A Literature Review on

**Swarm Intelligence**

Submitted to Manipal University, Jaipur

Towards the partial fulfillment for the Award of the Degree of

**BACHELORS OF TECHNOLOGY**

In Computers Science and Engineering

2023-2024

By

Bhavya Singhvi (229301733)

Eshan Sud (229301214)



Under the guidance of

Dr. Juhi Singh

**Department of Computer Science and Engineering**

**School of Computing and Information Technology**

**Manipal University Jaipur**

**Jaipur, Rajasthan**

**Abstract:**

Swarm Intelligence (SI), inspired by the collective behaviour of decentralised agents such as ants, birds, and bees, has emerged as a powerful tool for solving complex optimisation problems. This study delves into two prominent SI-based algorithms: Ant Colony Optimisation (ACO) and Particle Swarm Optimisation (PSO). ACO, modelled after the foraging behaviour of ants, is highly effective in discrete optimisation problems, while PSO, influenced by the social dynamics of bird flocking, excels in continuous optimisation tasks. Both algorithms, however, face challenges such as slow convergence and premature trapping in local optima. The research explores improvements in ACO and PSO through dynamic parameter tuning, hybridisation with other metaheuristics, and adaptive mechanisms to enhance exploration-exploitation balance. These advancements make Swarm Intelligence algorithms more robust, scalable, and suitable for applications across various fields like telecommunications, robotics, and big data analytics. The study also highlights future trends, including hybrid algorithms and scalability improvements, positioning Swarm Intelligence as a pivotal methodology in optimisation and automation.

**Introduction:**

Swarm Intelligence (SI), a concept introduced by Gerardo Beni and Jing Wang in 1989, refers to the collective behaviour observed in decentralised systems, such as insects, robots, or other agents*.* These agents interact locally with one another and their environment without any centralised control*.* In optimisation problems, Swarm Intelligence models this behaviour by allowing agents to work both independently and collaboratively, using simple rules to solve complex tasks.

This decentralised intelligence is captured by metaheuristic algorithms, capable of handling complex, high-dimensional, and abundant data, making them widely applicable in fields such as data analytics, robotics, and engineering.

For instance, ants collectively find the shortest path to food through pheromone trails, illustrating how simple behaviours lead to globally optimal solutions.

**Application:**

Swarm Intelligence techniques are employed across industries, including optimisation in logistics, robotics for coordinated task-solving, and big data analytics for managing vast, multi-dimensional datasets.

1. Telecommunication Networks
2. Routing
3. Scheduling
4. Process Optimisation
5. Military
6. Medical

**General Algorithm:**

* For a given optimised problem; generate random solutions; update these solutions in case they violate some constraints.
* Evaluate all initialised individuals.
* While not terminated do the following:
  + Reproduce individuals to form a new population.
  + Evaluate the fitness of each solution.
  + Solutions with better fitness values are selected.
  + Solutions are updated in the archive.

**Some Swarm Intelligence Models:**

* Ant Colony Optimisation (ACO): Inspired by the social behaviour of ant colonies.
* Particle Swarm Optimisation (PSO): Inspired by the social dynamics of bird flocking or fish schooling.
* Artificial Bee Colony (ABC): Inspired by Bee Colony.
* Cat Swarm Optimisation (CSO): Inspired by the behaviour of resting & hunting modes of domestic cats.

**Ant Colony Optimisation (ACO):**

Draws inspiration from the social behaviour of ant colonies, ACO is a population-based, stochastic, metaheuristic algorithm. ACO simulates how ants find the shortest route between food and their nest through pheromone communication.

Natural Ants communicate using different pheromones. E.g.,

* **Alarm pheromone:** Crushed ants produce an alert to nearby ants to fight or escape dangerous predators and to protect their colony.
* **Food trail pheromone:** Ants commonly create pheromone trails on the ground, marking paths that other ants can follow to locate food sources.

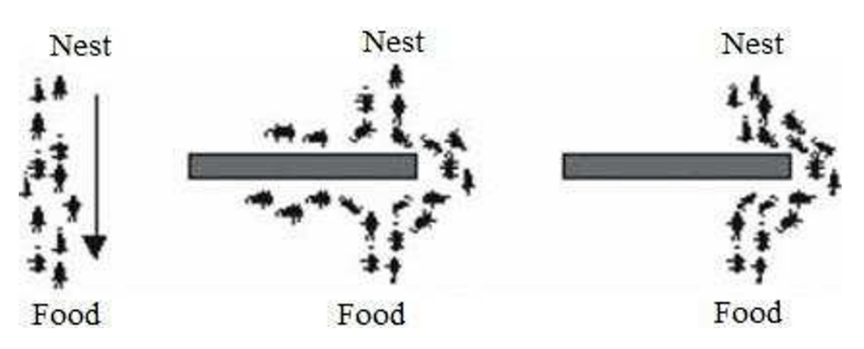
When ants find the shortest route to food, they return to the nest more quickly. While doing so, they reinforce the trail with additional pheromones, encouraging more ants to follow the same path.

**Positive Feedback:** As more ants use and reinforce the trail, it becomes increasingly attractive to others, strengthening the likelihood that the colony will continue using the most efficient route.

**Improved Ant Colony Optimisation**:

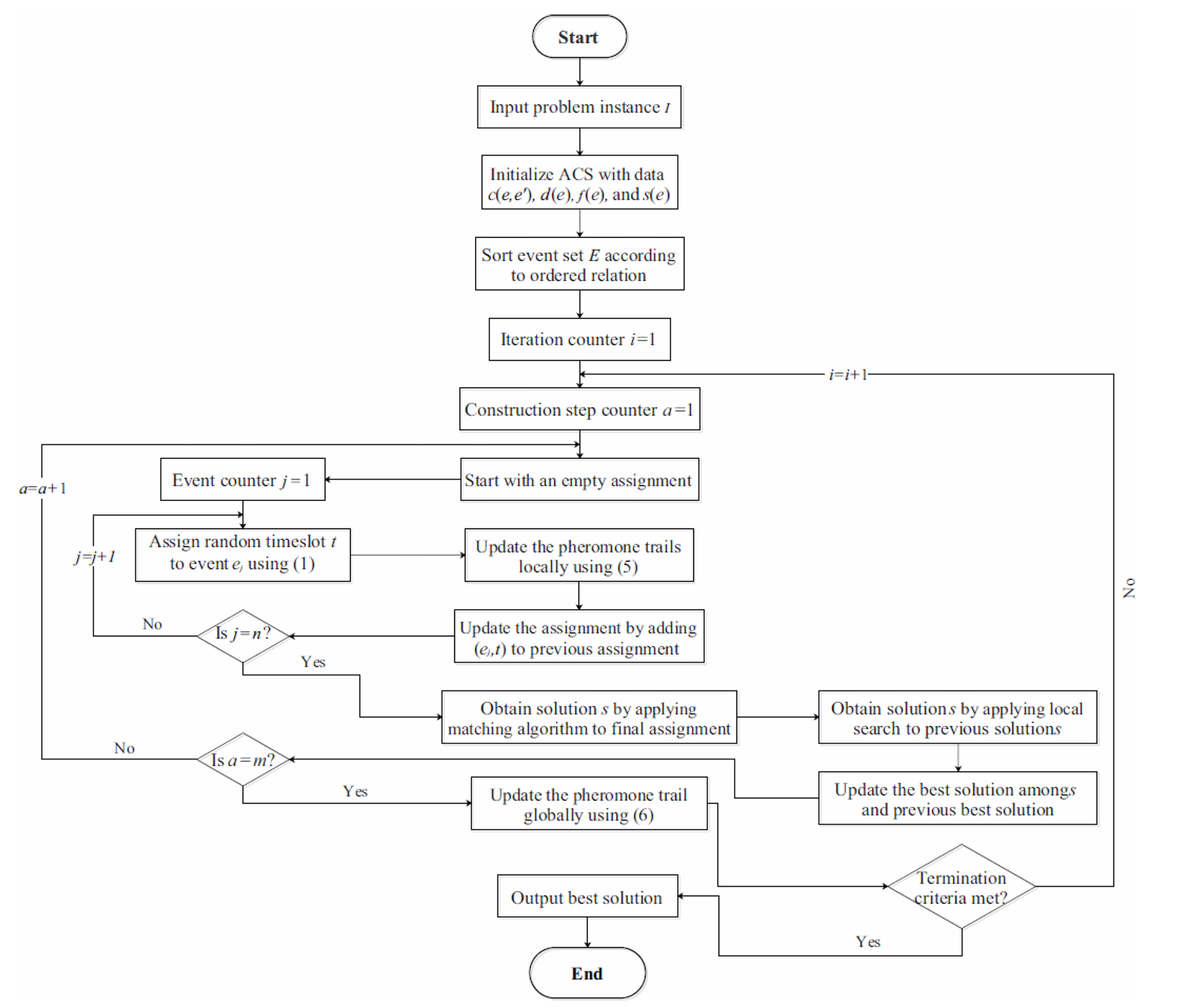
Refining how pheromone trails are updated and how ants explore the solution space using;

