The Ant Colony Optimization Meta-Heuristic[1]

Marco Dorigo and Gianni Di Caro
IRIDIA
Université Libre de Bruxelles
{mdorigo,gdicaro}@ulb.ac.be

[1]To appear in D. Corne, M. Dorigo and F. Glover, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.

Chapter 2

The Ant Colony Optimization Meta-Heuristic

2.1 Introduction

Ant algorithms are multi-agent systems in which the behavior of each single agent, called *artificial ant* or *ant* for short in the following, is inspired by the behavior of real ants. Ant algorithms are one of the most successful examples of swarm intelligent systems [3], and have been applied to many types of problems, ranging from the classical traveling salesman problem, to routing in telecommunications networks. In this section we will focus on the *ant colony optimization (ACO) meta-heuristic* [18], which defines a particular class of ant algorithms, called in the following *ACO algorithms*.

ACO algorithms have been inspired by the following experience run by Goss et al. [31] using a colony of real ants. A laboratory colony of Argentine ants (*Iridomyrmex humilis*) is given access to a food source in an arena linked to the colony's nest by a bridge with two branches of different length (see Figure 2.1). Branches are arranged in such a way that ants going in either direction (from the nest to the food source or vice versa) must choose between one branch or the other. The experimental observation is that, after a transitory phase which can last a few minutes, most of the ants use the shortest branch. It is also observed that the colony's probability of selecting the shortest branch increases with the difference in length between the two branches. The emergence of this shortest path selection behavior can be explained in terms of *autocatalysis* (*positive feedback*) and *differential path length*, and it is made possible by an indirect form of communication, known as stigmergy [32], mediated by local modifications of the environment.

In fact, Argentine ants, while going from the nest to the food source and vice versa, deposit a chemical substance, called *pheromone*, on the ground. When they arrive at a decision point, like the intersection between the shorter and the longer branch, they make a probabilistic choice biased by the amount of pheromone they smell on the two branches. This behavior has an autocatalytic effect because the very fact of choosing a path will increase the probability that it will be chosen again by future ants. At the beginning of the experiment there is no pheromone on the two branches and therefore ants going from the nest to the food source will choose any of the two branches with equal probability. Due to differential branch length, the ants choosing the shortest branch will be the first to reach the food source. When they, in their path back to the nest, reach the decision point, they will see some pheromone trail on the shorter path, the trail they released during the forward travel, and will choose it with higher probability than the longer one. New

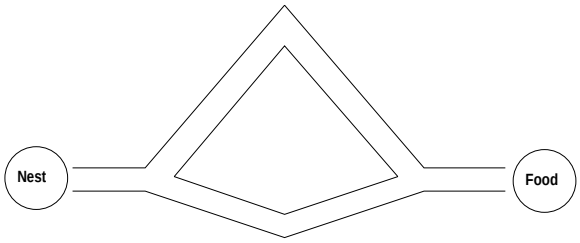

Figure 2.1: Experimental apparatus for the bridge experiment. Branches have different length. Ants move from the nest to the food source and back.

pheromone will be released on the chosen path, making it even more attractive for the subsequent ants. While the process iterates, pheromone on the shorter path is deposited at a higher rate than on the longer one, making the shorter path more and more selected until all ants end up using it.

In the next sections we show how these simple ideas can be engineered and put to work so that a colony of artificial ants can find good solutions to difficult optimization problems.

2.2 The simple ant colony optimization algorithm

In this section a very simple ant-based algorithm is presented to illustrate the basic behavior of the ACO meta-heuristic and to put in evidence its basic components.

The main task of each artificial ant, similarly to their natural counterparts, is to find a shortest path between a pair of nodes on a graph on which the problem representation is suitably mapped.

Let $G = (N, A)$ be a connected graph with $n = |N|$ nodes. The simple ant colony optimization (S-ACO) algorithm can be used to find a solution to the shortest path problem defined on the graph G, where a solution is a path on the graph connecting a source node s to a destination node d, and the path length is given by the number of hops in the path (see Figure 2.2).

To each arc (i, j) of the graph is associated a variable τ_{ij} called *artificial pheromone trail*, pheromone trail for short in the following. Pheromone trails are read and written by ants. The amount (intensity) of pheromone trail is proportional to the utility, as estimated by the ants, of using that arc to build good solutions.

Each ant applies a step-by-step constructive decision policy to build problem's solutions. At each node local information, maintained on the node itself and/or on its outgoing arcs, is used in a stochastic way to decide the next node to move to.

The decision rule of an ant k located in node i uses the pheromone trails τ_{ij} to compute the probability with which it should choose node $j \in \mathcal{N}_i$ as the next node to move to,[1] where $\mathcal{N}_i$ is the set of one-step neighbors of node i:

$$p_{ij}^k = \begin{cases} \dfrac{\tau_{ij}}{\sum_{j \in \mathcal{N}_i} \tau_{ij}} & \text{if } j \in \mathcal{N}_i \\ 0 & \text{if } j \notin \mathcal{N}_i \end{cases} \tag{2.1}$$

[1] At the beginning of the search process, a small amount of pheromone τ_0 is assigned to all the arcs.

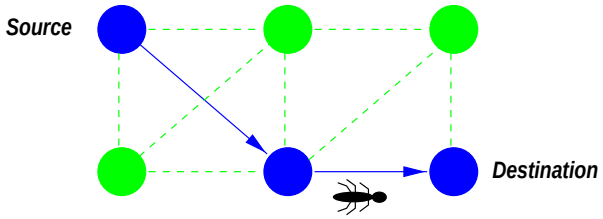

Figure 2.2: Ants build solutions, that is paths, from a source to a destination node. The ants choosing the solid line path will arrive sooner to the destination and will therefore be the first to bias search of ants moving back to the source node.

While building a solution ants deposit pheromone information on the arcs they use. In S-ACO ants deposit a constant amount $\Delta\tau$ of pheromone. Consider an ant that at time t moves from node i to node j. It will change the pheromone value τ_{ij} as follows:

$$\tau_{ij}(t) \leftarrow \tau_{ij}(t) + \Delta\tau \tag{2.2}$$

By this rule, which simulates real ants' pheromone depositing on arc (i, j), an ant using the arc connecting node i to node j increases the probability that ants will use the same arc in the future. As in the case of real ants, autocatalysis and differential path length are at work to favor the emergence of short paths.

To avoid a quick convergence of all the ants towards a sub-optimal path, an exploration mechanism is added: similarly to real pheromone trails, artificial pheromone trails "evaporate". In this way pheromone intensity decreases automatically, favoring the exploration of different arcs during the whole search process. The evaporation is carried out in a simple way, decreasing pheromone trails in an exponential way, $\tau \leftarrow (1 - \rho)\tau, \rho \in (0, 1]$ at each iteration of the algorithm. Preliminary experiments run with S-ACO using a simple graph modeling the experimental apparatus of Figure 2.1 have shown that the algorithm effectively finds the shortest path between the simulated nest and food sources. Experiments have also shown that if we increase the complexity of the searched graph, for example by connecting the nest to the food source by means of more than two possible paths, the behavior of the algorithm tends to become less stable and the value given to parameters becomes critical.

S-ACO must therefore be taken for what it is: a didactic example that, because of its simplicity, has a number of limitations. The algorithms defined in the following of this and of the following chapters share the basic properties of S-ACO, but are enriched with extra capabilities which help to overcome S-ACO limitations. For example, we can make the amount of pheromone deposited by ants proportional to the quality of the solution built or being generated by the ant so that pheromone information becomes more useful in directing ants search. Also, because in many problems some form of heuristic information is available at the nodes, it would be desirable to have ants able to use it.

Another important point is that it would be desirable to enlarge the class of problems that can be attacked by ACO algorithms. S-ACO can be applied only to shortest path problems without additional constraints: If we want to use it to find a shortest Hamiltonian path on a graph, that is, a path which visits all the nodes once and only once, we need to give our ants at least some limited form of memory.

In the next sections we will introduce the ACO meta-heuristic, which builds on the S-ACO model enriching artificial ants with a number of capacities that do not find their

counterpart in real ants, but that allow to overcome the above-listed limitations of the simple model.

