Journal of Telecommunication, Electronic and Computer Engineering
ISSN: 2180 – 1843 e-ISSN: 2289-8131 Vol. X No. X (202x) XX-XX
jtec.utem.edu.my
DOI: https://doi.org/10.54554/jtec.202x.xx.xx.xxx

Particle Swarm Optimization (PSO): OPTIMIZING GENETIC TRAITS AND HABITAT CONDITION

Aiman Haqeem Bin Baharuddin[1, 2], Yeo Kheong Jie[2*], Dina Sofea Binti Hilmy[3], Syah Hasrizal bin Haslizan[3]
[1]B022220003, [2]B022220074, [3]B022220011, and [4]B022220066
[2] FAKULTI KEJURUTERAAN ELEKTRONIK DAN KEJURUTERAAN KOMPUTER.
[3] UNIVERSITI TEKNIKAL MALAYSIA MELAKA, Jalan Hang Tuah Jaya, 76100 Durian Tunggal, Melaka

Article Info

Index Terms:
Particle Swarm Optimization
Genetic Traits Optimization
Swarm Intelligence
Environmental Sustainability

Abstract

Enhancing agricultural productivity and ecological sustainability necessitates the optimization of genetic traits and environmental conditions. The complexity and diverse nature of ecological systems pose challenges for traditional optimization methods. This study investigates the application of a swarm intelligence algorithm known as Particle Swarm Optimization (PSO) to predict and enhance the genetic characteristics of fruit-bearing trees, as well as environmental factors such as soil pH, temperature, and rainfall, with the aim of maximizing fruit yield. The methodology integrates dynamic adaptability and synthetic data modelling to address prevalent challenges related to Particle Swarm Optimization (PSO), such as scalability and premature convergence. A thorough examination of recent studies reveals possible solutions to these issues, including hybrid strategies that combine PSO with Artificial Neural Networks (ANNs).

*Corresponding Authors: B022220003@student.utem.edu.my, B022220074@student.utem.edu.my, B022220003@student.utem.edu.my , B022220003@student.utem.edu.my

I. INTRODUCTION

Obtaining the upmost fruit yield has been the goal since long before in agricultural science and research, this is due to high demand from the food and supermarket chain industry that is highly in need of sustainable food supply. Fruit-bearing trees' genetic characteristics such as plant height, leaf size, fruiting efficiency and other habitual factors such as soil pH, temperature and rainfall, play an important role in determining the fruit-bearing tree optimal yield production and exceptional quality. By focusing and optimizing these factors it can increase the agricultural output and sustainability.

Furthermore, traditional methods that are used before have several problems including the complex relationships between the tree genetic and environmental factors variable. These limitations are necessary to develop by applying advanced computational methods capable of navigating such complexity effectively which is PSO (Particle Swarm Optimization).

Particle Swarm Optimization (PSO) is a bio-inspired algorithm in swarm intelligence, that is able to generate a favorable solution. PSO is first developed by Kennedy and Eberhart in 1995, which Is used to simulates the similarity in behavior of social animals such as birds and fish to identify the best solutions in complex search spaces. The PSO method is well-suited for ecological optimization problems because of its simplicity, versatility, and effectiveness.

This research case is to explore the application of PSO algorithm to predict and optimize the early fruit yield by applying the genetic traits of fruit-bearing trees combined with habitual factors. Other than that, this study focuses on identifying and applying the optimal parameter combinations to increase the fruit yield to the max limit while also addressing problems such as tree scalability, premature convergence, and the adaptability trait to dynamic harsh conditions. By simulating and recreating the behavior of a swarm of particles, this study aims to provide useful findings into agricultural fruit yield optimization, that contributes to an enhanced resource management and ecological sustainability.

II. THEORY

A. Theory of PSO

To understand what is Particle Swarm Optimization (PSO), we first need to imagine a swarm of bees flying over a field of flowers. The bee's goal is to locate the field with the highest density of flowers but have never been to the field. They started their search by scattering randomly across the field of flowers with each bee remembering the location it found the most and dense with flowers and share this information with others of the swarm.

Over time, the bees needed to make a decison, whether they can either return to their own best location (the spot where they found the most flowers previously) or move toward the location found by the others of the swarm as having the highest flower density. This results in a balance between individual discovery and social influence. The bees adjusted their paths, flying somewhere near between these two points based on the individual success or collective information that plays a stronger role in their decision-making.

Furthermore, a bee might discover a spot with more flowers than any before known location. The entire swarm is drawn toward that new location with more flowers. This process is illustrated in Figure 1, where dashed lines represent the paths of the bees, and solid arrows indicate the velocity of their movement. In the example, Bee No. 2 has found the global best position (the location with the most flowers). Bee No. 1 has a velocity influenced by both its personal best position and the global best position. Meanwhile, Bee No. 3 represents a particle that has not yet found a strong personal best and is still being drawn toward the global best.