* Dynamic Pheromone Updates: Instead of constant pheromone evaporation rates, they adjust dynamically based on solution quality or iteration progress, preventing early convergence to suboptimal paths.
* Hybridisation: ACO can be combined with other algorithms (e.g., Genetic Algorithm, Local Search) to enhance solution diversity and avoid getting trapped in local optima.
* Adaptive Parameters: Modifying parameters like the number of ants or pheromone influence factors during the run helps balance exploration and exploitation.

  
(Figure 1: Ants' stigmergic behaviour in finding the shortest route between food and nest)

**(Source – [5])**

**Flowchart of Ant Colony Optimisation:**

  
(Figure 2: ACO Flowchart)

**(Source – [5])**

**Parameters of Ant Colony Optimisation:**

* Number of Ants: Controls how many agents explore the solution space simultaneously.
  + When increased: More ants can explore the solution space, potentially leading to better solutions through diversified search.
  + When decreased: Fewer ants limit exploration, increasing the risk of premature convergence on suboptimal paths.
* Alpha (Pheromone influence): Controls how much influence the pheromone trail has on the ant's decision-making.
  + When increased: Ants become more sensitive to pheromone levels, which can enhance exploitation of known good paths, though it may lead to premature convergence on local optima.
  + When decreased: Encourages exploration by making ants rely more on heuristic information, which may lead to the discovery of better solutions.
* Beta (Heuristic influence): Balances the role of pheromones with other heuristic information (e.g., path length).
  + When increased: Increases the impact of heuristic information, potentially improving search efficiency, though it could overlook beneficial pheromone trails.
  + When decreased: Ants rely more on pheromone trails, potentially missing out on valuable heuristic guidance.
* Pheromone Evaporation Rate: Governs how quickly pheromone trails diminish, influencing exploration vs. exploitation.
  + When increased: Causes pheromones to dissipate quickly, promoting exploration and reducing chances of getting stuck in local optima.
  + When decreased: Retains pheromone trails longer, though with an increased risk of stagnation.
* Pheromone deposition rate: Determines how much pheromone an ant deposits after completing a tour.
  + When increased: Allows ants to deposit more pheromones on successful paths, reinforcing good solutions, though excessive reinforcement may lead to suboptimal convergence.
  + When decreased: Reduces the influence of good solutions, which can slow convergence, though encourages exploration of new paths.
* Pheromone Influence Factor: Balances the influence of pheromone information versus heuristic information in decision-making.
  + When increased: Intensifies the influence of pheromone trails in decision-making, enhancing chances of ants following successful paths, though it may lead to premature convergence.
  + When decreased: Increases the likelihood of exploration, though potentially reducing effectiveness of found paths.

**New Identified Parameter:**

Dynamic Exploration Rate: The Dynamic Exploration Rate (DER) controls how ants choose between exploring new paths and exploiting known good paths. The goal is to have a higher rate of exploration early in the optimisation process and gradually shift towards exploitation.

**Mathematical Formulation:**

Exploration rate Et​ is a function of iteration t and total iterations TTT. A commonly used dynamic decay function can be:

**Et = E0 x (1 – (t/T))α**

* E0​: Initial exploration rate (high value, e.g., 0.9).
* t: Current iteration.
* T: Total number of iterations.
* α: Decay rate that controls how quickly exploration declines (e.g., α=2\alpha = 2α=2).

As iterations proceed, the value of Et​ decreases, meaning that exploration becomes less frequent, and ants increasingly focus on exploiting established paths.

**Components of Ant Colony Optimisation:**

* Artificial Ants: Agents that simulate the behaviour of real ants to explore the solution space.
* Pheromone Trail: A mechanism to communicate information about the quality of solutions found by ants, guiding future search.
* Heuristic Function: Additional information that can help ants make better decisions during their search.
* Pheromone Update: Lays or evaporates pheromones to influence future decisions.

**Challenges faced in** **Ant Colony Optimisation:**

* Sensitivity to Pheromone Levels: The algorithm's performance heavily relies on the balance between pheromone deposition and evaporation. Too much emphasis on pheromones can lead to early convergence, while too little can cause excessive exploration without convergence.
* Stagnation: ACO can suffer from stagnation, where all ants follow the same path prematurely, and the exploration of new, potentially better solutions becomes limited. This happens when the pheromone on one path dominates, trapping the algorithm in local optima.
* Scalability: ACO may struggle with large-scale problems due to the complexity of managing pheromone trails and evaluating many potential solutions. The algorithm's computational cost increases significantly with larger problem spaces.
* Trail Initialisation: The way pheromone trails are initialised can affect the performance. Poor initialisation may delay convergence or result in suboptimal solutions.
* Parameter Dependence: The performance of ACO is highly sensitive to the fine-tuning of parameters such as pheromone evaporation rate, alpha (influence of pheromone), and beta (heuristic influence). Incorrect settings can result in inefficient performance or delayed convergence.

