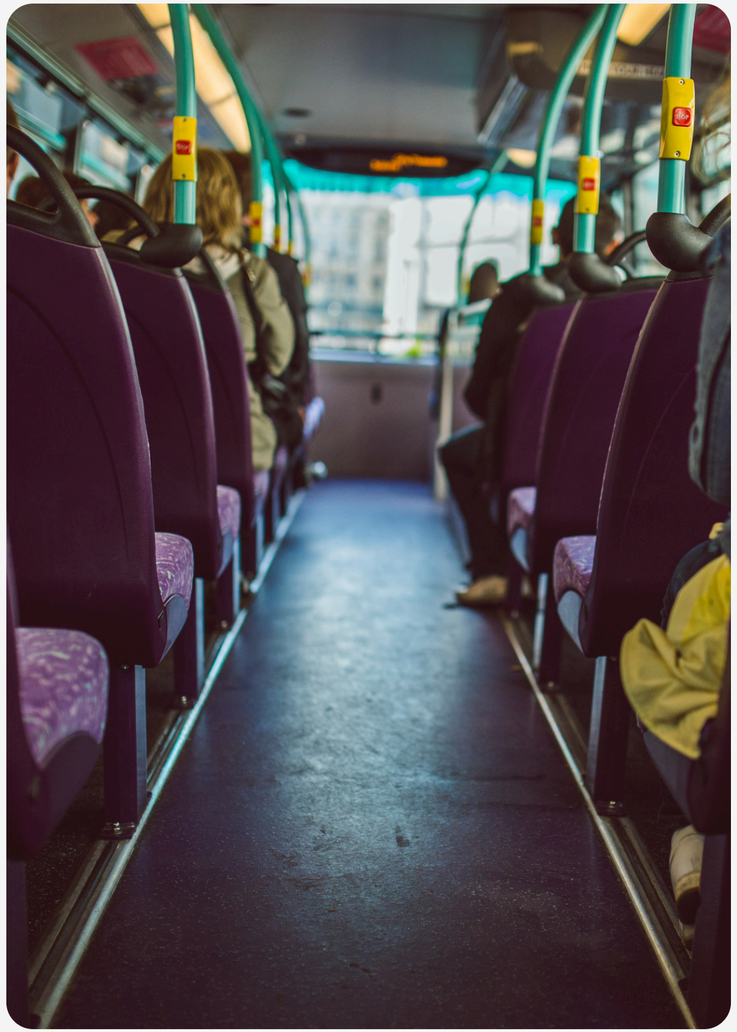
ROUTE OPTIMIZATION OF RAPIDKL BUS USING ACO ALGORITHM

TABLE OF CONTENTS

INTRODUCTION

RapidKL bus routes currently face inefficiencies leading to longer travel times. To address this, RapidKL is exploring Ant Colony Optimization (ACO) – an algorithm inspired by ant foraging behavior. ACO will be used to identify the most efficient routes for each bus, minimizing travel times. This involves finding the optimal sequence of bus stops, considering factors like distance (calculated using latitude and longitude) and potential variations in travel speed.



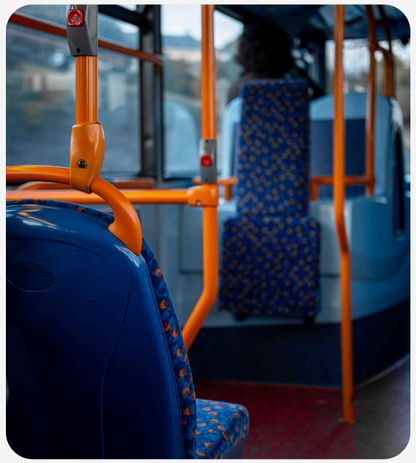
PROBLEM STATEMENT

RapidKL bus routes suffer from inefficiencies that result in:

- Increased travel times
- Passenger frustration
- Reduced ridership
- Scheduling issues

RapidKL aims to enhance its bus service efficiency by optimizing the routes taken by its buses.

