Ant Systems

Éric D. Taillard

Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA),
Corso Elvezia 36,
CH-6900 Lugano, Switzerland

Abstract

This article describes *Ant Systems*, a meta-heuristic based on an ant foraging metaphor. The presentation of Ant Systems has been somewhat generalized by adding a "Queen" process in charge of co-ordinating classical "Ant" processes, so that recent Ant Systems can be naturally included while remaining close to the metaphor. To illustrate how Ant Systems are practically implemented, a number of applications to the quadratic assignment problem are reviewed.

1 A model of real ants

The metaphor on which Ant Systems are based can be illustrated by observations of ants of the species *Linepithema humile* [2]. An ant colony nest is isolated, and a food source is provided which is accessible by a bridge composed of two branches of the same length. Although the ants are totally free to choose the left or the right branch of the bridge, it is rapidly observed that almost all ants use a given branch, even if there is no reason to prefer the left or the right one. This phenomenon is explained by the fact that ants deposit a chemical substance while traveling. They are able to detect this substance with their antennae. This substance carries informations and is called a *pheromone*.

A real ant is modeled as a probabilistic process: In the absence of pheromone, the ant explores the surrounding area in a totally random manner. If a pheromone trail is present, the ant follows the pheromone trail with a high probability. If two pheromone trails cross each other, the ant follows the trail with the larger amount of pheromone with a higher probability. Since the ant deposits additional pheromone when traveling, the foraging process evolves with positive feedback. Moreover, pheromones evaporate, meaning that a trail that is not used gradually disappears, amplifying the positive feedback effect.

The observation has been repeated with a bridge whose branches have different lengths [8]. Applying the same probabilistic behavior model, it can be deduced that the ants will rapidly choose the shorter branch and this is in fact observed. The phenomenon is explained as follows: at the beginning, when no pheromone is present, the ants choose the left or the right branch with equal probability. Since the ants which chose the shorter branch arrive earlier at the food (and come back to the nest earlier), the pheromone quantity on the shorter branch grows faster. So the ants are able to find the shortest path between nest and food, even if each ant has an extremely limited view of the surrounding area. Inspired by this natural optimization process, Colorni, Dorigo and Maniezzo [1] design a new meta-heuristic that can be applied to combinatorial optimization problems. The basic idea is to consider artificial ant processes that repeatedly build solutions to the optimization problem. Each process builds a solution with the help of a common memory that can be read and modified by any artificial ant. The common memory plays the role of trails in the real model.

However, when translated into a combinatorial optimization process, the simplified foraging process presented above is generally not able to create efficient heuristic methods and this (historical) model must be extended to cover recent and efficient applications inspired by ant colonies. Indeed, a simple positive feedback mechanism is difficult to tune correctly and often leads to degenerate situations: When the feedback is too strong, the only existing trail is that of the first ant which built a solution and when it is too weak, the trails are uniformly distributed in the whole search space. In the first situation, all ants use the same path and in the second, the the ants just perform a random walk.

The model is extended by considering an ant colony composed of differentiated ants such as workers, soldiers or sexual individuals. Each of these specialized individual is assigned a different task. The most important individual of the colony is the queen, which has the ability to create new ants and to choose the type of the created ants. Therefore, the queen co-ordinates the whole colony and has also a strategic role. Inclusion of an “intelligent” queen helps in the design of efficient algorithms, since search strategies can be included within the artificial ant system, while remaining close to the ant metaphor.

2 Artificial Ant Systems

Since the introduction of Ant Systems [1], the algorithms based on the ant metaphor have evolved. Instead of reviewing successive evolutions of these algorithms, a general Ant System model, developed from the real ant model, is presented. This general model is properly called the Ant System meta-heuristic, and is defined by a set of principles or a methodology that allows the development of heuristics for a wide range of combinatorial optimization problems.

Artificial ant systems are derived from the real ant model by making three analogies:

1. Real ants corresponds to processes in charge of building solutions to the combinatorial problem considered; these processes are often referred as *Artificial ants*, or simply *Ant* processes.

