The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances

Marco Dorigo
Université Libre de Bruxelles, IRIDIA,
Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium
mdorigo@ulb.ac.be

Thomas Stützle
TU Darmstadt, Computer Science, Intellectics Group
Alexanderstr. 10, D-64283 Darmstadt, Germany
stuetzle@informatik.tu-darmstadt.de

1 Introduction

Ant Colony Optimization (ACO) [31, 32] is a recently proposed metaheuristic approach for solving hard combinatorial optimization problems. The inspiring source of ACO is the pheromone trail laying and following behavior of real ants which use pheromones as a communication medium. In analogy to the biological example, ACO is based on the indirect communication of a colony of simple agents, called (artificial) ants, mediated by (artificial) pheromone trails. The pheromone trails in ACO serve as a distributed, numerical information which the ants use to probabilistically construct solutions to the problem being solved and which the ants adapt during the algorithm's execution to reflect their search experience.

The first example of such an algorithm is Ant System (AS) [29, 36, 37, 38], which was proposed using as example application the well known Traveling Salesman Problem (TSP) [58, 74]. Despite encouraging initial results, AS could not compete with state-of-the-art algorithms for the TSP. Nevertheless, it had the important role of stimulating further research on algorithmic variants which obtain much better computational performance, as well as on applications to a large variety of different problems. In fact, there exists now a considerable amount of applications obtaining world class performance on problems like the quadratic assignment, vehicle routing, sequential ordering, scheduling, routing in Internet-like networks, and so on [21, 25, 44, 45, 66, 83]. Motivated by this success, the ACO metaheuristic has been proposed [31, 32] as a common framework for the existing applications and algorithmic variants. Algorithms which follow the ACO metaheuristic will be called in the following ACO algorithms.

Current applications of ACO algorithms fall into the two important problem classes of static and dynamic combinatorial optimization problems. Static problems are those whose topology and cost do not change while the problems are being solved. This is the case, for example, for the classic TSP, in which city locations and intercity distances do not change during the algorithm's run-time. Differently, in dynamic problems the topology and costs can change while solutions are built. An example of such a problem is routing in telecommunications networks [25], in which traffic patterns change all the time. The ACO algorithms for solving these two classes of problems are very similar from a high-level perspective, but they differ significantly in implementation details. The ACO metaheuristic captures these differences and is general enough to comprise the ideas common to both application types.

The (artificial) ants in ACO implement a randomized construction heuristic which makes probabilistic decisions as a function of artificial pheromone trails and possibly available heuristic information based on the input data of the problem to be solved. As such, ACO can be interpreted as an extension of traditional construction heuristics which are readily available for many combinatorial optimization problems. Yet, an important difference with construction heuristics is the adaptation of the pheromone trails during algorithm execution to take into account the cumulated search experience.

The rest of this chapter is organized as follows. In Section 2, we briefly overview construction heuristics and local search algorithms. In Section 3 we define the ants' behavior, the ACO metaheuristic, and the type of problems to which it can be applied. Section 4 outlines the inspiring biological analogy and describes the historical developments leading to ACO. In Section 5 we illustrate how the ACO metaheuristic can be applied to different types of problems and we give an overview of its successful applications. Section 6 discusses several issues arising in the application of the ACO metaheuristic, while in Section 7 we present recent developments and conclude in Section 8 indicating future research directions.

2 Traditional approximation approaches

Many important combinatorial optimization problems are hard to solve. The notion of problem hardness is captured by the theory of computational complexity [47, 72] and for many important problems it is well known that they are $\mathcal{NP}$-hard, that is the time we needed to solve an instance in the worst case grows exponentially with instance size. Often, approximate algorithms are the only feasible way to obtain

procedure *Greedy Construction Heuristic*
  $s_p =$ *empty solution*
  while s_p no complete solution **do**
    $e =$ `GreedyComponent`(s_p)
    $s_p = s_p \otimes e$
  end
  return s_p
end *Greedy Construction Heuristic*

Figure 1: Algorithmic skeleton of a greedy construction heuristic. The addition of component e to a partial solution s_p is denoted by the operator $\otimes$.

near optimal solutions at relatively low computational cost.

Classically, most approximate algorithms are either *construction* algorithms or *local search* algorithms.[1] These two types of methods are significantly different, because construction algorithms work on partial solutions trying to extend these in the best possible way to complete problem solutions, while local search methods move in the search space of complete solutions.

2.1 Construction algorithms

Construction algorithms build solutions to a problem under consideration in an incremental way starting with an empty initial solution and iteratively adding opportunely defined solution components without backtracking until a complete solution is obtained. In the simplest case, solution components are added in random order. Often better results are obtained if a heuristic estimate of the myopic benefit of adding solution components is taken into account. *Greedy construction heuristics* add at each step a solution component which achieves the maximal myopic benefit as measured by some heuristic information. An algorithmic outline of a greedy construction heuristic is given in Figure 1. The function `GreedyComponent` returns the solution component e with the best heuristic estimate. Solutions returned by greedy algorithms are typically of better quality than randomly generated solutions. Yet, a disadvantage of greedy construction heuristics is that only a very limited number of solutions can be generated. Additionally, greedy decisions in early stages of the construction process strongly constrain the available possibilities at later stages, often determining very poor moves in the final phases of the

[1] Other approximate methods are also conceivable. For example, when stopping exact methods, like Branch & Bound, before completion [4, 56] (for example, after some given time bound, or when some guarantee on the solution quality is obtained through the use of lower and upper bounds), we can convert exact algorithms into approximate ones.

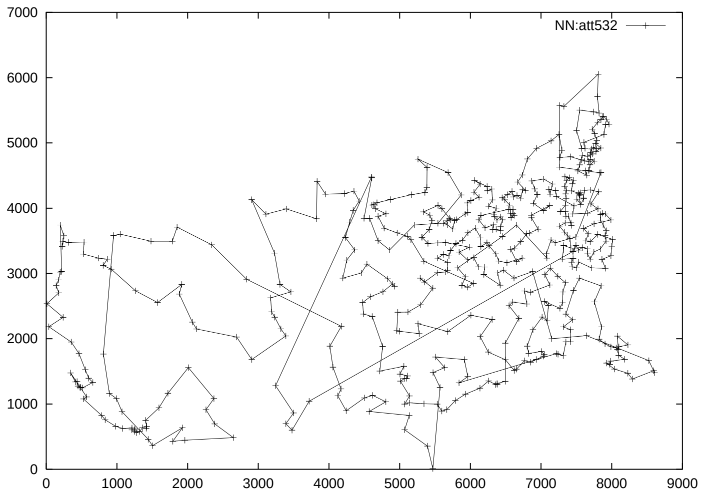

Figure 2: Tour returned by the nearest neighbor heuristic on TSP instance `att532` from TSPLIB.

solution construction.

As an example consider a greedy construction heuristic for the traveling salesman problem (TSP). In the TSP we are given a complete weighted graph $G = (N, A)$ with N being the set of nodes, representing the cities, and A the set of arcs fully connecting the nodes N. Each arc is assigned a value d_{ij}, which is the length of arc $(i, j) \in A$. The TSP is the problem of finding a minimal length Hamiltonian circuit of the graph, where an Hamiltonian circuit is a closed tour visiting exactly once each of the $n = |N|$ nodes of G. For symmetric TSPs, the distances between the cities are independent of the direction of traversing the arcs, that is, $d_{ij} = d_{ji}$ for every pair of nodes. In the more general asymmetric TSP (ATSP) at least for one pair of nodes i, j we have $d_{ij} \neq d_{ji}$.

A simple rule of thumb to build a tour is to start from some initial city and to always choose to go to the closest still unvisited city before returning to the start city. This algorithm is known as the *nearest neighbor* tour construction heuristic. Figure 2 shows a tour returned by the nearest neighbor heuristic on TSP instance `att532`, taken from TSPLIB,[2] with 532 cities in the US.

Noteworthy in this example is that there are some few very long links in the tour, leading to strongly suboptimal solutions. In fact, construction algorithms are typically the fastest approximate methods, but the solutions they generate often

[2]TSPLIB is a benchmark library for the TSP and related problems and is accessible via http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95

```
procedure IterativeImprovement (s ∈ S)
   s' = Improve(s)
   while s' ≠ s do
      s = s'
      s' = Improve(s)
   end
   return s
end IterativeImprovement
```

Figure 3: Algorithmic skeleton of iterative improvement.

are not of a very high quality and they are not guaranteed to be optimal with respect to small changes; the results produced by constructive heuristics can often be improved by local search algorithms.

2.2 Local search

Local search algorithms start from a complete initial solution and try to find a better solution in an appropriately defined *neighborhood* of the current solution. In its most basic version, known as *iterative improvement*, the algorithm searches the neighborhood for an improving solution. If such a solution is found, it replaces the current solution and the local search continues. These steps are repeated until no improving neighbor solution is found anymore in the neighborhood of the current solution and the algorithm ends in a *local optimum*. An outline of an iterative improvement algorithm is given in Figure 3. The procedure `Improve` returns a better neighbor solution if one exists, otherwise it returns the current solution, in which case the algorithm stops.

