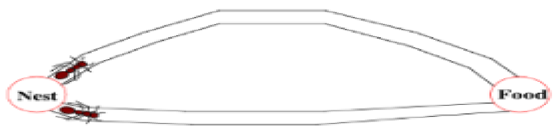
Ant Colony Optimization

Optimization problems are very important in the field of both scientific and industrial. Some real-life examples of these optimization problems are time table scheduling, nursing time distribution scheduling, train scheduling, capacity planning, traveling salesman problems, vehicle routing problems, Group-shop scheduling problem, portfolio optimization, etc. Many optimizations algorithms are developed for this reason. Ant colony optimization is one of them. Ant colony optimization is a probabilistic technique for finding optimal paths. In computer science and researches, the ant colony optimization algorithm is used for solving different computational problems.

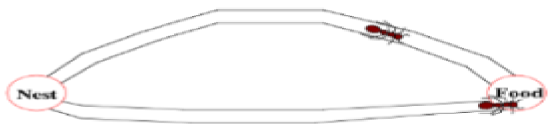
Ant colony optimization (ACO) was first introduced by Marco Dorigo in the 90. . This algorithm is introduced based on the foraging behavior of an ant for seeking a path between their colony and source food. Initially, it was used to solve the well-known traveling salesman problem. Later, it is used for solving different hard optimization problems.

Ants are social insects. They live in colonies. The behavior of the ants is controlled by the goal of searching for food. While searching, ants roaming around their colonies. An ant repeatedly hops from one place to another to find the food. While moving, it deposits an organic compound called pheromone on the ground. Ants communicate with each other via pheromone trails. When an ant finds some amount of food it carries as much as it can carry. When returning it deposits pheromone on the paths based on the quantity and quality of the food. Ant can smell pheromone. So, other ants can smell that and follow that path. The higher the pheromone level has a higher probability of choosing that path and the more ants follow the path, the amount of pheromone will also increase on that path.

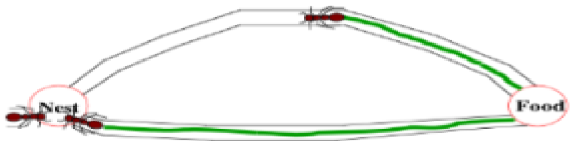
Let's see an example of this. Let consider there are two paths to reach the food from the colony. At first, there is no pheromone on the ground. So, the probability of choosing these two paths is equal that means 50%. Let consider two ants choose two different paths to reach the food as the probability of choosing these paths is fifty-fifty.



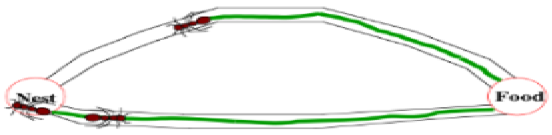
The distances of these two paths are different. Ant following the shorter path will reach the food earlier than the other.



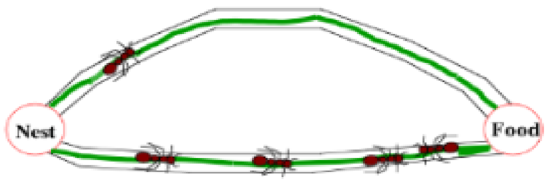
After finding food, it carries some food with itself and returns to the colony. When it tracking the returning path it deposits pheromone on the ground. The ant following the shorter path will reach the colony earlier.



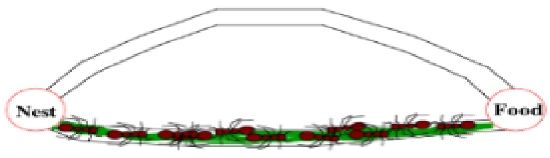
When the third ant wants to go out for searching food it will follow the path having shorter distance based on the pheromone level on the ground. As a shorter path has more pheromones than the longer, the third ant will follow the path having more pheromones.



By the time the ant following the longer path returned to the colony, more ants already have followed the path with more pheromones level. Then when another ant tries to reach the destination (food) from the colony it will find that each path has the same pheromone level. So, it randomly chooses one. Let consider it choose the above one



Repeating this process again and again, after some time, the shorter path has a more pheromone level than others and has a higher probability to follow the path, and all ants next time will follow the shorter path.



For solving different problems with ACO, there are three different proposed version of Ant-System:

Ant Density & **Ant Quantity:** Pheromone is updated in each movement of an ant from one location to another.

Ant Cycle: Pheromone is updated after all ants completed their tour.

$$\Delta\tau_{xy}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

Where L is the cost of an ant tour length and Q is a constant.

Given this, the functioning of an ACO algorithm can be summarized as follows. A set of computational concurrent and asynchronous agents (a colony of ants) moves through states of the problem corresponding to partial solutions of the problem to solve. They move by applying a stochastic local decision policy based on two parameters, called *trails* and *attractiveness*. By moving, each ant incrementally constructs a solution to the problem. When an ant completes a solution, or during the construction phase, the ant evaluates the solution and modifies the trail value on the components used in its solution. This pheromone information will direct the search of the future ants.

