Computational Intelligence Project: Dynamic Ant Colony Optimization

Tommaso Facchin

Ca' Foscari University of Venice

September 2025

1 Background

Ant Colony Optimization (ACO) is a population-based metaheuristic inspired by the foraging behavior of real ants, first introduced by Dorigo in the early 1990s [1]. In nature, ants deposit pheromone trails to communicate indirectly, enabling the colony to find the shortest paths between the nest and food sources. This emergent collective intelligence has been widely adapted for combinatorial optimization problems, including the Traveling Salesman Problem, vehicle routing, and network routing [2].
In ACO, a colony of m artificial ants iteratively constructs solutions on a problem graph $G = (N, E)$, where N is the set of nodes and E the set of edges. Each ant k moves from node i to node j with a probability defined as:

$$P_{ij}^k = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N_k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta} \tag{1}$$

where:

- τ_{ij} is the pheromone intensity on edge (i, j),
- η_{ij} is the heuristic desirability (often $1/d_{ij}$, with d_{ij} being the Euclidean distance),
- $\alpha \geq 0$ determines the relative importance of the pheromone trail τ_{ij}. Higher values of α increase the tendency of ants to follow previously successful paths, promoting exploitation.
- $\beta \geq 0$ determines the relative importance of the heuristic information η_{ij}. Higher values of β make ants more likely to choose edges based on local desirability (e.g., shorter distance), promoting exploration.
- N_k is the set of feasible nodes for ant k.

After all ants have completed a tour, pheromone levels are updated according to:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{2}$$

where $\rho \in (0, 1]$ is the pheromone evaporation rate, and $\Delta\tau_{ij}^k$ represents the pheromone deposited by ant k:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k}, & \text{if ant } k \text{ used edge } (i, j) \text{ in its tour,} \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

with Q being a positive constant and L_k the total length of ant k's tour. This iterative process leads to the emergence of high-quality solutions as pheromone accumulates on shorter paths, balancing **exploration** and **exploitation**.

2 My Approach

The project implements a *Dynamic Ant Colony Optimization (ACO)* system within a 2D interactive environment. The simulation is developed using the *Unity Game Engine* and programmed in *C#*, providing a real-time visualization of ant foraging behavior under dynamically changing conditions.
The primary objective is to model how virtual ants search for food while adapting to dynamic obstacles, such as walls that can be added or removed during the simulation. Each ant constructs its path based on a combination of:

- **Pheromone trails**, representing collective knowledge from previous ant explorations.
- **Heuristic information**, calculated as the inverse of the Euclidean distance to food sources.

Pheromone is deposited exclusively on the return trip to the nest. Upon returning, each ant's path is evaluated and compared to the current best-known path. The best path is preserved indefinitely to guide subsequent ants, while efficient paths receive a proportional pheromone boost, reinforcing successful routes without evaporation.
To facilitate experimentation and user interaction, the simulation provides an **interactive interface** with adjustable parameters:

- α, the pheromone influence factor
- β, the heuristic influence factor
- Amount of pheromone deposited per ant
- Pheromone evaporation rate
- Elitism boost factor
- Simulation speed

Additionally, a small inventory system allows users to dynamically modify the environment by placing:

- Walkable paths (black)
- Walls (brown)
- Food sources (yellow)

The simulation supports environment navigation and zooming using *WASD keys*, and a counter displays the total food collected by the colony in the top-left corner. While multiple food items can be added simultaneously, this may impact performance and increase computational lag.
This approach provides a flexible and interactive platform to observe the emergent behaviors of ants in a dynamic environment, highlighting the adaptability of ACO to real-time changes.

3 Implementation

The Dynamic ACO simulation is implemented in *Unity* using *C#*. The project is structured to separate core algorithmic logic from visualization and user interaction, providing modularity and maintainability. The main components of the implementation are summarized below.

3.1 Project Structure

- **Assets/Scripts/**: contains all C# scripts implementing the ACO algorithm, ant behavior, environment management, and user interface.
- **Assets/Prefabs/**: stores reusable objects such as ants, food sources, walls, and walkable paths.
- **Assets/Scenes/**: includes the main simulation scene and optional test scenarios.
- **Build/**: contains executable files for Windows and other platforms, allowing users to run the simulation without Unity.

3.2 Algorithm

The simulation follows the standard ACO procedure with modifications for dynamic updates. The main workflow for each simulation step is:

Algorithm 1 Dynamic ACO Simulation Loop

Initialize environment and place ants at the nest
Initialize pheromone levels τ_{ij} uniformly
while simulation is running **do**
    for each ant k **do**
        Choose next node j based on $P_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}$
        Move to node j
        if food reached **then**
            Pick up food and return to nest
            Deposit pheromone along the path
            Update best path if current path is shorter
        end if
    end for
    Evaporate pheromones: $\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}$
    Update visualization and user interface
    Handle dynamic environment changes (add/remove walls, paths, or food)
end while

3.3 Parameter Configuration

The simulation exposes several parameters that can be adjusted through the user interface, allowing you to experiment and fine-tune ant behavior in real time:

- α: the pheromone influence factor, which controls how strongly ants follow previously reinforced paths.
- β: the heuristic influence factor, which biases ants toward cells closer to food sources.
- Pheromone deposit amount per ant: the quantity of pheromone an ant leaves along its return path.
- Evaporation rate ρ: determines how quickly pheromone trails decay, affecting how long past paths influence ant decisions.
- Elitism boost factor: extra reinforcement applied to the colony's best-known path, encouraging ants to exploit high-quality solutions.
- Simulation speed: adjusts the pace of the simulation for faster or slower visualization.