OBJECTIVE



Primary Objective: To minimize the total travel time for each bus route.

RapidKL wants to accomplish:

- **Shorter passenger journeys** result in happier passengers since they commute for less time.
- **Improved schedule adherence**: Bus dependability is increased when routes are shorter since it is simpler for buses to stick to their timetables.
- **Possibility of higher ridership**: Quicker travel times may draw in new passengers who might have otherwise chosen quicker options.

DETAILED DESCRIPTION OF PROBLEM

It involves determining the most efficient bus routes to minimize total travel time. The problem entails allocating routes to buses so that the overall travel time is minimized while adhering to various constraints. Complexities include the permutation of routes and scheduling buses to optimize travel times, while uncertainties involve varying traffic conditions and passenger demands.

The objective function used:

Euclidean distance matrix

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

The constraints:

- Ensuring each stop is visited exactly once.
- Forming a cycle by returning to the starting point.
- Avoiding route overlaps.

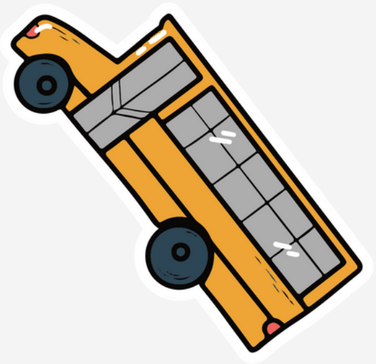
The details of objective function:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

- Node : coordinate bus stop represented as (y, x)
- Alpha : influence of pheromone
- Beta : influence of heuristic information
- Evaporation rate : the pheromone trail decreases over time
- Ants : A single route