The choice of an appropriate neighborhood structure is crucial for the performance of the local search algorithm and has to be done in a problem specific way. The neighborhood structure defines the set of solutions that can be reached from s in one single step of a local search algorithm. An example neighborhood for the TSP is the *k-opt* neighborhood in which neighbor solutions differ by at most k arcs. Figure 4 shows the example of a 2-opt neighborhood. The 2-opt algorithm systematically tests whether the current tour can be improved by replacing two edges. To fully specify a local search algorithm is needed a neighborhood examination scheme that defines how the neighborhood is searched and which neighbor solution replaces the current one. In the case of iterative improvement algorithms, this rule is called *pivoting rule* [90] and examples are the *best-improvement* rule, which chooses the neighbor solution giving the largest improvement of the objective function, and the *first-improvement* rule, which uses the first improved solution

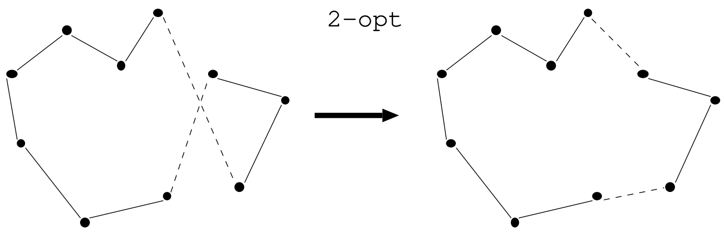

Figure 4: Schematic illustration of the 2-opt algorithm. The proposed move reduces the total tour length if we consider the Euclidean distance between the points.

found in the neighborhood to replace the current one. A common problem with local search algorithms is that they easily get trapped in local minima and that the result strongly depends on the initial solution.

3 ACO Metaheuristic

Artificial ants used in ACO are stochastic solution construction procedures that probabilistically build a solution by iteratively adding solution components to partial solutions by taking into account (i) heuristic information on the problem instance being solved, if available, and (ii) (artificial) pheromone trails which change dynamically at run-time to reflect the agents' acquired search experience.

The interpretation of ACO as an extension of construction heuristics is appealing because of several reasons. A stochastic component in ACO allows the ants to build a wide variety of different solutions and hence explore a much larger number of solutions than greedy heuristics. At the same time, the use of heuristic information, which is readily available for many problems, can guide the ants towards the most promising solutions. More important, the ants' search experience can be used to influence in a way reminiscent of reinforcement learning [87] the solution construction in future iterations of the algorithm. Additionally, the use of a colony of ants can give the algorithm increased robustness and in many ACO applications the collective interaction of a population of agents is needed to efficiently solve a problem.

The domain of application of ACO algorithms is vast. In principle, ACO can be applied to any discrete optimization problem for which some solution construction mechanism can be conceived. In the following of this section, we first define a generic problem representation which the ants in ACO exploit to construct solutions, then we detail the ants' behavior while constructing solutions, and finally we define the ACO metaheuristic.

3.1 Problem representation

Let us consider the minimization problem[3] $(\mathcal{S}, f, \Omega)$, where $\mathcal{S}$ is the *set of candidate solutions*, f is the *objective function* which assigns to each candidate solution $s \in \mathcal{S}$ an objective function (cost) value $f(s,t)$,[4] and Ω is a set of constraints. The goal is to find a *globally optimal* solution $s_{opt} \in \mathcal{S}$, that is, a minimum cost solution that satisfies the constraints Ω.

The problem representation of a combinatorial optimization problem $(\mathcal{S}, f, \Omega)$ which is exploited by the ants can be characterized as follows:

- A finite set $\mathcal{C} = \{c_1, c_2, \ldots, c_{N_C}\}$ of *components* is given.
- The *states* of the problem are defined in terms of sequences $x = \langle c_i, c_j, \ldots, c_k, \ldots \rangle$ over the elements of $\mathcal{C}$. The set of all possible sequences is denoted by $\mathcal{X}$. The length of a sequence x, that is, the number of components in the sequence, is expressed by $|x|$.
- The finite set of *constraints* Ω defines the set of feasible states $\tilde{\mathcal{X}}$, with $\tilde{\mathcal{X}} \subseteq \mathcal{X}$.
- A set $\mathcal{S}^*$ of feasible solutions is given, with $\mathcal{S}^* \subseteq \tilde{\mathcal{X}}$ and $\mathcal{S}^* \subseteq \mathcal{S}$.
- A *cost* $f(s,t)$ is associated to each candidate solution $s \in \mathcal{S}$.
- In some cases a cost, or the estimate of a cost, $J(x_i, t)$ can be associated to states other than solutions. If x_j can be obtained by adding solution components to a state x_i then $J(x_i,t) \leq J(x_j,t)$. Note that $J(s,t) \equiv f(s,t)$.

Given this representation, artificial ants build solutions by moving on the construction graph $G = (\mathcal{C}, \mathcal{L})$, where the vertices are the components $\mathcal{C}$ and the set $\mathcal{L}$ fully connects the components C (elements of $\mathcal{L}$ are called *connections*). The problem constraints Ω are implemented in the policy followed by the artificial ants, as explained in the next section. The choice of implementing the constraints in the construction policy of the artificial ants allows a certain degree of flexibility. In fact, depending on the combinatorial optimization problem considered, it may be more reasonable to implement constraints in a hard way allowing ants to build only feasible solutions, or in a soft way, in which case ants can build infeasible solutions (that is, candidate solutions in $\mathcal{S} \setminus \mathcal{S}^*$) that will be penalized, in dependence of their degree of infeasibility.

[3]The adaptation to a maximization problem is straightforward.

[4]The parameter *t* indicates that the the objective function can be time dependent, as it is the case, for example, in applications to dynamic problems.

3.2 Ant's behavior

Ants can be characterized as stochastic construction procedures which build solutions moving on the construction graph $G = (\mathcal{C}, \mathcal{L})$. Ants do not move arbitrarily on G, but rather follow a construction policy which is a function of the problem constraints Ω. In general, ants try to build feasible solutions, but, if necessary, they can generate infeasible solutions. Components $c_i \in \mathcal{C}$ and connections $l_{ij} \in \mathcal{L}$ can have associated a *pheromone trail* τ (τ_i if associated to components, τ_{ij} if associated to connections) encoding a long-term memory about the whole ant search process that is updated by the ants themselves, and a *heuristic value* η (η_i and η_{ij}, respectively) representing a priori information about the problem instance definition or run-time information provided by a source different from the ants. In many cases η is the cost, or an estimate of the cost, of extending the current state. These values are used by the ants heuristic rule to make probabilistic decisions on how to move on the graph.

More precisely, each ant k of the colony has the following properties:

- It exploits the graph $G = (\mathcal{C}, \mathcal{L})$ to search for feasible solutions s of minimum cost. That is, solutions s such that $\hat{f}_s = \min_s f(s, t)$.
- It has a memory $\mathcal{M}^k$ that it uses to store information about the path it followed so far. Memory can be used (i) to build feasible solutions (i.e., to implement constraints Ω), (ii) to evaluate the solution found, and (iii) to retrace the path backward to deposit pheromone.
- It can be assigned a *start state* x_s^k and one or more *termination conditions* e^k. Usually, the start state is expressed as a unit length sequence, that is, a single component or an empty sequence.
- When in state $x_r = \langle x_{r-1}, i \rangle$ it tries to move to any node j in its *feasible neighborhood* $\mathcal{N}_i^k$, that is to a state $\langle x_r, j \rangle \in \tilde{X}$. If this is not possible, then the ant can move to a node j in its infeasible neighborhood $\mathcal{IN}_i^k$, generating in this way an infeasible state (i.e., a state $\langle x_r, j \rangle \in X \setminus \tilde{X}\}$.
- It selects the move by applying a probabilistic decision rule. Its probabilistic decision rule is a function of (i) locally available pheromone trails and heuristic values, (ii) the ant's private memory storing its past history, and (iii) the problem constraints.
- The construction procedure of ant k stops when at least one of the termination conditions e^k is satisfied.

- When adding a component c_j to the current solution it can update the pheromone trail associated to it or to the corresponding connection. This is called *online step-by-step pheromone update*.
- Once built a solution, it can retrace the same path backward and update the pheromone trails of the used components or connections. This is called *online delayed pheromone update*.

It is important to note that ants move concurrently and independently and that each ant is complex enough to find a (probably poor) solution to the problem under consideration. Typically, good quality solutions emerge as the result of the collective interaction among the ants which is obtained via indirect communication mediated by the information ants read/write in the variables storing pheromone trail values. In a way, this is a distributed learning process in which the single agents, the ants, are not adaptive themselves but, on the contrary, they adaptively modify the way the problem is represented and perceived by other ants.

3.3 The metaheuristic

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 building paths on G. They move by applying a stochastic local decision policy that makes use of pheromone trails and heuristic information. 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 pheromone trails on the components or connections it used. This pheromone information will direct the search of the future ants.

Besides ants' activity, an ACO algorithm includes two more procedures: *pheromone trail evaporation* and *daemon actions* (this last component being optional). Pheromone evaporation is the process by means of which the pheromone trail intensity on the components 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 components used by the ant that built

```
procedure ACO metaheuristic
   ScheduleActivities
      ManageAntsActivity()
      EvaporatePheromone()
      DaemonActions() {Optional}
   end ScheduleActivities
end ACO metaheuristic
```

Figure 5: The ACO metaheuristic in pseudo-code. Comments are enclosed in braces. The procedure `DaemonActions()` is optional and refers to centralized actions executed by a daemon possessing global knowledge.

the best solution. Pheromone updates performed by the daemon are called *off-line pheromone updates*.

In Figure 5 the ACO metaheuristic behavior is described in pseudo-code. The main procedure of the ACO metaheuristic manages, via the **ScheduleActivities** construct, the scheduling of the three above discussed components of ACO algorithms: (i) management of ants' activity, (ii) pheromone evaporation, and (iii) daemon actions. The **ScheduleActivities** construct does not specify how these three activities are scheduled and synchronized. In other words, it does not say whether they should be executed in a completely parallel and independent way, or if some kind of synchronization among them is necessary. The designer is therefore free to specify the way these three procedures should interact.

