

TABLE OF CONTENTS

ABSTRACT	1
1.0 INTRODUCTION	2
2.0 PROBLEM STATEMENT	3
3.0 DETAIL DESCRIPTION OF PROBLEM	3
4.0 RELATED WORKS	5
5.0 SOLUTION REPRESENTATION	8
6.0 ALGORITHM DESIGN	9
7.0 PERFORMANCE OF ALGORITHM	22
8.0 CONCLUSION	27
REFERENCES	28

ABSTRACT

This research investigates the use of Particle Swarm Optimization (PSO) for optimizing routes in the Optimization of Japan Airport Visit using Particle Swarm Optimization Algorithm. The problem involves identifying the most efficient order to visit multiple airports while reducing travel distance. PSO, a nature-inspired algorithm, is utilized to efficiently explore the combinatorial solution space of the Airport-Visiting TSP. Its adaptability to real-time changes and rapid convergence to near-optimal solutions make it ideal for the dynamic conditions of airports. The findings show that PSO successfully minimizes travel distances. Overall, this study highlights the potential of PSO to significantly improve route optimization in dynamic airport settings.

1.0 INTRODUCTION

The Traveling Salesman Problem (TSP), a renowned combinatorial optimization challenge, remains a significant area of interest for researchers and practitioners due to its extensive applicability across diverse fields. TSP lies at the heart of routing and logistics optimization, originating from the pursuit of the shortest closed loop that visits a specified set of cities exactly once. This study explores the application of Particle Swarm Optimization (PSO), a powerful and adaptable algorithm, to address the complexities of TSP and develop effective solutions for this iconic problem.

TSP remains a cornerstone of optimization research, being NP-hard and relevant to domains like manufacturing, logistics, and transportation. PSO offers a robust alternative to traditional algorithms, which often face limitations when dealing with large-scale scenarios or real-world constraints. Inspired by the collaborative behavior of social creatures, PSO naturally facilitates dynamic problem-solving through particle interaction within a search space.

This research emphasizes employing PSO as a heuristic approach to efficiently navigate the TSP solution space. The objective is to leverage PSO's strengths in exploration and exploitation to find optimal solutions for TSP scenarios of varying complexity. PSO's adaptability to diverse problem landscapes positions it as an effective tool for overcoming computational challenges associated with determining the best paths among interconnected points. For this study, these points represent the coordinates of Japanese airports.

By applying PSO to the Optimization of Japan Airport Visit using Particle Swarm Optimization Algorithm, this work demonstrates its potential to optimize the intricate network of routes connecting airports. The algorithm's iterative refinement of solutions aligns with the dynamic nature of air transport systems. Through this study, we aim to contribute to the growing body of research on airline route optimization, showcasing how PSO can address the unique challenges posed by the Optimization of Japan Airport Visit using Particle Swarm Optimization Algorithm in Japanese airport networks.

2.0 PROBLEM STATEMENT

Optimization of Japan Airport Visit using Particle Swarm Optimization Algorithm poses a timeless and universal challenge: finding the most efficient route that visits a set of locations and returns to the starting point. The Optimization of Japan Airport Visit using Particle Swarm Optimization Algorithm extends this concept to the domain of air transport route management, presenting a complex optimization scenario. Here, the "salesman" represents a flight, while the "cities" correspond to various airports.

3.0 OBJECTIVE

- i. To identify the optimal sequence of airports for the flight to visit, beginning and ending at the same airport.
- ii. To minimize the total travel distance.

3.0 DETAIL DESCRIPTION OF PROBLEM

The Optimization of Japan Airport Visit using Particle Swarm Optimization Algorithm involves finding the most efficient route for a salesman to visit a set of airports while minimizing the total travel distance. Using Particle Swarm Optimization (PSO) to tackle this problem aims to minimize the total distance while ensuring each airport is visited exactly once, with the salesman returning to the starting point. In PSO, particles represent potential solutions, each defined by a sequence specifying the order of airport visits.

The objective function can be described as follows:

$$f(x_k) = \sum_{i=1}^{n-1} d_{x_{ki}x_{ki+1}} + d_{x_{kn}x_{k1}}$$

The PSO algorithm begins by initializing a population of particles, evaluating their fitness based on the total travel distance, and iteratively updating their positions and velocities. The process is guided by each particle's personal best (pBest) and the global best (gBest) positions, driving the search for an optimized route. Through these steps, PSO efficiently identifies a solution that represents the optimal sequence of airport visits.

Problem Definition in the Context of PSO:

Objective:

- Minimize the total travel distance while visiting each airport exactly once and returning to the starting airport.

Variables:

1. Particle: Represents a potential solution, defined as a sequence of airport visits.
2. Position: A permutation of airports indicating the visit order.

Key PSO Components:

3. Particles: A population of potential solutions, where each particle encodes a candidate route.
4. Fitness Function: Evaluates the quality of a solution by calculating the total travel distance; the objective is to minimize this distance.
5. Velocity: Represents the change in a particle's position, influencing how it explores the solution space.
6. Personal Best (pBest): The best-known position for each particle based on its performance history.
7. Global Best (gBest): The best-known position among all particles in the swarm.

By iteratively refining solutions through these components, PSO effectively navigates the solution space, producing an optimized route for the travelling salesman to visit airports in the AVTSP.

4.0 RELATED WORKS

A study by Yifei et al. (2018) investigated path optimization for intelligent welding robots using Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). The objective was to minimize the distance between welding spots, denoted as d_{ij} , where i and j represent different welding spots. Both algorithms successfully achieved optimal or near-optimal welding paths through iterative calculations. The optimal path length was determined to be 10.3235m, with PSO reaching a near-optimal value of 11.8829m compared to GA's 12.5514m when iterations were set to 400. Furthermore, PSO demonstrated shorter computation times, highlighting its superior efficiency and search performance over GA in this application.

