Mathware & Soft Computing 9 (2002)

A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends

Oscar Cordón[1], Francisco Herrera[1], Thomas Stützle[2]
[1]Dept. of Computer Science and A.I. E.T.S. de Ingeniería Informática
University of Granada. 18071 - Granada (Spain)
[2]Intellectics Group. Dept. of Computer Science
Darmstadt University of Technology. 64283 - Darmstadt (Germany)
e-mail: {ocordon,herrera}@decsai.ugr.es,
stuetzle@informatik.tu-darmstadt.de

Abstract

Ant Colony Optimization (ACO) is a recent metaheuristic method that is inspired by the behavior of real ant colonies. In this paper, we review the underlying ideas of this approach that lead from the biological inspiration to the ACO metaheuristic, which gives a set of rules of how to apply ACO algorithms to challenging combinatorial problems. We present some of the algorithms that were developed under this framework, give an overview of current applications, and analyze the relationship between ACO and some of the best known metaheuristics. In addition, we describe recent theoretical developments in the field and we conclude by showing several new trends and new research directions in this field.

1 Introduction

Complex combinatorial optimization problems arise in many different fields such as economy, commerce, engineering, industry or medicine. However, often these kinds of problems are very hard to solve in practice. This inherent difficulty of solving such problems is captured in theoretical computer science by the fact that many of them are known to be $\mathcal{NP}$-hard, which means that there is no algorithm known for solving them in polynomial time [40].

Still, many of these problems have to be solved in a huge number of practical settings and therefore a large number of algorithmic approaches were proposed to tackle them. The existing techniques can roughly be classified into exact and approximate algorithms. Exact algorithms try to find an optimal solution and to prove that the solution obtained is actually an optimal one; these algorithms include techniques such as backtracking, branch and bound, dynamic programming, etc. [75, 10]. Because exact algorithms show poor performance for many problems, several types of approximate algorithms were developed that provide high quality solutions to combinatorial problems in short computation time.

Approximate algorithms can be classified into to main types: construction algorithms and local search algorithms. The former are based on generating solutions from scratch by adding solution components step by step. The best known example are greedy construction heuristics [10]. Their advantage is speed: they are typically very quick and, in addition, often return reasonably good solutions. However, these solutions are not guaranteed to be optimal with respect to small local changes. Therefore, a typical approach is to further improve the solutions returned by greedy heuristics by a local search. Local search algorithms repeatedly try to improve the current solution by movements to (hopefully better) neighboring solutions. The simplest case are iterative improvement algorithms: if in the neighborhood of the current solution s, a better solution s' is found, it replaces the current solution and the search is continued from s'; if no better solution is found, the algorithm terminates in a local optimum.

Unfortunately, iterative improvement algorithms may become stuck in poor quality local optima. To allow for a further improvement in solution quality, in the last two decades the research in this field has moved attention to the design of general-purpose techniques for guiding underlying, problem-specific construction or local search heuristics. These techniques are often called *metaheuristics* [74, 41, 93] and they consist of concepts that can be used to define heuristic methods. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different (combinatorial) optimization problems with relatively few modifications if given some underlying, problem specific heuristic method. In fact, metaheuristics are now widely recognized as the most promising approaches for attacking hard combinatorial optimization problems [2, 67, 79].

Metaheuristics incorporate concepts from very different fields such as genetics, biology, artificial intelligence, mathematics and physics, and neuro-sciences, among others. Examples of metaheuristics include simulated annealing [1, 55], tabu search [42], iterated local search [57], variable neighborhood search algorithms [51], greedy randomized adaptive search procedures (GRASP) [34, 35], and evolutionary algorithms [4, 43, 52]. A rather recent metaheuristic is ant colony optimization (ACO), which is inspired by shortest path searching behavior of various ant species. However, since the initial work of Dorigo, Maniezzo, and Colorni on *Ant System* [28], ACO is now quickly becoming a mature research field: a large number of authors have developed more sophisticated models that were used to successfully solve a large number of complex combinatorial optimization problems and theoretical insights into the algorithm are now becoming available.

This paper reviews the basis of ACO algorithms. We first present the behavior of real ant colonies, which inspired ACO, in Section 2. Next, the transition from real to artificial ants is described in Section 3; there we discuss the kinds of problems solved by ACO, we summarize the similarities and differences between natural and artificial ants and the generic operation mode of an ACO algorithm, and finally indicate the required steps to solve a combinatorial optimization problem by ACO. Section 4 describes several of the existing ACO algorithms, while their applications are reviewed in Section 5. The relationship between ACO and other metaheuristics is analyzed in Section 6, and theoretical aspects of ACO are addressed in Section 7. Finally, Section 8 discusses some new trends in ACO and Section 9 presents the concluding remarks.