4 History of ACO algorithms

The first ACO algorithm proposed was Ant System (AS). AS was applied to some rather small instances of the traveling salesman problem (TSP) with up to 75 cities. It was able to reach the performance of other general-purpose heuristics like evolutionary computation [29, 38]. Despite these initial encouraging results, AS could not prove to be competitive with state-of-the-art algorithms specifically designed for the TSP when attacking large instances. Therefore, a substantial amount of recent research has focused on ACO algorithms which show better performance than AS when applied, for example, to the TSP. In the following of this section we first briefly introduce the biological metaphor on which AS and ACO are inspired, and then we present a brief history of the developments that took from the original AS to the most recent ACO algorithms. In fact, these more recent algorithms are direct extensions of AS which add advanced features to improve the algorithm performance.

4.1 Biological analogy

In many ant species, individual ants may deposit a pheromone (a particular chemical that ants can smell) on the ground while walking [50]. By depositing pheromone they create a trail that is used, for example, to mark the path from the nest to food sources and back. In fact, by sensing pheromone trails foragers can follow the path to food discovered by other ants. Also, they are capable of exploiting pheromone trails to choose the shortest among the available paths taking to the food.

Deneubourg and colleagues [22, 50] used a double bridge connecting a nest of ants and a food source to study pheromone trail laying and following behavior in controlled experimental conditions.[5] They ran a number of experiments in which they varied the ratio between the length of the two branches of the bridge. The most interesting, for our purposes, of these experiments is the one in which one branch was longer than the other. In this experiment, at the start the ants were left free to move between the nest and the food source and the percentage of ants that chose one or the other of the two branches was observed over time. The outcome was that, although in the initial phase random oscillations could occur, in most experiments all the ants ended up using the shorter branch.

This result can be explained as follows. When a trial starts there is no pheromone on the two branches. Hence, the ants do not have a preference and they select with the same probability any of the two branches. Therefore, it can be expected that, on average, half of the ants choose the short branch and the other half the long branch, although stochastic oscillations may occasionally favor one branch over the other. However, because one branch is shorter than the other, the ants choosing the short branch are the first to reach the food and to start their travel back to the nest.[6] But then, when they must make a decision between the short and the long branch, the higher level of pheromone on the short branch biases their decision in its favor.[7] Therefore, pheromone starts to accumulate faster on the short branch which will eventually be used by the great majority of the ants.

It should be clear by now how real ants have inspired AS and later algorithms: the double bridge was substituted by a graph and pheromone trails by artificial pheromone trails. Also, because we wanted artificial ants to solve problems more

[5]The experiment described was originally executed using a laboratory colony of Argentine ants (*Iridomyrmex humilis*). It is known that these ants deposit pheromone both when leaving and when returning to the nest [50].

[6]In the ACO literature this is often called *differential path length* effect.

[7]A process like this, in which a decision taken at time t increases the probability of making the same decision at time $T > t$ is said to be an *autocatalytic* process. Autocatalytic processes exploit *positive feedback*.

complicate than those solved by real ants, we gave artificial ants some extra capacities, like a memory (used to implement constraints and to allow the ants to retrace their path back to the nest without errors) and the capacity of depositing a quantity of pheromone proportional to the quality of the solution produced (a similar behavior is observed also in some real ants species in which the quantity of pheromone deposited while returning to the nest from a food source is proportional to the quality of the food source found [3]).

In the next section we will see how, starting from AS, new algorithms have been proposed that, although retaining some of the original biological inspiration, are less and less biologically inspired and more and more motivated by the need of making ACO algorithms competitive with or improve over state-of-the-art algorithms. Nevertheless, many aspects of the original Ant System remain: the need for a colony, the role of autocatalysis, the cooperative behavior mediated by artificial pheromone trails, the probabilistic construction of solutions biased by artificial pheromone trails and local heuristic information, the pheromone updating guided by solution quality, and the evaporation of pheromone trail, are present in all ACO algorithms. It is interesting to note that there is one well known algorithm that, although making use in some way of the ant foraging metaphor, cannot be considered an instance of the Ant Colony Optimization metaheuristic. This is HAS-QAP, proposed in [46], where pheromone trails are not used to guide the solution construction phase; on the contrary, they are used to guide modifications of complete solutions in a local search style. This algorithm belong nevertheless to *ant algorithms*, a new class of algorithms inspired by a number of different behaviors of social insects. Ant algorithms are receiving increasing attention in the scientific community (see for example [8, 9, 11, 30]) as a promising novel approach to distributed control and optimization.

4.2 Historical development

As we said, AS was the first example of an ACO algorithm to be proposed in the literature. In fact, AS was originally a set of three algorithms called *ant-cycle*, *ant-density*, and *ant-quantity*. These three algorithms were proposed in Dorigo's doctoral dissertation [29] and first appeared in a technical report [37, 36] that was published a few years later in the IEEE Transactions on Systems, Man, and Cybernetics [38]. Another early publication is [16].

While in *ant-density* and *ant-quantity* the ants updated the pheromone directly after a move from a city to an adjacent one, in *ant-cycle* the pheromone update was only done after all the ants had constructed the tours and the amount of pheromone deposited by each ant was set to be a function of the tour quality. Because *ant-cycle* performed better than the other two variants, it was later called simply Ant System (and in fact, it is the algorithm that we will present in the following subsection), while the other two algorithms were no longer studied.

The major merit of AS, whose computational results were promising but not competitive with other more established approaches, was to stimulate a number of researchers, mostly in Europe, to develop extensions and improvements of its basic ideas so to produce more performing, and often state-of-the-art, algorithms. It is following the successes of this collective undertaking that recently Dorigo and Di Caro [31] made the synthesis effort that took to the definition of the ACO metaheuristic presented in this chapter (see also [32]). In other words, the ACO metaheuristic was defined *a posteriori* with the goal of providing a common characterization of a new class of algorithms and a reference framework for the design of new instances of ACO algorithms.

4.2.1 The first ACO algorithm: Ant System and the TSP

The traveling salesman problem (TSP) is a paradigmatic $\mathcal{NP}$-hard combinatorial optimization problem which has attracted an enormous amount of research effort [55, 58, 74]. The TSP is a very important problem also in the context of Ant Colony Optimization because it is the problem to which the original AS was first applied [36, 29, 38], and it has later often been used as a benchmark to test new ideas and algorithmic variants.

In AS each ant is initially put on a randomly chosen city and has a memory which stores the partial solution it has constructed so far (initially the memory contains only the start city). Starting from its start city, an ant iteratively moves from city to city. When being at a city i, an ant k chooses to go to a still unvisited city j with a probability given by

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} \qquad \text{if } j \in \mathcal{N}_i^k \tag{1}$$

where $\eta_{ij} = 1/d_{ij}$ is a priori available heuristic information, α and β are two parameters which determine the relative influence of pheromone trail and heuristic information, and $\mathcal{N}_i^k$ is the feasible neighborhood of ant k, that is, the set of cities which ant k has not yet visited. Parameters α and β have the following influence on the algorithm behavior. If $\alpha = 0$, the selection probabilities are proportional to $[\eta_{ij}]^\beta$ and the closest cities will more likely be selected: in this case AS corresponds to a classical stochastic greedy algorithm (with multiple starting points since ants are initially randomly distributed on the cities). If $\beta = 0$, only pheromone amplification is at work: this will lead to the rapid emergence of a *stagnation* situation with the corresponding generation of tours which, in general, are strongly suboptimal [29]. (Search stagnation is defined in [38] as the situation where all the ants follow the same path and construct the same solution.)

The solution construction ends after each ant has completed a tour, that is, after each ant has constructed a sequence of length n. Next, the pheromone trails are updated. In AS this is done by first lowering the pheromone trails by a constant factor (this is pheromone evaporation) and then allowing each ant to deposit pheromone on the arcs that belong to its tour:

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \qquad \forall(i,j) \tag{2}$$

where $0 < \rho \leq 1$ is the pheromone trail evaporation rate and m is the number of ants. The parameter ρ is used to avoid unlimited accumulation of the pheromone trails and enables the algorithm to "forget" previously done bad decisions. On arcs which are not chosen by the ants, the associated pheromone strength will decrease exponentially with the number of iterations. $\Delta\tau_{ij}^{k}(t)$ is the amount of pheromone ant k deposits on the arcs; it is defined as

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} 1/L^{k}(t) & \text{if arc } (i,j) \text{ is used by ant } k \\ \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $L^{k}(t)$ is the length of the kth ant's tour. By Equation 3, the shorter the ant's tour is, the more pheromone is received by arcs belonging to the tour.[8] In general, arcs which are used by many ants and which are contained in shorter tours will receive more pheromone and therefore are also more likely to be chosen in future iterations of the algorithm.

4.2.2 Ant System and its extensions

As we said, AS was not competitive with state-of-the-art algorithms for TSP. Researchers then started to extend it to try to improve its performance.

A first improvement, called the *elitist strategy*, was introduced in [29, 38]. It consists in giving the best tour since the start of the algorithm (called T^{gb}, where gb stays for global-best) a strong additional weight. In practice, each time the pheromone trails are updated, those belonging to the edges of the global best tour get an additional amount of pheromone. For these edges Equation 3 becomes:

[8]Note that when applied to symmetric TSPs the arcs are considered to be bidirectional and arcs (i,j) and (j,i) are both updated. This is different for the ATSP, where arcs are directed; an ant crossing arc (i,j) only will update this arc and not the arc (j,i).

$$\Delta\tau_{ij}^{gb}(t) = \begin{cases} e/L^{gb}(t) & \text{if arc } (i,j) \in T^{gb} \\ \\ 0 & \text{otherwise} \end{cases} \tag{3a}$$

The arcs of T^{gb} are therefore reinforced with a quantity of $e \cdot 1/L^{gb}$, where L^{gb} is the length of T^{gb} and e is a positive integer. Note that this type of pheromone update is a first example of daemon action as described in Section 3.3.