Through this method, the bees explore the field, adjusting their speed and direction based on the success of finding flowers relative to the swarm. They also avoid areas with fewer flowers, searching instead for those with higher concentrations. Ultimately, the entire field is explored, and the bees converge around the location with the highest flower density.

Kennedy and Eberhart (1995a) modeled this behavior in a computer program. They soon realized that their model could be developed into an optimization algorithm, which they called PSO. The term "particle" is used to represent various natural agents, such as fish, bees, birds, or any other creatures that exhibit swarm behaviour.

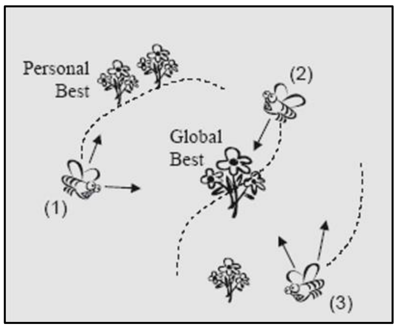


Figure 1 Simulation of PSO by Swarm of Bees

B. History

Particle Swarm Optimization (PSO) was first created and introduced in 1995 by James Kennedy, a social psychologist, and Russell Eberhart, an electrical engineer. Initially, the PSO concept was inspired by the behaviour of social animals such as birds and fish. Kennedy and Eberhart goal is to simulate these social behaviours in order to explore how individuals in a group might learn and adapt their movements based on their experiences gained and social interactions with others. Their original goal was to create a model of human social interaction behaviour, but they soon recognized its potential as a powerful optimization algorithm.

The foundation of PSO lies in mimicking the way swarms in nature operate. Each member of the swarm (or "particle") adjusts its position based on its own previous best position and the best-known positions discovered by others in the group. This collaborative behaviour leads the swarm to converge toward the optimal solution in a search space.

Today, PSO remains a widely used optimization technique due to its simplicity, versatility, and ability to solve challenging problems in diverse domains. Its origins continue to inspire new variations and applications, ensuring its relevance in modern computational science.

C. Application

Two key factors determine the entire PSO procedure, two acceleration factors and the weight of inertia. In PSO, the inertia weight is thought to play the most significant impact. To identify the best solution, it is therefore thought to be crucial to properly control the inertia weight. With linearly changing inertia weights over the iterations, Shi and Eberhart improved the PSO's convergence. This algorithm can be applied in a wide range of optimization domains. Among them are:

1. Optimization of energy storage.
2. Electrical load scheduling.
3. Flood route and control.
4. Identification and categorization of diseases.
5. Segmentation of medical images.
6. Monitoring of water quality.
7. Agriculture observation

III. LITERATURE REVIEW

A. Problem Statement

Given the visualization for the structure of ecological systems, the challenges of optimizing genetic traits and environmental circumstances are still on going. Genetic diversity and environmental restrictions are frequently poorly balanced by traditional methods. The swarm intelligence-based algorithm Particle Swarm Optimization (PSO), which draws inspiration from the social behaviors of creatures, presents a potential method. Despite its promise, PSO has multiple weakness points, including early convergence, constrained scalability, and trouble adjusting to changing conditions. There is currently no standard and extensible framework for the use of PSO in ecological environments, despite recent developments like hybrid algorithms and improved swarm intelligence techniques attempting to solve these issues.

B. Source Discussion

Julius Beneoluchi Odili, Mohd Nizam Mohmad Kahar, and A. Noraziah, in their paper "Swarm Intelligence Optimization Algorithms: A Review," gives a detailed analysis of swarm-based algorithms, including PSO. The study, conducted at Universiti Malaysia Pahang, highlights the versatility of PSO in solving complex optimization problems. It emphasizes PSO's advantages, such as simplicity and convergence speed, while acknowledging its weaknesses, including susceptibility to local optima and performance challenges in high-dimensional problem spaces. This aligns with the obstacles encountered in ecological optimization, underscoring the need for advanced techniques to enhance PSO's robustness and applicability.

Meanwhile, D. M. Soomro, M. Y. Al-izzi, H. A. Soodi, N. M. Elasager, and S. C. Chong, in their paper "Optimal Restoration for Distribution System using PSO and ANN," demonstrate the outcome of integrating PSO with Artificial Neural Networks (ANNs). It is tested across institutions such as the University Tun Hussein Onn Malaysia and the University of Gaziantep, their research showcases the synergy between PSO's global search capabilities and ANN's predictive power. This hybrid approach effectively addresses multi-objective optimization challenges, as evidenced in power distribution restoration. While the study focuses on electrical engineering applications, it provides valuable insights into how hybrid PSO models can be adapted for ecological optimization, particularly in scenarios requiring real-time adaptability and dynamic decision-making.

