setar → Penang → Ipoh

Image 8.3: The total distance between cities for the Held Karp Algorithm



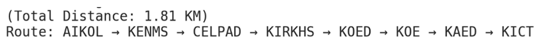
```
(Total Distance: 1690.0 KM)
Route: Kuala Lumpur → Seremban → Malacca → Johor Bahru → Kuantan

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

Image 8.4: The total distance between cities for the Nearest-Neighbour Algorithm



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

Image 8.5: The total distance between Kulliyyah in IIUM for the Held Karp Algorithm



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

Image 8.6: The total distance between Kulliyyah in IIUM for the Nearest-Neighbour Algorithm

This path minimises the total travel distance while ensuring that each city is visited exactly once before returning to the origin.

(3) 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

(4) Estimated Travel Time

Assuming an average driving speed of 80 km/h, the estimated travel time between cities was calculated and summed using the Held Karp & Nearest Neighbor Algorithm:

1. Dataset for the estimated travel time between major cities in Malaysia (in hours) using the Held Karp Algorithm

Travel route from major cities in Malaysia	Hours of speed (in km/h)
Kuala Lumpur → Seremban	0.8 hours
Seremban → Malacca	1.1 hours
Malacca → Johor Bahru	2.8 hours
Johor Bahru → Kuantan	4.3 hours
Kuantan → Kuala Terengganu	3.0 hours
Kuala Terengganu → Kota Bharu	2.1 hours
Kota Bharu → Alor Setar	3.6 hours
Alor Setar → Penang	1.3 hours
Penang → Ipoh	2.0 hours
Total Travel Time	21.0 hours

2. Dataset for the estimated travel time between major cities in Malaysia (in hours) using the Nearest-Neighbor Algorithm

Travel route from major cities in Malaysia	Hours of speed (in km/h)
Kuala Lumpur -> Seremban	0.8 hours
Seremban -> Malacca	1.1 hours
Malacca -> Johor Bahru	2.8 hours
Johor Bahru -> Kuantan	4.3 hours
Kuantan -> Kuala Terengganu	3.0 hours
Kuala Terengganu -> Kota Bharu	2.1 hours
Kota Bharu -> Penang	2.6 hours
Penang -> Alor Setar	1.3 hours
Alor Setar -> Ipoh	1.3 hours
Total Travel Time	21.3 hours

3. Dataset for the estimated travel time between major kulliyyahs in IIUM (in minutes) using the Held-Karp Algorithm

Travel route from major kuliyyahs in IIUM	Hours of speed (in minutes)
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

4. Dataset for the estimated travel time between major kulliyyahs in IIUM (in minutes) using the Nearest-Neighbor Algorithm

Travel route from major kuliyyahs in IIUM	Hours of speed (in minutes)
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

(5) 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.

(6) Application-Specific Algorithm Recommendations

Logistics and Supply Chain Management: A hybrid strategy is recommended where Held-Karp is used for small route optimization (≤15 locations) in cost-critical operations, while Nearest Neighbor handles large-scale distribution and real-time fleet management, as 1-2% route optimization can lead to thousands of dollars in annual fuel savings.

Robotics and Autonomous Systems: When choosing algorithms for robots and self-driving systems, the decision should be based on what the robot needs to do. The Held-Karp algorithm works best when the robot operates in places that don't change much, and when you need to save battery power to make the robot work longer. The Nearest Neighbor algorithm is better when the robot needs to avoid obstacles that move around, or when many robots work together and need to calculate new paths quickly.

Delivery Route Planning: Package and food delivery services should use different methods for different types of delivery. The Nearest Neighbor algorithm works best for same-day delivery and emergency deliveries when speed matters most. The Held-Karp algorithm is better for planned deliveries where you can take time to find the best route. For neighborhood deliveries, a combined approach works well where you use the exact method for small groups of stops, then use the faster method to connect between different neighborhoods.

Campus and Urban Navigation: For navigation in schools and cities, the choice should be based on how big the area is. The Held-Karp algorithm works well for small campuses with 20 locations or fewer and tourist routes where you want to give people the best experience. The Nearest Neighbor algorithm is better for large universities and public bus systems where schedules need to change quickly.

