Assignment 2 – Swarm Intelligence
CS 451 - Computational Intelligence, Spring 2025

Syeda Samah Daniyal 07838, Aina Shakeel 08430

March 15, 2025

Contents

1 Solving facility layout problem using ACO

1.1 Introduction

This section explores the application of Ant Colony Optimization (ACO) to a hospital layout problem, a specific case of facility layout optimization. The objective is to assign n facilities to n locations while minimizing patient movement costs, represented by a distance matrix and a flow matrix. The approach is tested on the QAP instance Els19. Various ACO parameter configurations are evaluated, iteratively refining them to improve results. The following sections discuss problem formulation, solution methodology, and the impact of parameter variations on convergence and solution quality. Effective facility layout design is crucial in hospitals, influencing patient flow, wait times, and operational efficiency. This study demonstrates how ACO can optimize a simplified version of this problem by minimizing total patient flow costs.

1.2 Problem Formulation

In this problem, there are n facilities and n available locations. Two $n \times n$ matrices are provided:

- **Distance Matrix:** Represents the distances between locations.
- **Flow Matrix:** Represents the flow (movement) of patients between facilities.

The objective is to minimize the cost defined as:

$$\text{Cost} = \sum_{i=1}^{n} \sum_{j=1}^{n} \text{flow}_{ij} \times \text{distance}_{\pi(i)\pi(j)} \tag{1}$$

where π is a permutation indicating the assignment of facility i to location $\pi(i)$.
For our dataset, Els19, $n = 19$.

1.3 Methodology

The ACO algorithm mimics the behavior of ants seeking paths between their colony and food sources (foraging). The key components of the ACO algorithm in this application include:

- **Pheromone Trail (τ):** A matrix that indicates the desirability of assigning a facility to a location.
- **Desirability/ Heuristic (η):** Derived from the total flow of each facility and the total distance of each location. Facilities with high flow are more "central" and should be mapped to locations with low cumulative distance. The desirability in the code is computed using the following formula:

$$\eta_{i,j} = \frac{\sum_k f_{i,k}}{\sum_k d_{j,k} + \epsilon}$$

where:

  - $\eta_{i,j}$ is the desirability value for assigning facility i to location j.
  - $f_{i,k}$ represents the flow between facility i and other facilities k, summing over all k.
  - $d_{j,k}$ represents the distance between location j and other locations k, summing over all k.
  - $\epsilon = 10^{-10}$ is added to the denominator to prevent division by zero.

  This formula favors assigning facilities with high total flow to locations with low total distance, optimizing patient movement efficiency.

- **Probability Function:** When assigning facility i to an available location j, the probability is calculated as:

$$p_{ij} = \frac{\tau_{ij}^{\alpha}\, \eta_{ij}^{\beta}}{\sum_{k \in \text{allowed}} \tau_{ik}^{\alpha}\, \eta_{ik}^{\beta}} \tag{2}$$

  where α and β control the relative importance of pheromone trails and desirability information, respectively.

- **Pheromone Update:** After each iteration, pheromones evaporate (controlled by the evaporation rate γ) and new pheromone is deposited proportionally to the quality of the solutions found.

- **Pheromone Limits:** To avoid stagnation and premature convergence, maximum and minimum limits (τ_{min} and τ_{max}) are imposed on the pheromone levels. After the evaporation and deposition phases in each iteration, the pheromone values are clipped to remain within these bounds.

The algorithm works by creating an effective balance between exploring new solutions and exploiting known good assignments as follows:

- **Reading the Instance:** A function reads the QAP instance from a text file, where the first line indicates the problem size, followed by the distance matrix (describing distances between locations) and the flow matrix (describing flows between facilities).

- **Creating the QAP Instance:** The ‘QAPInstance’ class stores the matrices and provides a method to compute the total cost for any facility-to-location assignment using the formula of cost mentioned above.

- **Simulating Ants:** Each ant (implemented in the ‘Ant’ class) builds a complete solution by:
  - Precomputing desirability values based on each facility’s total flow and each location’s total distance (formula given above).
  - Processing facilities in descending order of flow.
  - For each facility, calculating the probability of assigning it to each available location using the current pheromone levels and desirability values through the probability function mentioned above.

  - Selecting a location based on these probabilities, thereby constructing a solution.

- **Pheromone Update and Iteration:** The 'AntColony' class runs the ACO process for a fixed number of iterations. In each iteration:
  - A group of ants generates solutions.
  - The best solution and average cost are tracked.
  - The pheromone matrix is updated by applying evaporation (reducing pheromone levels by a factor determined by the evaporation rate) and then reinforcing the pheromone on the paths used by each ant based on the quality of their solutions.
  - The pheromone values are then clipped to remain within the bounds $\tau_{\min}$ and $\tau_{\max}$.