In summary, both case studies of paper emphasize PSO's strengths while mentioning those limitations through either critical analysis or innovative hybridization techniques. The work by Odili et al. serves as a foundational review, identifying gaps in existing swarm-based optimization methods, while the research by Soomro et al. visualize a applicable solution through different form of modeling. Together, these papers put a scope on the potential for leveraging advanced and integrated optimization methods to overcome the challenges of applying PSO to the optimization of genetic traits and habitat conditions. Bridging these gaps requires further exploration of hybrid models, scalability enhancements, and applications in variant of ecological contexts.

IV. METHODOLOGY

A. Overview

The system carries out Particle Swarm Optimization (PSO) to predict maximum fruit yield through dual optimization of plant genetic traits (plant height, leaf size, etc.) and environmental factors (soil pH, temperature, rainfall). Synthetic data from mathematical modeling receives added noise to replicate real-world variability through this approach. Synthetic data originates from code-generated numerical value. The methodology dealt with two categories of characteristics which included genotypic traits like height and leaf size alongside phenotypic traits including soil pH and environmental factors of temperature and rainfall. The model randomly distributes initial positions and velocities across defined boundaries of each cohort. Each particle within the population receives an initial "best position" which is evaluated by the fitness model of fruit yield. The fitness of each individual particle is evaluated through the fruit_yield_model that evaluates predicted yield from traits and habitat components.) along with the characteristics of the plant's habitat (soil pH, temperature and rainfall, etc.) in order to predict the optimal fruit yield. This methodology employs synthetic data obtained from mathematical modelling followed by the addition of noise to attain real-world variability.

B. Dataset

This dataset consists of artificially created data which programmers produced directly inside the code. We gathered genetically defined traits such as height and leaf size together with environmental traits that include soil pH along with temperature and rainfall measures'), as well as phenotypic traits (e.g., pH of soil, temperature and rainfall).

Range of Parameters:

- Height: 1 to 10 meters
- Leaf Size: 5 to 50 cm
- Soil pH: 5.0 to 9.0
- Temperature: 10°C to 35°C
- Rainfall: 50 mm to 200 mm

C. Python Libraries Used

- **NumPy**: For number computations and mostly random data.
- **Matplotlib**: For 2D and 3D visualization for the optimization results.
- **mpl_toolkits. mplot3d**: 3D plotting of particle movements

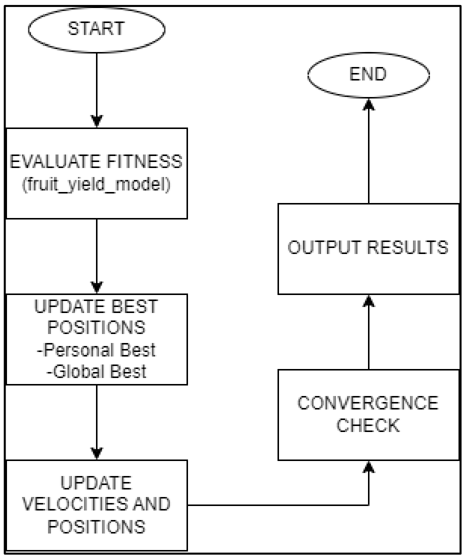
D. Algorithm

1. **Initialization**:
   - Initialize randomly the position of particles within the space
   - Assign each particle a starting point movement for the "best" position using the fruit yield model.
2. **Fitness Evaluation**:
   - Calculate fitness for each particle based on the fruit yield mode, which combines the traits and the habitat factors.
3. **Update Velocities and Positions**:
   - Adjust the particle velocities by using inertia, cognitive and social components
   - Update the particle position using the latest velocity, making sure they are within the boundary.
4. **Convergence**:
   - Tracking the best fitness value and particle position at each iteration
   - Stopping the algorithm if the maximum number of iterations is match, or the fitness is converging.

5. **Key Parameters:**

- **Inertia Weight (w)**: Controlling the influence of every particle's previous velocity.
- **Cognitive Coefficient (c1)**: Controls the particle's attraction to its own best position.
- **Social Coefficient (c2)**: Controls the particle's attraction to the global best position

E. Flowchart


START
EVALUATE FITNESS (fruit_yield_model)
UPDATE BEST POSITIONS -Personal Best -Global Best
UPDATE VELOCITIES AND POSITIONS
CONVERGENCE CHECK
OUTPUT RESULTS
END

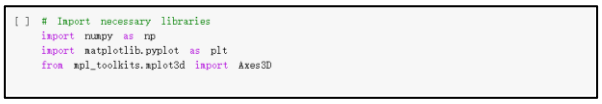
F. Visualizations

1. **Convergence Plot:** Illustrating the best yield iterations and demonstrating how the algorithm is converging towards a more optimized solution.