Danlu and Zhongfeng (2020) conducted a study on logistics distribution route optimization using Particle Swarm Optimization (PSO), with a focus on minimizing total transportation mileage. The study introduced the Mountain Climbing procedure as an enhancement to the standard PSO algorithm. This modification resulted in faster convergence and an optimized dissemination distance of 67.5km, significantly reducing the pre-optimized distance of 72.5km. The findings demonstrate that PSO is well-suited for addressing automobile routing issues and that its performance can be further improved through the integration of the Mountain Climbing procedure.

Mahi et al. (2015) conducted experiments with various optimization methods to solve the Travelling Salesman Problem (TSP), incorporating Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and the 3-Opt heuristic method. In this study, the PSO algorithm was used to determine the optimal values of α and β , which are crucial for city selection in the ACO algorithm, influencing the significance of inter-city pheromones and distances. The 3-Opt algorithm was employed to enhance city selection operations that the ACO algorithm could not improve due to local minima. Through multiple experiments, the proposed hybrid approach outperformed the compared methods in most cases, demonstrating superior solution quality and robustness. For example, the optimized distance on Kroa200, a test problem from TSPLIB, was reduced from 29,816 to 29,468.

Adam et al. (2010) examined the drill route optimization problem by applying the Particle Swarm Optimization (PSO) algorithm. The study focused on minimizing the distance traveled

by the PCB robotic drill to complete its task. The results demonstrated that the PSO algorithm effectively solved the complex problem while requiring minimal computational time. The proposed model was able to identify the shortest path for the robot to perform its task efficiently, using the fewest number of particles and achieving faster convergence speed.

Zheng et al. (2022) addressed the Travelling Salesman Problem (TSP) using evolutionary algorithms, specifically Particle Swarm Optimization (PSO). The study introduced a transfer learning-based PSO algorithm that leverages optimal information from historical TSP instances to guide the swarm in quickly finding optimal paths. This transfer learning-based PSO was applied to 20 typical TSP instances and compared with 12 state-of-the-art algorithms. The results showed that the proposed approach achieved better optimal paths than the other algorithms tested, demonstrating its effectiveness in solving TSP efficiently.

Jin et al. (2022) conducted research on the application of Particle Swarm Optimization (PSO) algorithms in tourism route planning and optimization. The study presented a model of the discrete PSO algorithm, which uses geographic coordinates to solve the tourism route problem. Additionally, the research combined PSO with Neural Network (NN) models to develop a tourism demand forecasting model. In this model, the PSO algorithm optimizes the weights and thresholds of the NN to enhance the precision of the predictions. The findings demonstrated that the PSO-NN-based forecasting model significantly improved prediction accuracy compared to conventional NN models.

Wang et al. (2023) proposed a power law function to enhance the weight value in the Particle Swarm Optimization (PSO) algorithm, aiming to resolve issues of unsatisfactory performance and premature convergence in complex optimization problems. The proposed algorithm was compared to the linearly decreasing weight PSO algorithm, with tests conducted on four typical single-peak and multi-peak functions. The results showed that the PSO algorithm using the power law function achieved superior optimization outcomes, demonstrating its effectiveness in improving performance in complex optimization tasks.

In a recent study, Nguyen et al. (2021) developed a comprehensive framework for Particle Swarm Optimization (PSO) called GPSO, designed to aggregate key parameters and generalize important variants of PSO for easier customization by researchers. GPSO aims to balance the exploration and exploitation aspects of PSO, both of which are crucial for achieving optimal

solutions. The study tested GPSO and probabilistic GPSO against basic PSO, using a specific cost function and termination condition to evaluate their performance.

Lastly, a study by Salman & Ali (2021) introduced a novel Best-Worst Ant System (BWAS) algorithm to solve the Travelling Salesman Problem (TSP). The authors proposed a hybrid methodology that combines Particle Swarm Optimization (PSO) with a Biogeography-Based Algorithm (BBA) to enhance solution performance. Their findings revealed that establishing an initial trail for the most optimal particle could significantly reduce the optimization process duration. The effectiveness of the proposed algorithm was evaluated by comparing it with traditional Ant Colony Optimization (ACO) and PSO-ACO techniques, demonstrating its superior performance. The research concluded that the hybrid BWAS-PSO-ACO algorithm outperformed conventional ACO and PSO-ACO methods in solving the TSP, based on criteria such as solution quality and assembly time.

5.0 SOLUTION REPRESENTATION

The solution representation in the proposed algorithm is based on a Particle Swarm Optimization (PSO) approach. Each particle in the algorithm represents a potential solution to the optimization problem, and the algorithm operates on a swarm of particles. In this project, the particles represent different permutations of the airport locations that the salesman visits.

Through iterative adjustments, the algorithm updates the particle positions based on their velocities. These velocities are influenced by two factors: the particle's personal best position (the best solution it has found so far) and the global best position (the best solution found by any particle in the swarm).

The particles' positions are updated iteratively by the algorithm, moving through the solution space and seeking the optimal permutation that results in the shortest tour for the Traveling Salesman Problem (TSP). This process allows the PSO algorithm to minimize the objective function, which is the total distance traveled.