Other improvements, described below, were the rank-based version of Ant System (AS$_{rank}$), $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS), and Ant Colony System (ACS). AS$_{rank}$ [14] is in a sense an extension of the elitist strategy: it sorts the ants according to the lengths of the tours they generated and, after each tour construction phase, only the $(w-1)$ best ants and the global-best ant are allowed to deposit pheromone. The rth best ant of the colony contributes to the pheromone update with a weight given by $\max\{0, w-r\}$ while the global-best tour reinforces the pheromone trails with weight w. Equation 2 becomes therefore:

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{r=1}^{w-1}(w-r)\cdot\Delta\tau_{ij}^{r}(t) + w\cdot\Delta\tau_{ij}^{gb}(t) \tag{2a}$$

where $\Delta\tau_{ij}^{r}(t) = 1/L^{r}(t)$ and $\Delta\tau_{ij}^{gb}(t) = 1/L^{gb}$.

ACS [42, 34, 33] improves over AS by increasing the importance of exploitation of information collected by previous ants with respect to exploration of the search space.[9] This is achieved via two mechanisms. First, a strong elitist strategy is used to update pheromone trails. Second, ants choose the next city to move to using a so-called *pseudo-random proportional* rule [34]: with probability q_0 they move to the city j for which the product between pheromone trail and heuristic information is maximum, that is, $j = \arg\max_{j\in\mathcal{N}_i^k}\{\tau_{ij}(t)\cdot\eta_{ij}^{\beta}\}$, while with probability $1-q_0$ they operate a biased exploration in which the probability $p_{ij}^{k}(t)$ is the same as in AS (see Equation 1). The value q_0 is a parameter: when it is set to a value close to 1, as it is the case in most ACS applications, exploitation is favored over exploration. Obviously, when $q_0 = 0$ the probabilistic decision rule becomes the same as in AS.

As we said, pheromone updates are performed using a strong elitist strategy: only the ant that has produced the best solution is allowed to update pheromone trails, according to a pheromone trail update rule similar to that used in AS:

[9] ACS was an offspring of Ant-Q [41], an algorithm intended to create a link between reinforcement learning [87] and Ant Colony Optimization. Computational experiments have shown that some aspects of Ant-Q, in particular the pheromone update rule, could be strongly simplified without affecting performance. It is for this reason that Ant-Q was abandoned in favor of the simpler and equally good ACS.

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \rho\cdot\Delta\tau_{ij}^{best}(t) \qquad (4)$$

The best ant can be the *iteration-best* ant, that is, the best in the current iteration, or the *global-best* ant, that is, the ant that made the best tour from the start of the trial.

Finally, ACS differs form previous ACO algorithms also because ants update the pheromone trails while building solutions (like it was done in *ant-quantity* and in *ant-density*. In practice ACS ants "eat" some of the pheromone trail on the edges they visit. This has the effect of decreasing the probability that a same path is used by all the ants (i.e., it favors exploration, counterbalancing this way the other two above-mentioned modifications that strongly favor exploitation of the collected knowledge about the problem). ACS has been made more performing also by the addition of local search routines that take the solution generated by ants to their local optimum just before the pheromone update.

$\mathcal{MM}$AS [82, 84, 85] introduces upper and lower bounds to the values of the pheromone trails, as well as a different initialization of their values. In practice, in $\mathcal{MM}$AS the allowed range of the pheromone trail strength is limited to the interval $[\tau_{\min}, \tau_{\max}]$, that is, $\tau_{\min} \leq \tau_{ij} \leq \tau_{\max}\ \forall\tau_{ij}$, and the pheromone trails are initialized to the upper trail limit, which causes a higher exploration at the start of the algorithm. Also, like in ACS, in $\mathcal{MM}$AS only the best ant (the global-best or the iteration-best ant) is allowed to add pheromone after each algorithm iteration. Computational results have shown that best results are obtained when pheromone updates are performed using the global-best solution with increasing frequency during the algorithm execution. Similar to ACS, also $\mathcal{MM}$AS often exploits local search to improve its performance.

4.2.3 Applications to dynamic problems

The application of ACO algorithms to dynamic problems, that is, problems whose characteristics change while being solved, is the most recent major development in the field. The first such application [77] concerned routing in circuit-switched networks (e.g., classical telephone networks). The proposed algorithm, called ABC, was demonstrated on a simulated version of the British Telecom network. The main merit of ABC was to stimulate the interest of ACO researchers in dynamic problems. In fact, only rather limited comparisons were made between ABC and state-of-the-art algorithms for the same problem so that it is not possible to judge on the quality of the results obtained.

A very successful application of ACO to dynamic problems is the AntNet algorithm, proposed by Di Caro and Dorigo [23, 25, 27, 24] and discussed in Section 5. AntNet was applied to routing in packet-switched networks (e.g., the Internet).

It contains a number of innovations with respect to AS and it was experimentally shown to outperform a whole set of state-of-the-art algorithms on numerous benchmark problems.

5 Examples of applications

The versatility and the practical use of the ACO metaheuristic for the solution of combinatorial optimization problems is best illustrated via example applications to a number of different problems.

The ACO application to the TSP has already been presented in the previous section. Here, we additionally discuss applications to three $\mathcal{NP}$-hard optimization problems, the single machine total weighted tardiness problem (SMTWTP), the generalized assignment problem (GAP), and the set covering problem (SCP). We have chosen these problems to make the application examples as comprehensive as possible with respect to different ways of representing solutions. While the TSP and the SMTWTP are permutation problems, that is, solutions are represented as permutations of solution components, solutions in the GAP are assignments of tasks to agents and in the SCP a solution is represented as a subset of the available solution components.

Applications of ACO to dynamic problems focus mainly on routing in data networks. As an example we present AntNet [25], a very successful algorithm for packet-switched networks like the Internet.

Example 1: The single machine total weighted tardiness scheduling problem (SMTWTP)

In the SMTWTP n jobs have to be processed sequentially without interruption on a single machine. Each job has a processing time p_j, a weight w_j, and a due date d_j associated and all jobs are available for processing at time zero. The tardiness of job j is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is its completion time in the current job sequence. The goal in the SMTWTP is to find a job sequence which minimizes the sum of the weighted tardiness given by $\sum_{i=1}^{n} w_i \cdot T_i$.

For the ACO application to the SMTWTP, the set of components $\mathcal{C}$ is the set of all jobs. As in the TSP case, the states of the problem are all possible partial sequences. In the SMTWTP case we do not have explicit costs associated with the connections because the objective function contribution of each job depends on the partial solution constructed so far.

The SMTWTP was attacked in [21] using ACS (ACS-SMTWTP). In ACS-SMTWTP, each ant starts with an empty sequence and then iteratively appends an unscheduled job to the partial sequence constructed so far. Each ant chooses the next job using the *pseudo-random-proportional* action choice rule, where the at each step the feasible neighborhood $\mathcal{N}_i^k$ of ant k is formed by the still unscheduled jobs. Pheromone trails are defined as follows: τ_{ij} refers to the desirability of scheduling job j at position i. This definition of the pheromone trails is, in fact, used in most ACO application to scheduling problems [2, 21, 66, 80]. Concerning the heuristic information, in [21] the use of three priority rules allowed to define three different types of heuristic information for the SMTWTP. The investigated priority rules were: (i) the earliest due date rule (EDD), which puts the jobs in non-decreasing order of the due dates d_j, (ii) the modified due date rule (MDD) which puts the jobs in non-decreasing order of the modified due dates given by $mdd_j = \max\{C + p_j, d_j\}$ [2], where C is the sum of the processing times of the already sequenced jobs, and (iii) the apparent urgency rule (AU) which puts the jobs in non-decreasing order of the apparent urgency [70], given by $au_j = (w_j/p_j) \cdot \exp(-(\max\{d_j - C_j, 0\})/k\overline{p})$, where k is a parameter of the priority rule. In each case, the heuristic information was defined as $\eta_{ij} = 1/h_j$, where h_j is either d_j, mdd_j, or au_j, depending on the priority rule used.

The global and the local pheromone update is done as in the standard ACS as described in Section 4.2, where in the global pheromone update T^{gb} is the total weighted tardiness of the global best solution.

In [21], ACS-SMTWTP was combined with a powerful local search algorithm. The final ACS algorithm was tested on a benchmark set available from ORLIB at `http://www.ms.ic.ac.uk/info.html`. Within the computation time limits given[10] ACS reached a very good performance and could find in each single run the optimal or best known solutions on all instances of the benchmark set. For more details on the computational results we refer to [21].

Example 2: The generalized assignment problem (GAP)

In the GAP a set of tasks $\mathcal{I}, i = 1, \ldots, n$ has to be assigned to a set of agents $\mathcal{J}, j = 1, \ldots, m$. Each agent j has only a limited capacity a_j and each task i consumes, when assigned to agent j, a quantity b_{ij} of the agent's capacity. Also, the cost d_{ij} of assigning task i to agent j is given. The objective then is to find a feasible task assignment with minimal cost.

Let y_{ij} be one if task i is assigned to agent j and zero otherwise. Then the GAP can formally be defined as

[10] The maximum time for the largest instances was 20 min on a 450MHz Pentium III PC with 256 MB RAM. Programs were written in C++ and the PC was run under Red Hat Linux 6.1.

$$\min f(y) = \sum_{j=1}^{m} \sum_{i=1}^{n} d_{ij} \cdot y_{ij} \tag{5}$$

subject to

$$\sum_{i=1}^{n} b_{ij} \cdot y_{ij} \leq a_j \quad j = 1, \ldots, m \tag{6}$$

$$\sum_{j=1}^{n} y_{ij} = 1 \quad i = 1, \ldots, n \tag{7}$$

$$y_{ij} \in \{0, 1\} \quad i, j = 1, \ldots, n \tag{8}$$

The constraints 6 implement the limited resource capacity of the agents, while constraints 7 and 8 impose that each task is assigned to exactly one agent and that a task cannot be split among several agents.