Furthermore, an ACO algorithm includes two more mechanisms: *trail evaporation* and, optionally, *daemon actions*. Trail evaporation decreases all trail values over time, in order to avoid unlimited accumulation of trails over some component. Daemon actions can be used to implement centralized actions which cannot be performed by single ants, such as the invocation of a local optimization procedure, or the update of global information to be used to decide whether to bias the search process from a non-local perspective.

More specifically, an *ant* is a simple computational agent, which iteratively constructs a solution for the instance to solve. Partial problem solutions are seen as *states*. At the core of the ACO algorithm lies a loop, where at each iteration, each ant *moves* (performs a *step*) from a state i to another one y, corresponding to a more complete partial solution. That is, at each step s, each ant *k* computes a set $\mathbf{A}_k^s$(i) of feasible expansions to its current state, and moves to one of these in probability. The probability distribution is specified as follows. For ant *k* , the probability p_{iy}^k of moving from state i to state y depends on the combination of two values:

- the *attractiveness* h_{iy} of the move, as computed by some heuristic indicating the *a priori* desirability of that move;
- the *trail level* t_{iy} of the move, indicating how proficient it has been in the past to make that particular move: it represents therefore an *a posteriori* indication of the desirability of that move.

Trails are *updated* usually when all ants have completed their solution, increasing or decreasing the level of trails corresponding to moves that were part of "good" or "bad" solutions, respectively.

Ant System

The importance of the original Ant System (AS) resides mainly in being the prototype of a number of ant algorithms which collectively implement the ACO paradigm. AS already follows the outline presented in the previous subsection, specifying its elements as follows.

The move probability distribution defines probabilities $p_i^k{}_\Psi$ to be equal to 0 for all moves which are infeasible (i.e., they are in the tabu list of ant *k* , that is a list containing all moves which are infeasible for ants *k* starting from state í), other-wise they are computed by means of formula (5 .1), where α and β are user-defined parameters ($(0 \leq \alpha, \beta \leq 1)$:

$$p^k_{\iota\psi} = \begin{cases} \dfrac{\tau^{\alpha}_{\iota\psi} + \eta^{\beta}_{\iota\psi}}{\sum\limits_{(\iota\zeta)\notin tabu_k} \left(\tau^{\alpha}_{\iota\zeta} + \eta^{\beta}_{\iota\zeta}\right)} & \text{if } (\iota\psi) \notin \text{tabu}_k \\ 0 & \text{otherwise} \end{cases}$$

tabu$_k$ is the tabu list of ant *k* , while parameters α and β specify the impact of trail and attractiveness, respectively.

After each iteration *t* of the algorithm, i.e., when all ants have completed a solution, trails are updated by equation:

$$\tau_{\iota\psi}(\tau) = \rho\ \tau_{\iota\psi}(\tau - 1) + \Delta\tau_{\iota\psi}$$

where $\Delta_{\iota\Psi}$ represents the sum of the contributions of all ants that used move (ι_Ψ) to construct their solution, ρ, $0 \leq \rho \leq 1$, is a user-defined parameter called *evaporation coefficient,* and $\Delta_{\iota\Psi}$ represents the sum of the contributions of all ants that used move (ιΨ) to construct their solution. The ants' contributions are proportional to the quality of the solutions achieved, i.e., the better solution is, and the higher will be the trail contributions added to the moves it used.

For example, in the case of the TSP, moves correspond to arcs of the graph, thus state ι could correspond to a path ending in node *i*, the state Ψ to the same path but with the arc (*ij*) added at the end and the move would be the traversal of arc (*ij*). The quality of the solution of ant *k* would be the length L of the tour *k* found by the ant and formula would become $\tau_{ij}(t)=\rho\ \tau_{ij}(t\text{-}1)+\Delta\tau_{ij}$, , with

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

where *m* is the number of ants and $\Delta\,\tau ij^{k}$ is the amount of trail laid on edge (*ij*) by ant *k* , which can be computed as

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} \text{ if ant } k \text{ uses arc (ij) in its tou} \\ \quad\quad 0 \quad \text{otherwise} \end{cases}$$

. Q being a constant parameter.

The ant system simply iterates a main loop where *m* ants construct in parallel their solutions, thereafter updating the trail levels. The performance of the algorithm depends on the correct tuning of several parameters, namely:α,β relative importance of trail and attractiveness, ρ, trail persistence, $\tau_{ij}(0)$, initial trail level, *m*, number of ants, and Q, used for defining to be of high quality solutions with low cost. The algorithm is the following.

```
{Initialization} •

Initialize $\tau_{\iota\psi}$ and $\eta_{\iota\psi}$, ∀(ιψ ).

{Construction} •

  For each ant k (currently in state ι) do
    repeat

         choose in probability the state to move into.
         append the chosen move to the k-th ant's set $tabu_k$

until ant k has completed its solution.

end for

{Trail update} •

For each ant move (ιψ ) do
compute $\Delta\tau_{\iota\psi}$

update the trail matrix.
end for

{Terminating condition} •

If not (end test) go to step 2
```