2. **2D Particle Movements**: Showing the distribution and movement of every particles in the 2D space over iterations, converging all of it toward the best optimal solution.

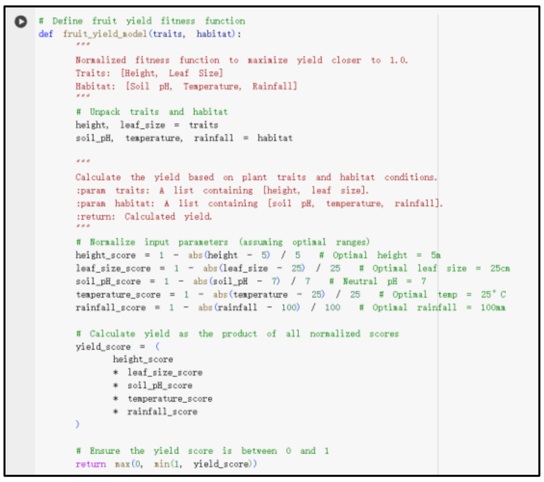
3. **3D particle movements**: Visualizing the particle movement in 3D space and highlighting the converging movement towards optimized position

V. RESULT & DISCUSSION



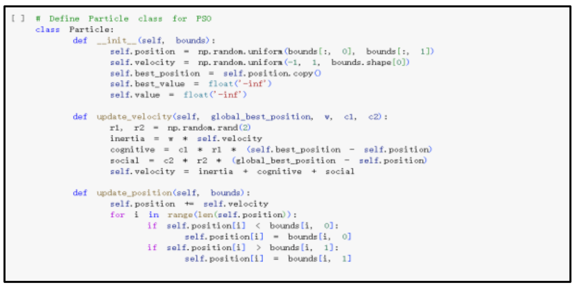
```
[ ] # Import necessary libraries
    import numpy as np
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D
```

Figure 2:Import NumPy & matplotlib library



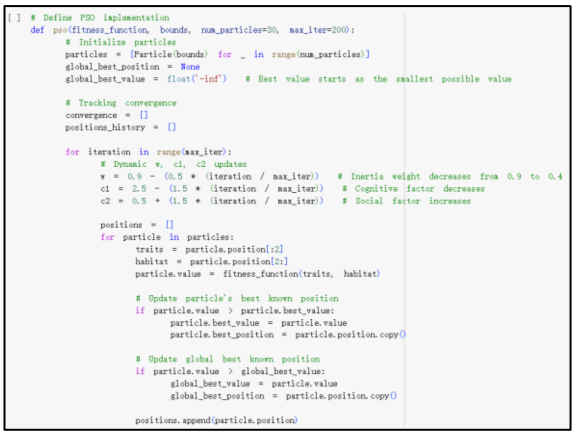
```
# Define fruit yield fitness function
def fruit_yield_model(traits, habitat):
    """
    Normalized fitness function to maximize yield closer to 1.0.
    Traits: [Height, Leaf Size]
    Habitat: [Soil pH, Temperature, Rainfall]
    """
    # Unpack traits and habitat
    height, leaf_size = traits
    soil_pH, temperature, rainfall = habitat

    """
    Calculate the yield based on plant traits and habitat conditions.
    :param traits: A list containing [height, leaf size].
    :param habitat: A list containing [soil pH, temperature, rainfall].
    :return: Calculated yield.
    """
    # Normalize input parameters (assuming optimal ranges)
    height_score = 1 - abs(height - 5) / 5   # Optimal height = 5m
    leaf_size_score = 1 - abs(leaf_size - 25) / 25   # Optimal leaf size = 25cm
    soil_pH_score = 1 - abs(soil_pH - 7) / 7   # Neutral pH = 7
    temperature_score = 1 - abs(temperature - 25) / 25   # Optimal temp = 25° C
    rainfall_score = 1 - abs(rainfall - 100) / 100   # Optimal rainfall = 100mm

    # Calculate yield as the product of all normalized scores
    yield_score = (
        height_score
        * leaf_size_score
        * soil_pH_score
        * temperature_score
        * rainfall_score
    )

    # Ensure the yield score is between 0 and 1
    return max(0, min(1, yield_score))
```