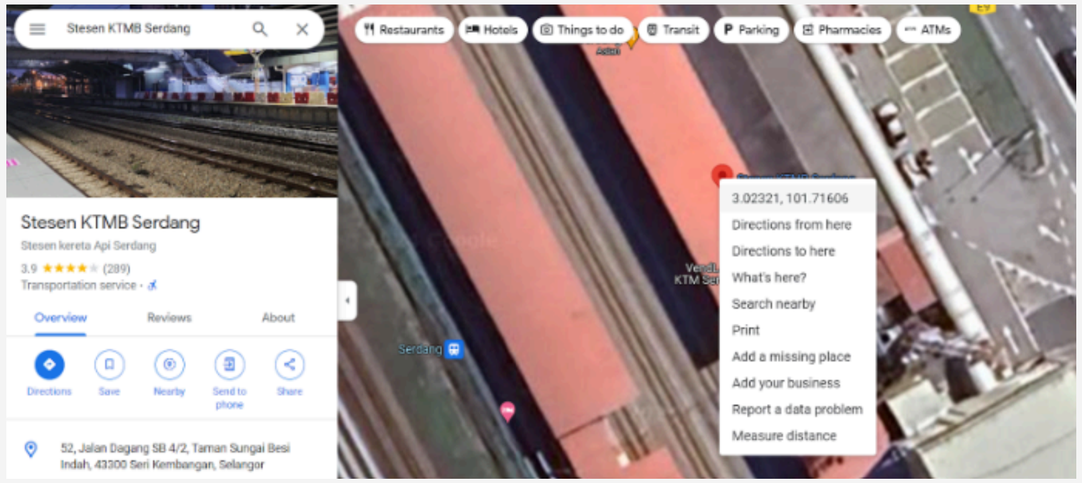
RELATED WORKS

Study	Approach	Objective	Results
Nabila Dwi Indria (2021)	Used the store and coordinate points for each Alfamidi outlet location in Palu City.	Minimizing the total distance travelled, improving time and cost.	The shortest route for the distribution of goods is 86.98 km long.
Israel Edem (2024)	Used a nature-inspired search strategy, specifically the Kestrel-based Search Algorithm (KSA)	To optimize the location of emergency response wagons to minimize the expected time it takes to reach the scene of a railway crossing accident.	The Kestrel-based Search Algorithm (KSA) outperformed the comparative algorithms, which were the Bat Algorithm (BAT) and Ant Colony Optimization (ACO).
Abira Massi Armond (2022)	Used coordinates (latitude and longitude) of each bus stop and the distances between each pair of bus stops.	To find the shortest and most efficient bus routes by reducing operational costs and improving the punctuality of the bus service.	The optimal parameters for the ACO algorithm were determined through various tests, and the results showed a distance of 29.8 km.
Mingyan Li (2019)	Calculated using a formula that takes into account the pheromone concentration and the heuristic information.	Finding the shortest path by minimizing the travel distance, reducing travel time, fuel consumption, and overall transportation costs.	The shortest distance found was 88.10 units, and the optimal path was discovered after approximately 55 iterations.



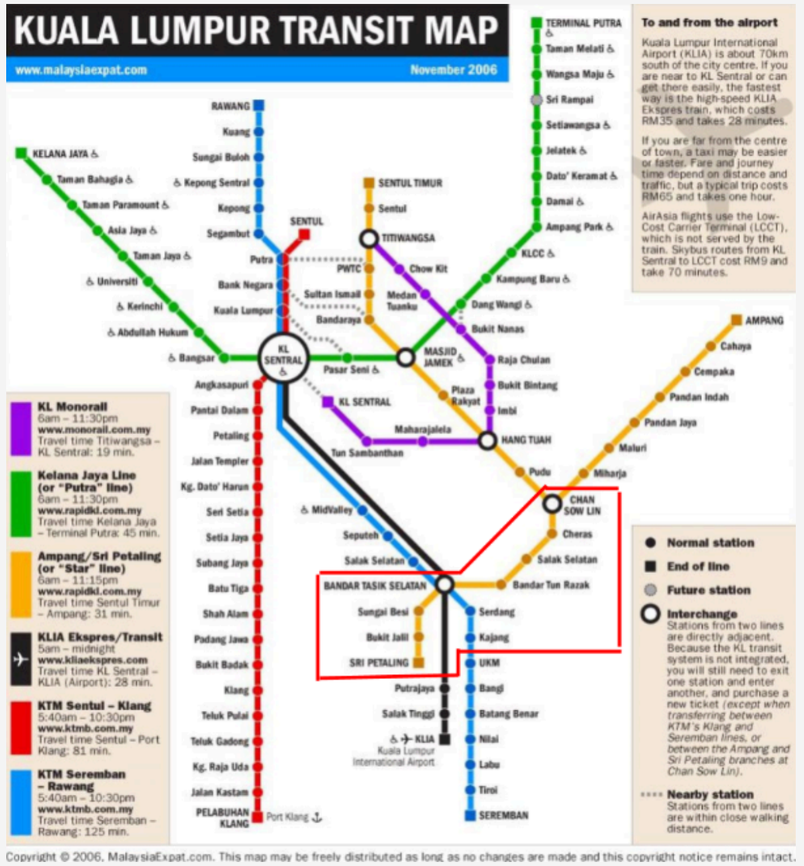
SOURCE DATASET

Data Collection : In google Map



Rapid KL Bus MAP:

https://maps-kuala-lumpur.com/rapidkl-map

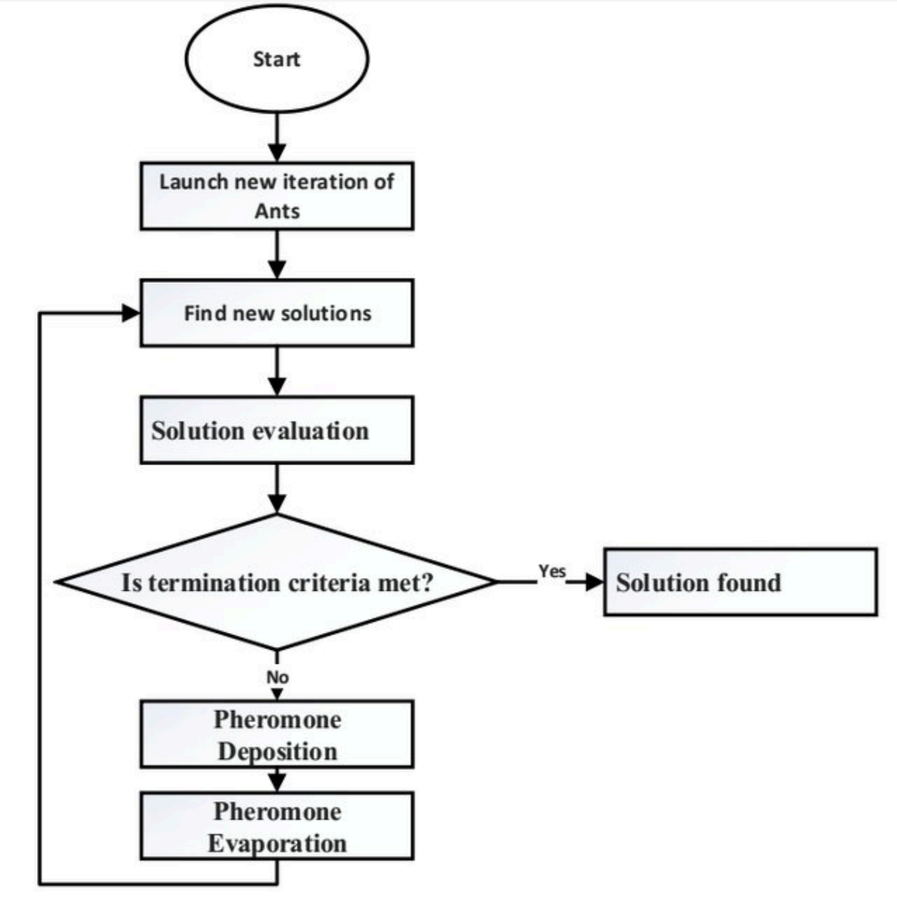


DATASET

A dataset of coordinates for the bus stops are created consisting of 3 attributes which are place, longitude, latitude and place

Latitude	Longitute	Place
3.016793	101.7835	Kajang
3.023206	101.7161	Serdang
3.08962	101.7125	Bandar Tun Razak
3.10203	101.7065	Salak Selatan
3.11243	101.7145	Cheras
3.1289	101.716	Chan Sow Lin
3.072	101.7096	Bandar Tasik Selatan
3.0639	101.708	Sungai Besi
3.058049	101.6914	Bukit Jalil
3.0614	101.6871	Sri Petaling
3.016793	101.7835	Kajang