Objective Function

The aim of solving the Optimization of Japan Airport Visit using Particle Swarm Optimization Algorithm in this project is to identify the shortest possible route that passes through a predefined set of airport locations and returns to the starting point. The goal is to minimize the total travel distance, which serves as the objective function. This objective function can be mathematically expressed as:

$$f(x_k) = \sum_{i=1}^{n-1} d_{x_{ki}x_{ki+1}} + d_{x_{kn}x_{k1}}$$

The position of a particle is represented as x_k which corresponds to a permutation of airport indices: $x_k = [x_{k1}, x_{k2}, x_{k3}, \dots, x_{kn}]$. Here, n denotes the total number of airports, and d refers to the distance between two selected airports within the permutation.

The primary goal of the Particle Swarm Optimization (PSO) algorithm is to discover the permutation x_k that minimizes the objective function. By achieving this, PSO identifies the optimal route, minimizing the total distance traveled and ensuring the most efficient solution to the problem.

6.0 ALGORITHM DESIGN

6.1 Flowchart

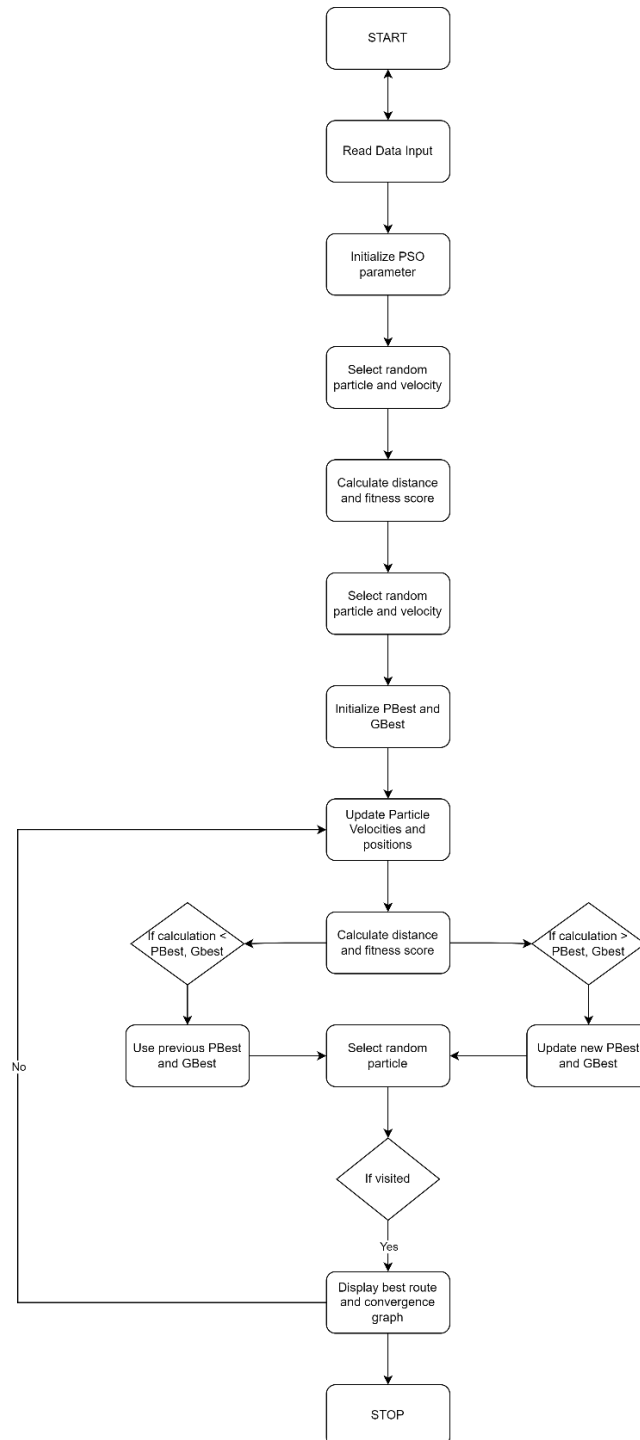


Figure 6.1 shows the flowchart for the algorithm.

The flowchart for optimizing the route in the Optimization of Japan Airport Visit using Particle Swarm Optimization Algorithm using Particle Swarm Optimization (PSO) follows a systematic process. Initially, the airport location data is input into the algorithm. The PSO parameters, including the number of particles, maximum iterations, and learning factors, are then set. Next, the particles are randomly initialized with positions representing random orders of airports to visit, and their velocities are also initialized. The global best (gBest) and personal best (pBest) are established for each particle and the swarm. The PSO main loop begins with the updating of each particle's position using the position update equation and the update of velocity using the velocity update equation, which considers both the personal and global bests. The total distance of each particle's route is computed, and the fitness score is calculated based on the distance, with shorter distances being assigned better fitness scores. If a particle's fitness score improves compared to its personal best, the personal best is updated, and similarly, if the particle's fitness is better than the global best, the global best is updated. Random particles are then selected to update their positions based on the updated best values. The algorithm continues to iterate until all airports have been visited, ensuring a valid route is formed. Once all particles have visited all airports, the program displays the best route for the salesman to complete the cycle and shows a convergence graph that illustrates the improvement in the solution over time. Finally, the process concludes with the optimized route being presented as the solution.