Figure 3: Definition of fruit yield fitness function



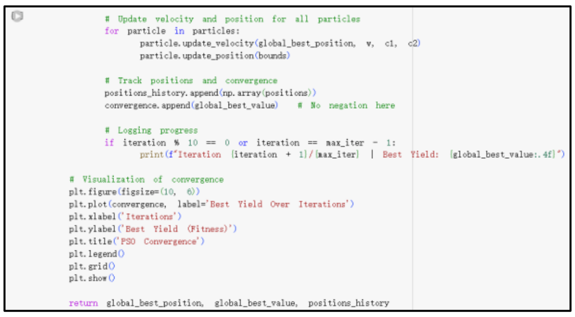
```
[ ] # Define Particle class for PSO
    class Particle:
        def __init__(self, bounds):
            self.position = np.random.uniform(bounds[:, 0], bounds[:, 1])
            self.velocity = np.random.uniform(-1, 1, bounds.shape[0])
            self.best_position = self.position.copy()
            self.best_value = float('-inf')
            self.value = float('-inf')

        def update_velocity(self, global_best_position, w, c1, c2):
            r1, r2 = np.random.rand(2)
            inertia = w * self.velocity
            cognitive = c1 * r1 * (self.best_position - self.position)
            social = c2 * r2 * (global_best_position - self.position)
            self.velocity = inertia + cognitive + social

        def update_position(self, bounds):
            self.position += self.velocity
            for i in range(len(self.position)):
                if self.position[i] < bounds[i, 0]:
                    self.position[i] = bounds[i, 0]
                if self.position[i] > bounds[i, 1]:
                    self.position[i] = bounds[i, 1]
```

Figure 4:Definition of particle class of PSO



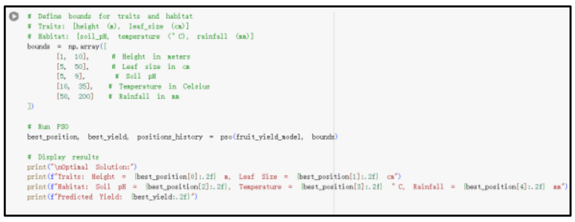
```
[ ] # Define PSO implementation
    def pso(fitness_function, bounds, num_particles=30, max_iter=200):
        # Initialize particles
        particles = [Particle(bounds) for _ in range(num_particles)]
        global_best_position = None
        global_best_value = float('-inf')   # Best value starts as the smallest possible value

        # Tracking convergence
        convergence = []
        positions_history = []

        for iteration in range(max_iter):
            # Dynamic w, c1, c2 updates
            w = 0.9 - (0.5 * (iteration / max_iter))   # Inertia weight decreases from 0.9 to 0.4
            c1 = 2.5 - (1.5 * (iteration / max_iter))   # Cognitive factor decreases
            c2 = 0.5 + (1.5 * (iteration / max_iter))   # Social factor increases

            positions = []
            for particle in particles:
                traits = particle.position[:2]
                habitat = particle.position[2:]
                particle.value = fitness_function(traits, habitat)

                # Update particle's best known position
                if particle.value > particle.best_value:
                    particle.best_value = particle.value
                    particle.best_position = particle.position.copy()

                # Update global best known position
                if particle.value > global_best_value:
                    global_best_value = particle.value
                    global_best_position = particle.position.copy()

                positions.append(particle.position)
```

Figure 5:Definition of PSO implementation (1)



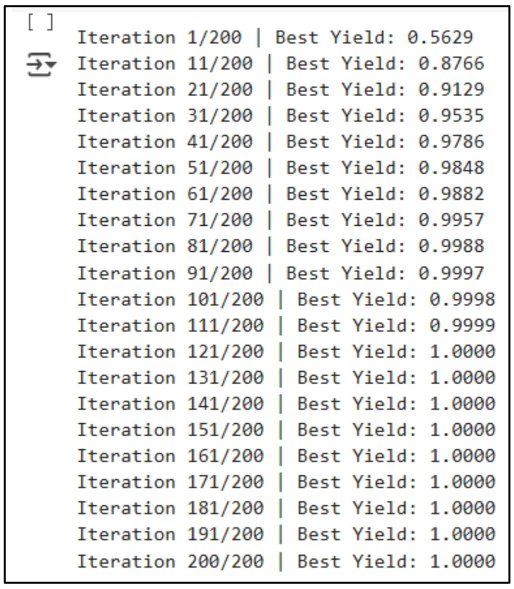
```
            # Update velocity and position for all particles
            for particle in particles:
                particle.update_velocity(global_best_position, w, c1, c2)
                particle.update_position(bounds)

            # Track positions and convergence
            positions_history.append(np.array(positions))
            convergence.append(global_best_value)   # No negation here

            # Logging progress
            if iteration % 10 == 0 or iteration == max_iter - 1:
                print(f"Iteration {iteration + 1}/{max_iter} | Best Yield: {global_best_value:.4f}")

        # Visualization of convergence
        plt.figure(figsize=(10, 6))
        plt.plot(convergence, label='Best Yield Over Iterations')
        plt.xlabel('Iterations')
        plt.ylabel('Best Yield (Fitness)')
        plt.title('PSO Convergence')
        plt.legend()
        plt.grid()
        plt.show()

        return global_best_position, global_best_value, positions_history
```