2.3 The ACO meta-heuristic

ACO algorithms, that is, instances of the ACO meta-heuristic introduced in the following of this section, can be applied to discrete optimization problems that can be characterized as follows:

- A finite set of *components* $C = \{c_1, c_2, \ldots, c_{N_C}\}$ is given.
- A finite set L of possible *connections/transitions* among the elements of C is defined over a subset $\tilde{C}$ of the Cartesian product $C \times C$, $L = \{l_{c_i c_j} \mid (c_i, c_j) \in \tilde{C}\}$, $|L| \leq N_C^2$.
- For each $l_{c_i c_j} \in L$ a *connection cost* function $J_{c_i c_j} \equiv J(l_{c_i c_j}, t)$, possibly parametrized by some time measure t, can be defined.
- A finite set of *constraints* $\Omega \equiv \Omega(C, L, t)$ is assigned over the elements of C and L.
- The *states* of the problem are defined in terms of sequences $s = \langle c_i, c_j, \ldots, c_k, \ldots \rangle$ over the elements of C (or, equivalently, of L). If S is the set of all possible sequences, the set $\tilde{S}$ of all the (sub)sequences that are feasible with respect to the constraints $\Omega(C, L, t)$, is a subset of S. The elements in $\tilde{S}$ define the problem's *feasible states*. The length of a sequence s, that is, the number of components in the sequence, is expressed by $|s|$.
- A *neighborhood structure* is defined as follows: the state s_2 is said to be a neighbor of s_1 if (i) both s_1 and s_2 are in S, (ii) the state s_2 can be reached from s_1 in one logical step, that is, if c_1 is the last component in the sequence determining the state s_1, it must exists $c_2 \in C$ such that $l_{c_1 c_2} \in L$ and $s_2 \equiv \langle s_1, c_2 \rangle$.
- A *solution* ψ is an element of $\tilde{S}$ satisfying all the problem's requirements. A solution is said *multi-dimensional* if it is defined in terms of multiple distinct sequences over the elements of C.
- A *cost* $J_\psi(L, t)$ is associated to each solution ψ. $J_\psi(L, t)$ is a function of all the costs $J_{c_i c_j}$ of all the connections belonging to the solution.

Consider the graph $G = (C, L)$ associated to a given discrete optimization problem instance as above defined. The solutions to the optimization problem can be expressed in terms of feasible paths on the graph G. ACO algorithms can be used to find minimum cost paths (sequences) feasible with respect to the constraints Ω.[2]

In ACO algorithms a population (colony) of agents (or ants) collectively solve the optimization problem under consideration by using the above graph representation. Information collected by the ants during the search process is encoded in *pheromone trails* τ_{ij} associated to connection l_{ij}.[3] Pheromone trails encode a long-term memory about the whole ant search process. Depending on the problem representation chosen, pheromone

[2]For example, in the traveling salesman problem defined in Section 2.4.1, C is the set of cities, L is the set of arcs connecting cities, and a solution ψ is an Hamiltonian circuit.

[3]Here and in the following, we simplify notation by setting $l_{c_i c_j} = l_{ij}$.

trails can be associated to all problem's arcs, or only to some of them. Arcs can also have an associated *heuristic value* η_{ij} representing a priori information about the problem instance definition or run-time information provided by a source different from the ants.

The ants' colony presents the following general characteristics:

- Although each ant is complex enough to find a (probably poor) solution to the problem under consideration, good quality solutions can only emerge as the result of the collective interaction among the ants.
- Each ant makes use only of private information and of information local to the node[4] it is visiting.
- Ants communicate with other ants only in an indirect way, mediated by the information they read/write in the variables storing pheromone trail values.
- Ants are not adaptive themselves. On the contrary, they adaptively modify the way the problem is represented and perceived by other ants.

Ants of the colony have the following properties:

- An ant searches for minimum cost feasible solutions $\hat{J}_\psi = \min_\psi J_\psi(L, t)$.
- An ant k has a memory $\mathcal{M}^k$ that it can use to store information on the path it followed so far. Memory can be used (i) to build feasible solutions, (ii) to evaluate the solution found, and (iii) to retrace the path backward.
- An ant k in state $s_r = \langle s_{r-1}, i \rangle$ can move to any node j in its *feasible neighborhood* $\mathcal{N}_i^k$, defined as $\mathcal{N}_i^k = \{j \mid (j \in \mathcal{N}_i) \wedge (\langle s_r, j \rangle \in \tilde{S})\}$.
- An ant k can be assigned a *start state* s_s^k and one or more *termination conditions* e^k.[5]
- Ants start from the start state and move to feasible neighbor states, building the solution in an incremental way. The construction procedure stops when for at least one ant k at least one of the termination conditions e^k is satisfied.
- An ant k located on node i can move to a node j chosen in $\mathcal{N}_i^k$. The move is selected applying a probabilistic decision rule.
- The ants' probabilistic decision rule is a function of (i) the values stored in a node local data structure $\mathcal{A}_i = [a_{ij}]$ called *ant-routing table*, obtained by a functional composition of node locally available pheromone trails and heuristic values, (ii) the ant's private memory storing its past history, and (iii) the problem constraints.
- When moving from node i to neighbor node j the ant can update the pheromone trail τ_{ij} on the arc (i, j). This is called *online step-by-step pheromone update*.

[4] In the following, the terms node and component, as well as arc and connection/transition, will be used interchangeably.

[5] Usually, the start state is expressed as a unit length sequence, that is, a single component.

- Once built a solution, the ant can retrace the same path backward and update the pheromone trails on the traversed arcs. This is called *online delayed pheromone update*.

- Once it has built a solution, and, if the case, after it has retraced the path back to the source node, the ant dies, freeing all the allocated resources.

Informally, the behavior of ants in an ACO algorithm can be summarized as follows. A colony of ants concurrently and asynchronously move through adjacent states of the problem by moving through neighbor nodes of G, as shown in the S-ACO algorithm. They move by applying a stochastic local decision policy that makes use of the information contained in the node-local ant-routing tables. By moving, ants incrementally build solutions to the optimization problem. Once an ant has built a solution, or while the solution is being built, the ant evaluates the (partial) solution and deposits information about its goodness on the pheromone trails of the connections it used. This pheromone information will direct the search of the future ants.

Besides ants' activity, an ACO algorithm include two more procedures: *pheromone trail evaporation* and *daemon actions*.[6] Pheromone evaporation is the process by means of which the pheromone trail intensity on the connections automatically decreases over time. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm towards a sub-optimal region. It implements a useful form of *forgetting*, favoring the exploration of new areas of the search space. Daemon actions can be used to implement centralized actions which cannot be performed by single ants. Examples are the activation of a local optimization procedure, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective. As a practical example, the daemon can observe the path found by each ant in the colony and choose to deposit extra pheromone on the arcs used by the ant that made the shortest path. Pheromone updates performed by the daemon are called *offline pheromone updates*.

In Figure 2.3 the ACO meta-heuristic behavior is described in pseudo-code. The main procedure of the ACO meta-heuristic manages, via the `schedule_activities` construct, the scheduling of the three above discussed components of an ACO algorithm: (i) ants' generation and activity, (ii) pheromone evaporation, and (iii) daemon actions. It is important to note that the `schedule_activities` construct does not specify how these three activities are scheduled and synchronized and, in particular, whether they should be executed in a completely parallel and independent way, or if some kind of synchronization among them is necessary. This leaves the designer the freedom to specify the way these three procedures should interact.