Manufacturing and Production Planning: In factories and production settings, the choice should be based on what's most important - quality or speed. The Held-Karp algorithm is best for precise work like drilling circuit boards, manufacturing processes, and maintenance schedules, where being exact saves money and prevents equipment wear. The Nearest Neighbor algorithm works better in busy warehouses where workers need to pick items quickly, and small improvements in route quality aren't worth the extra time.

VII.CONCLUSION

This research successfully implemented and evaluated both exact and heuristic approaches to solving the Travelling Salesman Problem (TSP) using real-world datasets from Malaysian cities and the IIUM Gombak campus kulliyyahs. The comparative analysis between the Held-Karp algorithm and the Nearest Neighbor Algorithm provides valuable insights into the fundamental trade-offs between solution optimality and computational efficiency in combinatorial optimization problems.

Key Findings

The **Held-Karp algorithm** consistently found the best possible routes for both datasets, achieving 21.0 hours for Malaysian cities and 29.0 minutes for IIUM kulliyyahs. While it guarantees optimal solutions, it becomes slower as the number of cities increases, making it suitable only for smaller problems (up to 25 cities).

The **Nearest Neighbor Algorithm** was much faster, producing results almost instantly for both datasets. However, it found slightly longer routes: 21.3 hours for Malaysian cities and 26.0 minutes for IIUM kulliyyahs. This algorithm works well when speed is more important than finding the perfect route.

Practical Implications

The choice of algorithm depends on what you need most. Use Held-Karp when you need the shortest possible route and have time to wait for the calculation, such as in cost-sensitive logistics operations. Use Nearest Neighbor when you need quick results for real-time applications or when dealing with many locations.

Both algorithms worked well on the smaller IIUM campus dataset, showing that simple methods can be sufficient for local navigation systems.

Limitations and Future Work

This study had some limitations. We only tested small datasets (10 cities and 8 buildings), so we don't know how well these methods work with hundreds of locations. We also only looked at basic distance calculations and didn't consider other factors like traffic, fuel costs, or road conditions.

Future studies could test larger datasets, include real-time traffic data, and compare other solving methods like Genetic Algorithms or Simulated Annealing.

Final Remarks

This research shows that choosing the right algorithm depends on your specific needs. For small problems where you need the best solution, use exact methods like Held-Karp. For larger problems or when you need fast results, heuristic methods like Nearest Neighbor work well enough.

Our study successfully compared both approaches using real Malaysian city data and campus locations. The results help understand when to use each method and provide useful insights for transportation planning and route optimization in Malaysia.

References

Abid, M. M., & Muhammad, I. (2015). Heuristic approaches to solve the travelling salesman problem. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, *2–2*, 390–396. https://doi.org/10.11591/telkomnika.v15i2.8301

Alkadri, S. P. A., & Fakhruzi, I. (2024). Travel application itinerary using the travelling salesman problem method and the Held-Karp algorithm. *TEKNOSAINS: Jurnal Sains, Teknologi & Informatika*, *1*, 103–111. https://doi.org/10.37373/tekno.v11i1.682

Benhida, S., Mir, A., & Laboratory of Industrial Engineering and Computer Science, National School of Applied Sciences ENSA, B.P. 1136, AGADIR, MOROCCO. (2017). Solving the Travelling Salesman Problem with Exact Methods. *International Journal of Enhanced Research in Science, Technology & Engineering*, 61.

Chauhan, Ankit. "Random Forest Classifier and Its Hyperparameters." *Analytics Vidhya*, 23 Feb. 2021, medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6.

Prasetyo, Y. B., & Romli, Moh. A. (2023). Implementation of Held-Karp algorithm to calculate the shortest route path. In International Journal of New Technology and Research (IJNTR), *International Journal of New Technology and Research (IJNTR)* (Vol. 9, Issue 12, pp. 01–07) [Journal-article]. https://doi.org/10.31871/IJNTR.9.12.4

Rahman, M. Z., Sheikh, S. R., Islam, A., & Rahman, M. A. (2024). Improvement of the nearest neighbor heuristic search algorithm for the travelling salesman problem. *Journal of Engineering Advancements*, 19–26. https://doi.org/10.38032/jea.2024.01.004