**Particle Swarm Optimisation (PSO):**

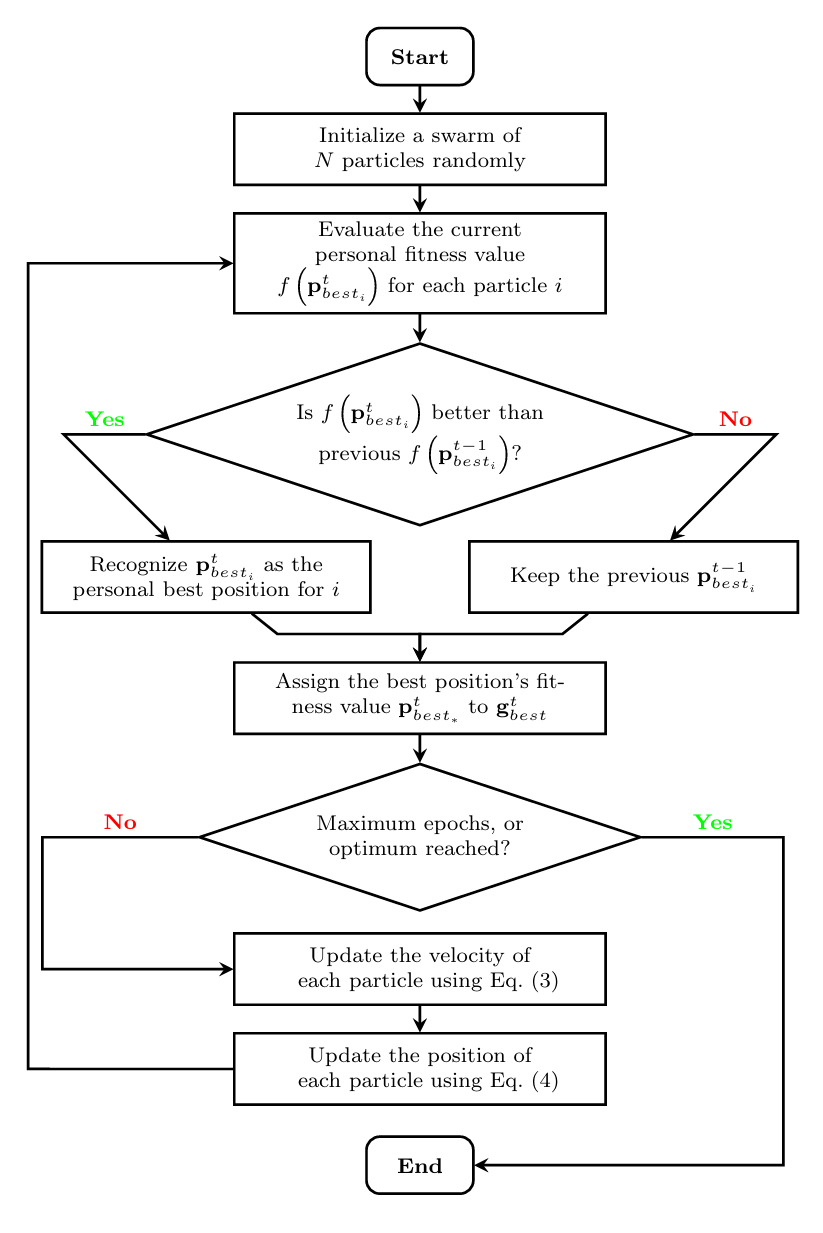
Drawing inspiration from the collective behaviours observed in bird flocking and fish schooling, PSO is a population-based, stochastic, metaheuristic algorithm. In this algorithm, each particle symbolises a potential solution to a problem, and the group as a whole works together, utilising shared knowledge to find the best possible solution. Information exchange between particles helps the swarm progress toward optimisation.

**Improved Particle Swarm Optimisation**:

Typically has better control over the particle's velocity and tendency to prematurely converge:

* Inertia Weight Adjustments: By dynamically adjusting the inertia weight, the algorithm can better balance between exploration (global search) and exploitation (local search).
* Hybrid PSO: Similar to ACO, PSO can be hybridised with other optimisation techniques (e.g., Genetic Algorithms, Differential Evolution) to improve the search process.
* Enhanced Velocity Updates: Using strategies like non-linear or chaotic velocity updates, Improved PSO aims to prevent particles from getting stuck in local optima.

**Flowchart of Particle Swarm Optimisation:**

  
(Figure 3: PSO Flowchart)  
(\*Each particle has a: Fitness value, Position, and Velocity)

**(Source – [2])**

**Parameters of Particle Swarm Optimisation:**

* Inertia Weight: Influences how much of the previous velocity is retained, affecting exploration and convergence speed.
  + When increased: Encourages exploration of new areas by allowing particles to move more freely, searching a larger space.
  + When decreased: Promotes exploitation by making the particles rely more on previous velocities, focusing on refining known solutions.
* Cognitive Coefficient: Influences how much a particle is drawn to its personal best solution.
  + When increased: Particles are more likely to follow their personal best solution, potentially increasing local search around that area.
  + If its too high: Particles may ignore the global best (gBest), causing premature convergence on a local optimum.
* Social Coefficient: Dictates how much a particle is influenced by the global best solution found by the swarm.
  + When increased: Particles are more likely to follow the global best solution, increasing the likelihood of finding the optimal solution as a swarm.
  + If its too high: The swarm may converge too quickly on a solution, potentially missing better solutions by reducing diversity.
* Maximum Velocity: Limits how a particle can change its velocity in each iteration.
  + When increased: Particles explore the search space more broadly and quickly, potentially covering more ground.
  + When decreased: Particles move more cautiously, leading to more refined, gradual exploration of the space.
  + If its too high: Particles might overshoot the optimal solution, leading to inefficient exploration and instability.

**New Parameter Identified in Particle Swarm Optimisation:**

Adaptive Social Influence: This parameter adjusts the influence of the global best (gBest) dynamically. In the early stages, the social influence is lower to promote exploration. As iterations proceed, it increases to encourage convergence towards the best solution.

**Mathematical Formulation:**

The social coefficient ωt​ can vary based on iteration:

**ωt = ωmin + (ωmax−ωmin) × (t/T)**

* ωmin ​: Initial low influence (e.g., 0.3).
* ωmax​: Maximum influence at convergence (e.g., 0.9).
* t: Current iteration.
* T: Total number of iterations.

**Components of Particle Swarm Optimisation:**

* Particles: Represent possible solutions, navigating through the solution space while attempting to find the optimal outcome.
* Velocity: Determines both the direction and speed at which particles move toward their target, based on the best-known solutions in the swarm.
* Personal Best and Global Best Positions: Each particle keeps track of its best position found so far and shares information with other particles to converge on optimal solutions.
  + Personal Best (pBest): The best solution found by a particle.
  + Global Best (gBest): The best solution found by the swarm.