- **Convergence Analysis:** The algorithm plots the best and average costs over iterations to visualize how the solution quality improves (or plateaus) over time. This helps determine if the algorithm has converged to a stable solution or if further iterations yield diminishing returns.

1.4 Experimental Setup and Results

Several different parameter configurations were tested.

1.4.1 Initial Run

The first experiment used:

- **Number of Ants:** 20
- **Iterations:** 100
- $\alpha = 1$ (Pheromone Influence)
- $\beta = 1$ (Desirability Influence)
- $\gamma = 0.8$ (Evaporation Rate)
- **Q:** 1 (Constant for pheromone update)
- **Initial Pheromone:** 1.0

Output:

```
Iteration 1: Best Cost = 30857216.0, Average Cost = 40264659.3
...
Iteration 100: Best Cost = 23690642.0, Average Cost = 27151480.0
```

The best assignment obtained was:

```
[18, 10, 8, 2, 9, 14, 3, 4, 17, 5, 1, 0, 7, 15, 13, 12, 11, 16, 6]
```

with a cost of **23690642.0**. Figure 1 shows the convergence behavior of initial setup.

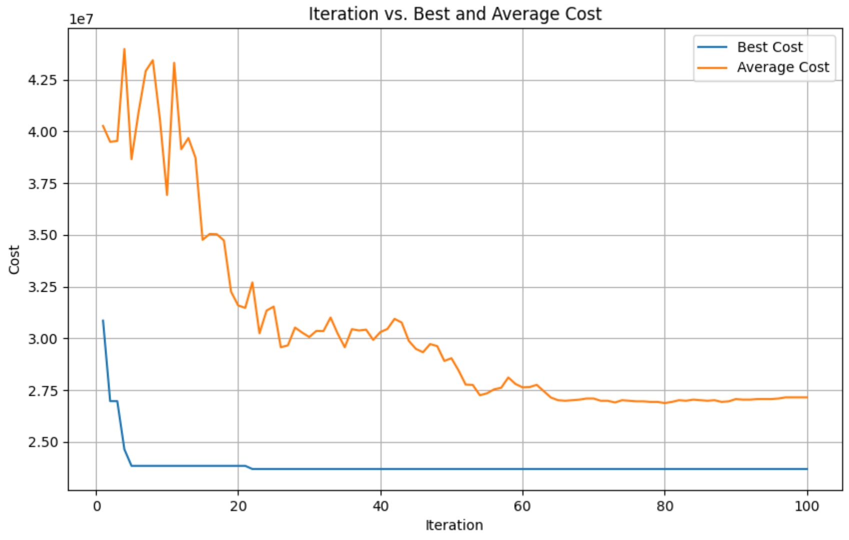

Figure 1: Graph showing best and average costs over iterations.

1.4.2 Best Solution Run (After Fine-tuning)

After experimenting with different parameter values, the best results were achieved using the following parameters and refining ACO by introducing maximum and minimum limits on the pheromone levels:

- **Number of Ants:** 80
- **Iterations:** 200
- $\alpha = 7$ (Pheromone Influence)
- $\beta = 4$ (Desirability Influence)
- $\gamma = 0.6$ (Evaporation Rate)
- **Q:** 1 (Constant for pheromone update)
- **Initial Pheromone:** 1.0
- **Minimum Pheromone:** 0.1
- **Maximum Pheromone:** 10.0

Output:

```
Iteration 1: Best Cost = 24266632.0, Average Cost = 38963890.45
...
Iteration 200: Best Cost = 19115126.0, Average Cost = 38051251.65
```

The best assignment obtained was:

```
[16, 17, 18, 0, 1, 2, 3, 6, 13, 10, 4, 12, 8, 15, 14, 5, 7, 11, 9]
```

with a best cost of **17415440.0**.

Figure 2 shows the convergence behavior of the best outcome.