Although ACO algorithms are suitable to find minimum cost (shortest) paths on a graph in general, it is important to note that they are an interesting approach only for those shortest path problems to which more classical algorithms like dynamic programming or label correcting methods [1] cannot be efficiently applied. This is the case, for example, for the following types of shortest path problems:

- NP-hard problems, for which the dimension of the full state-space graph is exponential in the dimension of the problem representation. In this case, ants make use of the much smaller graph G, built from the problem's components, and use their

[6] The daemon actions component is optional.

```
1  procedure ACO_meta-heuristic()
2    while (termination_criterion_not_satisfied)
3      schedule_activities
4        ants_generation_and_activity();
5        pheromone_evaporation();
6        daemon_actions();   {optional}
7      end schedule_activities
8    end while
9  end procedure
```

```
1  procedure ants_generation_and_activity()
2    while (available_resources)
3      schedule_the_creation_of_a_new_ant();
4      new_active_ant();
5    end while
6  end procedure
```

```
1  procedure new_active_ant() {ant lifecycle}
2    initialize_ant();
3    M = update_ant_memory();
4    while (current_state ≠ target_state)
5      A = read_local_ant-routing_table();
6      P = compute_transition_probabilities(A, M, Ω);
7      next_state = apply_ant_decision_policy(P, Ω);
8      move_to_next_state(next_state);
       if (online_step-by-step_pheromone_update)
9        deposit_pheromone_on_the_visited_arc();
10       update_ant-routing_table();
       end if
11     M = update_internal_state();
12   end while
     if (online_delayed_pheromone_update)
13     foreach visited_arc ∈ ψ  do
14       deposit_pheromone_on_the_visited_arc();
15       update_ant-routing_table();
16     end foreach
     end if
17   die();
18 end procedure
```

Figure 2.3: The ACO meta-heuristic in pseudo-code. Comments are enclosed in braces. The procedure `daemon_actions()` at line 6 of the `ACO_meta_heuristic()` procedure is optional and refers to centralized actions executed by a daemon possessing global knowledge. In the `new_active_ant()` procedure, the `target_state` (line 4) refers to a complete solution built by the ant, while the step-by-step and delayed pheromone updating procedures at lines 9-10 and 14-15 are often mutually exclusive. When both of them are absent the pheromone is deposited by the daemon.

memory to generate feasible solutions which in most ACO implementations are then taken to a local optimum by a problem specific local optimizer.

- Those shortest path problems in which the properties of the problem's graph representation change over time concurrently with the optimization process, that has to adapt to the problem's dynamics. In this case, the problem's graph can even be physically available (like in networks problems), but its properties, like the value of connection costs $J_{c_i c_j}(t)$, can change over time. In this case we conjecture that the use of ACO algorithms becomes more and more appropriate as the variation rate of costs $J_{c_i c_j}(t)$ increases and/or the knowledge about the variation process diminishes.

- Those problems in which the computational architecture is spatially distributed, as in the case of parallel and/or network processing. Here ACO algorithms, due to their intrinsically distributed and multi-agent nature that well matches these types of architectures, can be very effective.

In the following section we will consider the application of the ACO-meta-heuristic to two paradigmatic problems belonging to the above defined classes of problems: the traveling salesman problem (TSP) and the adaptive routing in communications networks. TSP is the prototypical representative of NP-hard combinatorial optimization problems [28] where the problem instance is statically assigned and the information is globally available. On the contrary, in the problem of adaptive routing in communications networks an exogenous process (the incoming data traffic) makes the problem instance change over the time, and temporal constraints impose to solve the problem in a distributed way.

Chapters 3, 4, and 5 of this book illustrate further applications of the ACO meta-heuristic to NP-hard combinatorial optimization problems, while further and more detailed examples of applications to adaptive routing can be found in [13, 15, 45, 53].

2.4 ACO for the traveling salesman problem

2.4.1 The traveling salesman problem

The traveling salesman problem plays an important role in ant colony optimization because it was the first problem to be attacked by these methods (see [16, 21, 22]). The TSP was chosen for many reasons: (i) it is a problem to which the ant colony metaphor is easily adapted, (ii) it is one of the most studied *NP*-hard [37, 43] problems in combinatorial optimization, and (iii) it is very easily explained, so that the algorithm behavior is not obscured by too many technicalities.

The traveling salesman problem, using the terminology introduced in the previous section, can be defined as follows. Let C be a set of components, representing cities, L be a set of connections fully connecting the elements in C, and $J_{c_i c_j}$ be the cost (length) of the connection between c_i and c_j, that is, the distance between cities i and j. The TSP is the problem of finding a minimal length Hamiltonian circuit on the graph $G = (C, L)$. An Hamiltonian circuit of graph G is a closed tour ψ visiting once and only once all the N_C nodes of G. Its length is given by the sum of the lengths $J_{c_i c_j}$ of all the arcs of which it is composed. Distances need not be symmetric (in an asymmetric TSP $J_{c_i c_j}$ can be different from $J_{c_j c_i}$), and the graph need not be fully connected: if it is not, the missing arcs can be added giving them a very high length.

In the following we will present Ant System, a paradigmatic example of how ACO algorithms can be applied to the TSP. Extensions of Ant System can be found in [6, 20, 48].

2.4.2 Ant System for the TSP

Ant System (AS), which was the first (1991) [16, 21] ACO algorithm, was designed as a set of three ant algorithms differing for the way the pheromone trail was updated by ants. Their names were: *ant-density*, *ant-quantity*, and *ant-cycle*. A number of ant algorithms, including the ACO meta-heuristic, have later been inspired by ant-cycle, the most performing of the three[7]. Many of these implementations have found interesting and successful applications (see Section 2.8, or [18] for a more detailed overview).

In Figure 2.4 the `new_active_ant()` procedure used by the AS algorithm is shown. This can be informally described as follows. A number $m \leq N_C$ of ants is positioned in

```
1  procedure new_active_ant(ant_identifier)
2    k = ant_identifier; i = get_city(); s_s^k = i; s^k(t) = s_s^k ;
3    M^k(t) = i;
4    while ( | s^k(t) | ≠ N_C )
5      foreach j ∈ N_i^k do read(a_ij);
6      foreach j ∈ N_i^k do [P]_ij = p_ij = a_ij / Σ_{l∈N_i^k} a_il;
7      next_node = apply_probabilistic_rule(P, N_i^k ) ;
8      i = next_node; s^k(t) = ⟨s^k(t), i⟩ ;
9      −
10     −
11     M^k(t) = i;
12   end while
13   foreach l_ij ∈ ψ^k(t) do
14     τ_ij(t) ← τ_ij(t) + 1/J_ψ^k ;
15     a_ij(t) ← [τ_ij(t)]^α [η_ij]^β / Σ_{l∈N_i^k} [τ_il(t)]^α [η_il]^β ;
16   end foreach
17   free_all_allocated_resources();
18 end procedure
```

Figure 2.4: Pseudo-code of Ant System's `new_active_ant()` procedure. Line numbers are put in one-to-one correspondence with those of the ACO meta-heuristic pseudo-code of Figure 2.3. Instruction at lines 9 and 10 are empty because no online step-by-step pheromone update is performed. Pheromone evaporation is performed between between lines 14 and 15 by the `pheromone_evaporation()` procedure of Figure 2.3, which is activated by an appropriate synchronization mechanism (not shown in the pseudo-code).

parallel on m cities. The ants' start state, that is, the start city, can be chosen randomly, and the memory $\mathcal{M}^k$ of each ant k is initialized by adding the current start city to the set of already visited cities (initially empty). Ants then enter a cycle (Figure 2.4, lines $4 \rightarrow 12$) which lasts N_C iterations, that is, until each ant has completed a tour.

[7]Hereafter, as it has been done in most published papers, we identify Ant System with ant-cycle.

During each step an ant located on node i considers the feasible neighborhood, reads the entries a_{ij}'s of the ant-routing table $\mathcal{A}_i$ of node i (figure 2.4, line 5), computes the transition probabilities (line 6), and then applies its decision rule to choose the city to move to (line 7), moves to the new city (line 8), and updates its memory (line 11).

Once ants have completed a tour (which happens synchronously, given that during each iteration of the while loop each ant adds a new city to the tour under construction), they use their memory to evaluate the built solution and to retrace the same tour backward and increase the intensity of the pheromone trails τ_{ij} of visited connections l_{ij} (lines 13 $\rightarrow$ 16). This has the effect of making the visited connections become more desirable for future ants. Then the ants die, freeing all the allocated resources. In AS all the ants deposit pheromone and no problem-specific daemon actions are performed. The triggering of pheromone evaporation happens after all ants have completed their tours. Of course, it would be easy to add a local optimization daemon action, like a 3-opt procedure [38]; this has been done in most of the ACO algorithms for TSP that have developed after AS (see for example [20, 50]).