2. The pheromone trails corresponds to a common memory that is updated each time a new solution is built; more precisely, the memory records a value associated with each component of a solution.

3. The queen corresponds to a central process in charge of activating and co-ordinating artificial ants and of managing the common memory.

A meta-heuristic based on ant behavior can be described by a set of processes that collaborate through a common memory. The first set of processes, corresponding to the ants, built solutions in a probabilistic way, with probabilities depending on information stored in memory. All these artificial Ant processes are activated and co-ordinated by a Queen process that also manages the common memory. Very schematically, an ant system can be specified by two different processes:

- *Ant process*
  1. Receive problem data, memory state and, perhaps other parameters from the Queen process,
  2. With the help of the memory, build a new solution probabilistically,
  3. Send the new solution to Queen process.
- *Queen process*
  1. Initialize the memory,
  2. Repeat, (in parallel) until a stopping criterion is met:
     (a) Choose parameters for a new Ant process and activate the Ant process,
     (b) Receive a solution from an Ant process and update the memory,
  3. Return the best solution produced by the system.

This formulation of Ant Systems extends and generalizes previous formulations. Indeed, the very first systems did not mention a Queen process and the memory consisted of "trails", i.e. quantities associated with elements constituting a solution (for example, these elements could be edges for problems in which a path in a graph is searched). In the first Ant System applications, there was a fixed number of Ant processes that were activated simultaneously. These processes were executed in parallel and directly updated the trail values. Therefore, there was no explicit co-ordination of Ant processes, but rather an implicit one, realized through the trails. Unfortunately, such low-level interactions between simple artificial ants that work asynchronously and that do not know explicitly what the other ants are doing is not really efficient for designing practical applications. Recent developments of Ant Systems have first introduced a kind of synchronization by waiting for the completion of all Ant processes before updating the trails. Updates are then performed in a more intelligent way, following search strategies. For example, only an elite solution may be used to update the trails, thus implementing an intensification strategy (see [7]). Another example is to decrease the value of the trails corresponding to the elements chosen in the solution which has been built (negative feedback), thus implementing a diversification strategy. More recently, the concept of artificial trails has been interpreted as a memory [14] storing information about the use of particular components in the solutions previously built. Finally, in order to integrate a stronger, more intelligent form of co-operation between Ant processes, the Queen process has been introduced [13]. This process is also called ACO by other authors [4] (Ant Colony Optimization).

The art of designing a heuristic based on Ant Systems resides principally in the choice of the information contained in the memory (and their update) and in the way this information is used for constructing a new solution. In order to give a flavor of how a practical Ant Systems based algorithm may be designed, few implementations which discuss the design choices are briefly reviewed. These implementations illustrate very different ways of designing an Ant Systems based heuristic. The first implementation discussed is the "historical" one that first introduced the Ant System metaphor. Then we review implementations for the quadratic assignment problem for which Ant Systems have proven to be among the best heuristic methods for some problem instance classes.

2.1 Historical application to the TSP

The first Ant System application to a combinatorial optimization problem was programmed for the traveling salesman problem [1]. In this application, the memory is a vector $\mathbf{m}$ of real numbers (m_e) associated with each edge $e \in E$ of the graph $G(V, E)$ in which a shortest tour passing exactly once by each vertex of V is sought. The values contained in $\mathbf{m}$ are interpreted as the pheromone quantity left on a trail. Initially all the entries of $\mathbf{m}$ are set to the same, small value. The constructive procedure simulates the behavior of an ant that, starting from a vertex, can only move to a vertex it has not already visited (unless all vertices have been visited; in this case the ant completes its tour by returning to the first vertex). At each step, the edge e that is used to move to another vertex is randomly chosen with a probability proportional to $(m_e)^\alpha (d_e)^\beta$, where e is an edge with one end being the vertex on which the ant stands and the other being a vertex not yet visited, $0 \leq \alpha, 0 \geq \beta$ are two parameters and d_e is the length of edge e. This probabilistic rule favors the choice of short edges that frequently appear in the solution generated by the system. The very first implementation of the algorithm had no Ant process co-ordination: A fixed number k of totally asynchronous Ant processes repeatedly constructed new solutions. The memory updates were also performed asynchronously by the Ant processes during the solution construction. Unfortunately, this system was found to be less efficient than another implementation with synchronized Ant processes that can be described as follows:

In the best implementation of the algorithm [1], the Queen process activates k parallel Ant processes at each loop. Each Ant process builds a solution with the probabilistic rule described above, send the solution to the Queen process and dies. The Queen therefore receives k solutions $s_1, \ldots, s_k$ of length $L_1, \ldots, L_k$ at each loop. The memory is updated as follows: First $\mathbf{m} \leftarrow \rho\mathbf{m}$ where $0 \leq \rho \leq 1$ is a parameter that simulates the evaporation of the trails. Then, $m_e \leftarrow m_e + Q/L_i$, $\forall s_i$ and $\forall e \in s_i$, where $0 \leq Q$ is another parameter controlling the amount of positive feedback. The value m_e associated with each edge e gives therefore a stronger weight to edges which appear in good solutions and to edges that are often chosen.

This approach was able to produce, for small problem instances, solutions of slightly better quality than Lin and Kernighan's [9] heuristic. This approach gives a very limited role to the Queen process, which is why it was not initially integrated into Ant Systems explicitly. Recently, the system has been greatly improved by:

1. The use of a more elaborated construction procedure, including candidate lists and additional parameters.
2. The improvement of each constructed solution with a local search.
3. The addition of intensification and diversification strategies in memory management, i.e. the introduction of a stronger co-ordination between the Ant processes.

The new method has been named ACS (Ant Colony System) [3].

2.2 Applications to the quadratic assignment problem

It is possible to design completely different Ant System-based heuristics by:

1. Modifying the type of information memorized,
2. Changing memory update rules (or learning mechanism),
3. Modifying the constructive procedure,
4. Including alternative search strategies.

The quadratic assignment problem (QAP) is perhaps the problem for which the largest range of Ant System-based heuristics have now been designed. This section reviews few of these methods, focusing on various choices made by their designers. In essence, the QAP can be described by the search of a permutation $\pi \in \Pi$ minimizing $\sum_{i=1}^{n} a_{ij} b_{\pi_i \pi_j}$, where Π is the set of all the permutations of n elements and $\mathbf{A} = (a_{ij})$ and $\mathbf{B} = (b_{ij})$ are two matrices of size $n \times n$. A typical QAP application is the placement of interacting facilities in spatial locations. The interaction (or flow) between facilities i and j is given by a_{ij} and the distance between location r and s is given by b_{rs}. The aim is to find locations for the facilities such that the sum of the *distance* $\times$ *flow* products is minimized.

For all applications reviewed in this section, the memory is implemented by a matrix $\mathbf{M}$ of size $n \times n$. Matrix entry m_{ir} records that facility i was placed on location r in solutions previously generated by the algorithm. The higher the quality and the larger the number of solutions generated with facility i placed on location r, the higher the m_{ir} value is.

- *Max-Min Ant System (MMAS)*

  The main idea of the Max-Min Ant System [11] is to bound the values contained in the memory in an interval: $m_{min} \leq m_{ir} \leq m_{max}$, $\forall m_{ir}$, where m_{min} and m_{max} are two parameters. By choosing appropriate m_{min} and m_{max} values, the search is prevented from stagnating, that is repeatedly producing the same solution.

  Ant process

  The constructive procedure works as follows: at each step, a facility i which has not yet been assigned is randomly chosen. This facility is assigned to an unoccupied location r using a "pseudo-random-proportional" rule: Either location r which maximizes m_{ir} is chosen or the location is chosen from several alternatives, which associated probabilities are proportional to m_{ir}. The former choice has a probability q, the latter $(1-q)$. This step is repeated until all facilities are allocated. Then a local search is applied to improve the solution that is finally sent to the Queen process. There are two version of MMAS: the first one uses a descent procedure stopping at the first local optimum as local search and the second one uses the taboo search of Taillard [12].