The GAP can easily be cast into the framework of the ACO metaheuristic. The problem can be represented by a graph in which the set of components comprises the set of tasks and agents, that is $\mathcal{C} = \mathcal{I} \cup \mathcal{J}$ and the set of connections fully connect the graph. Each assignment, which consists of n couplings (i, j) of tasks and agents, corresponds to an ant's walk on this graph. Such a walk has to observe the constraints 7 and 8 to obtain an assignment. One particular way of generating such an assignment is by an ant's walk which iteratively switches from task nodes (nodes in the set $\mathcal{J}$) to agent nodes (nodes in the set $\mathcal{I}$) without repeating any task node but using possibly several times an agent node (several tasks may be assigned to a same agent).

At each step of the construction process, an ant has to make one of the following two basic decisions [19]: (i) it has to decide which task to assign next and (ii) it has to decide to which agent a chosen task should be assigned. Pheromone trail and heuristic information can be associated with both tasks. With respect to the first step the pheromone information can be used to learn an appropriate assignment order of the tasks, that is τ_{ij} gives the desirability of assigning task j directly after task i, while the pheromone information in the second step is associated with the desirability of assigning a task to a specific agent.

For simplicity let us consider an approach in which the tasks are assigned in a random order. Then, at each step a task has to be assigned to an agent. Intuitively, it is better to assign tasks to agents such that small assignment costs are incurred and that the agent needs only a relatively small amount of its available resource to perform the task. Hence, one possible heuristic information is $\eta_{ij} = a_j / d_{ij} b_{ij}$ and a probabilistic selection of the assignments can follow the AS probabilistic rule (Equation 1) or the pseudo-random proportional rule of ACS. Yet, a complication in the construction process is that the GAP involves resource capacity constraints and, in fact, for the GAP no guarantee is given that an ant will construct a feasible solution which obeys the resource constraints given by Equation 6. In fact, to have a bias towards generating feasible solutions, the resource constraints should be taken into account in the definition of $\mathcal{N}_i^k$, the feasible neighborhood of ant k. For the GAP, we define $\mathcal{N}_i^k$ to consist of all those agents to which the task i can be assigned without violating the agents' resource capacity. If no agent can meet the task's resource requirement, we are forced to build an infeasible solution and in this case we simply can choose $\mathcal{N}_i^k$ as the set of all agents. Infeasibilites can then be handled, for example, by assigning penalties proportional to the amount of resource violations.

A first application of $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [85], to the GAP was presented in [73]. The approach shows some particularities, like the use of a single ant and the lack of any heuristic information. The infeasibility of solutions is only treated in the pheromone update: the amount of pheromone deposited by an ant is set to a high value if the solution it generated is feasible, to a low value if it is infeasible. These values are constants independent of the solution quality. Additionally, $\mathcal{MM}$AS was coupled with a local search based on tabu search and ejection chain elements [49] and it obtained very good performance on benchmark instances available at ORLIB.

Example 3: The set covering problem (SCP)

In the set covering problem (SCP) we are given a $m \times n$ matrix $A = (a_{ij})$ in which all the matrix elements are either zero or one. Additionally, each column is given a non-negative cost b_j. We say that a column j covers a row i if $a_{ij} = 1$. The goal in the SCP is to choose a subset of the columns of minimal weight which covers every row. Let $\mathcal{J}$ denote a subset of the columns and y_j is a binary variable which is one, if $j \in \mathcal{J}$, and zero otherwise. The SCP can be defined formally as follows.

$$\min f(y) = \sum_{j=1}^{n} b_j \cdot y_j \tag{9}$$

subject to

$$\sum_{j=1}^{n} a_{ij} \cdot y_j \geq 1 \quad i = 1, \ldots, m \tag{10}$$

$$y_j \in \{0, 1\} \quad j = 1, \ldots, n \tag{11}$$

The constraints 10 enforce that each row is covered by at least one column.

ACO can be applied in a very straightforward way to the SCP. The columns are chosen as the solution components and have associated a cost and a pheromone trail. The Ω constraints say that each column can be visited by an ant once and only once and that a final solution has to cover all rows. A walk of an ant over the graph representation corresponds to the iterative addition of columns to the partial solution obtained so far. Each ant starts with an empty solution and adds columns until a cover is completed. A pheromone trail τ_i and a heuristic information η_i are associated to each column i a. A column to be added is chosen with probability

$$p_i^k(t) = \frac{[\tau_i(t)] \cdot [\eta_i]^\beta}{\sum_{l \in \mathcal{N}^k} [\tau_l(t)] \cdot [\eta_l]^\beta} \qquad \text{if } i \in \mathcal{N}^k \tag{12}$$

where $\mathcal{N}^k$ is the feasible neighborhood of ant k which consists of all columns which cover at least one still uncovered row. The heuristic information η_i can be chosen in several different ways. For example, a simple static information could be used, taking into account only the column cost: $\eta_i = 1/b_i$. A more sophisticate approach would be to consider the total number of rows d_i covered by a column i and to set $\eta_i = d_i/b_i$. The heuristic information could also be made dependent on the partial solution y_k of an ant k. In this case, it can be defined as $\eta_i = e_i/b_i$, where e_i is the so-called *cover value*, that is, the number of additional rows covered when adding column i to the current partial solution. In other words, the heuristic information measures the unit cost of covering one additional row.

An ant ends the solution construction when all rows are covered. In a post-optimization step, an ant can remove redundant columns—columns that only cover rows which are also covered by a subset of other columns in the final solution—or apply some additional local search to improve solutions. The pheromone update can be done again in a standard way like it has already been described in the previous sections.

When applying ACO to the SCP we have two main differences with the previously presented applications: (i) pheromone trails are associated only to components and, (ii) the length of the ant's walks (corresponding to the lengths of the sequences) may differ among the ants and, hence, the solution construction only ends when all the ants have terminated their corresponding walks.

There exist already some first approaches applying ACO to the SCP. In [1], ACO has been used only as a construction algorithm and the approach has only been tested on some small SCP instances. A more recent article [53] applies Ant System to the SCP and uses techniques to remove redundant columns and local search to improve solutions. Good results are obtained on a large set of benchmark instances taken from ORLIB, but the performance of Ant System could not fully reach that of the best performing algorithms for the SCP.

Example 4: AntNet for network routing applications

Given a graph representing a telecommunications network, the problem solved by AntNet is to find the minimum cost path between each pair of nodes of the graph. It is important to note that, although finding a minimum cost path on a graph is an easy problem (it can be efficiently solved with algorithms with polynomial worst case complexity [5]), it becomes extremely difficult when the costs on the edges are time-varying stochastic variables. This is the case of routing in packet-switched networks, the target application for AntNet.

Here we briefly describe a simplified version of AntNet (the interested reader should refer to [25], where the AntNet approach to routing is explained and evaluated in detail). As we said, in AntNet each ant searches for a minimum cost path between a given pair of nodes of the network. To this goal, ants are launched from each network node towards randomly selected destination nodes. 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 and of local pheromone and heuristic information (very much like to what happened, for example, in AS).

Differently from AS, where pheromone trails are associated to edges, in AntNet pheromone trails are associated to arc-destination pairs. That is, each directed arc (i, j) has $n - 1$ trail values $\tau_{ijd} \in [0, 1]$ associated, where n is the number of nodes in the graph associated to the routing problem; in general, $\tau_{ijd} \neq \tau_{jid}$. In other words, there is one trail value τ_{ijd} for each possible destination node d an ant located in node i can have. Each arc has also associated an heuristic value $\eta_{ij} \in [0, 1]$ independent of the final destination. The heuristic values can be set for example to the values $\eta_{ij} = 1 - q_{ij} / \sum_{l \in \mathcal{N}_i} q_{il}$, 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: links with a shorter queue have a higher heuristic value.

In AntNet, as well as in most other implementations of ACO algorithms for routing problems [77, 88], the daemon component (see Figure 5) is not present.

Ants choose their way probabilistically, using as probability a functional composition of the local pheromone trails τ_{ijd} and of the heuristic values η_{ij}. While building the path to their destinations, ants move using the same link queues as data and experience the same delays as data packets. Therefore, 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 quality of the path they built. The overall quality of a path is 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 quality of the path it built. To deposit pheromone, after reaching its destination node, the ant moves back to its source node along the same path but backward and using high priority queues, to allow a fast propagation of the collected information. The pheromone trail intensity of each arc l_{ij} the ant used while it was moving from s to d is increased as follows: $\tau_{ijd}(t) \leftarrow \tau_{ijd}(t)+\Delta\tau^k(t)$. After the pheromone trail on the visited arcs has been updated, the pheromone value of all the outgoing connections of the same node i, relative to the destination d, evaporates in such a way that the pheromone values are normalized and can continue to be usable as probabilities: $\tau_{ijd}(t) \leftarrow \tau_{ijd}(t)/(1 + \Delta\tau^k(t))$, $\forall j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is the set of neighbors of node i.

AntNet was compared with many state-of-the-art algorithms on a large set of benchmark problems under a variety of traffic conditions. It always compared favorably with competing approaches and it was shown to be very robust with respect to varying traffic conditions and parameter settings. More details on the experimental results can be found in [25].

5.1 Applications of the ACO metaheuristic

ACO has recently raised a lot of interest in the scientific community. There are now available numerous successful implementations of the ACO metaheuristic applied to a wide range of different combinatorial optimization problems. These applications comprise two main application fields.

- $\mathcal{NP}$-hard problems, for which the best known algorithms have exponential time worst case complexity. For these problems very often ACO algorithms are coupled with extra capabilities, as implemented by the daemon actions, like a problem specific local optimizer which takes the ants' solutions to a local optimum.