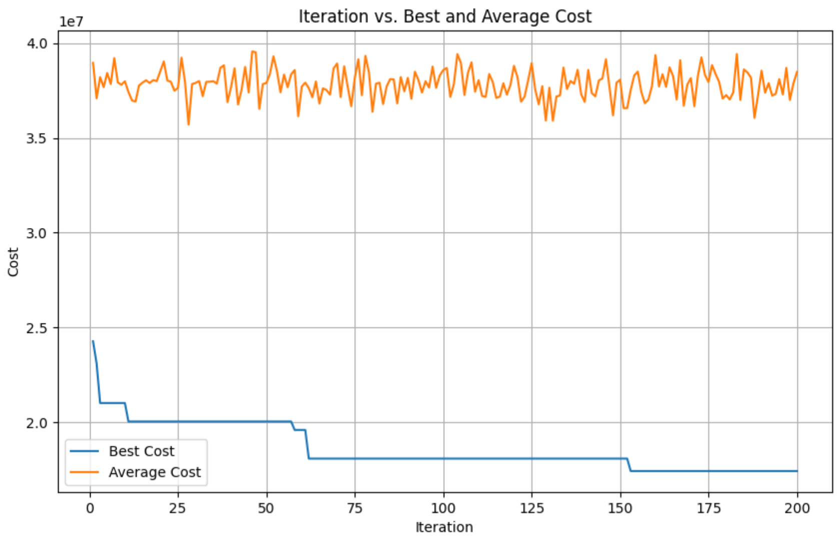

Figure 2: Graph showing best and average costs over iterations.

1.5 Analysis and Discussion

The experiments conducted indicate that the performance of the ACO algorithm is sensitive to the parameter settings. We noted the following observations on the simple ACO implementation:

(a) **Pheromone (α) vs. Desirability (β) Influence:**

- When the difference between the pheromone and desirability parameters was minimal, on average, there was little change in performance. This balance resulted in a stable search process but did not encourage sufficient exploration, leading to only modest improvements. The following was tested while keeping all other parameters the same as that of the optimal solution and $\alpha = 5$, $\beta = 4$.

  ```
  Iteration 1: Best Cost = 24563306.0, Average Cost = 37880291.2
  ...
  Iteration 100: Best Cost = 20454244.0, Average Cost = 27470662.0
  ```

  The best assignment obtained was:

  ```
  [16, 17, 7, 11, 2, 4, 12, 18, 9, 13, 1, 0, 10, 14, 15, 3, 8, 6, 5]
  Best cost: 20454244.0
  ```

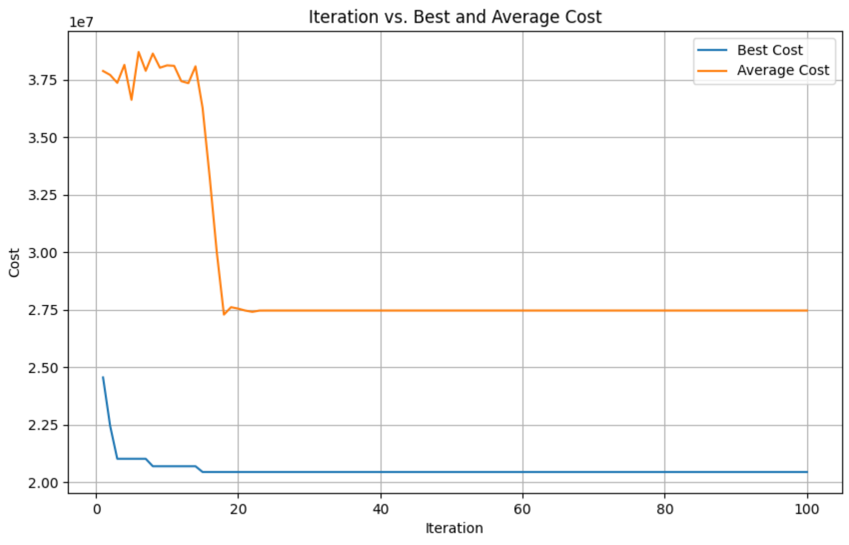

Figure 3: Graph showing best and average costs over iterations for $\alpha = 5$ and $\beta = 4$

- Conversely, when the desirability influence (β) was significantly greater than the pheromone influence (α), the algorithm's improvements were limited. An overemphasis on desirability information caused the search to be too greedy, often converging prematurely to suboptimal solutions, on average. The following was tested while keeping all other parameters the same as that of the optimal solution and $\alpha = 4$, $\beta = 7$.

```
Iteration 1: Best Cost = 21212004.0, Average Cost = 34658415.75
...
Iteration 100: Best Cost = 20045160.0, Average Cost = 28327346.0
```

  The best assignment obtained was:

```
[17, 18, 10, 12, 16, 14, 11, 15, 7, 8, 2, 9, 6, 3, 4, 5, 13, 1, 0]
Best cost: 20045160.0
```