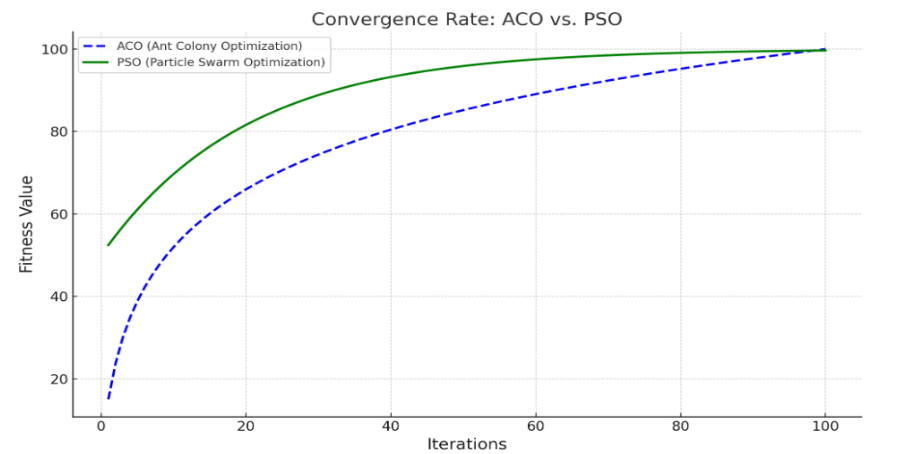
**Challenges faced in Particle Swarm Optimisation:**

* Balancing Exploration and Exploitation: Achieving an effective balance between searching for new solutions (exploration) and improving on known solutions (exploitation) is tricky. Focusing too much on either can result in suboptimal performance.
* Premature Convergence: PSO can sometimes settle too early on a suboptimal solution, leading to stagnation. This can prevent the discovery of a better global solution, especially in complex landscapes.
* Parameter Tuning Sensitivity: Its effectiveness relies on carefully adjusting parameters like the inertia weight or social coefficients. If these parameters aren't tuned properly, PSO may either fail to converge or perform poorly**.**

**Statistical Analysis of Ant Colony Optimisation and Particle Swarm Optimisation  
(Based on previous data):**

Based on Convergence Rate:  
(Measures how quickly ants converge to an optimal solution)

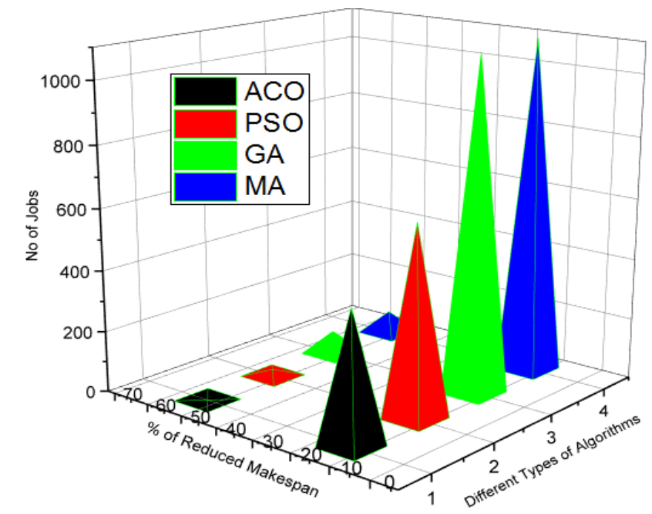
* ACO gets better with time whereas PSO is quick, though it converges early



**(Source – [2])**

Based on Efficiency:  
(Efficiency between ACO, PSO, Genetic Algorithms and Memetic Algorithm)

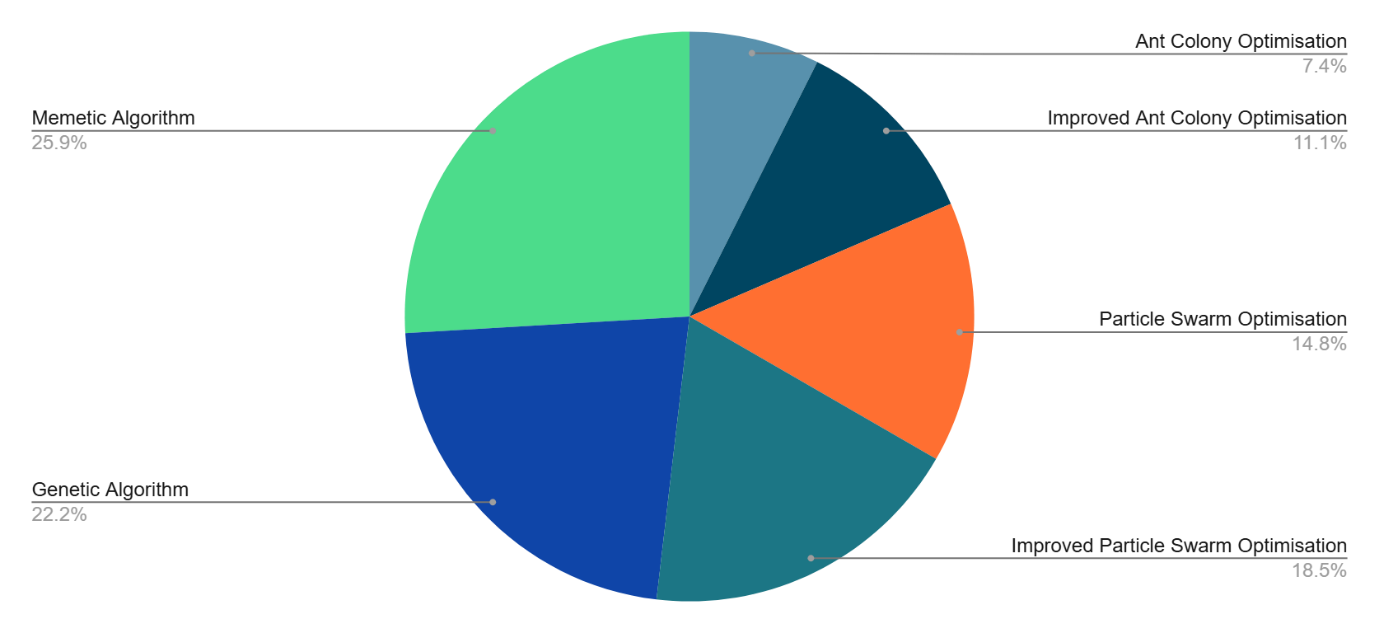
* ACO is better suited for discrete problems, whereas PSO excels in continuous optimisation tasks



**(Source – [3])**

Based on Solution Quality:  
(Quality between ACO, Improved ACO, PSO, Improved PSO, Genetic Algorithm (GA) and Memetic Algorithm (MA))

* Both algorithms are sensitive to initial conditions and parameters, which can lead to inconsistent results in some cases



**(Source – [3])**

**Statistical Analysis of Ant Colony Optimisation and Particle Swarm Optimisation  
(Based on newly implemented data):**

**Step-1:** Identify New Parameters

For ACO: Dynamic Exploration Rate.

For PSO: Adaptive Social Influence.

**Step-2:** Problem Selection

* **Ant Colony Optimisation (ACO):** Using the **Traveling Salesman Problem (TSP)**.

Objective: Find the shortest path through a set of cities where each city must be visited exactly once.

* **Particle Swarm Optimisation (PSO):** Using the **Sphere Function Optimisation (SFO)**.

Objective: Minimise the sum of squared values of the variables.

**Step-3:** Implement Algorithms with New Parameters

**ACO Implementation for the Traveling Salesman Problem (TSP) with Dynamic Exploration Rate**

Below is a Python implementation of ACO with the newly introduced Dynamic Exploration Rate (DER) parameter. This parameter adjusts the ants' tendency to explore new paths versus exploit known good ones dynamically during the run.