- 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 be physically available (like in networks problems), but its properties, like the costs of components or of connections, 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 the costs increases and/or the knowledge about the variation process diminishes.

Table 1. Current applications of ACO algorithms. Applications are listed by class of problems and in chronological order.

Problem name	*Authors*	*Algorithm name*	*Year*	*Main references*
Traveling salesman	Dorigo, Maniezzo & Colorni	AS	1991	[29, 37, 38]
	Gambardella & Dorigo	Ant-Q	1995	[41]
	Dorigo & Gambardella	ACS & ACS-3-opt	1996	[33, 34, 42]
	Stützle & Hoos	$\mathcal{MM}$AS	1997	[84, 82, 85]
	Bullnheimer, Hartl & Strauss	AS_{rank}	1997	[14]
	Cordón, et al.	BWAS	2000	[18]
Quadratic assignment	Maniezzo, Colorni & Dorigo	AS-QAP	1994	[65]
	Gambardella, Taillard & Dorigo	HAS-QAP[a]	1997	[46]
	Stützle & Hoos	$\mathcal{MM}$AS-QAP	1997	[79, 85]
	Maniezzo	ANTS-QAP	1998	[62]
	Maniezzo & Colorni	AS-QAP[b]	1999	[64]
Scheduling problems	Colorni, Dorigo & Maniezzo	AS-JSP	1994	[17]
	Stützle	AS-FSP	1997	[80]
	Bauer et *al.*	ACS-SMTTP	1999	[2]
	den Besten, Stützle & Dorigo	ACS-SMTWTP	1999	[21]
	Merkle, Middendorf & Schmeck	ACO-RCPS	2000	[66]
Vehicle routing	Bullnheimer, Hartl & Strauss	AS-VRP	1997	[12, 13]
	Gambardella, Taillard & Agazzi	HAS-VRP	1999	[45]
Connection-oriented network routing	Schoonderwoerd *et al.*	ABC	1996	[77, 76]
	White, Pagurek & Oppacher	ASGA	1998	[89]
	Di Caro & Dorigo	AntNet-FS	1998	[26]
	Bonabeau *et al.*	ABC-smart ants	1998	[10]
Connection-less network routing	Di Caro & Dorigo	AntNet & AntNet-FA	1997	[23, 25, 28]
	Subramanian, Druschel & Chen	Regular ants	1997	[86]
	Heusse *et al.*	CAF	1998	[54]
	van der Put & Rothkrantz	ABC-backward	1998	[88]
Sequential ordering	Gambardella & Dorigo	HAS-SOP	1997	[43, 44]
Graph coloring	Costa & Hertz	ANTCOL	1997	[19]
Shortest common supersequence	Michel & Middendorf	AS-SCS	1998	[67, 68]
Frequency assignment	Maniezzo & Carbonaro	ANTS-FAP	1998	[63]
Generalized assignment	Ramalhinho Lourenço & Serra	MMAS-GAP	1998	[73]
Multiple knapsack	Leguizamón & Michalewicz	AS-MKP	1999	[59]
Optical networks routing	Navarro Varela & Sinclair	ACO-VWP	1999	[71]
Redundancy allocation	Liang & Smith	ACO-RAP	1999	[60]
Constraint satisfaction	Solnon	Ant-P-solver	2000	[78]

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

These applications are summarized in Table 1. In some of these applications, ACO algorithms have obtained world class performance, which is the case, for example, for quadratic assignment [62, 85], sequential ordering [43, 44], vehicle routing [45], scheduling [21, 66] or packet-switched network routing [25].

6 Discussion of application principles

Despite being a rather recent metaheuristic, ACO algorithms have already been applied to a large number of different combinatorial optimization problems. Based on this experience, we have identified some basic issues which play an important role in several of these applications. These are discussed in the following.

6.1 Pheromone trails definition

A first, very important choice when applying ACO is the definition of the intended meaning of the pheromone trails. Let us explain this issue with an example. When applying ACO to the TSP, the standard interpretation of a pheromone trail τ_{ij}, used in all available ACO applications to the TSP, is that it refers to the desirability of visiting city j directly after a city i. That is, it provides some information on the desirability of the relative positioning of city i and j. Yet, another possibility, not working so well in practice, would be to interpret τ_{ij} as the desirability of visiting city i as the jth city in a tour, that is, the desirability of the absolute positioning. Differently, when applying ACO to the SMTWTP (see Section 5) better results are obtained when using the absolute position interpretation of the pheromone trails, where τ_{ij} is the desirability of putting job j on the ith position [20]. This is intuitively due to the different role that permutations have in the two problems. In the TSP, permutations are cyclic, that is, only the relative order of the solution components is important and a permutation $\pi = (1\ 2\ \ldots\ n)$ has the same tour length as the permutation $\pi' = (n\ 1\ 2\ \ldots\ n-1)$—it represents the same tour. Therefore, a relative position based pheromone trail is the appropriate choice. On the contrary, in the SMTWTP (as well as in many other scheduling problems), π and π' represent two different solutions with most probably very different costs. Hence, in the SMTWTP the absolute position based pheromone trails are a better choice. Nevertheless, it should be noted that, in principle, both choices are possible, because any solution of the search space can be generated with both representations.

The definition of the pheromone trails is crucial and a poor choice at this stage of the algorithm design will probably result in poor performance. Fortunately, for many problems the intuitive choice is also a very good one, as it was the case for the previous example applications. Yet, sometimes the use of the pheromones can be somewhat more involved, which is, for example, the case in the ACO application to the shortest common supersequence problem [68].

6.2 Balancing exploration and exploitation

Any high performing metaheuristic algorithm has to achieve an appropriate balance between the exploitation of the search experience gathered so far and the exploration of unvisited or relatively unexplored search space regions. In ACO several ways exist of achieving such a balance, typically through the management of the pheromone trails. In fact, the pheromone trails induce a probability distribution over the search space and determine which parts of the search space are effectively sampled, that is, in which part of the search space the constructed solutions are located with higher frequency. Note that, depending on the distribution of the pheromone trails, the sampling distribution can vary from a uniform distribution to a degenerate distribution which assigns a probability of one to a single solution and zero probability to all the others. In fact, this latter situation corresponds to the stagnation of the search as explained on page 13.

The simplest way to exploit the ants' search experience is to make the pheromone update a function of the solution quality achieved by each particular ant. Yet, this bias alone is often too weak to obtain good performance, as was shown experimentally on the TSP [82, 85]. Therefore, in many ACO algorithms (see Section 4) an *elitist strategy* whereby the best solutions found during the search strongly contribute to pheromone trail updating, was introduced.

A stronger exploitation of the "learned" pheromone trails can also be achieved during solution construction by applying the pseudo-random proportional rule of Ant Colony System, as explained in Section 4.2.2.

Search space exploration is achieved in ACO primarily by the ants' randomized solution construction. Let us consider for a moment an ACO algorithm that does not use heuristic information (this can be easily achieved by setting $\beta = 0$). In this case, the pheromone updating activity of the ants will cause a shift from the initial uniform sampling of the search space to a sampling focused on specific search space regions. Hence, exploration of the search space will be higher in the initial iterations of the algorithm, and will decrease as the computation goes on. Obviously, attention must be put to avoid that a too strong focus on apparently good regions of the search space causes the ACO algorithm to enter a stagnation situation.

There are several ways to try to avoid such stagnation situations, maintaining this way a reasonable level of exploration of the search space. For example, in ACS the ants use a local pheromone update rule during the solution construction to make the path they have taken less desirable for following ants and, thus, to diversify search. $\mathcal{MM}$AS introduces an explicit lower limit on the pheromone trail level so that a minimal level of exploration is always guaranteed. $\mathcal{MM}$AS also uses a reinitialization of the pheromone trails, which is a way of enforcing search space exploration. Experience has shown that pheromone trail reinitialization, when combined with appropriate choices for the pheromone trail update [85] can be very useful to refocus the search on a different search space region.

Finally, an important, though somewhat neglected, role in the balance of exploration and exploitation is that of the parameters α and β, which determine the relative influence of pheromone trail and heuristic information. Consider first the influence of the parameter α. For $\alpha > 0$, the larger the value of α the stronger the exploitation of the search experience, for $\alpha = 0$ the pheromone trails are not taken into account at all, and for $\alpha < 0$ the most probable choices done by the ants are those that are less desirable from the point of view of pheromone trails. Hence, varying α could be used to shift from exploration to exploitation and vice versa. The parameter β determines the influence of the heuristic information in a similar way. In fact, systematic variations of α and β could, similarly to what is done in the strategic oscillations approach [48], be part of simple and useful strategies to balance exploration and exploitation.

6.3 ACO and local search

In many applications to $\mathcal{NP}$-hard combinatorial optimization problems like the TSP, the QAP, or the VRP, ACO algorithms perform best when coupled with local search algorithms (which is, in fact, a particular type of daemon action of the ACO metaheuristic). Local search algorithms locally optimize the ants' solutions and these locally optimized solutions are used in the pheromone update.

The use of local search in ACO algorithms can be very interesting as the two approaches are complementary. In fact, ACO algorithms perform a rather coarse-grained search, and the solutions they produce can then be locally optimized by an adequate local search algorithm. The coupling can therefore greatly improve the quality of the solutions generated by the ants.

On the other side, generating initial solutions for local search algorithms is not an easy task. For example, it has been shown that, for most problems, repeating local searches from randomly generated initial solutions is not efficient (see for example [55]). In practice, ants probabilistically combine solution components which are part of the best locally optimal solutions found so far and generate new, promising initial solutions for the local search. Experimentally, it has been found that such a combination of a probabilistic, adaptive construction heuristic with local search can yield excellent results [6, 34, 84].

Despite the fact that the use of local search algorithms has been shown to be crucial for achieving best performance in many ACO applications, it should be noted that ACO algorithms also show very good performance where local search algorithms cannot be applied easily. One such example are the network routing applications described in Section 5 or the shortest common supersequence problem [68].