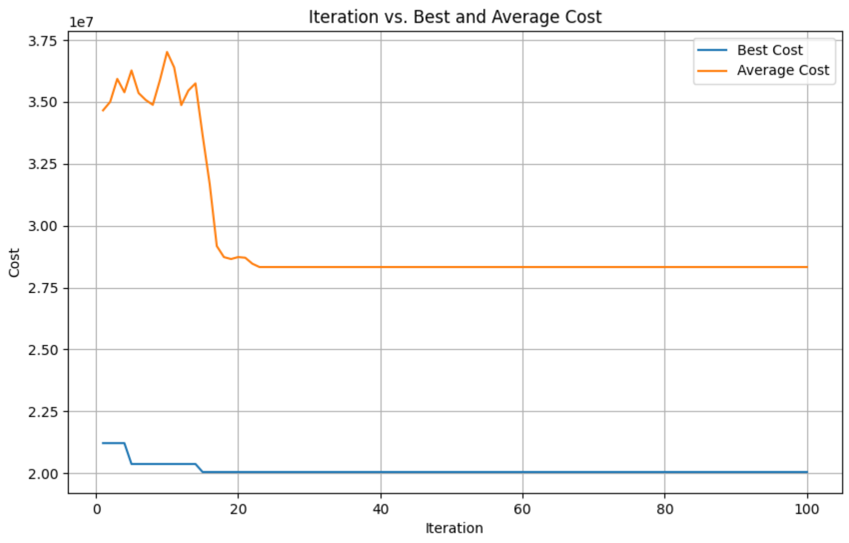

Figure 4: Graph showing best and average costs over iterations for $\alpha = 4$ and $\beta = 7$

- Hence, making pheromone influence (α) greater than desirability influence (β) showed the best result.

(b) **Evaporation Rate (γ):**

- When tuning the evaporation rate (γ), our experiments indicate that neither a high ($\gamma = 0.8$) nor a low ($\gamma = 0.4$) setting is optimal, as both lead to subpar performance compared to a balanced setting ($\gamma = 0.6$).
- At $\gamma = 0.8$, pheromone trails dissipate too quickly, reducing the algorithm's ability to reinforce good solutions. This leads to excessive exploration, preventing strong convergence. Our results show an initial sharp improvement in cost but little further improvement, with the average cost plateauing around 30215990.0.

```
Iteration 1: Best Cost = 23943692.0, Average Cost = 36877234.475
...
Iteration 100: Best Cost = 20809486.0, Average Cost = 30215990.0
```

The best assignment obtained was:

```
[17, 18, 10, 12, 8, 3, 6, 1, 0, 16, 14, 9, 2, 11, 4, 7, 13, 5, 15]
Best cost: 20809486.0
```

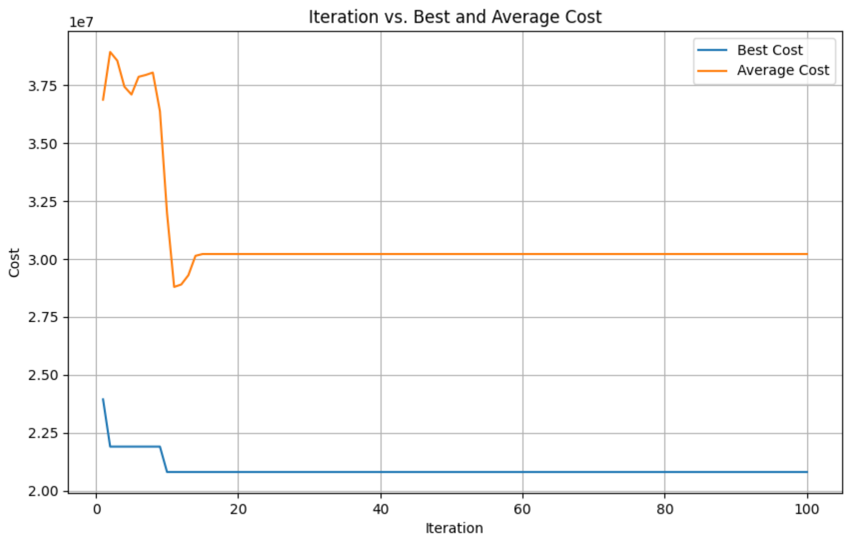

Figure 5: Graph showing best and average costs over iterations for $\gamma = 0.8$

- Conversely, at $\gamma = 0.4$, pheromone information lingers too long, causing over-exploitation of early solutions and limiting exploration. The best cost gradually improves to 19888006.0 by iteration 15 but then stagnates, with the average cost stabilizing at 25137208.0. This suggests premature convergence.

  ```
  Iteration 1: Best Cost = 21310768.0, Average Cost = 38572966.55
  ...
  Iteration 100: Best Cost = 19888006.0, Average Cost = 25137208.0
  ```

  The best assignment obtained was:

  ```
  [17, 16, 7, 2, 8, 11, 3, 6, 12, 10, 4, 9, 1, 5, 18, 15, 13, 0, 14]
  Best cost: 19888006.0
  ```