Figure 6: Definition of PSO implementation (2)



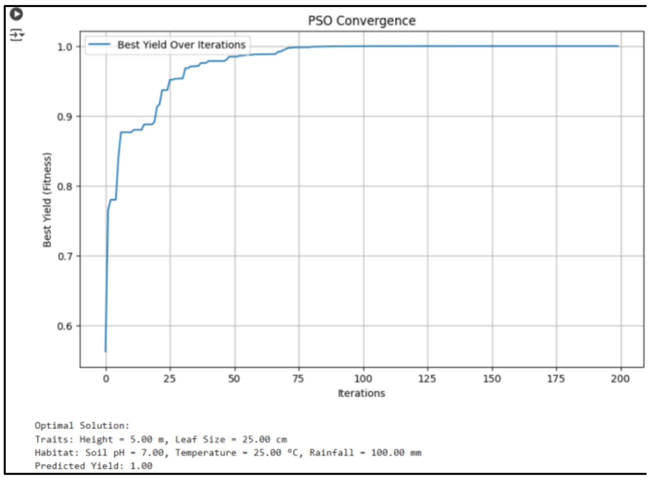
```
# Define bounds for traits and habitat
# Traits: [height (m), leaf_size (cm)]
# Habitat: [soil_pH, temperature (° C), rainfall (mm)]
bounds = np.array([
        [1, 10],     # Height in meters
        [5, 50],     # Leaf size in cm
        [5, 9],       # Soil pH
        [10, 35],    # Temperature in Celsius
        [50, 200]    # Rainfall in mm
])

# Run PSO
best_position, best_yield, positions_history = pso(fruit_yield_model, bounds)

# Display results
print("\nOptimal Solution:")
print(f"Traits: Height = {best_position[0]:.2f} m, Leaf Size = {best_position[1]:.2f} cm")
print(f"Habitat: Soil pH = {best_position[2]:.2f}, Temperature = {best_position[3]:.2f} ° C, Rainfall = {best_position[4]:.2f} mm")
print(f"Predicted Yield: {best_yield:.2f}")
```

Figure 7:Definition of PSO bounds of traits and habitat



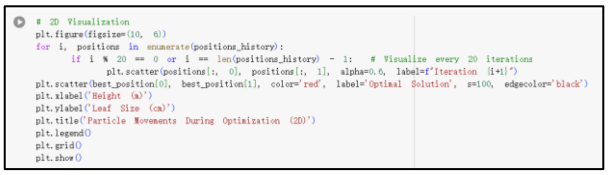
```
[ ]
      Iteration 1/200 | Best Yield: 0.5629
      Iteration 11/200 | Best Yield: 0.8766
      Iteration 21/200 | Best Yield: 0.9129
      Iteration 31/200 | Best Yield: 0.9535
      Iteration 41/200 | Best Yield: 0.9786
      Iteration 51/200 | Best Yield: 0.9848
      Iteration 61/200 | Best Yield: 0.9882
      Iteration 71/200 | Best Yield: 0.9957
      Iteration 81/200 | Best Yield: 0.9988
      Iteration 91/200 | Best Yield: 0.9997
      Iteration 101/200 | Best Yield: 0.9998
      Iteration 111/200 | Best Yield: 0.9999
      Iteration 121/200 | Best Yield: 1.0000
      Iteration 131/200 | Best Yield: 1.0000
      Iteration 141/200 | Best Yield: 1.0000
      Iteration 151/200 | Best Yield: 1.0000
      Iteration 161/200 | Best Yield: 1.0000
      Iteration 171/200 | Best Yield: 1.0000
      Iteration 181/200 | Best Yield: 1.0000
      Iteration 191/200 | Best Yield: 1.0000
      Iteration 200/200 | Best Yield: 1.0000
```

Figure 8: Result of optimal solution for best yield from first to 200th iteration of AI processing



```
Optimal Solution:
Traits: Height = 5.00 m, Leaf Size = 25.00 cm
Habitat: Soil pH = 7.00, Temperature = 25.00 °C, Rainfall = 100.00 mm
Predicted Yield: 1.00
```