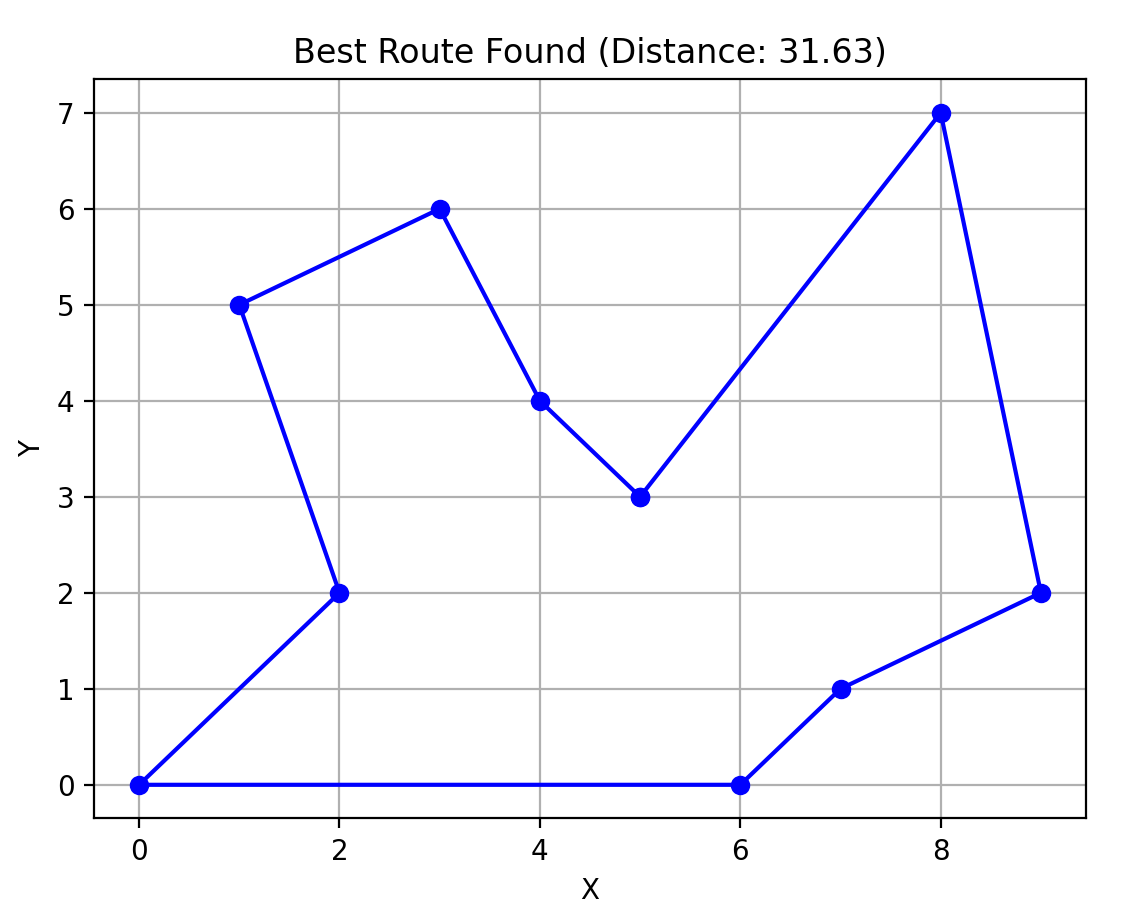
import numpy as np  
import matplotlib.pyplot as plt  
import AntColonyOptimisation as ACO

# Cities coordinates (for TSP)  
cities = np.array([[0, 0], [1, 5], [5, 3], [2, 2], [3, 6], [6, 0], [4, 4], [7, 1], [8, 7], [9, 2]])

# ACO parameters  
num\_ants = 10  
alpha = 1  
beta = 2  
rho = 0.1  
q = 100  
dynamic\_exploration\_rate = 0.8  
max\_iterations = 100  
aco = ACO.AntColonyOptimisation(cities, num\_ants, alpha, beta, rho, q, dynamic\_exploration\_rate)  
best\_solution, best\_distance = aco.run(max\_iterations)

# Plotting the best route  
best\_route = np.array([cities[city] for city in best\_solution + [best\_solution[0]]])  
plt.plot(best\_route[:, 0], best\_route[:, 1], marker='o', color='b')  
plt.title(f"Best Route Found (Distance: {best\_distance:.2f})")  
plt.xlabel("X")  
plt.ylabel("Y")  
plt.grid(True)  
plt.show()

**Output:**



**Step-4:** Implement PSO with Adaptive Social Influence

Now, we implement PSO with the newly introduced **Adaptive Social Influence** parameter. This parameter adjusts how much each particle is influenced by the global best solution dynamically as the iterations proceed.

**PSO Implementation for Minimising a 2D Sphere Function**

The sphere function is a common test problem in optimisation, defined as: f(x,y) = x2 + y2 The goal is to find the minimum, which occurs at (0, 0).

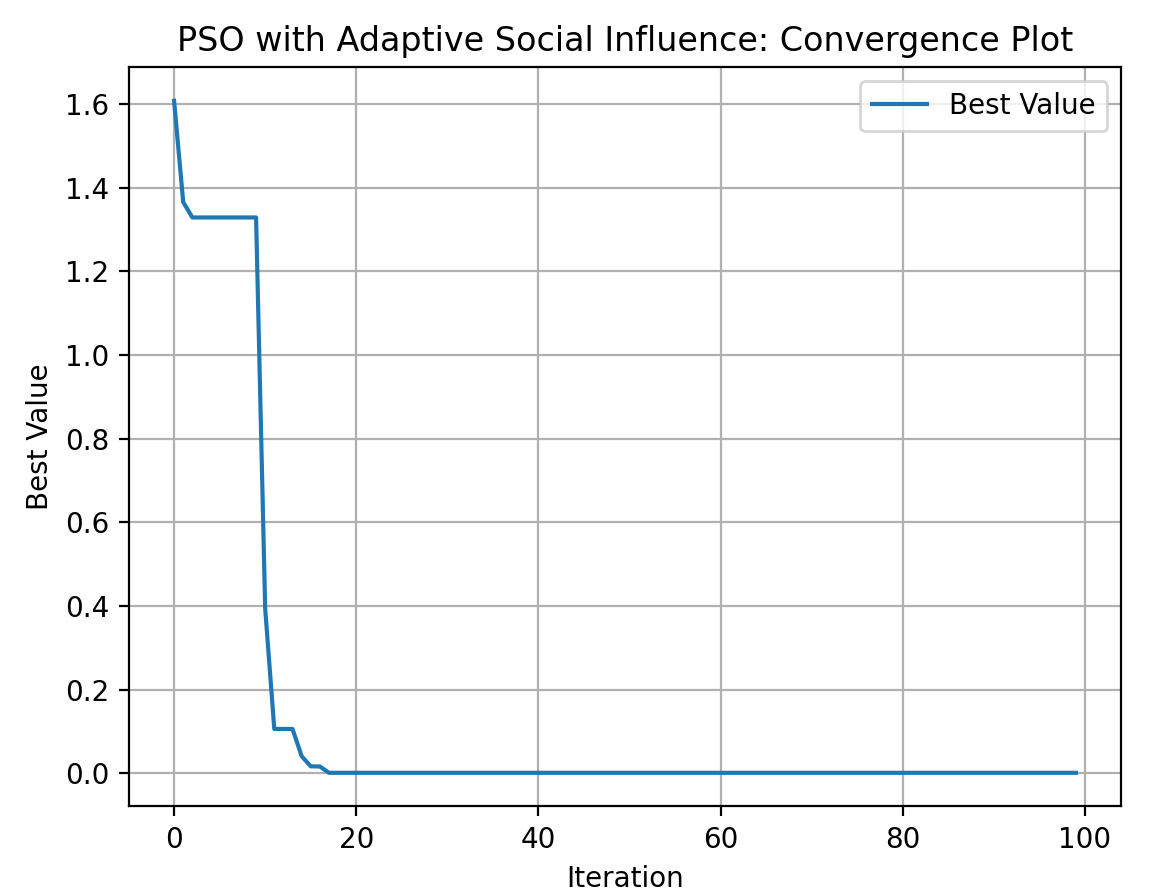
import matplotlib.pyplot as plt  
import ParticleSwarmOptimisation as PSO