The amount of pheromone trail $\tau_{ij}(t)$ maintained on connection l_{ij} is intended to represent the learned desirability of choosing city j when in city i (which also corresponds to the desirability that the arc l_{ij} belong to the tour built by an ant). The pheromone trail information is changed during problem solution to reflect the experience acquired by ants during problem solving. Ants deposit an amount of pheromone proportional to the quality of the solutions ψ they produced: the shorter the tour generated by an ant, the greater the amount of pheromone it deposits on the arcs which it used to generate the tour.[8] This choice helps to direct search towards good solutions.

The memory (or internal state) $\mathcal{M}^k$ of each ant contains the already visited cities and is called *tabu list*[9]. The memory $\mathcal{M}^k$ is used to define, for each ant k, the set of cities that an ant located on city i still has to visit. By exploiting $\mathcal{M}^k$ an ant k can build feasible solutions, that is, it can avoid to visit a city twice. Also, memory allows the ant to compute the length of the tour generated and to cover the same path backward to deposit pheromone on the visited arcs.

The ant-routing table $\mathcal{A}_i = [a_{ij}(t)]$ of node i, where $\mathcal{N}_i$ is the set of all the neighbor nodes of node i, is obtained by the following functional composition of pheromone trails $\tau_{ij}(t)$ and local heuristic values η_{ij}:[10]

$$a_{ij} = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i} [\tau_{il}(t)]^\alpha [\eta_{il}]^\beta} \quad \forall j \in \mathcal{N}_i \tag{2.3}$$

where α and β are two parameters that control the relative weight of pheromone trail and heuristic value.

The probability $p_{ij}^k(t)$ with which at the t-th algorithm iteration an ant k located in city i chooses the city $j \in \mathcal{N}_i^k$ to move to is given by the following probabilistic decision rule:

[8] As for most of the ACO implementations, there is no per-connection credit assignment: all the connections belonging to a solution receive the same amount of pheromone depending on the quality of the solution the connection is part of.

[9] The term tabu list is used here to indicate a simple memory that contains the set of already visited cities, and has no relation with tabu search [29, 30].

[10] The heuristic values used are $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the distance between cities i and j. In other words, the shorter the distance between two cities i and j, the higher the heuristic value η_{ij}.

$$p_{ij}^k(t) = \frac{a_{ij}(t)}{\sum_{l \in \mathcal{N}_i^k} a_{il}(t)} \tag{2.4}$$

where $\mathcal{N}_i^k \subseteq \mathcal{N}_i$ is the feasible neighborhood of node i for ant k (that is, the set of cities ant k has not yet visited) as defined by using the ant private memory $\mathcal{M}^k$ and the problem constraints.

The role of the parameters α and β is the following. If $\alpha = 0$, the closest cities are more likely to be selected: this corresponds to a classical stochastic greedy algorithm (with multiple starting points since ants are initially randomly distributed on the nodes). If on the contrary $\beta = 0$, only pheromone amplification is at work: this method will lead to the rapid emergence of a *stagnation*, that is, a situation in which all ants make the same tour which, in general, is strongly sub-optimal [22]. An appropriate trade-off has to be set between heuristic value and trail intensity.

After all ants have completed their tour, each ant k deposits a quantity of pheromone $\Delta\tau^k(t) = 1/J_\psi^k(t)$ on each connection l_{ij} that it has used, where $J_\psi^k(t)$ is the length of tour $\psi^k(t)$ done by ant k at iteration t:

$$\tau_{ij}(t) \leftarrow \tau_{ij}(t) + \Delta\tau^k(t), \quad \forall\ l_{ij} \in \psi^k(t), \quad k = 1, \ldots, m \tag{2.5}$$

where m is the number of ants at each iteration (maintained constant) and the total number of ants is set to $m = N_C$.[11] This way of setting the value $\Delta\tau^k(t)$ makes it a function of the ant's performance: the shorter the tour done, the greater the amount of pheromone deposited.

After pheromone updating has been performed by the ants[12], pheromone evaporation is triggered: the following rule is applied to all the arcs l_{ij} of the graph G

$$\tau_{ij}(t) \leftarrow (1 - \rho)\tau_{ij}(t) \tag{2.6}$$

where $\rho \in (0, 1]$ is the pheromone trail decay coefficient (the initial amount of pheromone $\tau_{ij}(0)$ is set to a small positive constant value τ_0 on all arcs).

2.5 ACO for routing in communications networks

2.5.1 The routing problem

The generic routing problem in communications networks can be informally stated as the problem of building and using *routing tables* to direct data traffic so that some measure of network performance[13] is maximized.

We can use the terminology introduced in Section 2.3 to give a formal definition of the routing problem. Let the sets C and L correspond respectively to the sets of processing

[11] These parameters settings, and those for α, β and ρ, set respectively set to 1, 5 and 0.5, were experimentally found to be good by Dorigo [16].

[12] In the original ant system [16, 21] pheromone evaporation was performed before pheromone updating. The algorithm presented here and the original one are exactly the same if the values $\Delta\tau^k(t)$ used in Equation 2.5 are set to $\Delta\tau^k(t) = 1/((1 - \rho) \cdot J_\psi^k(t))$.

[13] The choice of a measure of network performance is a function of the type of network and of the provided services. For example, in a packet-switching network, performance can be measured by throughput (amount of correctly delivered bits per time unit), and by the distribution of data packet delays.

nodes and of communication links of the real network. Let $G = (C, L)$ be a directed graph, where each node in the set C represents a network node with processing/queuing and forwarding capabilities, and each oriented arc in L is a directional transmission system (link). Each link has associated a cost measure defined by its physical properties and crossing traffic flow. Network applications generate data flows from source to destination nodes. For each node in the network, the local routing component uses the local routing table to choose the best outgoing link to direct incoming data towards their destination nodes. The routing table $\mathcal{R}_i = [r_{ijd}]$ of a generic node i, where $\mathcal{N}_i$ is the set of neighbors of i, says to data packets entering node i and directed towards destination node d which should be the next node $j \in \mathcal{N}_i$ to move to. Routing tables are bi-dimensional because the choice of the neighbor node to which a data packet entering a generic node i should be forwarded is a function of the packet destination node d. Ant-routing tables possess the same bi-dimensional structure: pheromone trails associated to each connection are vectors of $N_C - 1$ values. In fact, ant-routing tables, in all the ACO implementations for routing, are used to build the routing tables by means of implementation-dependent transformations. These vectorial pheromone trails are the natural extension of the scalar trails used for the TSP.

Other important differences with the TSP implementation arise from the different nature of the two problems: (i) each ant is assigned a defined pair (s, d) of start-destination nodes and, discovering a path between s and d, the ant builds only a part of the whole problem solution, defined in terms of paths between all the pairs (i, j) in the network, (ii) the costs associated to the connections are not statically assigned: they depend on the connection's physical properties and on the traffic crossing the connection, that interacts recursively with the routing decisions.

In the following subsection we present S-AntNet, a simplified version of the AntNet algorithm. A detailed description of AntNet can be found in [13], while a more performing extension of it is described in [15].

2.5.2 S-AntNet

In S-AntNet, each ant searches for a minimum cost path between a pair of nodes of the network. Ants are launched from each network node towards destination nodes randomly selected to match the traffic patterns. Each ant has a source node s and a destination node d, and moves from s to d hopping from one node to the next till node d is reached. When ant k is in node i, it chooses the next node j to move to according to a probabilistic decision rule which is a function of the ant's memory $\mathcal{M}^k$ and of the local ant-routing table $\mathcal{A}_i$.