  Queen process

  Initially, the Queen process sets all the m_{ir} values to m_{max}. Then, the process enters the main iteration that consists of activating k Ant processes, where k is a parameter, and of waiting for k solutions $\pi^1, \ldots, \pi^k$ of cost $c^1, \ldots, c^k$ before updating the memory. The memory is updated as follows: First, all the entries of $\mathbf{M}$ are decreased by setting $\mathbf{M} \leftarrow \rho\mathbf{M}$, where $0 \leq \rho \leq 1$ is a parameter. Let π^b be the best solution among $\pi^1, \ldots, \pi^k$ and let π^* be the best solution produced by the search so far (c^b and c^* are their corresponding costs). Every odd iteration, the Queen process update the memory by setting $m_{i\pi_i^b} \leftarrow m_{i\pi_i^b} + 1/c^b$, $\forall i$ while every even iteration, the memory is updated with π^* and c^* instead of π^b and c^b. Finally, the entries of $\mathbf{M}$ that are lower than m_{min} are set to m_{min} and those larger than m_{max} are set to m_{max}.

The Max-Min Ant System has also been applied to the traveling salesman problem.

- *Approximate Nondeterministic Tree Search (ANTS)*

  The main innovation of the Approximate Nondeterministic Tree Search [10] lies in the constructive procedure: At each step, the procedure randomly places a facility on a location with a probability depending on the memory, as usual for Ant Systems, but also depending on the computation of a lower bound, similar to what is done in implicit enumeration methods, such as branch and bound.

  Ant process

  First, the Ant process receives, from the Queen process, an order in which the locations must be filled by the facilities. At each step, the constructive procedure assigns a facility to the next empty location r as follows: first a lower bound LB_{ir} of the cost of assigning facility i to location r is computed, taking into account the assignments already done in previous steps. Then, the facility to assign is chosen with a probability proportional to $\alpha m_{ir} + (1-\alpha)LB_{ir}$, where $0 \leq \alpha \leq 1$ is a parameter. Once all facilities are allocated, the solution is improved with a local search that stops at the first local optimum.

  Queen process

  First, the Queen process computes a lower bound LB on the solution cost by solving a linear relaxation of the problem. The dual variables obtained during the computation are used on the one hand for initializing the m_{ir} values of the memory (noted m_{ir}^0) and, on the other hand, these variables are used to compute the order in which the Ant process constructive procedure must fill the location. The order depends only on the problem data and is computed only once. The rationale behind this ordering is that the higher the value of a dual variable associated with a location is, the higher the impact of the assignment of a facility to that location should be on the solution cost, and therefore the earlier the location must be filled.

  The Queen process then activates k Ant processes and receives k solutions $\pi^1, \ldots, \pi^k$of cost $c^1, \ldots, c^k$. A moving average $\bar{c}$ of the values of the last m solutions received is computed and the memory entries are updated as follows:

$$m_{i\pi_i^b} \leftarrow m_{i\pi_i^b} + m_{i\pi_i^b}^0 \frac{\bar{c} - c^b}{\bar{c} - LB}, \ \forall\, 1 \leq i \leq n, \ \forall\, 1 \leq b \leq k$$

- *Hybrid Ant System (HAS)*

  A first principle of the Hybrid Ant System [6] is to replace the Ant process constructive procedure by a probabilistic local search: Instead of building a brand new solution each time an Ant process is activated, the Ant process receives a starting solution and applies a given number of local search moves to that solution. The moves are randomly chosen with a probability which depends on the values in memory. The second basic idea of the Hybrid Ant System is to introduce search strategies in the Queen process.

Ant process

First, the Ant process receives a solution π from the Queen, in addition to the problem data. The process then repeats m local search steps as follows: A facility i is randomly chosen in $1, \ldots, n$. A second facility j is chosen with the "pseudo-random-proportional" rule and facilities i and j exchange their location. The pseudo-random-proportional rule chooses either the facility j maximizing $m_{i\pi_j} + m_{j\pi_i}$ (with probability q) or chooses j randomly, with a probability proportional to $m_{i\pi_j} + m_{j\pi_i}$. After m probabilistic local search steps, the solution is improved with a local search (driven by the objective function) that ends at the first local optimum before being sent to the Queen process.