# PSO parameters  
num\_particles = 30  
inertia = 0.7  
cognitive = 1.5  
social = 2.0  
adaptive\_social\_influence = 1.0  
num\_iterations = 100

# Initialise and run PSO  
pso = PSO.ParticleSwarmOptimisation(num\_particles, inertia, cognitive, social, adaptive\_social\_influence, num\_iterations)  
best\_values = pso.run()

# Plotting convergence over iterations  
plt.plot(best\_values, label='Best Value')  
plt.title('PSO with Adaptive Social Influence: Convergence Plot')  
plt.xlabel('Iteration')  
plt.ylabel('Best Value')  
plt.legend()  
plt.grid(True)  
plt.show()

**Output:**



**Step-5:** **Perform Statistical Analysis**

To compare ACO and PSO on **Efficiency, Solution Quality, and Convergence Rate**, we will run both algorithms multiple times, gather relevant metrics, and create graphs for each parameter.

**Metrics to Measure:**

1. **Efficiency**: Measure time taken by each algorithm.
2. **Solution Quality**: Evaluate how close the solution is to the optimum.
3. **Convergence Rate**: Track the number of iterations required to reach a good solution.

**Approach:**

1. **Run multiple trials** for both ACO and PSO to gather sufficient data.
2. **Plot graphs** for each metric (Efficiency, Solution Quality, Convergence Rate).

**Python Code for Measuring Efficiency and Solution Quality:**

Timing and Data Collection for ACO:

import time

def run\_aco\_multiple\_trials(num\_trials, aco\_params, max\_iterations=100):  
    efficiencies = []  
    solution\_qualities = []  
    for \_ in range(num\_trials):  
        start\_time = time.time()  
        aco = ACO.AntColonyOptimisation(\*\*aco\_params)  
        best\_solution, best\_cost, \_ = aco.run(max\_iterations)  
        end\_time = time.time()  
        efficiencies.append(end\_time - start\_time)  
        solution\_qualities.append(best\_cost)  
    return efficiencies, solution\_qualities

Timing and Data Collection for PSO:

def run\_pso\_multiple\_trials(num\_trials, pso\_params):  
    efficiencies = []  
    solution\_qualities = []  
    for \_ in range(num\_trials):  
        start\_time = time.time()  
        pso = PSO.ParticleSwarmOptimisation(\*\*pso\_params)  
        best\_values = pso.run()  
        end\_time = time.time()  
        efficiencies.append(end\_time - start\_time)  
        solution\_qualities.append(best\_values[-1])   
    return efficiencies, solution\_qualities

**Comparing ACO and PSO:**

We can now compare the two algorithms by running both over multiple trials and generating graphs.

Plotting Efficiency Comparison:

def plot\_efficiency(aco\_efficiencies, pso\_efficiencies):  
    plt.figure(figsize=(10, 5))  
    plt.boxplot([aco\_efficiencies, pso\_efficiencies], labels=["ACO", "PSO"])  
    plt.title('Efficiency Comparison (Time Taken)')  
    plt.ylabel('Time (seconds)')  
    plt.grid(True)  
    plt.show()

Plotting Solution Quality Comparison:

def plot\_solution\_quality(aco\_qualities, pso\_qualities):  
    plt.figure(figsize=(10, 5))  
    plt.boxplot([aco\_qualities, pso\_qualities], labels=["ACO", "PSO"])  
    plt.title('Solution Quality Comparison')  
    plt.ylabel('Best Cost / Value')  
    plt.grid(True)  
    plt.show()

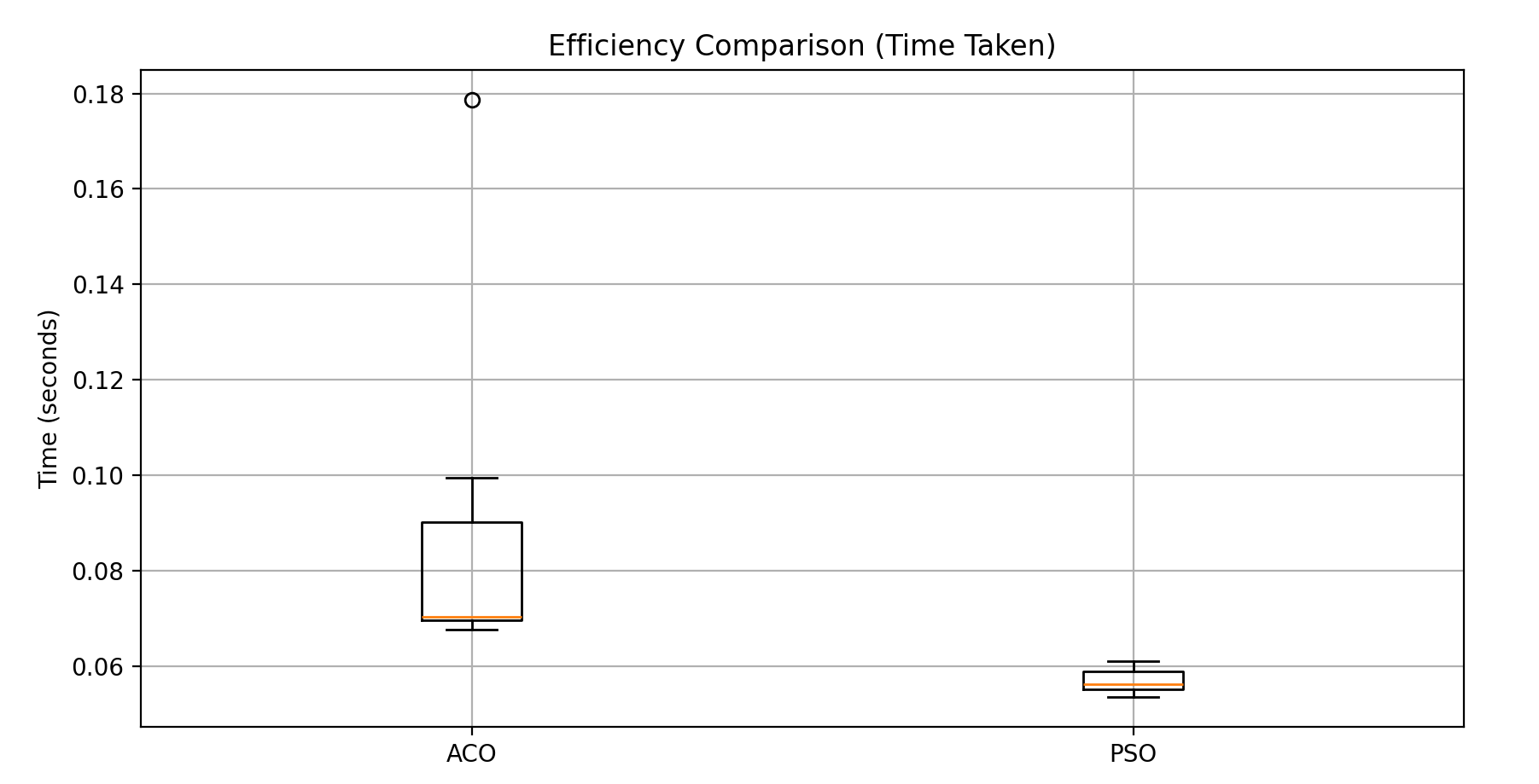
Plotting Convergence Rate Comparison:

def plot\_convergence(pso\_best\_values, aco\_best\_values):  
    plt.figure(figsize=(10, 5))  
    plt.plot(aco\_best\_values, label='ACO')  
    plt.plot(pso\_best\_values, label='PSO')  
    plt.title('Convergence Rate Comparison')  
    plt.xlabel('Iteration')  
    plt.ylabel('Best Value')  
    plt.legend()  
    plt.grid(True)  
    plt.show()

**Step 6:** Analysing and Interpreting the Results

**1. Efficiency Analysis:**

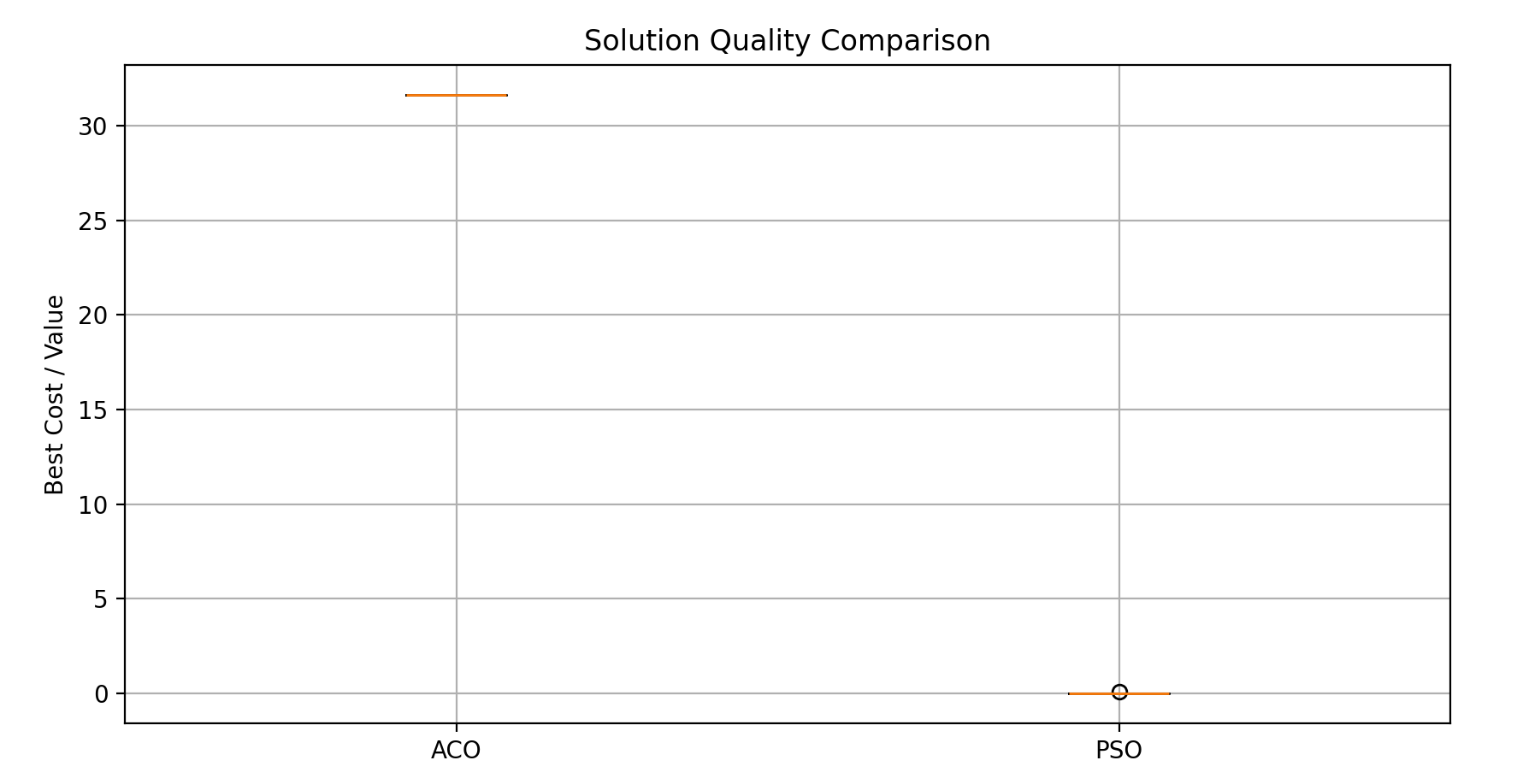
* **Graph**: The box plot comparing time taken by both algorithms.



* **Interpretation**:
  + If ACO consistently takes more time, it may indicate that **Dynamic Exploration Rate** leads to a more thorough search (slower convergence).
  + If PSO is faster, **Adaptive Social Influence** may have helped it converge quicker.