Pheromone trails are still connected to arcs, but are memorized in variables associated to arc-destination pairs. That is, each directed arc (i, j) has $N_C - 1$ trail values $\tau_{ijd} \in [0, 1]$ associated, one for each possible destination node d an ant located in node i can have (therefore, in general, $\tau_{ijd} \neq \tau_{jid}$). Each arc has also associated an heuristic value $\eta_{ij} \in [0, 1]$ independent of the final destination. The heuristic values are set to the following values:

$$\eta_{ij} = 1 - \frac{q_{ij}}{\sum_{l \in \mathcal{N}_i} q_{il}} \tag{2.7}$$

where q_{ij} is the length (in bits waiting to be sent) of the queue of the link connecting node i with its neighbor j.

In S-AntNet, as well as in most other implementations of ACO algorithms for routing problems, the daemon component (line 6 of the ACO meta-heuristic of Figure 2.3) is not present.

The local ant-routing table $\mathcal{A}_i$ is obtained by a functional composition of the local pheromone trails τ_{ijd} and heuristic values η_{ij}. While building the path to the destination, ants move using the same link queues as data. In this way ants experience the same delays as data packets and the time T_{sd} elapsed while moving from the source node s to the destination node d can be used as a measure of the path quality. The overall "goodness" of a path can be evaluated by an heuristic function of the trip time T_{sd} and of a local adaptive statistical model maintained in each node. In fact, paths need to be evaluated relative to the network status because a trip time T judged of low quality under low congestion conditions could be an excellent one under high traffic load. Once the generic ant k has completed a path, it deposits on the visited nodes an amount of pheromone $\Delta\tau^k(t)$ proportional to the goodness of the path it built. To this purpose, after reaching its destination node, the ant moves back to its source nodes along the same path but backward and using high priority queues, to allow a fast propagation of the collected information.

```
1  procedure new_active_ant(ant_identifier)
2    k = ant_identifier; i = get_start_node(); t = get_end_node(); s^k(t) = s_s^k ;
3    M^k(t) = i;
4    while (i ≠ t)
5      foreach j ∈ N_i^k do read(a_ij);
6      foreach j ∈ N_i^k do [P]_ij = p_ij = a_ij / Σ_{l∈N_i^k} a_il;
7      next_node = apply_probabilistic_rule(P, N_i^k) ;
8      i = next_node; s^k(t) = ⟨s^k(t), i⟩ ;
9      −
10     −
11     M^k(t) = i;
12   end while
13   foreach l_ij ∈ ψ^k(t) do
14     τ_ij(t) ← τ_ij(t) + Δτ^k(t) ;
15     a_ij(t) ← (w τ_ij(t) + (1 − w) η_ij) / (w + (1 − w)(|N_i^k| − 1)) ;
16   end foreach
17   free_all_allocated_resources();
18 end procedure
```

Figure 2.5: Pseudo-code of S-AntNet's `new_active_ant()` procedure. Line numbers are put in one-to-one correspondence with those of the ACO meta-heuristic pseudo-code of Figure 2.3. Instruction at lines 9 and 10 are empty because no online step-by-step pheromone update is performed. The update of pheromone trails and of ant-routing tables is done by the ant during its backward path toward the origin node (lines 13 → 16). Pheromone evaporation is performed between between lines 14 and 15 by the `pheromone_evaporation()` procedure of Figure 2.3, which is activated by an appropriate synchronization mechanism (not shown in the pseudo-code).

During this backward path from d to s the ant k increases the pheromone trail value

$\tau_{ijd}(t)$ of each connection l_{ij} previously used while it was moving from s to d. The pheromone trail intensity is increased by applying the following rule:

$$\tau_{ijd}(t) \leftarrow \tau_{ijd}(t) + \Delta\tau^k(t) \tag{2.8}$$

The reason the ant updates the pheromone trails during its backward trip is that, before it can compute the amount of pheromone $\Delta\tau^k(t)$ to deposit on the visited arcs it needs to complete a path from source to destination to evaluate it.

After the pheromone trail on the visited arc has been updated, the pheromone value of all the outgoing connections of the same node i, relative to the destination d, evaporates:[14]

$$\tau_{ijd}(t) \leftarrow \frac{\tau_{ijd}(t)}{(1 + \Delta\tau^k(t))}, \quad \forall j \in \mathcal{N}_i \tag{2.9}$$

where $\mathcal{N}_i$ is the set of neighbors of node i.

As we said, S-AntNet's ant-routing table $\mathcal{A}_i = [a_{ijd}(t)]$ of node i is obtained, as usual, by the composition of the pheromone trail values with the local heuristic values. This is done as follows:

$$a_{ijd}(t) = \frac{w\tau_{ijd}(t) + (1-w)\eta_{ij}}{w + (1-w)(|\mathcal{N}_i| - 1)} \tag{2.10}$$

where $j \in \mathcal{N}_i$, d is the destination node, $w \in [0, 1]$ is a weighting factor and the denominator is a normalization term.

The ants decision rule is then defined as follows. Let, at time t, ant k be located on node i and be directed towards node d. If $\mathcal{N}_i \not\subseteq \mathcal{M}^k$, that is, if there is at least one city in the ant's current location neighborhood[15] that ant k has not visited yet, then the ant chooses the next node $j \in \mathcal{N}_i$ with probability

$$p^k_{ijd}(t) = \begin{cases} a_{ijd}(t) & \text{if } \; j \notin \mathcal{M}^k \\ 0 & \text{if } \; j \in \mathcal{M}^k \end{cases} \tag{2.11}$$

otherwise, the ant chooses a city $j \in \mathcal{N}_i$ with uniform probability: $p^k_{ijd}(t) = 1/(|\mathcal{N}_i|)$.

In other words, ants try to avoid cycles (Equation 2.11) but, in the case all the nodes in i's neighborhood have already been visited by the ant, the ant has no choice and it has to re-visit a node, generating in this way a cycle. In this case the generated cycle is deleted from the ant memory, that forgets completely about it. Considering the stochasticity of the decision rule and the evolution in the traffic conditions, it is very unlikely that the ant repeats the same cycle over and over again.

2.6 Parallel implementations

The population-oriented nature of ACO algorithms makes them particularly suitable to parallel implementation. In particular, it is at least in principle possible to exploit three different types of parallelism: (i) parallelism at the level of ants, (ii) parallelism

[14] In this case the decay factor is chosen so that it operates a normalization of the pheromone values which continue therefore to be usable as probabilities.

[15] In S-AntNet, differently from what happens in Ant System, the neighborhood and the feasible neighborhood are the same (i.e., $\mathcal{N}^k_i \equiv \mathcal{N}_i$).

at the level of data, and (iii) functional parallelism. Parallelism at the level of ants, which is probably the most obvious way of parallelizing an ACO algorithm, consists in considering a number $\mathcal{NC}$ of colonies, $\mathcal{NC} > 1$, each applied to the same problem instance. Colonies can be allowed or not to exchange information on the search process. Parallelism at the level of data consists in splitting the considered problem in a number of subproblems in the data domain, each one solved by a colony of ants. Last, functional parallelism could be easily obtained by letting the three procedures `ants_generations_and_activity()`, `pheromone_evaporation()`, and `daemon_actions()` (lines 4-6 of the `ACO_meta-heuristic()` procedure of Figure 2.3) perform their activities concurrently, maybe exchanging synchronization signals. Obviously, functional parallelism can be combined with the other two types of parallelism. Currently, all the parallel implementations of ACO algorithms use ant level parallelism. These implementations are briefly reviewed in the following.

- The first parallel versions of an ACO algorithm was Bolondi and Bondanza's implementation of AS for the TSP on the Connection Machine CM-2 [34]. The approach taken was that of attributing a single processing unit to each ant [2]. Unfortunately, experimental results showed that communication overhead can be a major problem with this approach on fine grained parallel machines, since ants spend most of their time communicating to other ants the modifications they did to pheromone trails. As a result, the algorithm's performance was not impressive and scaled up very badly when increasing the problem dimensions.