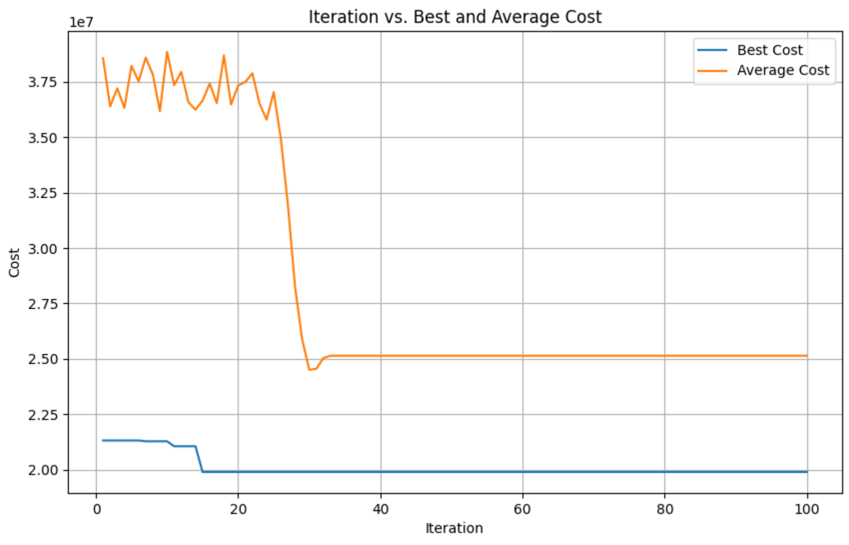

Figure 6: Graph showing best and average costs over iterations for $\gamma = 0.4$

- A balanced γ (around 0.6) helps maintain a better trade-off between exploration and exploitation, allowing the algorithm to refine solutions effectively without excessive exploration or premature convergence. This highlights the importance of proper parameter tuning in ACO for optimal performance.

(c) **Number of Iterations:**

- Reducing the number of iterations led to insufficient convergence, on average, as the pheromone trails did not have enough time to accumulate meaningful differences, resulting in lower-quality solutions. The following was tested while keeping all other parameters the same as that of the optimal solution and iterations = 50.

```
Iteration 1: Best Cost = 19996234.0, Average Cost = 36611111.2
...
Iteration 50: Best Cost = 19996234.0, Average Cost = 31312940.0
```

Best assignment:

```
[17, 18, 10, 1, 4, 11, 7, 14, 3, 6, 15, 2, 8, 9, 13, 0, 16, 5, 12]
Best cost: 19996234.0
```

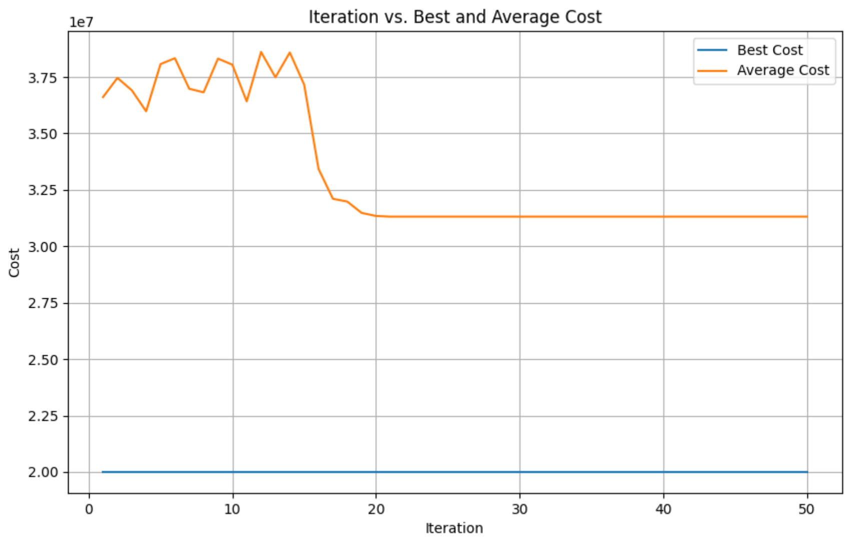

Figure 7: Graph showing best and average costs over iterations = 50

- Increasing the number of iterations beyond a certain threshold did not yield further improvements, on average, indicating that the algorithm reached a plateau where additional iterations provided diminishing returns. The following was tested while keeping all other parameters the same as that of the optimal solution and iterations = 150.

  ```
  Iteration 1: Best Cost = 20856210.0, Average Cost = 38256403.875
  ...
  Iteration 150: Best Cost = 20495626.0, Average Cost = 32586054.0
  ```