ALGORITHM DESIGN



ALGORITHM DESIGN

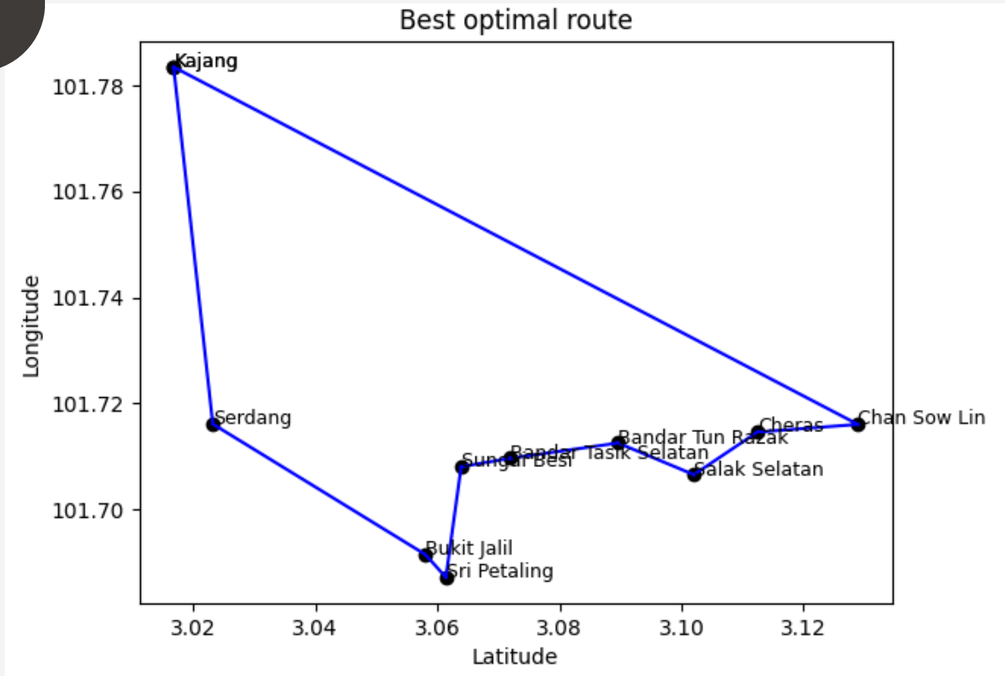
```python
# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Data RapidKL Bus.csv')
```

```python
# Extract coordinates and place names
coord_x = data['Latitude'].tolist()
coord_y = data['Longitute'].tolist()
places = data['Place'].tolist()

# Step 1. Initialization of parameters
stop_num = len(coord_x)  # the number of stops
dis = [[0 for _ in range(stop_num)] for _ in range(stop_num)]  # distance matrix
epsilon = 1e-10  # Small value to avoid division by zero

# Calculate the Euclidean distance matrix
for i in range(stop_num):
    for j in range(i, stop_num):
        temp_dis = math.sqrt((coord_x[i] - coord_x[j]) ** 2 + (coord_y[i] - coord_y[j]) ** 2)
        dis[i][j] = temp_dis if temp_dis != 0 else epsilon  # Avoid zero distance
        dis[j][i] = dis[i][j]

# Initialize pheromone matrix
pheromone = [[1 for _ in range(stop_num)] for _ in range(stop_num)]
iter_best = []  # the shortest path of each iteration
best_path = []
best_length = 1e6

# Function to construct a new path
def construct_path(dis, pheromone, alpha, beta):
    path = [0]
    cur_node = 0
    unvisited_stops = [i for i in range(1, len(dis))]
    for _ in range(len(dis) - 1):
        roulette_pooling = []
        for stop in unvisited_stops:
            roulette_pooling.append(math.pow(pheromone[cur_node][stop], alpha) * math.pow(1 / dis[cur_node][stop], beta))
        index = roulette(roulette_pooling)
        cur_node = unvisited_stops[index]
        path.append(cur_node)
        unvisited_stops.pop(index)
    path.append(0)
    return path

# Function to calculate the length of the path
def cal_dis(dis, path):
    length = 0
    for i in range(len(path) - 1):
        length += dis[path[i]][path[i + 1]]
    return length
```

```python
[24] for _ in range(iterations):
        ant_path = []
        ant_path_length = []
        for _ in range(pop):
            new_path = construct_path(dis, pheromone, alpha, beta)
            new_length = cal_dis(dis, new_path)
            ant_path.append(new_path)
            ant_path_length.append(new_length)
        iter_best_path_length = min(ant_path_length)
        if iter_best_path_length < best_length:
            best_length = iter_best_path_length
            best_path = ant_path[ant_path_length.index(iter_best_path_length)]
        iter_best.append(best_length)

        # Update pheromone
        for i in range(stop_num):
            for j in range(i, stop_num):
                pheromone[i][j] *= (1 - rho)
                pheromone[j][i] *= (1 - rho)
        for i in range(pop):
            delta = Q / ant_path_length[i]
            path = ant_path[i]
            for j in range(len(path) - 1):
                pheromone[path[j]][path[j + 1]] += delta
                pheromone[path[j + 1]][path[j]] += delta

    # Convert best path indices to place names
    best_path_places = [places[i] for i in best_path]

    # Remove duplicate start node if present
    if best_path_places[0] == best_path_places[1]:
        best_path_places.pop(1)

    # Ensure the best path starts and ends at the same place
    if best_path_places[0] != best_path_places[-1]:
        best_path_places.append(best_path_places[0])

    # Plot convergence graph and display shortest distance (best path) obtained
    plt.figure()
    plt.scatter(coord_x, coord_y, color='black')
    for i in range(len(best_path) - 1):
        temp_x = [coord_x[best_path[i]], coord_x[best_path[i + 1]]]
        temp_y = [coord_y[best_path[i]], coord_y[best_path[i + 1]]]
        plt.plot(temp_x, temp_y, color='blue')
    for i, place in enumerate(places):
        plt.text(coord_x[i], coord_y[i], place, fontsize=9)
    plt.xlabel('Latitude')
    plt.ylabel('Longitude')
    plt.title('Best optimal route')
    plt.show()
```