- Better results were obtained on a coarse grained parallel network of 16 transputers [2, 17]. In this implementation, Bolondi and Bondanza divided the colony in $\mathcal{NC}$ subcolonies, where $\mathcal{NC}$ was set to be the same as the number of available processors. Each subcolony acted as a complete colony and implemented therefore a standard AS algorithm. Once each subcolony completed an iteration of the algorithm, a hierarchical broadcast communication process collected the information about the tours of all the ants in all the subcolonies and then broadcast this information to all the $\mathcal{NC}$ processors. In this way, a concurrent update of the pheromone trails was performed. The speed-up obtained with this approach was nearly linear with the number of processors and this behavior was shown to be rather stable for increasing problem dimensions.

- More recently, Bullnheimer, Kotsis and Strauss [8] proposed two coarse grained parallel versions of AS called Synchronous Parallel Implementation (SPI) and Partially Asynchronous Parallel Implementation (PAPI). SPI is basically the same as the one implemented on transputers by Bolondi and Bondanza, while in PAPI pheromone information is exchanged among subcolonies every fixed number of iterations done by each subcolony. The two algorithms have been evaluated by simulation. The findings show that PAPI performs better than SPI, where performance was measured by running time and speedup. This is probably due to PAPI's reduced communication caused by the less frequent exchange of pheromone trail information among subcolonies. More experimentation will be necessary to compare the quality of the results produced by the SPI and the PAPI implementations.

- An interesting aspect of any ant level parallel implementation is the type of pheromone trail information that should be exchanged between the $\mathcal{NC}$ subcolonies and how

this information should be used to update the subcolony's trail information. Krüger, Merkle and Middendorf [36] considered: (i) the exchange of the global best solution: every subcolony uses the global best solution to choose where to add pheromone trail; (ii) the exchange of the local best solutions: every subcolony receives the local best solution from all other subcolonies and uses it to update pheromone trails; and (iii) the exchange of the total trail information: every colony computes the average over the trail information of all colonies (i.e., if $\tau^r = [\tau^r_{ij}]$ is the trail information of subcolony r, $1 \leq r \leq \mathcal{NC}$, then every colony r sends τ^r to the other colonies and afterwards computes $\tau^r_{ij} = \sum_{h=1}^{\mathcal{NC}} \tau^h_{ij}$, $1 \leq, i, j \leq n$). Preliminary results indicate that methods (i) and (ii) are faster and give better solutions than method (iii).

- The execution of parallel independent runs is the easiest way to obtain a parallel algorithm and, obviously, it is a reasonable approach when the underlying algorithm, as it is the case with ACO algorithms, is randomized. Stützle [47] presents computational results for the execution of parallel independent runs on up to ten processors of his $\mathcal{MM}$AS algorithm [48, 49]. His results show that the performance of $\mathcal{MM}$AS improves with the number of processors.

2.7 Ants, pheromones, and solutions evaluation

In this section we discuss some of the most characterizing aspects of ACO algorithms. In particular, we focus on the way solutions generated by ants are evaluated, the way these evaluations are used to direct, via pheromone trail laying, ants' search, and on the importance of using a colony of ants.

Implicit and explicit solution evaluation. In ACO algorithms solutions generated by ants provide feedback to direct the search of future ants entering the system. This is done by two mechanisms. The first one, which is common to all ACO algorithms, consists of an *explicit* solution evaluation. In this case some measure of the quality of the solution generated is used to decide how much pheromone should be deposited by ants. The second one is a kind of *implicit* solution evaluation. In this case, ants exploit the differential path length (DPL) effect of real ants foraging behavior. That is, the fact that if an ant chooses a shorter path then it is the first to deposit pheromone and to bias the search of forthcoming ants.

It turns out that in geographically distributed problems, like network problems, implicit solution evaluation based on the DPL effect can play an important role. In fact, as it was shown, for example, in [12, 13] where explicit solution evaluation was switched off by setting the amount of pheromone deposited by ants to a constant value independent of the cost of the path built by the ant, it is possible to find good solutions to network problems just exploiting the DPL effect. Quite obviously, it has also been shown that coupling explicit and implicit solution evaluation (by making the amount of pheromone deposited proportional to the cost of the solution generated) improves performance.

The fact that the DPL effect can be exploited only in geographically distributed network problems is due to efficiency reasons. In fact, the distributed nature of nodes in routing problems allows the exploitation of the DPL effect in a very natural way, without incurring in any additional computational costs. This is due both to

the decentralized nature of the system and to the inherently asynchronous nature of the dynamics of a real network.

On the contrary, this is not the case in combinatorial optimization problems where the most natural way to implement ACO algorithms is by a colony of synchronized ants, that is, ants that synchronously add elements to the solution they are building. Of course, it would in principle be possible to have asynchronous ants, in the sense explained above, also in combinatorial optimization problems. The problem is that the computational inefficiencies introduced by the computational overhead necessary to have independent, asynchronous ants can outweigh the gains due to the exploitation of the DPL effect (this was, for example, the case of the asynchronous implementation of an ACO algorithm for the TSP reported in [11]).

Explicit solution evaluation and pheromone laying. As we said above, after an ant has built a solution the cost of the built solution is used to compute the amount of pheromone the ant should deposit on the visited edges. In Ant System, for example, each ant deposits an amount of pheromone inversely proportional to the cost of the solution it generated. Obviously, this is only one of the possible choices and many implementations of ACO algorithms for the TSP or other combinatorial optimization problems exist that use different functional forms of the solution cost to decide how much pheromone the ants, or the daemon, should deposit. A problem which arises in routing problems, and in general in any problem where the characteristics of the problem change in unpredictable ways during problem solution, is that there is no simple way to evaluate a solution and therefore to decide how much pheromone ants should deposit. A way out to this problem, which is used by AntNet [12, 13], is to use ants to learn online a model of the network status that can be used to evaluate how good the solutions found by ants are.

Number of ants. The exact number of ants to be used is a parameter that, most of the times, must be set experimentally. Fortunately, ACO algorithms seem to be rather robust to the actual number of ants used. Here we will therefore limit our discussion to the following question: Why to use a colony of ants (i.e., the setting of $m > 1$) instead of using one single ant? In fact, although a single ant is capable of generating a solution, efficiency considerations suggest that the use of a colony of ants can be a desirable choice. This is particularly true for geographically distributed problems, because the differential length effect exploited by ants in the solution of this class of problems can only arise in presence of a colony of ants. It is also interesting to note that in routing problems ants solve $\hat{N} < N_C^2$ shortest path problems, and a colony of ants should be used for each of these problems.

On the other hand, in the case of combinatorial optimization problems in which ants move synchronously, the use of m ants that build θ solutions each (i.e., the ACO algorithm is run for θ iterations) could be equivalent, at least in principle, to the use of one ant that generates $m \cdot \theta$ solutions. Nevertheless, experimental evidence suggests that the algorithm's performance is at its best when the number m of ants is set to a value $M > 1$, where M is, in general, dependent on the class of problems to which the algorithm is applied.

2.8 Other ACO meta-heuristic applications

There are now available numerous successful implementations of the ACO meta-heuristic (Figure 2.3) applied to a number of different combinatorial optimization problems (see Table 2.1). The list is ordered by application problem and chronologically. The most studied problems have been the traveling salesman, the quadratic assignment (QAP) and routing in telecommunication networks. For all of these problems, ACO algorithms are competitive with the best available heuristic approaches. In particular:

- For the particularly important class of quadratic assignment problems which model real world problems, ACO algorithms are currently one of the most performing heuristics available. The next chapter of this book presents an overview of the available ACO algorithms for the QAP.
- Results obtained by the application of ACO algorithms to the TSP are very encouraging (see [51] for an overview of applications of ACO algorithms to the TSP): they are often better than those obtained using other general purpouse heuristics like evolutionary computation or simulated annealing. Also, when adding to ACO algorithms rather unsophisticated local search procedures based on 3-opt [38], the quality of the results obtained [20, 46, 50] is close to that obtainable by much more sophisticated methods. More research will be necessary to assess whether ACO algorithms can reach the performance of state-of-the-art algorithms like Iterated Lin-Kernighan [35].
- An ACO algorithm called AntNet [13, 15] outperformed a number of state-of-the-art routing algorithms for packet-switching networks on a set of benchmark problems.