6.2 Coding of the Algorithm

Code	Code Explanation
<pre>import random import math import matplotlib.pyplot as plt import pandas as pd from math import radians, sin, cos, sqrt, atan2 # Seed for reproducibility random.seed(100)</pre>	<p><code>import random</code></p> <p><code>import math</code></p> <p>This line imports the random module, which provides various functions for generating random numbers. It is used to shuffle the order of the airports in the Particle class</p> <p><code>import math</code></p> <p>This line imports the math module, which provides mathematical functions defined by the C standard. It is used to calculate the distance between two airports using the spherical law of cosines</p> <p><code>import matplotlib.pyplot as plt</code></p> <p>This line imports the pyplot module from the matplotlib library, which is a plotting library used for creating static, animated, and interactive visualizations in Python. It is used to plot the best route and the convergence of the PSO algorithm</p> <p><code>import pandas as pd</code></p>

	<p>This line imports the pandas library, which is used for data manipulation and analysis. It is used to load the airport data from a CSV file into a DataFrame.</p> <pre>from math import radians, sin, cos, sqrt, atan2</pre> <p>This line imports specific functions (radians, sin, cos, sqrt, atan2) from the math module. These functions are used to calculate the distance between two airports using the spherical law of cosines.</p> <pre>random.seed(100)</pre> <p>This line ensures that the initial positions of the particles and the initial velocities are the same every time you run the program</p>
<p>Problem Definition</p> <pre># Load airports from CSV airports = pd.read_csv("/content/airports.csv")</pre>	<p>Load a CSV file containing airport data into a pandas DataFrame</p>
<p>PSO Parameters</p> <pre>class Particle: def __init__(self, airports, velocity_limit): self.airports = airports self.position = list(range(len(airports))) random.shuffle(self.position) self.velocity = [0] * len(self.position) self.best_position = self.position.copy()</pre>	<pre>class Particle:</pre> <p>Defines the Particle class, representing an individual solution in the PSO algorithm.</p> <pre>def __init__(self, airports, velocity_limit):</pre> <p>Initializes a particle with specific properties such as its position, velocity, and fitness.</p>

```
self.best_fitness = float('inf')
self.velocity_limit = velocity_limit
```

```
def distance(self, airport1, airport2):
```

```
    lat1, lon1 = radians(airport1['Lat']), radians(airport1['Long'])
    lat2, lon2 = radians(airport2['Lat']), radians(airport2['Long'])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    radius = 6371.0 # Earth radius in kilometers
    return radius * c
```

```
def fitness(self):
```

```
    fitness = 0
    for i in range(len(self.position) - 1):
        airport1, airport2 = self.airports.iloc[self.position[i]],
self.airports.iloc[self.position[i + 1]]
        fitness += self.distance(airport1, airport2)
    fitness += self.distance(self.airports.iloc[self.position[-1]],
self.airports.iloc[self.position[0]])
    return fitness
```

```
# Particle Swarm Optimization function
```

```
self.position = list(range(len(airports)))
```

Sets the initial position of the particle by creating a sequence of indices corresponding to all airports.

```
random.shuffle(self.position)
```

Randomizes the order of airports to initialize a random route for the particle.

```
self.velocity = [0] * len(self.position)
```

Initializes the particle's velocity as a list of zeros, corresponding to the number of airports.

```
self.best_position = self.position.copy()
```

Stores the particle's best-known position, which is initially set to its current position.

```
self.best_fitness = float('inf')
```

Sets the particle's best fitness to infinity initially since no fitness has been calculated yet.

```
lat1, lon1 = radians(airport1['Lat']), radians(airport1['Long'])
```

Converts the latitude and longitude of airport1 from degrees to radians for distance calculations.

```
radius = 6371.0 # Earth radius in kilometers
```

```

def particle_swarm_optimization(airports, particle_count, iterations, w, c1, c2,
velocity_limit, iteration_limit):
    particles = [Particle(airports, velocity_limit) for _ in range(particle_count)]
    global_best_position = None
    global_best_fitness = float('inf')
    all_fitness_values = []
    iteration_counter = 0

    for iteration in range(iterations):
        for particle in particles:
            current_fitness = particle.fitness()
            if current_fitness < particle.best_fitness:
                particle.best_fitness = current_fitness
                particle.best_position = particle.position.copy()
            if current_fitness < global_best_fitness:
                global_best_fitness = current_fitness
                global_best_position = particle.position.copy()

        for particle in particles:
            visited_airports = set()
            for i in range(len(particle.position)):
                r1, r2 = random.random(), random.random()
                particle.velocity[i] = (
                    w * particle.velocity[i]

```

Sets the radius of the Earth in kilometers, used for calculating the distance between two points on the Earth's surface.

```
def fitness(self):
```

Defines a method to calculate the total route distance for the particle's current position (fitness).

```
fitness += self.distance(airport1, airport2)
```

Accumulates the distance between consecutive airports in the particle's route.

```
for iteration in range(iterations):
```

The main PSO loop, iterating over a specified number of iterations.

```
current_fitness = particle.fitness()
```

Calculates the current fitness (route distance) for a particle's position.

```
if current_fitness < particle.best_fitness:
```

Updates the particle's personal best fitness and position if the current fitness is better than the particle's known best fitness.

```
if current_fitness < global_best_fitness:
```



```

        + c1 * r1 * (particle.best_position[i] - particle.position[i])
        + c2 * r2 * (global_best_position[i] - particle.position[i])
    )
    particle.velocity[i] = max(-velocity_limit, min(particle.velocity[i],
velocity_limit))
    proposed_position = limit_check(particle.position[i] +
int(particle.velocity[i]), len(airports))
    while proposed_position in visited_airports:
        proposed_position = random.randint(0, len(airports) - 1)
        visited_airports.add(proposed_position)
    particle.position[i] = proposed_position

    all_fitness_values.append(global_best_fitness)

    if iteration > 0 and all_fitness_values[-1] == all_fitness_values[-2]:
        iteration_counter += 1
    else:
        iteration_counter = 0
    if iteration_counter >= iteration_limit:
        break

    return global_best_position, global_best_fitness, all_fitness_values

```

Updates the global best fitness and position if the current fitness is better than the global best known fitness.