ALGORITHM PERFORMANCE

1

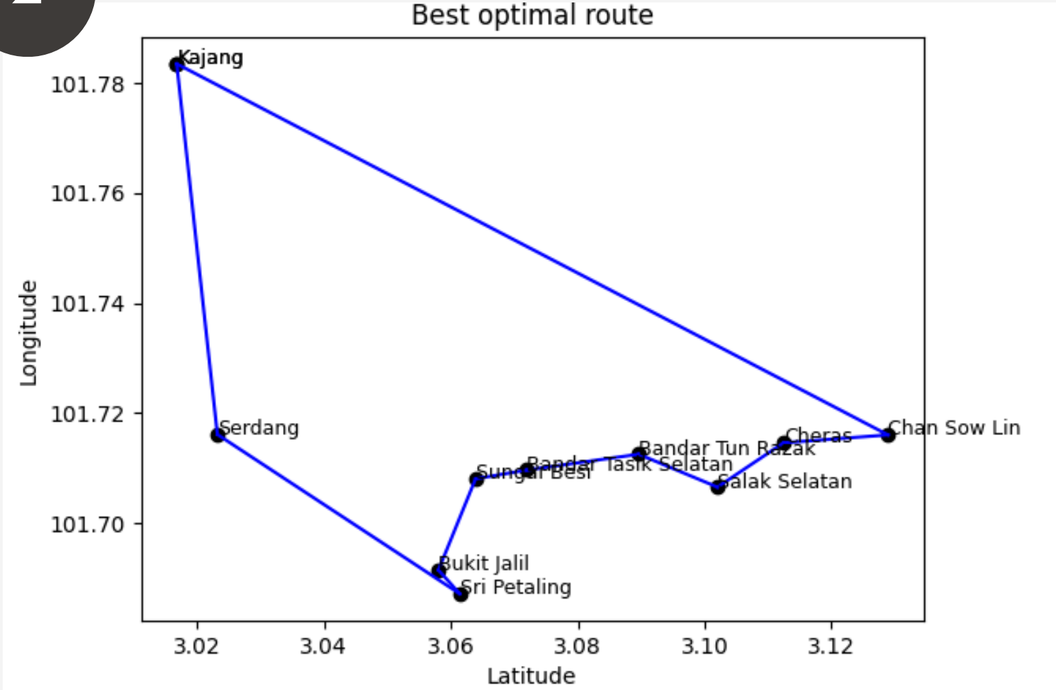


Shortest length: 0.3373

Alpha	1.0
Beta	5.0
rho (Evaporation rate)	0.5
Q (pheromone_deposit_amount)	100
iterations	100
pop (num ants)	10