6.4 Importance of heuristic information

The possibility of using heuristic information to direct the ants' probabilistic solution construction is important because it gives the possibility of exploiting problem specific knowledge. This knowledge can be available a priori (this is the most frequent situation in static problems) or at run-time (this is the typical situation in dynamic problems). In static problems, the heuristic information η is computed once at initialization time and then is the same throughout the whole algorithm's run. An example is the use, in the TSP applications, of the length d_{ij} of the arc connecting cities i and j to define the heuristic information $\eta_{ij} = 1/d_{ij}$. Static heuristic information has the advantage that (i) it is easy to compute, (ii) it has to be computed only once at initialization time, and (iii) in each iteration of the ACO algorithm, a table can be pre-computed with the values of $\tau_{ij}(t) \cdot \eta_{ij}^{\beta}$, which can result in a very significant saving of computation time. In the dynamic case, the heuristic information does depend on the partial solution constructed so far and therefore, has to be computed at each step of an ant's walk. This determines a higher computational cost that may be compensated by the higher accurateness of the computed heuristic values. For example, in the ACO application to the SMTWTP we found that the use of dynamic heuristic information based on the MDD or the AU heuristics (see Section 5) resulted in a better overall performance.

Another way of computing heuristic information was introduced in the ANTS algorithm [61], where it is computed using lower bounds on the solution cost of the completion of an ant's partial solution. This method has the advantage that it allows to exclude certain choices because they lead to solutions that are worse than the best found so far. It allows therefore the combination of knowledge on the calculation of lower bounds from mathematical programming with the ACO paradigm. Nevertheless, a disadvantage is that the computation of the lower bounds can be time consuming, especially because they have to be calculated at each single step by each ant.

Finally, it should be noted that while the use of heuristic information is rather important for a generic ACO algorithm, its importance is strongly reduced if local search is used to improve solutions. This is due to the fact that local search takes into account the cost information to improve solutions in a more direct way. Luckily, this means that ACO algorithms can achieve, in combination with a local search algorithm, very good performance also for problems for which it is difficult to define a priori a very informative heuristic information.

6.5 Number of ants

Why to use a colony of ants 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 is often 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 many shortest path problems in parallel (one between each pair of nodes) and a colony of ants must be used for each of these problems.

On the other hand, in the case of combinatorial optimization problems the differential length effect is not exploited and the use of m ants, $m > 1$, that build r solutions each (i.e., the ACO algorithm is run for r iterations) could be equivalent to the use of one ant that generates $m \cdot r$ solutions. Nevertheless, in this case theoretical results on the convergence of some specific ACO algorithms, which will be presented in Section 7, as well as experimental evidence suggest that ACO algorithms perform better when the number m of ants is set to a value $m > 1$.

In general, the best value for m is a function of the particular ACO algorithm chosen as well as of the class of problems being attacked, and most of the times it must be set experimentally. Fortunately, ACO algorithms seem to be rather robust to the actual number of ants used.

6.6 Candidate lists

One possible difficulty encountered by ACO algorithms is when they are applied to problems with big-sized neighborhood in the solution construction. In fact, an ant that visits a state with a big-sized neighborhood has a huge number of possible moves among which to choose. Possible problems are that the solution construction is significantly slowed down and that the probability that many ants visit the same state is very small. Such a situation can occur, for example, in the ACO application to large TSPs or large SCPs.

In such situations, the above-mentioned problem can be considerably reduced by the use of candidate lists. Candidate lists comprise a small set of promising neighbors of the current state. They are created using a priori available knowledge on the problem, if available, or dynamically generated information. Their use allows ACO algorithms to focus on the more interesting components, strongly reducing the dimension of the search space.

As an example, consider the ACO application to the TSP. For the TSP it is known that very often optimal solutions can be found within a surprisingly small subgraph consisting of all the cities and of those edges that connect each city to only a few of its nearest neighbors. For example, for the TSPLIB instance `pr2392.tsp` with 2392 cities an optimal solution can be found within a subgraph of the 8 nearest neighbors [74]. This knowledge can be used for defining candidate lists, which was first done in the context of ACO algorithms in [42]. There a candidate list included for each city its cl nearest neighbors. During solution construction an ant tries to choose the city to move to only among the cities in the candidate list. Only if all these cities have already been visited, the ant can choose among the other cities.

So far, in ACO algorithms the use of candidate lists or similar approaches is still rather unexplored. Inspiration from other techniques like Tabu Search [49] or GRASP [40], where strong use of candidate lists is made, could be useful for the development of effective candidate list strategies for ACO.

7 Other developments

7.1 Proving convergence

The simplest stochastic optimization algorithm is random search. Besides simplicity, random search has also the nice property that it guarantees that it will find, sooner or later, the optimal solution to your problem. Unfortunately, it is very inefficient. Stochastic optimization algorithms can be seen as ways of biasing random search so to make it more efficient. Unfortunately, once a stochastic algorithm is biased, it is no longer guaranteed that it will, at some point, find the optimal solution. In fact, the bias could simply rule out this possibility. It is therefore interesting to have convergence proofs that assure you that this does not happen.

The problem of convergence to the optimal solution of a generic ACO algorithm is open (and it will most probably remain so, given the generality of the ACO metaheuristic). Nevertheless, it should be noted that in some cases (e.g., Stützle's $\mathcal{MM}$AS [84]) we can be sure that the optimal solution does not become unreachable after the repetitive application of the algorithm. In fact, in the case of $\mathcal{MM}$AS the bound on the minimum value of pheromone trails makes it impossible that the probability of some moves becomes null, so that all solutions continue to remain reachable during the algorithm run.

Gutjahr [52] has recently proved convergence to the optimal solution for a particular ACO algorithm he calls Graph-based Ant System (GBAS). GBAS is very similar to AS: the only important difference between AS and GBAS is that GBAS puts some additional constraints on how the pheromone values should be updated.

In fact, in GBAS updates are allowed only when an improving solution is found. Gutjahr's convergence proof states that, given a small $\epsilon > 0$ and for fixed values of some algorithm parameters, after a number of cycles $t \geq t_0$ the algorithm will find the optimal solution with probability $P_t \geq 1 - \epsilon$, where $t_0 = f(\epsilon)$. This is an important result, and may open up the door to further convergence results for other instances of ACO algorithms.

Recently, Rubinstein [75] has introduced an algorithm called Cross-Entropy (CE) method that, while being very similar to AS, seems to have some nice properties (like a limited number of parameters and the possibility of determining their optimal value). It still has to be seen, however, whether the CE method will have a performance similar to that of ACO algorithms, or if, as it is more reasonable to expect, it will be necessary to renounce to its simplicity, which simplifies its theoretical study, to obtain state-of-the-art performance.

7.2 Parallel implementations

The very nature of ACO algorithms lends them to be parallelized in the data or population domains. In particular, many parallel models used in other population-based algorithms can be easily adapted to the ACO structure. Most parallelization strategies can be classified into *fine-grained* and *coarse-grained* strategies. Characteristic of fine-grained parallelization is that very few individuals are assigned to one processors and that frequent information exchange among the processors takes place. On the contrary, in coarse grained approaches larger subpopulations or even full populations are assigned to single processors and information exchange is rather rare. We refer, for example, to [35] for a review.

Fine-grained parallelization schemes have been investigated with parallel versions of AS for the TSP on the Connection Machine CM-2 adopting the approach of attributing a single processing unit to each ant [7]. Experimental results showed that communication overhead can be a major problem with this approach on fine grained parallel machines, since ants end up spending most of their time communicating to other ants the modifications they did to pheromone trails. Similar negative results have also been reported in [15].

As shown by several researches [7, 15, 57, 69, 81], coarse grained parallelization schemes are much more promising for ACO. When applied to ACO, coarse grained schemes run p subcolonies in parallel, where p is the number of available processors. Information among the subcolonies is exchanged at certain intervals. For example, in the Partially Asynchronous Parallel Implementation (PAPI) of Bullnheimer, Kotsis and Strauss [15], for which high speed-up was observed, the subcolonies exchange pheromone information every fixed number of iterations done by each subcolony. Krüger, Merkle and Middendorf [57] investigated which information should be exchanged between the m subcolonies and how this information should be used to update the subcolony's trail information. Their results showed that it was better to exchange the best solutions found so far and to use them in the pheromone update than to exchange complete pheromone matrices for modifications of the pheromone matrix of a local subcolony. Middendorf, Reischle, and Schmeck [69] investigate different ways of exchanging solutions among m ant colonies. They consider an exchange of the global best solutions among all colonies and local exchanges based on a virtual neighborhood among subcolonies which corresponds to a directed ring. Their main observation was that the best solutions, with respect to computing time and solution quality, were obtained by limiting the information exchange to a local neighborhood of the colonies. In the extreme case, no communication is done among the subcolonies, resulting in parallel independent runs of an algorithm. This is the easiest way to parallelize randomized algorithms and can be very effective as has been shown with computational results presented by Stützle [81].

8 Conclusions

The field of ACO algorithms is very lively, as testified for example by the successful biannual workshop (ANTS – From Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms; http://iridia.ulb.ac.be/˜ants/) where researchers meet to discuss the properties of ACO and other ant algorithms [8, 9, 30], both theoretically and experimentally.

From the theory side, researchers are trying either to extend the scope of existing theoretical results [51], or to find principled ways to set parameters values [75].

From the experimental side, most of the current research is in the direction of increasing the number of problems that are successfully solved by ACO algorithms, including real-word, industrial applications [39].