  Best assignment:

  ```
  [16, 17, 10, 18, 11, 1, 12, 0, 14, 2, 3, 13, 7, 6, 9, 15, 8, 5, 4]
  Best cost: 20495626.0
  ```

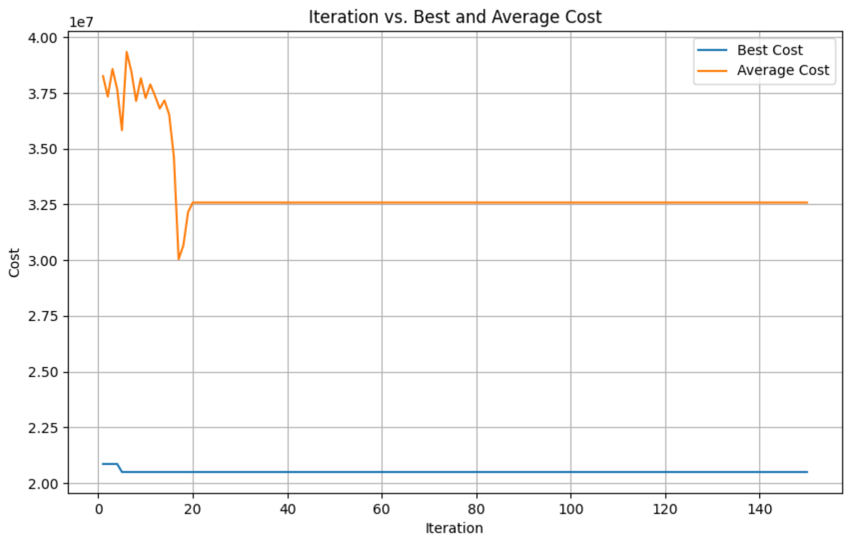

Figure 8: Graph showing best and average costs over iterations = 150

- **Effect of Pheromone Limits on Stagnation:** In addition to tuning the number of iterations, imposing maximum and minimum limits on pheromone values (τ_{min} and τ_{max}) plays a crucial role in preventing stagnation. Without these limits, pheromone trails can become excessively high or too low, leading the algorithm to over-exploit certain paths and ignore others. Clipping the pheromone values ensures that no path dominates too early, thereby preserving diversity in the search space. This balance between exploration and exploitation allows the algorithm to escape local optima and improve overall solution quality over successive iterations. Hence, since the refined ACO algorithm continued to yield improvements even beyond 100 iterations, we decided to extend the iteration count to 200. Figure 2 shows the result for refined ACO.

(d) **Number of Ants:**

- Increasing the number of ants generally allows for more comprehensive exploration of the solution space. For example, our fine-tuned run with 80 ants resulted in a significant improvement in the solution quality compared to the initial run.
- However, experiments with 50 ants and 120 ants, while keeping other parameters the same as that on optimal solution, revealed that there is an optimal range. With 50 ants, the algorithm converged quickly to a solution:

```
Iteration 1: Best Cost = 20529740.0, Average Cost = 36775395.32
...
Iteration 100: Best Cost = 20529740.0, Average Cost = 29020308.0
```

Best assignment:

```
[17, 18, 8, 16, 1, 10, 14, 12, 2, 7, 4, 5, 13, 15, 0, 9, 11, 3, 6]
```

```
Best cost: 20529740.0
```

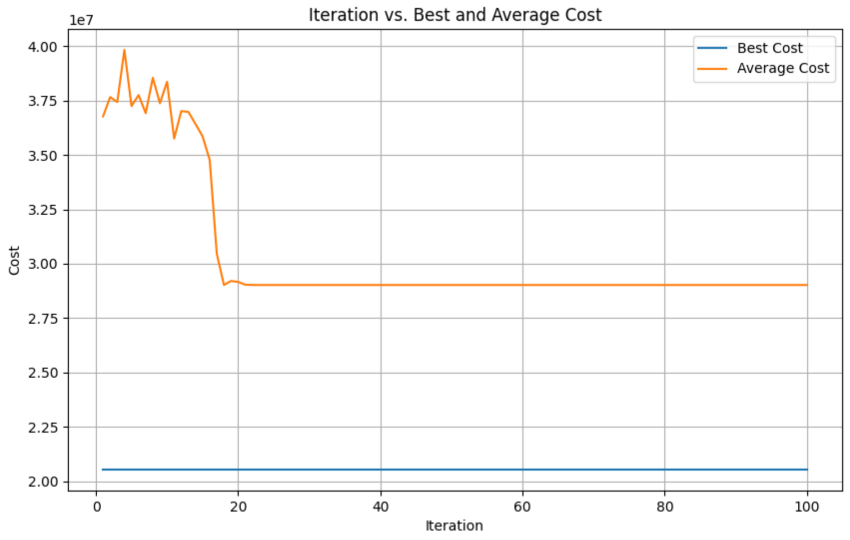

Figure 9: Graph showing best and average costs over iterations for Ants = 50

When increasing the number of ants to 120, the output did not improve as expected; in fact, the solution quality was slightly lower:

```
Iteration 1: Best Cost = 24301358.0, Average Cost = 38721568.5
...
Iteration 100: Best Cost = 20265380.0, Average Cost = 28051000.0
```