Figure 9:Plotted PSO Convergence



```
# 2D Visualization
plt.figure(figsize=(10, 6))
for i, positions in enumerate(positions_history):
        if i % 20 == 0 or i == len(positions_history) - 1:   # Visualize every 20 iterations
                plt.scatter(positions[:, 0], positions[:, 1], alpha=0.6, label=f"Iteration {i+1}")
plt.scatter(best_position[0], best_position[1], color='red', label='Optimal Solution', s=100, edgecolor='black')
plt.xlabel('Height (m)')
plt.ylabel('Leaf Size (cm)')
plt.title('Particle Movements During Optimization (2D)')
plt.legend()
plt.grid()
plt.show()
```

Figure 10: Coding of implementation for 2D visualization

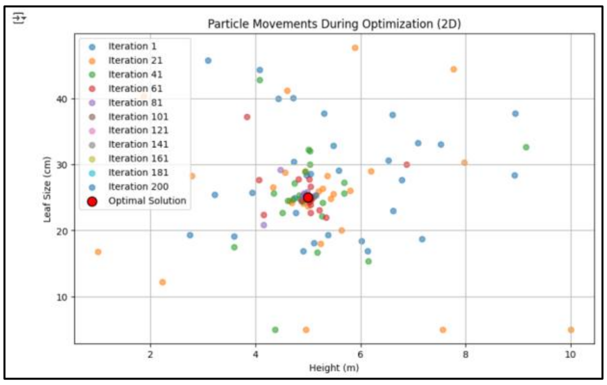
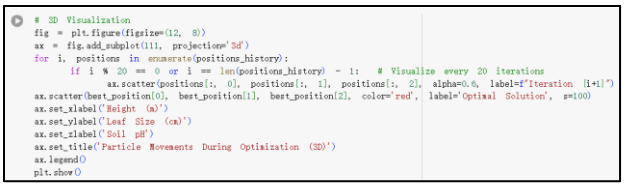


Figure 11: Result of 2D visualization



```
# 3D Visualization
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
for i, positions in enumerate(positions_history):
        if i % 20 == 0 or i == len(positions_history) - 1:   # Visualize every 20 iterations
                ax.scatter(positions[:, 0], positions[:, 1], positions[:, 2], alpha=0.6, label=f"Iteration {i+1}")
ax.scatter(best_position[0], best_position[1], best_position[2], color='red', label='Optimal Solution', s=100)
ax.set_xlabel('Height (m)')
ax.set_ylabel('Leaf Size (cm)')
ax.set_zlabel('Soil pH')
ax.set_title('Particle Movements During Optimization (3D)')
ax.legend()
plt.show()
```

Figure 12: Coding of implementation for 3D visualization

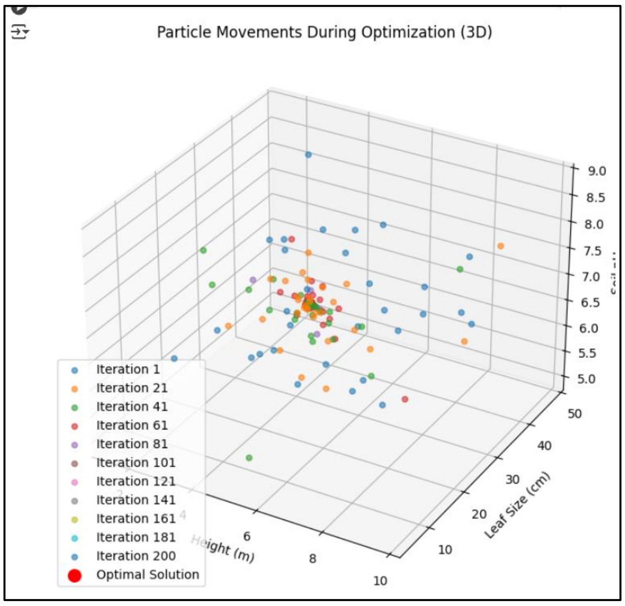


Figure 13: Result of 3D visualization

VI. DISCUSSION

The results demonstrate the effectiveness of the implemented optimization algorithm in identifying optimal parameters to maximize the desired outcome. Over successive iterations, the solutions converged toward an optimal region in the search space, indicating the fitness function successfully guided the process toward maximized results.

The algorithm identified specific parameter ranges that contributed significantly to the improved outcome. These include optimized values for key variables, such as physical traits and environmental conditions. The results show that the system achieved near-optimal values for all parameters, leading to high overall performance.

The evaluation highlights the optimization algorithm's ability to determine the best settings for improving the intended results. As the iterations went on, the solutions gradually approached an ideal area of the search space, proving that the fitness function was effective in promoting the best possible outcomes. A global optimum is approached by the solutions, and the fitness curve shows a consistent improvement over the iterations. Visual depictions of the parameter space show a prominent peak that denotes the ideal set of variables, highlighting the algorithm's ability to balance search space expansion with the improvement of workable solutions.