ALGORITHM PERFORMANCE

2

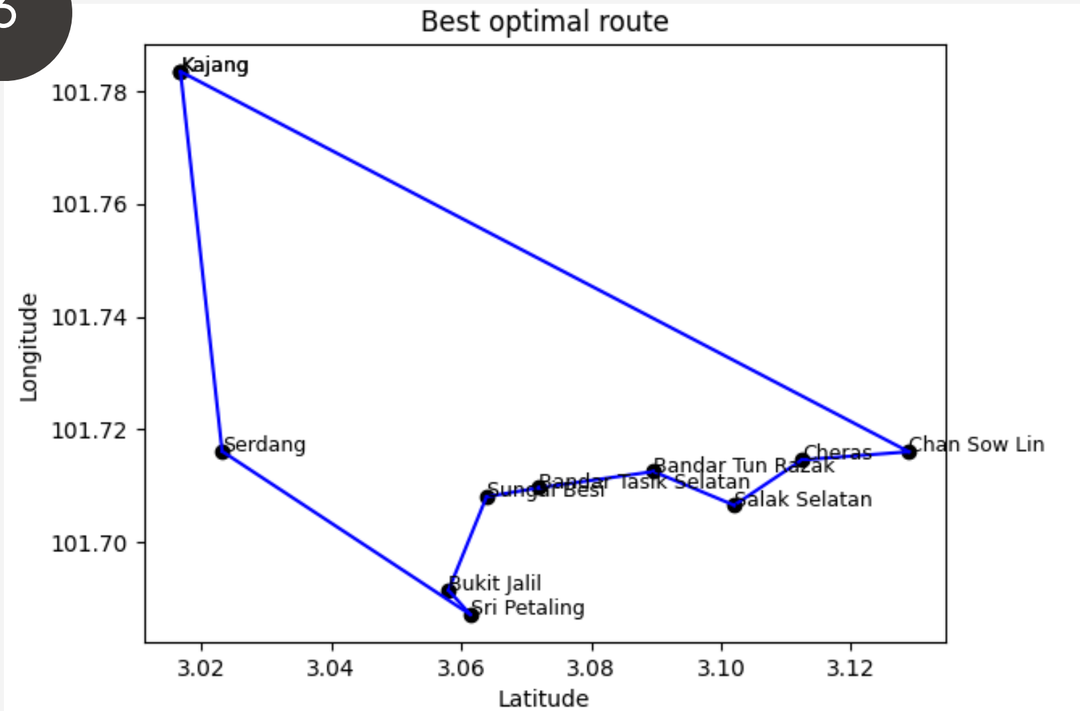


Shortest length: 0.3391

Alpha	1.0
Beta	5.0
rho (Evaporation rate)	0.5
Q (pheromone_deposit_amount)	100
iterations	50
pop (num ants)	10

ALGORITHM PERFORMANCE

3



Shortest length: 0.3391

Alpha	1.0
Beta	5.0
rho (Evaporation rate)	1
Q (pheromone_deposit_amount)	100
iterations	20
pop (num ants)	10

COMPARISON ALGORITHM PERFORMANCE

	Performance 1	Performance 2	Performance 3
Alpha	1.0	1.0	1.0
Beta	5.0	5.0	5.0
rho (Evaporation rate)	0.5	0.5	0.7
Q (pheromone_deposit_amount)	100	100	100
iterations	100	50	20
pop (num ants)	10	10	10
Shortest length	0.3373	0.3391	0.3391

CONCLUSIONS

- The ACO algorithm effectively finds near-optimal or optimal routes for the bus
- Increasing iterations tends to improve the accuracy of the solution but may require more computation time.
- For future improvement we can expand more rapidkl bus stop location

THANK YOU

HAVE A GOOD DAY:)