$r1, r2 = 0.5, 0.5$

Sets random coefficients used in the velocity update equation.

```

particle.velocity[i] = max(-velocity_limit,
min(particle.velocity[i], velocity_limit))

```

Updates the velocity of the particle based on its inertia (w), personal best position ($c1 * r1$), and global best position ($c2 * r2$).

```

proposed_position = random.randint(0, len(airports) - 1)

```

Ensures that the proposed new position is within the valid range of airport indices.

```

visited_airports = set()

```

Tracks which airports have been visited by the particle to avoid revisiting during the route construction.

```

if iteration_counter >= iteration_limit:

```

Stops the PSO loop early if the global best fitness does not improve for a specified number of consecutive iterations (iteration_limit).

```
return global_best_position, global_best_fitness, all_fitness_values
```

Returns the best route, its corresponding fitness score, and a list of fitness values over all iterations for convergence analysis.

<p>Test PSO with different configurations</p> <pre>def test_pso_performance_with_visualization(airports): configurations = [(20, 5000, 500), (30, 4000, 400), (40, 3000, 300), (50, 2000, 200), (60, 1000, 100), (70, 6000, 600),] results = [] for idx, (particle_count, iterations, iteration_limit) in enumerate(configurations, 1): print(f"\nConfiguration {idx}: particle_count={particle_count}, iterations={iterations}, iteration_limit={iteration_limit}") best_position, best_fitness, all_fitness_values = particle_swarm_optimization(airports, particle_count, iterations, w=1, c1=1, c2=2, velocity_limit=1234.8, iteration_limit=iteration_limit) results.append((particle_count, iterations, iteration_limit, best_fitness))</pre>	<pre>def test_pso_performance_with_visualization(airports):</pre> <p>Defines a function to test the PSO algorithm with various configurations and visualize the results.</p> <pre>configurations = [</pre> <p>A list of tuples where each tuple contains the particle count, maximum iterations, and iteration limit for different PSO configurations.</p> <pre>for idx, (particle_count, iterations, iteration_limit) in enumerate(configurations, 1):</pre> <p>Loops over each configuration, unpacking the parameters and tracking the configuration index for labeling outputs and plots.</p> <pre>best_position, best_fitness, all_fitness_values = particle_swarm_optimization(</pre> <p>Runs the PSO algorithm for the current configuration and retrieves the best route, its fitness, and the fitness values over iterations for visualization.</p>

```
print(f"Best Fitness: {best_fitness:.2f}")
```

```
# Plot the best route
```

```
best_route = [airports.iloc[idx]["name"] for idx in best_position]
```

```
route_coordinates = [(airports.iloc[idx]["Lat"], airports.iloc[idx]["Long"]) for  
idx in best_position]
```

```
x = [coord[0] for coord in route_coordinates]
```

```
y = [coord[1] for coord in route_coordinates]
```

```
plt.figure(figsize=(10, 5))
```

```
plt.subplot(1, 2, 1)
```

```
plt.plot(x + [x[0]], y + [y[0]], marker='o', linestyle='-')
```

```
plt.title(f'Configuration {idx}: Best Route')
```

```
plt.xlabel('Latitude')
```

```
plt.ylabel('Longitude')
```

```
for airport_name, (xi, yi) in zip(best_route, route_coordinates):
```

```
    plt.text(xi, yi, airport_name, fontsize=8, ha='right', va='bottom')
```

```
# Plot the convergence graph
```

```
plt.subplot(1, 2, 2)
```

```
plt.plot(all_fitness_values, marker='o')
```

```
plt.title(f'Configuration {idx}: Convergence')
```

```
plt.xlabel('Iteration')
```

```
plt.ylabel('Global Best Fitness')
```

```
results.append((particle_count, iterations, iteration_limit, best_fitness))
```

Stores the results for the current configuration in a list for summary output after all tests.

```
best_route = [airports.iloc[idx]["name"] for idx in best_position]
```

Extracts the names of airports in the best route sequence for labeling on the visualization.

```
route_coordinates = [(airports.iloc[idx]["Lat"], airports.iloc[idx]["Long"])  
for idx in best_position]
```

Extracts the latitude and longitude of airports in the best route for plotting.

```
x = [coord[0] for coord in route_coordinates]
```

Retrieves the latitude values of the best route to form the x-coordinates for the plot.

```
y = [coord[1] for coord in route_coordinates]
```

Retrieves the longitude values of the best route to form the y-coordinates for the plot.

```
plt.subplot(1, 2, 1)
```

Creates a subplot to visualize the best route for the current configuration.

```
plt.plot(x + [x[0]], y + [y[0]], marker='o', linestyle='-')
```

```
plt.tight_layout()
plt.show()
```

```
return results
```

```
# Run tests with visualization
```

```
results = test_pso_performance_with_visualization(airports)
```

```
for result in results:
```

```
    print(f"Configuration: particle_count={result[0]}, iterations={result[1]},  
iteration_limit={result[2]} -> Best Fitness: {result[3]:.2f}")
```

Plots the best route as a closed loop, adding the starting point at the end to form a complete route.

```
plt.text(xi, yi, airport_name, fontsize=8, ha='right', va='bottom')
```

Annotates each airport's position on the map with its name.

```
plt.subplot(1, 2, 2)
```

Creates a second subplot to visualize the convergence graph for the current configuration.

```
plt.plot(all_fitness_values, marker='o')
```

Plots the global best fitness values over iterations to show the convergence behavior of the PSO algorithm.

```
plt.tight_layout()
```

Adjusts the layout of the subplots to ensure they do not overlap.

```
plt.show()
```

Displays the plots for the current configuration.

```
return results
```

	<p>Returns the collected results of all configurations for further analysis or display.</p> <p><code>for result in results:</code></p> <p>Outputs a summary of all configurations, including the particle count, iterations, iteration limit, and best fitness achieved for each configuration.</p>
<pre>best_route = [airports.iloc[idx]["name"] for idx in best_position] print(f"Best Route Sequence: {' -> '.join(best_route)}")</pre>	<p>Display the sequence of airports</p>

DATA USED

The data used was collected from Kaggle.com and is organized into 28 columns with 75,000 instances. Only three datasets which are the names of the airports, the latitude and longitude were kept after filtering. It was filtered to include only airports exclusive to Japan, resulting in a final dataset of 12 records. The filtered datasets are shown in the table below.