In summary**,** the code simulates the behavior of a swarm of particles to find the optimal combination of traits and habitat conditions that maximize fruit yield. The PSO algorithm guides the particles towards the best solution by iteratively updating their positions and velocities based on their individual and collective knowledge. The visualization tools help to understand the optimization process and the final solution.

VII. CONCLUSION

The study that was made demonstrates that Particle Swarm Optimization (PSO) is effective to optimize the traits and the habitat conditions to maximize all the fruit yielded. By simulating the behavior of the particles to explore the space for search, this PSO algorithm identifies the best combination by comparing the plant height, leaf size, soil pH level, the temperature and how much rainfall it produces. The results that were obtained confirmed that the robustness of the fitness function when guiding the swarm is completely follows the global optimum, evidenced by the steady convergence patterns and also the visualization that were shown at the parameter space.

By adjusting most of the inertia weight, cognitive and social coefficients, most of the particles can maintain its own balance between the exploring and exploiting thus successfully going towards the most optimal solution. This configuration that was used has the best fitness and is extremely close to almost one hundred percent, showing that PSO can effectively solve optimization problems faced by agricultural and ecological systems.

Now, the visualizations depicting 2D and 3D motions of particles gave useful information about the optimization progress and convergence. This study shows that most people underestimate the potential of PSO in practical uses, which can be used as an example in crop yield optimization. This study also shows that this PSO offers a big approach for preventing similar challenges across most industry.

APPENDIX

Project coding of Particle Swarm Optimization (PSO): https://colab.research.google.com/drive/1cblANk2gMC6uuFWXpv3aNkSHSC4ZUYIt?usp=sharing#scrollTo=lIal8Kv-kzeo

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Universiti Teknikal Malaysia Melaka (UTeM), particularly the Department of Faculty of Electronics & Computer Technology & Engineering (FTKEK), for providing us with the resources and support necessary to carry out this project.

We are especially thankful to our lecturer, Prof. Madya Ir. Dr. Syafeeza binti Ahmad Radzi, for her guidance, encouragement, and constructive feedback throughout the course of this study. Her expertise in optimization techniques played a significant role in refining our research approach.

We also wish to acknowledge our classmates for their ethical and peer support. Their valuable insights, feedback, and collaborative discussions were instrumental in helping us improve and complete this work

.

CONFLICT OF INTEREST

We confirmed that there is no conflict of interest regarding the publication of the paper.

AUTHOR CONTRIBUTION

We confirmed contribution to the paper as follows: Introduction and Theory: Aiman; Literature Review and Result with discussion: Aiman, Yeo Kheong Jie, Dina and Syah; Methodology: Dina and Syah; Abstract and Conclusion: Yeo Kheong Jie. All authors had reviewed the findings and approved the final manuscript of the project.

REFERENCES

[1] J. B. Odili, Mohd, and A. Noraziah, “Swarm Intelligence Optimization Algorithms: A Review,” *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 10, no. 1–4, pp. 139–142, 2018, Accessed: Jan. 22, 2025. [Online]. Available: https://jtec.utem.edu.my/jtec/article/view/3606

[2] D. M. Soomro, M. Y. Al-izzi, H. A. Soodi, N. M. Elasager, and S. C. Chong, “Optimal Restoration for Distribution System using PSO and ANN,” *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 3–7, pp. 1–6, 2017, Accessed: Jan. 22, 2025. [Online]. Available: https://jtec.utem.edu.my/jtec/article/view/3057

[3] M. A. El-Shorbagy and A. E. Hassanien, “Particle Swarm Optimization from Theory to Applications,” *International Journal of Rough Sets and Data Analysis*, vol. 5, no. 2, pp. 1–24, Apr. 2018, doi: https://doi.org/10.4018/ijrsda.2018040101.

[4] E. H. Houssein, A. G. Gad, K. Hussain, and P. N. Suganthan, “Major Advances in Particle Swarm Optimization: Theory, Analysis, and Application,” *Swarm and Evolutionary Computation*, vol. 63, p. 100868, Jun. 2021, doi: https://doi.org/10.1016/j.swevo.2021.100868.

[5] C. Witt, “Theory of Particle Swarm Optimization,” *Theoretical computer science*, pp. 197–223, Feb. 2011, doi: https://doi.org/10.1142/9789814282673_0007.

[6] A. G. Gad, “Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review,” *Archives of Computational Methods in Engineering*, vol. 29, no. 5, pp. 2531–2561, Apr. 2022, doi: https://doi.org/10.1007/s11831-021-09694-4.

[7] “PARTICLE SWARM OPTIMIZATION AND IT’S APPLICATIONS,” *Lingaya’s Vidyapeeth*, Aug. 08, 2023. https://www.lingayasvidyapeeth.edu.in/particle-swarm-optimization-and-its-applications/