Very interesting results have been obtained also for:

- The sequential ordering problem, that is, the problem of finding the shortest Hamiltonian path on a graph which satisfies a set of precedence constraints on the order in which cities are visited. When applied to this problem HAS-SOP, an ACO algorithm coupled to a local search routine, has improved many of the best known results on a wide set of benchmark problems [25].
- The shortest common supersequence problem and the vehicle routing problem. These two problems, as well as the ACO algorithms proposed for their solution, are the subject of the two concluding chapters of this book section dedicated to ACO algorithms.

Last, ACO algorithms have also been applied to the graph coloring problem, for which reasonably good, although not state-of-the-art results were obtained.

2.9 A short overview of the applications presented in the following chapters of this book section

This section of the book includes three further chapters dedicated to applications of ACO algorithms to some important and difficult combinatorial optimization problems: the quadratic assignment problem, the vehicle routing problem, and the shortest common

Table 2.1. List of current applications of ACO algorithms. Classification by application and chronologically ordered.

Problem name	Authors	Year	Main references	Algorithm name
Traveling salesman	Dorigo, Maniezzo & Colorni	1991	[16, 21, 22]	AS
	Gambardella & Dorigo	1995	[23]	Ant-Q
	Dorigo & Gambardella	1996	[19, 20, 24]	ACS & ACS-3-opt
	Stützle & Hoos	1997	[48, 49]	$\mathcal{MM}$AS
	Bullnheimer, Hartl & Strauss	1997	[6]	AS_{rank}
Quadratic assignment	Maniezzo, Colorni & Dorigo	1994	[41]	AS-QAP
	Gambardella, Taillard & Dorigo	1997	[27]	HAS-QAP[a]
	Stützle & Hoos	1998	[50]	$\mathcal{MM}$AS-QAP
	Maniezzo & Colorni	1998	[40]	AS-QAP[b]
	Maniezzo	1998	[39]	ANTS-QAP
Vehicle routing	Bullnheimer, Hartl & Strauss	1996	[9, 5, 7]	AS-VRP
	Gambardella, Taillard & Agazzi	1999	[26]	HAS-VRP
Connection-oriented network routing	Schoonderwoerd, Holland, Bruten & Rothkrantz	1996	[45, 44]	ABC
	White, Pagurek & Oppacher	1998	[55]	ASGA
	Di Caro & Dorigo	1998	[14]	AntNet-FS
	Bonabeau, Henaux, Guérin, Snyers, Kuntz & Théraulaz	1998	[4]	ABC-smart ants
Connection-less network routing	Di Caro & Dorigo	1997	[12, 13, 15]	AntNet & AntNet-FA
	Subramanian, Druschel & Chen	1997	[52]	Regular ants
	Heusse, Guérin, Snyers & Kuntz	1998	[33]	CAF
	van der Put & Rothkrantz	1998	[53, 54]	ABC-backward
Sequential ordering	Gambardella & Dorigo	1997	[25]	HAS-SOP
Graph coloring	Costa & Hertz	1997	[10]	ANTCOL
Shortest common supersequence	Michel & Middendorf	1998	[42]	AS-SCS

[a] HAS-QAP is an ant algorithm which does not follow all the aspects of the ACO meta-heuristic.
[b] This is a variant of the original AS-QAP.

supersequence problem. Although these problems are introduced and discussed in detail in the forthcoming chapters, it is interesting here to show how they can be cast in the ACO meta-heuristic framework.

2.9.1 The quadratic assignment problem

The quadratic assignment problem can be stated as follows. Consider a set of n activities that have to be assigned to n locations. A matrix $\mathcal{D} = [d_{ij}]$ gives distances between locations, where d_{ij} is the distance between location i and location j, and a matrix $\mathcal{F} = [f_{hk}]$ characterizes flows among activities (transfers of data, material, humans, etc.), where f_{hk} is the flow between activity h and activity k. An assignment is a permutation π of $\{1, \ldots, n\}$, where $\pi(i)$ is the activity that is assigned to location i. The problem is to find a permutation π_m such that the product of the flows among activities by the distances between their locations be minimized. Here it is interesting to note that the TSP can be seen a particular case of the QAP: the items are the set of integers between 1 and n, while the locations are the cities to be visited. The TSP is then the problem of assigning a different integer number to each city in such a way that the tour which visits the cities ordered according to their assigned number has minimal length.

In the ACO algorithms presented in Chapter 3 of this book the QAP is represented as follows. The set C of the components is composed of the activities and of the locations. Transitions are from activities to locations and from locations to activities. Typically, an ant starts building a solution by first choosing an activity, then a location to which to assign the activity, then another activity, and so on, until all activities have been assigned. Activities, as well as locations, are chosen within the feasible neighborhood,

that is, within the set of activities (locations) not yet assigned. Typically, in AS-QAP and $\mathcal{MM}$AS-QAP for example, pheromone trails are associated to transitions from activities to locations (that is, to the choice of the location to which to assign an activity), but not to transitions from locations to activities (which are chosen by some probabilistic or heuristic rule that is not a function of pheromone trails). Obviously, nothing prevents from defining an ACO algorithm in which also transitions from locations to activities are a function of pheromone trails.

2.9.2 The shortest common supersequence problem

Given a set L of strings over an alphabet Σ, the shortest common supersequence problem consists in finding a string of minimal length that is a supersequence of each string in L. A string B is a supersequence of a string A if B can be obtained from A by inserting in A zero or more characters. Consider for example the set $L = bbbaaa, bbaaab, cbaab, cbaaa$. The string *cbbbaaab* is a shortest supersequence. Ants build solutions by repeatedly removing symbols from the front of the strings in L and appending them to the supersequence under construction. In practice, each ant maintains a vector of pointers to the front of the strings (where the front of a string is the first character in the string not yet removed) and moves in the space of the feasible vectors. Here the representation used by the ACO algorithm is the following. The components are the vectors of pointers, the transitions are implicitly defined by the rules which govern the way in which characters can be removed from the string fronts, and the constraints are implicitly defined by the ordering of the characters in the strings.

2.9.3 The vehicle routing problem

Vehicle routing problems (VRPs) are a class of problems in which a set of vehicles has to serve a set of customers minimizing a cost function and subject to a number of constraints. The characteristics of the vehicles and of the constraints determines the particular type of VRP. The VRP considered in Chapter 5 is called vehicle routing problem with time windows (VRPTW): Let $G = (N, A)$ be a complete directed graph, where $N = \{n_0, \ldots, n_n\}$ is the set of nodes, and $A = (i, j) : i \neq j$ is the set of arcs. Node n_0 represents a depot, while the other nodes represent customers locations. A weight $t_{ij} \geq 0$, representing the travel time from node n_i to node n_j, is associated to each arc (i, j). A demand $q_i \geq 0$ ($q_0 = 0$) and a service window $[b_i, e_i]$, with $e_i \geq b_i \geq 0$, are associated to each customer n_i. The objective is to find minimum cost vehicle routes such that (i) every customer is visited exactly once by exactly one vehicle, (ii) every customer is visited during its time window, (iii) for every vehicle the total demand does not exceed the vehicle capacity Q, and (iv) every vehicle starts and ends its tour in the depot. The particular VRPTW considered in Chapter 5 uses a hierarchical cost function: the first goal is to minimize the number of vehicles used, while the second goal is to minimize the total travel times. A solution with a lower number of vehicles is always preferred to a solution with a higher number of tours even if the travel time is higher. It is easy to see that VRPs and TSPs are closely related: a VRP consists of the solution of many TSPs with common start and end city (that is, the depot). As in the TSP, ants build their solutions by sequentially visiting all the cities: the problem's components are the cities, while the transitions can be associated to the arcs. The feasible neighborhood is given by the set of unvisited cities and pheromone trails are associated to arcs.

2.10 Acknowledgments

We are grateful to Thomas Stützle for critical reading of a draft version of this article. Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Research Associate. This work was supported by a Madame Curie Fellowship awarded to Gianni Di Caro (CEC-TMR Contract N. ERBFMBICT 961153).