Queen process

In addition to the memory matrix $\mathbf{M}$, the Queen process manages a pool of k solutions $\pi^1, \ldots, \pi^k$. Initially, the first k solutions are randomly chosen and improved with a local search. In addition to the pool of solutions, the Queen manages π^*, the best solution found by the system (c^* denotes its value). The memory matrix entries are initially set to the same small value. The Queen repeats the following cycle:

First, k Ant processes are activated, each of them receiving a different solution from the pool. Then, the Queen process waits for k solutions $\mu^1, \ldots, \mu^k$ from the Ant processes and updates π^*. Then, the memory is updated as follows: First, all entries are weakened by setting $\mathbf{M} \leftarrow \rho\mathbf{M}$, where $0 \leq \rho \leq 1$ is a parameter. Then, the entries corresponding to the best solution are reinforced by setting $m_{i\pi_i^*} \leftarrow m_{i\pi_i^*} + (1-\rho)/c^*$, $\forall i$. Finally, the pool of solutions is updated using diversification and intensification strategies: Normally, the Queen process is in intensification phase and replaces π^b by the best solution among π^b and μ^b, $\forall\ 1 \leq b \leq k$. The Queen process remains in intensification phase while at least one μ^b is better than π^b. But if no μ^b is better than π^b, the Queen process performs a diversification steps that consists of replacing $k-1$ solutions in the pool by random ones and by replacing the last one with π^*.

- *Fast Ant System (FANT)*

  The basic idea of the Fast Ant System [13] is to design a method which is as simple as possible while incorporating diversification and intensification strategies. This is realized on the one hand by systematically reinforcing the attractiveness of the m_{ir} values corresponding to the best solution found so far by the search and on the other hand by clearing the memory while giving less weight to the best solution if the process appears to be stagnating.

Ant process

The Ant process constructs a new solution by randomly choosing the location r of facility i with a probability proportional to m_{ir}. Then the solution is improved with a local search and sent to the Queen process.

Queen process

While implementing an automatic intensification and diversification, the Queen process requires only one parameter R and manages, in addition to the memory matrix $\mathbf{M}$, a variable v and the best solution π^* found by the system so far. Initially $v = 1$ and $m_{ir} = v, \ \forall i, r$. The Queen then repeats the following cycle:

1. Activate an Ant process,
2. Wait for a solution π from an Ant process,
3. If $\pi = \pi^*$ then set $v \leftarrow v + 1$ and $m_{ir} \leftarrow v, \ \forall i, r$,
4. If π is better than π^* then set $\pi^* \leftarrow \pi, v \leftarrow 1$ and $m_{ir} \leftarrow v, \ \forall i, r$,
5. Set $m_{i\pi_i} \leftarrow m_{i\pi_i} + v, \ \forall i$,
6. Set $m_{i\pi_i^*} \leftarrow m_{i\pi_i^*} + R, \ \forall i$.

To intensify the search in the neighborhood of π^*, the memory entries corresponding to π^* are reinforced at each iteration by R (Step 6). In order to "memorize" the solution π generated by the Ant process, the memory entries corresponding to π are reinforced by v (Step 5). When π^* is improved, variable v is set to 1 (to give a relatively higher weight to R) and the memory is re-initialized (Step 4). When the solution constructed by the Ant process is equal to π^*, meaning that the memory entries that correspond to π^* have been reinforced too much, variable v is incremented (to give a relatively lower weight to R in subsequent Queen iterations) and the memory is re-initialized (Step 3). This simple Ant System has been applied to other combinatorial optimization problems with success, such as the p-median, the balancing of turbine runners and the bi-quadratic assignment problems. It has been implemented on a sequential computer and the activation of only one Ant process at a time allows relatively good solutions to be obtained with a very low computational effort. However, the method can be easily extended to the case of an arbitrary number of Ant processes.