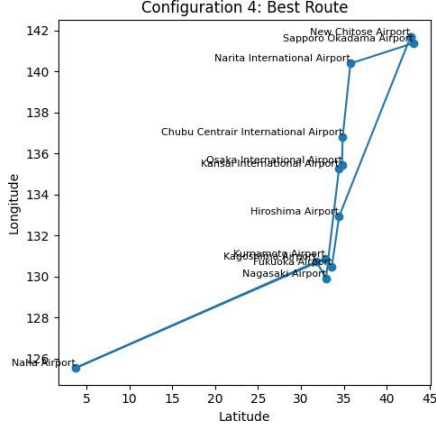
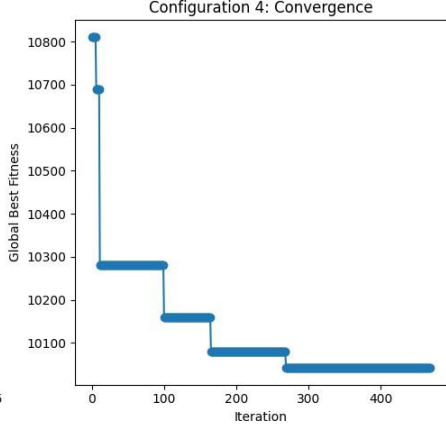
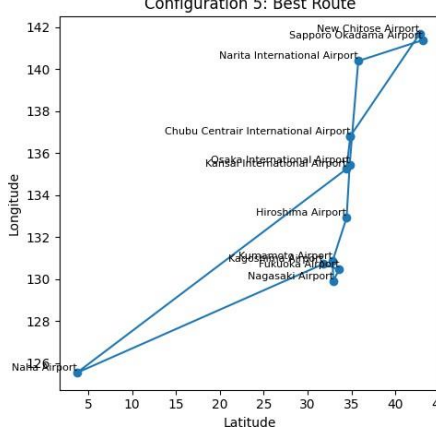
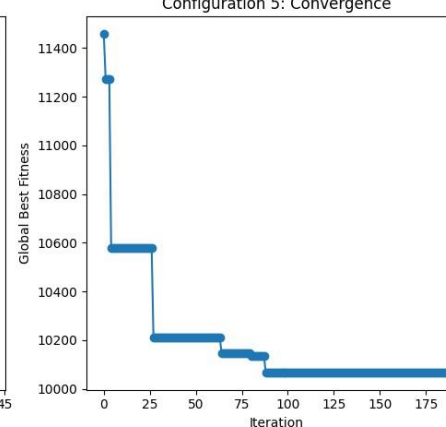
Airports (name)	Latitude (Lat)	Longitude (Long)
Narita International Airport	35.764702	140.386002
Kansai International Airport	34.427299	135.244003
Chubu Centrair International Airport	34.85839844	136.8049927
New Chitose Airport	42.7752	141.692001
Fukuoka Airport	33.58589935	130.451004
Kagoshima Airport	31.80340004	130.7189941
Hiroshima Airport	34.4361	132.919006
Kumamoto Airport	32.83729935	130.8549957
Naha Airport	3.683209896	125.5279999
Osaka International Airport	34.78549957	135.4380035
Nagasaki Airport	32.91690063	129.9140015
Sapporo Okadama Airport	43.117447	141.38134

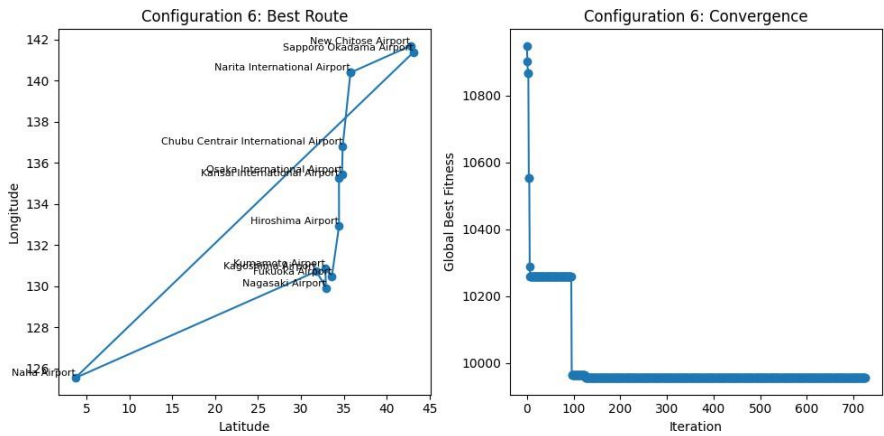
Dataset : <https://www.kaggle.com/datasets/mexwell/world-airports?resource=download>