Currently, the great majority of problems attacked by ACO are static and well-defined combinatorial optimization problems, that is, problems for which all the necessary information is available and does not change during problem solution. For this kind of problems ACO algorithms must compete with very well established algorithms, often specialized for the given problem. Also, very often the role played by local search is extremely important to obtain good results (see for example [44]). Although rather successful on these problems, we believe that ACO algorithms will really evidentiate their strength when they will be systematically applied to "ill-structured" problems for which it is not clear how to apply local search, or to highly dynamic domains with only local information available. A first step in this direction has already been done with the application to telecommunications networks routing, but more research is necessary.

9 Acknowledgments

Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Senior Research Associate. This work was partially supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] D.A. Alexandrov and Y.A. Kochetov. The behavior of the ant colony algorithm for the set covering problem. In K. Inderfurth, G. Schwödiauer, W. Domschke, F. Juhnke, P. Kleinschmidt, and G. Wäscher, editors, *Operations Research Proceedings 1999*, pages 255–260. Springer Verlag, 2000.

[2] A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss. An ant colony optimization approach for the single machine total tardiness problem. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, pages 1445–1450. IEEE Press, Piscataway, NJ, 1999.

[3] R. Beckers, J.-L. Deneubourg, and S. Goss. Modulation of trail laying in the ant *Lasius niger* (hymenoptera: Formicidae) and its role in the collective selection of a food source. *Journal of Insect Behavior*, 6(6):751–759, 1993.

[4] R. Bellman, A. O. Esogbue, and I. Nabeshima. *Mathematical Aspects of Scheduling and Applications*. Pergamon Press, New York, NJ, 1982.

[5] D. Bertsekas. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, Belmont, MA, 1998.

[6] K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimization. *Operations Research Letters*, 16:101–113, 1994.

[7] 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.

[8] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NJ, 1999.

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from social insect behavior. *Nature*, 406:39–42, 2000.

[10] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. 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.

[11] E. Bonabeau and G. Theraulaz. Swarm smarts. *Scientific American*, 282(3):54–61, 2000.

[12] 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 Academic Publishers, Dordrecht, 1999.

[13] B. Bullnheimer, R. F. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999.

[14] B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank-based version of the Ant System: A computational study. *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999.

[15] B. Bullnheimer, G. Kotsis, and C. Strauss. Parallelization strategies for the Ant System. In R. De Leone, A. Murli, P. Pardalos, and G. Toraldo, editors, *High Performance Algorithms and Software in Nonlinear Optimization*, volume 24 of *Applied Optimization*, pages 87–100. Kluwer Academic Publishers, Dordrecht, NL, 1998.

[16] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In F. J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, pages 134–142. MIT Press, Cambridge, MA, 1992.

[17] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant System for job-shop scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.

[18] O. Cordón, I. Fernández de Viana, F. Herrera, and L. Moreno. A new ACO model integrating evolutionary computation concepts: The best-worst ant system. In M. Dorigo, M. Middendorf, and T. Stützle, editors, *Abstract proceedings of ANTS2000 – From Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms*, pages 22–29. Université Libre de Bruxelles, 2000.

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

[20] M. den Besten. Ants for the single machine total weighted tardiness problem. Master's thesis, University of Amsterdam, 2000.

[21] M. den Besten, T. Stützle, and M. Dorigo. Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.S. Schwefel, editors, *Proceedings of PPSN-VI, Sixth International Conference on Parallel Problem Solving from Nature*, volume 1917 of *Lecture Notes in Computer Science*, pages 611–620. Springer Verlag, Berlin, Germany, 2000.

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

[23] 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.

[24] G. Di Caro and M. Dorigo. Ant colonies for adaptive routing in packet-switched communications networks. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 673–682. Springer Verlag, Berlin, Germany, 1998.

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

[26] 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.

[27] G. Di Caro and M. Dorigo. Mobile agents for adaptive routing. In H. El-Rewini, editor, *Proceedings of the 31st International Conference on System Sciences (HICSS-31)*, pages 74–83. IEEE Computer Society Press, Los Alamitos, CA, 1998.

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

[29] M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992. pp. 140.

[30] M. Dorigo, E. Bonabeau, and G. Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, 2000.

[31] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, London, UK, 1999.

[32] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

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

[34] 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.

[35] M. Dorigo and V. Maniezzo. Parallel genetic algorithms: Introduction and overview of current research. In J. Stenders, editor, *Parallel Genetic Algorithms: Theory and Applications*, pages 5–42. IOS Press, Amsterdam, The Netherlands, 1992.

[36] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: An autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[37] 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.

[38] 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.

[39] M. Dorigo, M. Middendorf, and T. Stützle, editors. *Abstract proceedings of ANTS2000 – From Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms*. Université Libre de Bruxelles, 7–9 September 2000.

[40] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[41] L. M. Gambardella and M. Dorigo. Ant-Q: A reinforcement learning approach to the traveling salesman problem. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, pages 252–260. Morgan Kaufmann Publishers, Palo Alto, CA, 1995.

[42] L. M. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, pages 622–627. IEEE Press, Piscataway, NJ, 1996.

[43] 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.

[44] L. M. Gambardella and M. Dorigo. Ant Colony System hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.

[45] L. M. Gambardella, È. D. Taillard, and G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw Hill, London, UK, 1999.

[46] L. M. Gambardella, È. D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2):167–176, 1999.

[47] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, CA, 1979.

[48] F. Glover. Tabu search – part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[49] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, MA, 1997.

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

[51] W. J. Gutjahr. A generalized convergence result for the graph-based Ant System metaheuristic. Technical Report 99-09, Department of Statistics and Decision Support Systems, University of Vienna, Austria, 1999.

[52] W. J. Gutjahr. A graph-based Ant System and its convergence. *Future Generation Computer Systems*, 16(8):873–888, 2000.

[53] R. Hadji, M. Rahoual, E. Talbi, and V. Bachelet. Ant colonies for the set covering problem. In M. Dorigo, M. Middendorf, and T. Stützle, editors, *Abstract proceedings of ANTS2000 – From Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms*, pages 63–66. Université Libre de Bruxelles, 2000.

[54] 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*.

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

[56] M. Jünger, G. Reinelt, and S. Thienel. Provably good solutions for the traveling salesman problem. *Zeitschrift für Operations Research*, 40:183–217, 1994.

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

[58] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Travelling Salesman Problem*. John Wiley & Sons, Chichester, UK, 1985.

[59] G. Leguizamón and Z. Michalewicz. A new version of Ant System for subset problems. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, pages 1459–1464. IEEE Press, Piscataway, NJ, 1999.

[60] Y.-C. Liang and A. E. Smith. An Ant System approach to redundancy allocation. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1478–1484. IEEE Press, Piscataway, NJ, 1999.

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

[62] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.

[63] V. Maniezzo and A. Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(8):927 – 935, 2000.

[64] V. Maniezzo and A. Colorni. The Ant System applied to the quadratic assignment problem. *IEEE Transactions on Data and Knowledge Engineering*, 11(5):769–778, 1999.

[65] 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.

[66] D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 893–900. Morgan Kaufmann Publishers, San Francisco, CA, 2000.

[67] R. Michel and M. Middendorf. An island model based Ant System with lookahead for the shortest supersequence problem. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 692–701. Springer Verlag, Berlin, Germany, 1998.

[68] R. Michel and M. Middendorf. An ACO algorithm for the shortest supersequence problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 51–61. McGraw Hill, London, UK, 1999.

[69] M. Middendorf, F. Reischle, and H. Schmeck. Information exchange in multi colony ant algorithms. In J. Rolim, editor, *Parallel and Distributed Computing, Proceedings of the 15 IPDPS 2000 Workshops, Third Workshop on*

Biologically Inspired Solutions to Parallel Processing Problems (BioSP3), volume 1800 of *Lecture Notes in Computer Science*, pages 645–652. Springer Verlag, Berlin, Germany, 2000.

[70] T. E. Morton, R. M. Rachamadugu, and A. Vepsalainen. Accurate myopic heuristics for tardiness scheduling. GSIA Working Paper 36-83-84, Carnegie–Mellon University, PA, 1984.

[71] G. Navarro Varela and M. C. Sinclair. Ant colony optimisation for virtual-wavelength-path routing and wavelength allocation. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, pages 1809–1816. IEEE Press, Piscataway, NJ, 1999.

[72] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

[73] H. Ramalhinho Lourenço and D. Serra. Adaptive approach heuristics for the generalized assignment problem. Technical Report Technical Report Economic Working Papers Series No.304, Universitat Pompeu Fabra, Dept. of Economics and Management, Barcelona, Spain, 1998.

[74] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany, 1994.

[75] R. Y. Rubinstein. Combinatorial optimization via the simulated cross-entropy method. In *Encyclopedia of Management Sciences*. 2000. in press.

[76] 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.

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

[78] C. Solnon. Solving permutation constraint satisfaction problems with artificial ants. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 118–122. IOS Press, Amsterdam, The Netherlands, 2000.

[79] T. Stützle. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System for the quadratic assignment problem. Technical Report AIDA–97–4, FG Intellektik, FB Informatik, TU Darmstadt, July 1997.

[80] T. Stützle. An ant approach to the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, volume 3, pages 1560–1564. Verlag Mainz, Aachen, 1998.

[81] T. Stützle. Parallelization strategies for ant colony optimization. In Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 722–731. Springer Verlag, Berlin, Germany, 1998.

[82] T. Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*. Infix, Sankt Augustin, Germany, 1999.

[83] T. Stützle and M. Dorigo. ACO algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 33–50. McGraw Hill, London, UK, 1999.

[84] T. Stützle and H. H. Hoos. The $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and local search for the traveling salesman problem. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 309–314. IEEE Press, Piscataway, NJ, 1997.

[85] T. Stützle and H. H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[86] 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.

[87] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

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

[89] 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.

[90] M. Yannakakis. Computational complexity. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55. John Wiley & Sons, Chichester, 1997.