In practice, I limited the number of ants to 300 to avoid performance issues, more than this caused noticeable lag during real-time simulation. Adjusting these parameters lets you explore the balance between **exploration** (trying new paths) and **exploitation** (following successful trails), and observe how quickly the colony converges to effective solutions under different settings.

3.4 Ant Foraging and Returning Behavior

Each ant in the simulation operates in one of two states: **Foraging** or **Returning**. The ant behavior is implemented in the `Ant` class, which encapsulates movement, path selection, and pheromone deposition.

Foraging: While foraging, ants explore the environment to locate food sources. The next cell is chosen based on a combination of:

- Pheromone intensity on neighboring cells
- Heuristic desirability calculated as the inverse of the distance to the nearest food
- Local exploration constraints, including a tabu list to avoid revisiting recently traversed cells

The `ChooseNextCell()` method implements this probabilistic selection. Ants also use a limited forward and side range to simulate a radar-like vision, prioritizing cells that are reachable and walkable. If food is detected within the radar radius, the ant moves directly towards it. Once a food source is reached, the ant picks up one unit of food and switches to the **Returning** state.

Returning: In the returning state, ants attempt to navigate back to the nest efficiently while depositing pheromone along the path. The method `AcoReturnToNest()` evaluates candidate cells in the forward and lateral directions relative to the previous movement, assigning weights based on pheromone intensity and distance to the nest.

- Pheromone is incrementally deposited along the return path to reinforce successful routes.
- Cells in the tabu list are generally avoided to prevent loops.
- A directional bias discourages immediate reversal of the previous step unless necessary.

Upon reaching the nest, the ant deposits the collected food, clears its tabu list, and evaluates whether the completed path is the new best-known path for the colony. The ant then re-enters the **Foraging** state to continue exploration.
This dual-state mechanism allows the simulation to capture the core dynamics of ACO, with foraging ants exploring the environment and returning ants reinforcing successful paths through pheromone deposition, adapting in real-time to dynamic obstacles and changes in the environment

3.5 Pheromone Management

The simulation keeps track of pheromone levels on a grid over the environment, which guides ant movement. To balance realism and performance, pheromone management is handled with a few key strategies:

Deposition: Ants deposit pheromone along their return path to the nest. For a cell (i, j), the deposited amount is:

$$\Delta\tau_{ij} = \tau_{base} \cdot \frac{v_{sim}}{v_{max}} \tag{4}$$

Here, τ_{base} is the base pheromone deposit per ant, v_{sim} is the current simulation speed, and v_{max} is the maximum speed. This normalization ensures that pheromone deposition remains consistent even if the simulation speed changes.

Evaporation: Pheromone gradually fades over time at a rate ρ, reducing the influence of older trails. For each cell not part of the elitist path:

$$\tau_{ij} \leftarrow (1 - \rho)\,\tau_{ij} \tag{5}$$

To improve performance, evaporation and visual updates are processed in batches rather than all at once, which helps maintain smooth real-time simulation even with hundreds of ants. Cells on the elitist path are exempt from evaporation to preserve the colony's best-known routes.

Elitism Boost: When a newly found path is shorter than the current best-known path, pheromone along that path is increased according to an elitism factor E:

$$\tau_{ij} \leftarrow \tau_{ij} + \tau_{base} \cdot E, \quad \forall (i,j) \in \text{best path} \tag{6}$$

This selectively strengthens high-quality paths while leaving other areas free for exploration.

Visualization: Pheromone levels are displayed on a dedicated tilemap, with intensity normalized as:

$$I_{ij} = \min\left(1, \frac{\tau_{ij}}{\tau_{max}}\right) \tag{7}$$

Cells on the elitist path are shown at full intensity, making the optimal route easily visible. Updating in batches also helps reduce lag and keeps the simulation responsive.

3.6 Interactive Environment

The simulation environment is fully dynamic and designed to support real-time user interaction. Users can modify the environment by placing different types of objects through an integrated inventory system:

- **Walkable paths** (black), defining areas accessible to the ants.
- **Walls** (brown), which act as obstacles and dynamically alter the search space.
- **Food sources** (yellow), which serve as targets for the foraging ants.

To help users get started, I created four initial maps with varying layouts and difficulty levels. These pre-configured maps provide a good baseline for observing ant behavior, testing parameter settings, and experimenting with dynamic obstacles. Users can also create or modify maps on the fly by adding walls, paths, or food sources.
The interface allows multiple food sources to be added simultaneously, although increasing their number may impact performance due to the computational load. The environment can be navigated and zoomed using the **WASD keys**, enabling detailed observation of ant behavior. A counter displayed in the top-left corner provides real-time feedback on the total quantity of food successfully transported to the nest by the colony, allowing users to monitor emergent collective behaviors.

4 Final Notes

The simulation is designed for real-time visualization and interactive experimentation. Performance may degrade if a large number of ants or food items are present due to increased computational load. To mitigate this, the system maintains only the best-known path permanently in memory while using simple heuristics for path selection, allowing adaptive behavior in dynamic scenarios.
The project can be executed directly from the provided GitHub repository. Users can download the repository and extract the `.rar` archive, which contains the complete Unity project and the executable. The simulation can be launched by running the executable file (`ACO.exe`) located in the `Build/` folder. The environment includes a set of pre-configured maps, but users can also modify the environment dynamically by adding walls, walkable paths, or food sources through the interactive interface.

References

[1] Marco Dorigo. Optimization, learning and natural algorithms. *Ph.D. Thesis, Politecnico di Milano*, 1992.

[2] L. M. Gambardella, M. Dorigo, and F. Di Caro. Ant colonies for dynamic traveling salesman problems. In *Proceedings of the 1999 IEEE International Conference on Evolutionary Computation*, pages 81–86. IEEE, 1999.