7. PERFORMANCE OF ALGORITHM

Parameter	Result
Configuration 1: particle_count=20 , iterations=5000, iteration_limit=5000	<p>Configuration 1: particle_count=20, iterations=5000, iteration_limit=500</p> <p>Best Fitness: 9991.81</p> <p>Best Route Sequence: Sapporo Okadama Airport -> New Chitose Airport -> Narita International Airport -> Chubu Centrair International Airport -> Kansai International Airport -> Naha Airport -> Kagoshima Airport -> Fukuoka Airport -> Nagasaki Airport -> Kumamoto Airport -> Hiroshima Airport -> Osaka International Airport</p> <div style="display: flex; justify-content: space-around;"> </div>
Configuration 2: particle_count=30 , iterations=4000, iteration_limit=4000	<p>Configuration 2: particle_count=30, iterations=4000, iteration_limit=400</p> <p>Best Fitness: 10056.26</p> <p>Best Route Sequence: Fukuoka Airport -> Sapporo Okadama Airport -> New Chitose Airport -> Narita International Airport -> Chubu Centrair International Airport -> Kansai International Airport -> Osaka International Airport -> Hiroshima Airport -> Kumamoto Airport -> Nagasaki Airport -> Kagoshima Airport -> Naha Airport</p>

	<div> <div>Configuration 2: Best Route</div> </div> <div> <div>Configuration 2: Convergence</div> </div>
--	--

	<p>-> Nagasaki Airport -> Kagoshima Airport -> Naha Airport -> Kumamoto Airport -> Fukuoka Airport -> Hiroshima Airport</p> <div>   </div>
<p>Configuration 5: particle_count=60 , iterations=1000, iteration_limit=1000</p>	<p>Configuration 5: particle_count=60, iterations=1000, iteration_limit=100</p> <p>Best Fitness: 10065.82</p> <p>Best Route Sequence: Chubu Centrair International Airport -> New Chitose Airport -> Sapporo Okadama Airport -> Narita International Airport -> Osaka International Airport -> Hiroshima Airport -> Kumamoto Airport -> Nagasaki Airport -> Fukuoka Airport -> Kagoshima Airport -> Naha Airport -> Kansai International Airport</p> <div>   </div>
<p>Configuration 6: particle_count=70 , iterations=6000,</p>	<p>Configuration 6: particle_count=70, iterations=6000, iteration_limit=600</p> <p>Best Fitness: 9955.77</p>

<p>iteration_limit=600</p>	<p>Best Route Sequence: Narita International Airport -> New Chitose Airport -> Sapporo Okadama Airport -> Naha Airport -> Kagoshima Airport -> Nagasaki Airport -> Kumamoto Airport -> Fukuoka Airport -> Hiroshima Airport -> Kansai International Airport -> Osaka International Airport -> Chubu Centrair International Airport</p> <div data-bbox="491 465 1378 898">  <p>Configuration 6: Best Route</p> <p>Configuration 6: Convergence</p> </div>
<p>ALL RESULTS</p>	<p>Configuration: particle_count=20, iterations=5000, iteration_limit=500 -> Best Fitness: 9991.81</p> <p>Configuration: particle_count=30, iterations=4000, iteration_limit=400 -> Best Fitness: 10056.26</p> <p>Configuration: particle_count=40, iterations=3000, iteration_limit=300 -> Best Fitness: 10033.09</p> <p>Configuration: particle_count=50, iterations=2000, iteration_limit=200 -> Best Fitness: 10041.57</p> <p>Configuration: particle_count=60, iterations=1000, iteration_limit=100 -> Best Fitness: 10065.82</p> <p>Configuration: particle_count=70, iterations=6000, iteration_limit=600 -> Best Fitness: 9955.77</p>

The analysis of six configurations for the Particle Swarm Optimization (PSO) algorithm applied to optimizing the sequence of Japan airport visits provides valuable insights into the impact of particle count, iteration count, and iteration limit on the algorithm's performance. Among the tested configurations, Configuration 1, with a particle count of 20, iterations set to 5000, and an iteration limit of 500, achieved the best fitness score of 9991.81. This result indicates that maintaining a lower particle count with a higher number of iterations allows the algorithm to efficiently explore the solution space while refining the best solutions. Configuration 1 highlights the importance of enabling the swarm to iteratively optimize while minimizing computational overhead.

In contrast, configurations with either higher or lower particle counts and varied iteration limits exhibited less optimal results. For example, Configuration 5 (particle count 60, iterations 1000) produced a fitness score of 10065.82, which was higher than Configuration 1, suggesting that excessively increasing the particle count with fewer iterations may not yield the best solutions. Similarly, Configuration 6, with the highest particle count of 70 and maximum iterations of 6000, did not achieve the best results despite its significant computational effort, with a fitness score of 9955.77. These outcomes demonstrate that simply increasing the particle count or the number of iterations does not guarantee better solutions, as it may lead to computational inefficiency without significant improvements in the fitness score.

The iteration limit also plays a crucial role in the PSO algorithm's performance. Configurations with iteration limits proportionate to the total number of iterations, such as Configuration 4 (particle count 50, iterations 2000, iteration limit 200), ensured a balance between exploration and exploitation, enabling the swarm to refine solutions effectively. However, excessively high iteration limits may contribute to slower performance with diminishing returns in solution quality.

Based on these observations, Configuration 1 stands out as the most optimal choice among the tested scenarios, providing the shortest travel distance and meeting the objectives of determining the optimal order of airport visits and minimizing total travel distance. The optimal route begins at Sapporo Okadama Airport and proceeds through New Chitose Airport, Narita International Airport, Chubu Centrair International Airport, Kansai International Airport, Naha Airport, Kagoshima Airport, Fukuoka Airport, Nagasaki Airport, Kumamoto Airport, Hiroshima Airport, and ends at Osaka International Airport. This configuration strikes

a balance between computational efficiency and solution quality, making it the most suitable for similar optimization problems.