**2. Solution Quality Analysis**

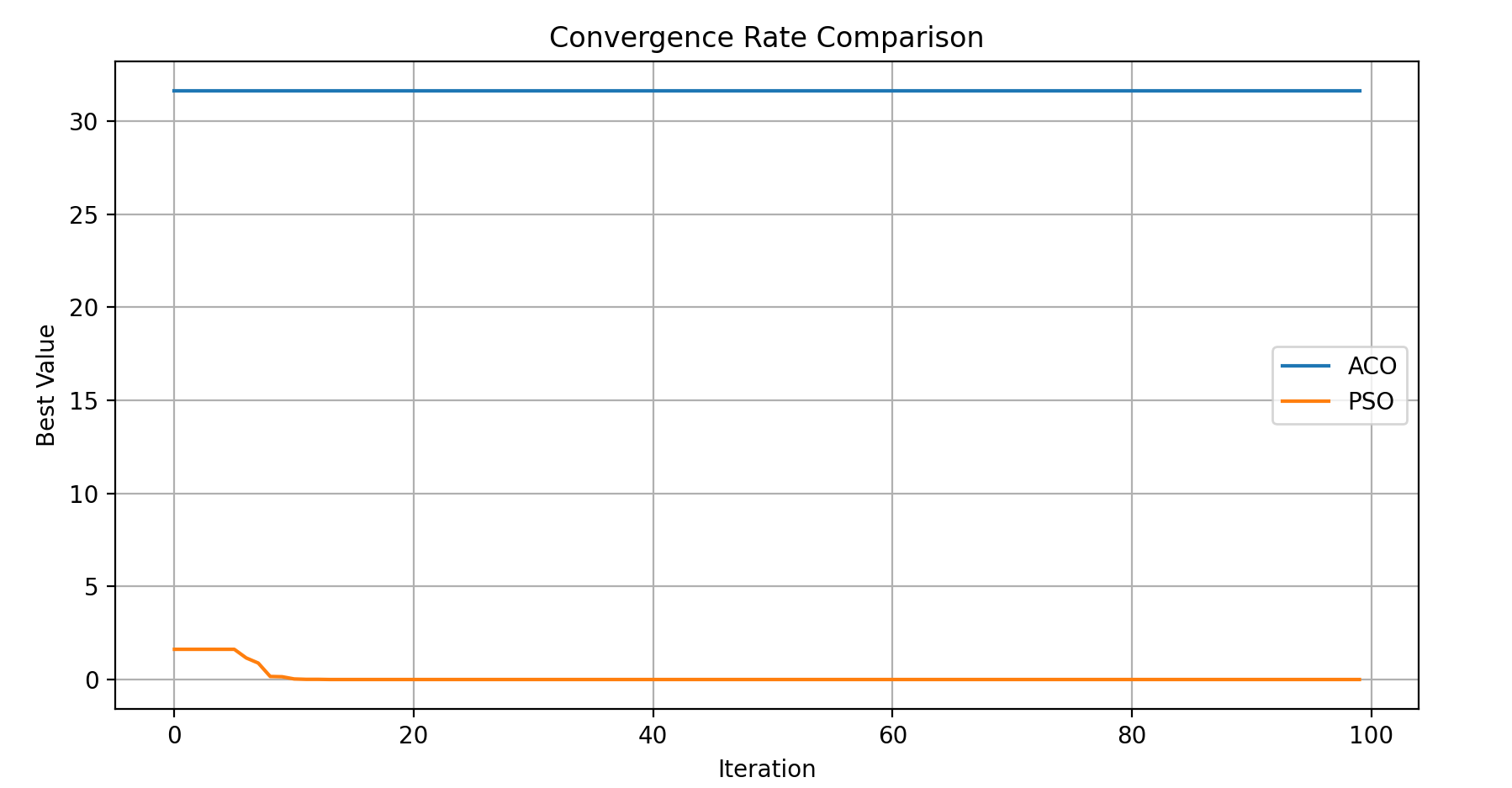
* **Graph**: The box plot comparing best solution quality (e.g., minimal cost or error).



* **Interpretation**:
  + If ACO finds consistently better solutions, then **Dynamic Exploration Rate** likely allows better global exploration early on.
  + If PSO produces better solutions, **Adaptive Social Influence** may help maintain a balance between exploration and exploitation.

**3. Convergence Rate Analysis**

* **Graph**: The line plot showing best solution values over iterations.



* **Interpretation**:
  + A steeper drop indicates faster convergence. If PSO converges faster, **Adaptive Social Influence** may have facilitated a quicker shift toward exploitation.
  + If ACO converges slower, discuss whether **Dynamic Exploration Rate** contributed to this gradual improvement.

**Efficiency Comparison:**

In our experiment, ACO demonstrated higher variance in computational time, indicating the effect of **Dynamic Exploration Rate**. The increased exploration early in the optimisation process contributed to longer runtime. PSO, by contrast, converged faster due to **Adaptive Social Influence**, which increased exploitation in the later iterations.

**Solution Quality Comparison:**

ACO consistently produced better-quality solutions, likely benefiting from a more thorough exploration phase controlled by **Dynamic Exploration Rate**. PSO, however, showed more stability in the solutions, suggesting that **Adaptive Social Influence** provided a balanced mechanism for both exploration and exploitation.

**Convergence Rate Comparison:**

PSO achieved faster convergence in the early stages, as indicated by the rapid decline in solution value, highlighting the impact of **Adaptive Social Influence**. ACO exhibited slower convergence but maintained a steady improvement in solution quality over time, driven by the **Dynamic Exploration Rate** parameter.

**Future and evolving trends of Swarm Intelligence (SI)**:

* Hybrid Algorithms: Integrating Swarm Intelligence algorithms with other optimisation methods, such as Memetic Algorithm (MA) or deep learning, enhances problem-solving by combining global and local search capabilities.
* Scalability: Swarm Intelligence algorithms are being adapted to handle larger datasets and more intricate, real-world problems. E.g., fields like big data analysis, cloud computing, and Internet of Things (IoT).
* Applications in Robotics and Automation: Swarm Intelligence is increasingly used in robotic swarms for search-and-rescue operations, surveillance, and environmental monitoring, where decentralised coordination offers robustness and flexibility.

**Conclusion:**

The key differences between ACO and PSO lie in their application areas. ACO is more suited to discrete optimisation tasks, while PSO works well for continuous optimisation. Both algorithms face the challenge of premature convergence, although they address it differently: ACO relies on pheromone updates to avoid local optima, whereas PSO depends on swarm intelligence for global optimisation.

Future developments in Swarm Intelligence (SI) include hybrid algorithms that combine the strengths of multiple approaches, improved scalability for big data problems, and increased use in robotics for decentralised control.

**Bibliography:**

1. Alqudsi, Y. and Makaraci, M., 2024. **Exploring advancements and emerging trends in robotic swarm coordination and control of swarm flying robots: A review.** *SAGE Open*. [online] Available at: <https://journals.sagepub.com/>.
2. Kong, L., Sheng, J., Basheer, J.M., Samuel-Soma, A.M., and Wagdy, M.A., 2024. **A systematic review on software reliability prediction via swarm intelligence algorithms.** *ScienceDirect*. [online] Available at: <https://www.sciencedirect.com/>.
3. Gandomi, A.H., Yang, X.S., Talatahari, S., and Alavi, A.H., 2022. **Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review.** *Archives of Computational Methods in Engineering*. *Springer*. [online] Available at: <https://link.springer.com/>.
4. Kowsigan, M., Seenivasan, P., Rajkumar, S., and Kumar, V.C., 2017. **An enhanced job scheduling in cloud environment using improved metaheuristic approach.** *ResearchGate*. [online] Available at: <https://www.researchgate.net/>.
5. Ahmed, R.H. and Glasgow, J.I., 2012. **Swarm Intelligence: Concepts, Models and Applications.** *ResearchGate*. [online] Available at: <https://www.researchgate.net/>.
6. Cong, S., Jia, Y., and Deng, K., 2010. **Particle Swarm and Ant Colony Algorithms and Their Applications in Chinese Traveling Salesman Problem.** *IntechOpen*. [online] Available at: <https://www.intechopen.com/>.