Best assignment:

```
[18, 17, 10, 16, 2, 1, 15, 6, 3, 7, 5, 4, 8, 14, 12, 11, 13, 9, 0]
Best cost: 20265380.0
```

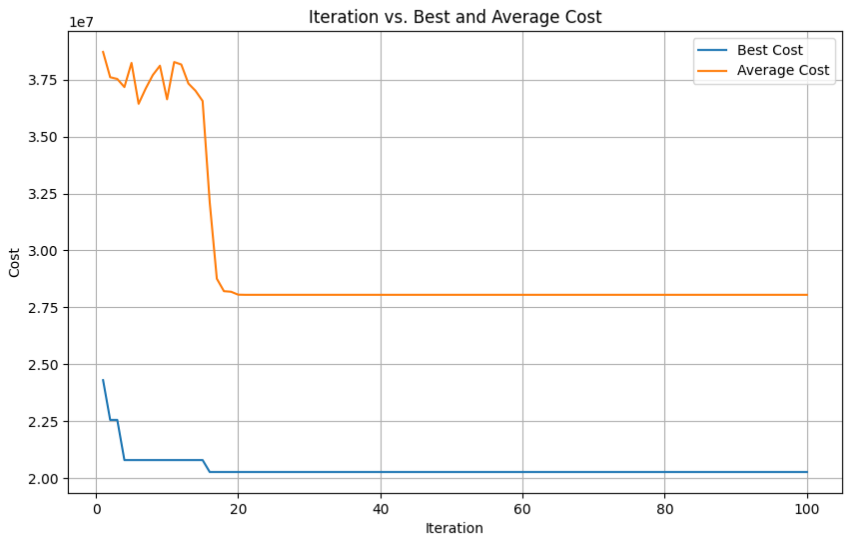

Figure 10: Graph showing best and average costs over iterations for Ants = 120

- These observations suggest that while increasing the number of ants can enhance exploration, too many ants may introduce excessive noise in the pheromone update process, thereby impeding convergence. Similarly, too few ants may cause premature convergence to a local optimum. Thus, there is an optimal range for the number of ants that balances exploration and exploitation.

These findings underscore the importance of fine-tuning ACO parameters. A balanced emphasis between pheromone and desirability influences, along with an optimal evaporation rate and a sufficient number of iterations and ants, is crucial for effective convergence toward high-quality solutions.

1.6 Conclusion

This demonstrated that Ant Colony Optimization can effectively address the hospital layout problem—a specialized facility layout challenge. By fine-tuning the parameters (notably increasing the number of ants and adjusting α, β, and γ), the algorithm achieved a significant reduction in the overall cost, improving the facility assignment. The convergence analysis reinforces the robustness and potential of the ACO approach for solving such optimization problems.

2 Visualizing Swarms

2.1 Introduction

Swarm behavior, observed in natural phenomena such as bird flocks and fish schools, has inspired numerous algorithms in artificial intelligence and computer graphics. This section presents a Processing-based simulation that visualizes swarm behavior by integrating concepts from Particle Swarm Optimization (PSO) and particle systems. The simulation features two distinct particle types: Aurora particles, which exhibit luminescent, trailing behavior reminiscent of natural auroras, and Sea particles, which demonstrate flocking dynamics influenced by PSO and environmental factors. A predator and a food source introduce additional complexity, creating an engaging and interactive experience. Real-time controls via sliders and keyboard/mouse inputs allow users to manipulate parameters, explore emergent behaviors, and gain insights into swarm dynamics.
The simulation can be seen here.

2.2 Concept and Problem Formulation

The primary objectives of this simulation are to:

- **Visualize complex swarm dynamics:** By combining PSO and flocking behavior to model interactions among particles.
- **Explore parameter sensitivity:** Users can adjust values aurora speed, sea brightness, plankton density, and behavioral coefficients to see their impact on the overall simulation.
- **Enhance interactivity and aesthetics:** Through environmental forces (wind, tide, solar activity), interactive elements (mouse disturbances, food source toggling), and visually appealing particle rendering.