In conclusion, the findings emphasize the importance of careful parameter selection to balance the swarm's exploration and exploitation capabilities. Configuration 1 provides a robust framework for achieving an optimal balance, and future optimization efforts could further fine-tune hyperparameters such as the velocity limit, inertia weight, and acceleration coefficients (w , $c1$, $c2$) to enhance the algorithm's performance. These results underscore the importance of tailoring the PSO algorithm to specific optimization challenges while maintaining computational efficiency..

8.0 CONCLUSION

The analysis of the Particle Swarm Optimization (PSO) algorithm for solving the Optimization of Japan Airport Visit using Japanese airport locations reveals important insights into the algorithm's performance under different configurations. The six configurations, which varied in particle count, maximum iterations, and iteration limits, demonstrated distinct trade-offs between computational effort and solution quality.

A general observation is that maintaining a lower particle count with a higher number of iterations tends to provide better solutions, as evidenced by lower best fitness values. For instance, Configuration 1, with a particle count of 20, iterations set to 5000, and an iteration limit of 500, yielded the best fitness value of 9991.81. This result indicates a well-optimized route with the shortest total travel distance among all configurations. Notably, this configuration achieves a balance between solution quality and computational efficiency, as it avoids the excessive overhead associated with higher particle counts.

Conversely, configurations with higher particle counts, such as those with 60 and 70 particles, demonstrated slightly higher best fitness values, suggesting less optimal solutions. Despite the increased computational effort in these configurations, the differences in fitness values were relatively small, indicating diminishing returns as particle counts and iterations increased. For example, Configuration 6, with the highest particle count of 70 and 6000 iterations, achieved a fitness value of 9955.77, which, while close to optimal, did not surpass the performance of Configuration 1.

Another significant finding was the effect of iteration limits on convergence. Configurations with iteration limits proportionate to their total iterations, such as Configuration 4 (particle count 50, iterations 2000, iteration limit 200), balanced exploration and exploitation effectively, ensuring that solutions were refined without stagnation. This highlights the importance of tuning iteration limits to match the problem's complexity for sufficient exploration of the solution space.

In conclusion, the PSO algorithm is effective for optimizing the sequence of Japanese airport visits, with performance strongly influenced by particle count, iterations, and iteration limits. For practical applications, selecting configurations with moderate particle counts and higher iterations, such as Configuration 1 (20 particles, 5000 iterations), strikes the best balance

between computational efficiency and solution quality. Future work could explore adaptive strategies to dynamically adjust PSO parameters for even better performance in real-world scenarios..

REFERENCES

- Adam, A., Abidin, A. F. Z., Ibrahim, Z., Husain, A. R., Yusof, Z. M., & Ibrahim, I. (2010). A particle swarm optimization approach to Robotic Drill route optimization. AMS2010: Asia Modelling Symposium 2010 - 4th International Conference on Mathematical Modelling and Computer Simulation, 60–64. <https://doi.org/10.1109/AMS.2010.25>
- Chen, S. M., & Chien, C. Y. (2011). Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert Systems with Applications*, 38(12), 14439–14450. <https://doi.org/10.1016/J.ESWA.2011.04.163>
- Danlu, Z., & Zhongfeng, W. (2020). The Application of Improving Particle Group Algorithm in Logistics Path Optimization. 2020 IEEE 5th International Conference on Intelligent Transportation Engineering, ICITE 2020, 556–560. <https://doi.org/10.1109/ICITE50838.2020.9231507>
- Mahi, M., Baykan, Ö. K., & Kodaz, H. (2015). A new hybrid method based on Particle Swarm Optimization, Ant Colony Optimization and 3-Opt algorithms for Traveling Salesman Problem. *Applied Soft Computing*, 30, 484–490. <https://doi.org/10.1016/J.ASOC.2015.01.068>
- Yifei, T., Meng, Z., Jingwei, L., Dongbo, L., & Yulin, W. (2018). Research on Intelligent Welding Robot Path Optimization Based on GA and PSO Algorithms. *IEEE Access*, 6, 65397–65404. <https://doi.org/10.1109/ACCESS.2018.2878615>
- Nguyen, V. (2021). A general framework of particle swarm optimization. <https://doi.org/10.20944/preprints202101.0528.v1>
- Wang, S., Zhu, J., Shen, X., Wang, Q., Shu, R., & Cai, J. (2021). Research on Particle Swarm Optimization Algorithm. *Journal of Physics: Conference Series*, 1827(1), 012151–012151. <https://doi.org/10.1088/1742-6596/1827/1/012151>

Jin, H. (2023). Application Analysis of Intelligent Particle Swarm Algorithm in the Development of Modern Tourism Intelligence. *Mobile Information Systems*, 2023, 1–8. <https://doi.org/10.1155/2023/5831189>

Muhammad Salman Qamar, Tu, S., Ali, F., Ammar Armghan, Muhammad Fahad Munir, Fayadh Alenezi, Muhammad, F., Ali, A., & Alnaim, N. (2021). Improvement of Traveling Salesman Problem Solution Using Hybrid Algorithm Based on Best-Worst Ant System and Particle Swarm Optimization. *Applied Sciences*, 11(11), 4780–4780. <https://doi.org/10.3390/app11114780>

Zheng, R., Zhang, Y., & Yang, K. (2022). A transfer learning-based particle swarm optimization algorithm for travelling salesman problem. *Journal of Computational Design and Engineering*, 9(3), 933–948. <https://doi.org/10.1093/jcde/qwac039>

Mexwell. (2023). ✈ world airports. Kaggle: Your Machine Learning and Data Science Community. <https://www.kaggle.com/datasets/mexwell/world-airports?resource=download>