Bibliography

[1] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[2] M. Bolondi and M. Bondanza. Parallelizzazione di un algoritmo per la risoluzione del problema del commesso viaggiatore. Master's thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1993.

[3] E. Bonabeau, M. Dorigo, and G. Théraulaz. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, 1999.

[4] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Théraulaz. Routing in telecommunication networks with "Smart" ant-like agents. In *Proceedings of IATA'98, Second Int. Workshop on Intelligent Agents for Telecommunication Applications*. Lectures Notes in AI vol. 1437, Springer Verlag, 1998.

[5] B. Bullnheimer, R. F. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. Technical Report POM-10/97, Institute of Management Science, University of Vienna, Austria, 1997. Accepted for publication in the *Annals of Operations Research*.

[6] B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank-based version of the ant system: a computational study. Technical Report POM-03/97, Institute of Management Science, University of Vienna, Austria, 1997. Accepted for publication in the *Central European Journal for Operations Research and Economics*.

[7] B. Bullnheimer, R. F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In S. Voß, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 285–296. Kluwer Academics, 1999.

[8] B. Bullnheimer, G. Kotsis, and C. Strauss. Parallelization strategies for the ant system. Technical Report POM 9-97, Institute of Management Science, University of Vienna, Austria, 1997. To appear in: Kluwer Series of Applied Optmization (selected papers of HPSNO'97), A. Murli, P. Pardalos and G. Toraldo, editors.

[9] B. Bullnheimer and C. Strauss. Tourenplanung mit dem Ant System. Technical Report 6, Institut für Betriebwirtschaftslehre, University of Vienna, Austria, 1996.

[10] D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.

[11] M. Cottarelli and A. Gobbi. Estensioni dell'algoritmo formiche per il problema del commesso viaggiatore. Master's thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, 1997.

[12] G. Di Caro and M. Dorigo. AntNet: A mobile agents approach to adaptive routing. Technical Report IRIDIA/97-12, IRIDIA, Université Libre de Bruxelles, Belgium, 1997.

[13] G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research (JAIR)*, 9:317–365, December 1998.

[14] G. Di Caro and M. Dorigo. Extending AntNet for best-effort Quality-of-Service routing. Unpublished presentation at *ANTS'98 - From Ant Colonies to Artificial Ants: First International Workshop on Ant Colony Optimization* http://iridia.ulb.ac.be/ants98/ants98.html, October 15-16 1998.

[15] G. Di Caro and M. Dorigo. Two ant colony algorithms for best-effort routing in datagram networks. In *Proceedings of the Tenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98)*, pages 541–546. IASTED/ACTA Press, 1998.

[16] M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[17] M. Dorigo. Parallel ant system: An experimental study. Unpublished manuscript, 1993.

[18] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. Technical Report IRIDIA/98-10, IRIDIA, Université Libre de Bruxelles, Belgium, 1998. Accepted for publication in *Artificial Life*.

[19] M. Dorigo and L. M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–81, 1997.

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

[21] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[22] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 26(1):29–41, 1996.

[23] L. M. Gambardella and M. Dorigo. Ant-Q: A reinforcement learning approach to the traveling salesman problem. In *Proceedings of the Twelfth International Conference on Machine Learning, ML-95*, pages 252–260. Palo Alto, California: Morgan Kaufmann, 1995.

[24] L. M. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC96*, pages 622–627. IEEE Press, 1996.

[25] L. M. Gambardella and M. Dorigo. HAS-SOP: An hybrid ant system for the sequential ordering problem. Technical Report IDSIA-11-97, IDSIA, Lugano, Switzerland, 1997.

[26] L. M. Gambardella, E. Taillard, and G. Agazzi. Ant colonies for vehicle routing problems. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.

[27] L. M. Gambardella, E. D. Taillard, and M. Dorigo. Ant colonies for the QAP. Technical Report IDSIA-4-97, IDSIA, Lugano, Switzerland, 1997. Accepted for publication in the *Journal of the Operational Research Society (JORS)*.

[28] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, 1979.

[29] F. Glover. Tabu search, part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[30] F. Glover. Tabu search, part II. *ORSA Journal on Computing*, 2:4–32, 1989.

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

[32] P. P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez *bellicositermes natalensis et cubitermes sp.* La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[33] M. Heusse, S. Guérin, D. Snyers, and P. Kuntz. Adaptive agent-driven routing and load balancing in communication networks. Technical Report RR-98001-IASC, Départment Intelligence Artificielle et Sciences Cognitives, ENST Bretagne, 1998. Accepted for publication in the *Journal of Complex Systems*.

[34] W. D. Hillis. *The Connection Machine*. MIT Press, 1982.

[35] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study. In E.H. Aarts and J. K. Lenstra, editors, *In Local Search in Combinatorial Optimization*, pages 215–310. Chichester: John Wiley & Sons, 1997.

[36] F. Krüger, D. Merkle, and M. Middendorf. Studies on a parallel ant system for the BSP model. Unpublished manuscript.

[37] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy-Kan, and D. B. Shmoys, editors. *The Travelling Salesman Problem*. Wiley, 1985.

[38] S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Journal*, 44:2245–2269, 1965.

[39] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. Technical Report CSR 98-1, C. L. In Scienze dell'Informazione, Universitá di Bologna, sede di Cesena, Italy, 1998.

[40] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *IEEE Trans. Knowledge and Data Engineering*, 1999, in press.

[41] V. Maniezzo, A. Colorni, and M. Dorigo. The ant system applied to the quadratic assignment problem. Technical Report IRIDIA/94-28, IRIDIA, Université Libre de Bruxelles, Belgium, 1994.

[42] R. Michel and M. Middendorf. An island model based ant system with lookahead for the shortest supersequence problem. In A. E. Eiben, T. Back, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, pages 692–701. Springer-Verlag, 1998.

[43] G. Reinelt. *The Traveling Salesman Problem: Computational Solutions for TSP Applications*. Berlin: Springer-Verlag, 1994.

[44] R. Schoonderwoerd, O. Holland, and J. Bruten. Ant-like agents for load balancing in telecommunications networks. In *Proceedings of the First International Conference on Autonomous Agents*, pages 209–216. ACM Press, 1997.

[45] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1996.

[46] T. Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*. PhD thesis, Fachbereich Informatik, TU Darmstadt, Germany, 1998.

[47] T. Stützle. Parallelization strategies for ant colony optimization. In A. E. Eiben, T. Back, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, pages 722–731. Springer-Verlag, 1998.

[48] T. Stützle and M. Dorigo. ACO Algorithms for the Traveling Salesman Problem. In K. Miettinen P. Neittaanmäki, J. Periaux and M.M. Mäkelä, editors, *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications*. John Wiley & Sons, 1999.

[49] T. Stützle and H. Hoos. The $\mathcal{MAX}$–$\mathcal{MIN}$ ant system and local search for the traveling salesman problem. In T. Baeck, Z. Michalewicz, and X. Yao, editors, *Proceedings of IEEE-ICEC-EPS'97, IEEE International Conference on Evolutionary Computation and Evolutionary Programming Conference*, pages 309–314. IEEE Press, 1997.

[50] T. Stützle and H. Hoos. Improvements on the ant system: Introducing $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 245–249. Springer Verlag, Vienna, 1997.

[51] T. Stützle and H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant system and local search for combinatorial optimization problems. In S. Voß, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 313–329. Kluwer Academics, Boston, 1998.

[52] D. Subramanian, P. Druschel, and J. Chen. Ants and reinforcement learning: A case study in routing in dynamic networks. In *Proceedings of IJCAI-97, International Joint Conference on Artificial Intelligence*, pages 832–838. Morgan Kaufmann, 1997.

[53] R. van der Put. Routing in the faxfactory using mobile agents. Technical Report R&D-SV-98-276, KPN Research, The Netherlands, 1998.

[54] R. van der Put and L. Rothkrantz. Routing in packet switched networks using agents. *Simulation Practice and Theory*, 1999, in press.

[55] T. White, B. Pagurek, and F. Oppacher. Connection management using adaptive mobile agents. In H.R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, pages 802–809. CSREA Press, 1998.