3 General considerations

Ant Systems belongs to the wide class of meta-heuristics which works with a memory or *Adaptive Memory Programming* [14]. Ant Systems share common features with Genetic Algorithms, Greedy Randomized Adaptive Search Procedures [5], Tabu Search and Scatter Search. Indeed, Ant Systems are based on a population of artificial agents, the Ant processes, but the interaction between individuals, through a common memory, is quite unlike classical genetic operators. The constructive procedure embedded in the Ant processes looks a bit like those of a greedy randomized adaptive search procedure and can also be compared to a probabilistic Scatter Search, since the solutions generated are a kind of combination of solutions previously built. Finally, the Ant processes are coordinated by a process that can incorporate various search strategies proposed in Tabu Search.

Ant Systems have now matured and successful applications are more and more numerous. Efficient Ant Systems very often embed elements that are unrelated to the metaphor which inspired the first Ant Systems. Typical foreign elements are the use of a local search or the memory central management. This is perhaps a natural development for meta-heuristics: it has been noted [14] that the best heuristic methods are often inspired by many basic meta-heuristics. For example, most efficient genetic algorithms include a local search. Therefore, it is not surprising that Ant Systems are evolving in a similar way and incorporate components of other meta-heuristics; conversely, components of Ant Systems will certainly be combined with methods based on other meta-heuristics. Future developments of Ant Systems will include the treatment of dynamic and stochastic problems and the design of new distributed systems.

References

[1] Colorni, A., M. Dorigo and V. Maniezzo, “Distributed optimization by ant colonies”, in *Toward a practice of autonomous systems: proceedings of the First European Conference on Artificial Life (ECAL 91)*, F.J. Varela and P. Bourgine (eds.), MIT Press, Cambridge, 1992, 134–142.

[2] Deneubourg, J.-L., S. Aron, S. Goss and J.M. Pasteels, “The self-organizing exploratory pattern of the Argentine ant”, *Journal of Insect Behavior* 3, 1990, 159–168.

[3] Dorigo, M. and L.-M. Gambardella, “Ant colony system: A cooperative learning approach to the traveling salesman problem”, *IEEE Transactions on Evolutionary Computation* 1, 1997, 53–66.

[4] Dorigo, M. and G. Di Caro, “The Ant Colony Optimization Meta-Heuristic”, in *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover (eds.), McGraw-Hill, 1999.

[5] Feo, T. and M. Resende, “Greedy randomized adaptive search procedures”, *Journal of Global Optimization 16*, 1995, 109–133.

[6] Gambardella, L. M., É. D. Taillard and M. Dorigo, “Ant colonies for the quadratic assignment problem”, *Journal of the Operational Research Society* 50, 1999, 167–176.

[7] Glover, F. and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston/Dordrecht/London, 1997.

[8] Goss, S., S. Aron, J.-L. Deneubourg and J.M. Pasteels, "Self-organized shortcuts in the Argentine ant", *Naturwissenschaften* 76, 1989, 579–581.

[9] Lin, S. and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem", *Operations Research* 21, 1973, 498–516.

[10] Maniezzo, V., "Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem", Technical Report *CSR* 98-1, C. L. in Scienze dell'Informazione, Università di Bologna, Sede di Cesena, Italy, 1998. To appear in *INFORMS Journal on Computing*.

[11] Stützle, T. and H. H. Hoos, "The Max-Min Ant System and Local Search for Combinatorial Optimization Problems", in *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. H. Osman and C. Roucairol (eds.), Kluwer Academic Publishers, Boston/Dordrecht/London, 1999, 313–329.

[12] Taillard É. D., "Robust taboo search for the quadratic assignment problem", *Parallel computing* 17, 1991, 443–455.

[13] Taillard, É. D., "FANT: Fast ant system", Technical report *IDSIA-46-98*, IDSIA, Lugano, 1998.

[14] Taillard, É. D., L.-M. Gambardella, M. Gendreau, and J.Y. Potvin, "Adaptive Memory Programming: A Unified View of Meta-Heuristics", EURO XVI Conference Tutorial and Research Reviews booklet (semi-plenary sessions), Brussels, July 1998.