The problem is formulated around simulating particles with varying behaviors:

1. **Aurora Particles** These particles exhibit motion driven by Perlin noise and sinusoidal functions, constrained to the upper portion of the canvas. They feature finite lifetimes and colorful, fading trails influenced by solar activity.
2. **Sea Particles** display dual-mode behavior. When a food source is active, they employ PSO mechanisms—adapting velocities based on personal and global best positions. In the absence of food, they follow flocking rules (cohesion, separation, and alignment) while reacting to external forces and predators.
3. **Predator and Environmental Forces** A predator influences sea particles by triggering escape responses, while wind, tides, and wave dynamics adds realism to their motion.

2.3 Algorithm and Implementation

The simulation is implemented using Processing and is structured into several key components:

2.3.1 AuroraParticle Class

- **Properties:** Each Aurora particle has a position, velocity, acceleration, lifetime, a dynamically updated color, and a trail array of 20 previous positions (trail).
- **Dynamics:** Motion is driven by:
  - Perlin noise (x-axis) and sinusoidal oscillations (y-axis) adjusted by solarActivity.
  - Alignment with nearby particles (within 50 units), contributing a small velocity adjustment.
  - Velocity capped at 3 + 0.5 * solarActivity, scaled by auroraSpeed.

  Particles reset to the top (y = 0) if they exceed y = height/2.25, with wrapping around horizontal boundaries.
- **Visual Effects:** A smooth, glowing trail is rendered, with alpha fading based on trail position, lifetime, and dynamic effects like banding and flicker. Colors shift between green, purple, pink, and red based on noise and solarActivity.

2.3.2 SeaParticle Class

- **Properties:** Sea particles have position, velocity, acceleration, brightness, lifetime, and a size (default 3). They also track PSO-specific personal best position and fitness.
- **Dual Behavior:**
  - *PSO-Based Movement:* When `foodActive` is true, particles compute fitness as the inverse distance to the target (food source). Velocity updates use inertia ($w = 0.7$), cognitive ($c_1 = 2.0$), and social ($c_2 = 2.0$) components, weighted 30% and 70% respectively, with a velocity limit of 2.
  - *Flocking Behavior:* Without food, particles use a grid-based neighbor search (25x25 grid) to compute:
    - *Cohesion:* Move toward the average position of neighbors within 50 units, scaled by cohesion.
    - *Separation:* Avoid crowding, scaled by separation.
    - *Alignment:* Match velocity with neighbors, scaled by alignment.

    They also respond to wind, tides, waves, and predator proximity (flee within 100 units, die within 20 units).
- **Visual Effects:** Particles are rendered as bezier-shaped blobs with a glowing halo. Brightness fades over time unless disturbed.

2.3.3 Predator Class

- The predator is a simple class with position and velocity, initialized at the canvas center. It moves randomly, bouncing off horizontal and vertical boundaries. Displayed as a red ellipse, it influences sea particles within 100 units and eliminates them within 20 units when predatorMode is active.

2.3.4 Interactivity and Control

Interactive elements are implemented using the ControlP5 library, which provides sliders to adjust parameters including:

- Aurora speed, sea brightness, and plankton density.
- Flocking coefficients: cohesion, separation, and alignment.
- Environmental factors: wind influence, tide influence, and solar activity.

Additionally, mouse interactions (pressing and dragging) and keyboard inputs allow the user to:

- **Mouse:** Pressing or dragging disturbs sea particles within a radius (50, doubled in predator mode), brightening and scattering them.
- **Keyboard:**
  - 'P' toggles predatorMode.
  - 'F' toggles foodActive, setting target to the mouse position.
  - Spacebar toggles isPaused.
  - 'R' resets particles and sliders.

2.4 Visualization and Interactivity

The aesthetic appeal of the simulation is enhanced through several design choices:

- **Blending Modes:** Additive blending (ADD) creates glowing effects, especially for aurora trails and sea particle halos, against a dark background with random star-like dots.
- **Dynamic Trails:** Aurora particles feature 20-point trails. Sea particles have subtle glow halos, and waves based on tideInfluence.
- **Real-Time Feedback:** Interactive sliders and mouse/keyboard inputs allow users to see immediate visual changes when adjusting parameters, making it a powerful tool for experimentation.

These elements work together to create an immersive simulation that is both informative and artistically engaging.

2.5 Conclusion

This simulation effectively demonstrates the complexity and beauty of swarm dynamics by merging PSO and particle system techniques. The integration of aurora and sea particles, along with interactive controls and environmental interactions, offers a rich platform for exploring emergent behavior.