2 Natural ant colonies

Ants are social insects that live in colonies and, because of their collaborative interaction, they are capable of showing complex behaviors and to perform difficult tasks from an ant's local perspective. A very interesting aspect of the behavior of several ant species is their ability to find shortest paths between the ants' nest and the food sources. This fact is specially noticeable having in mind that in many ant species ants are almost blind, which avoids the exploitation of visual clues.

While walking between their nest and food sources, some ant species deposit a chemical called *pheromone* (an odorous substance). If no pheromone trails are available, ants move essentially at random, but in the presence of pheromone they have a tendency to follow the trail. In fact, experiments by biologists have shown [78, 44] that ants probabilistically prefer paths that are marked by a high pheromone concentration. In practice, choices between different paths occur when several paths intersect. Then, ants choose the path to follow by a probabilistic decision biased by the amount of pheromone: the stronger the pheromone trail, the higher its desirability. Because ants in turn deposit pheromone on the path they are following, this behavior results in a self-reinforcing process leading to the formation of paths marked by high pheromone concentrations. This behavior also allows ants to identify shortest paths between their nest and a food source [44].[1]

How this mechanism allows the ants to reach shortest paths is illustrated in Figure 1. Initially, there is no pheromone trail on the environment and, when the ants arrive at an intersection, they randomly choose one of the branches. However, as ants are traveling, the most promising paths receive a greater amount of pheromone after some time. This is due to the fact that, because these paths are shorter, the ants following them are able to reach the goal (i.e., the food) quicker and to start their return-trip earlier. Since on the shorter branch already a slightly stronger pheromone trail exists, the ants' decision is biased towards the shorter branch, which, thus, receives a larger proportion of the pheromone of the returning ants than the longer branch. This process finally results in an increasingly stronger bias towards the shorter branch and, in the end, to convergence to the shortest.

The latter procedure is complemented in the natural environment by the fact that the pheromone evaporates after some time. This way, less promising paths progressively loose pheromone because of being visited by less and less ants. However, several biological studies show that the pheromone trails are very persistent (the pheromone can stay from several hours to several months depending on aspects such as the ant species, the floor type, ... [9]), thus making less significant the influence of the evaporation in the shortest path searching behavior.

[1] Note that ants only communicate indirectly, through modifications of the physical environment they perceive. This form of communication is called *artificial stigmergy* in [25].

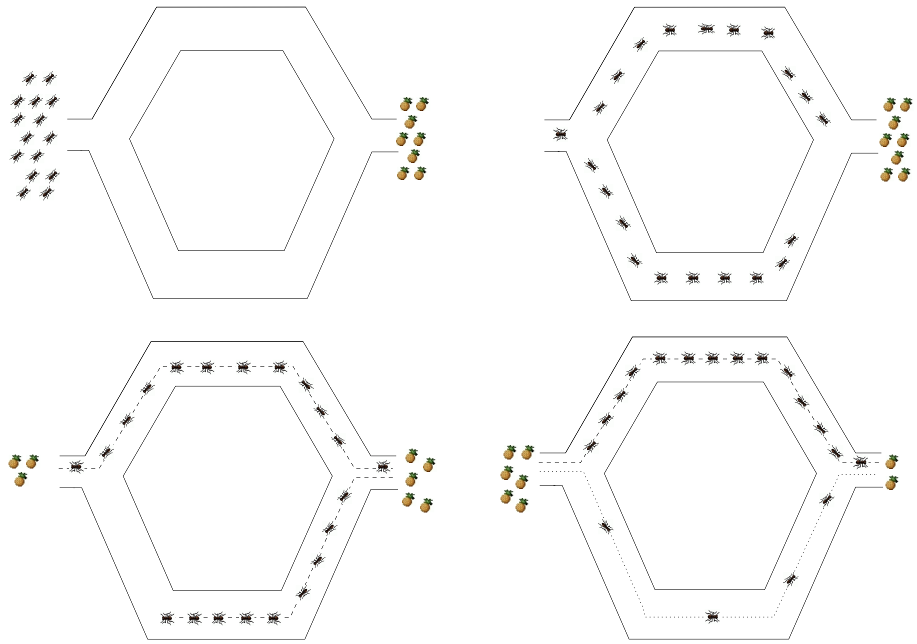

Figure 1: Emergent behavior of the colony that ends by obtaining the shortest path between two points (mass recruitment). Based on the figure in [9]

In [9], several experiments are reported showing that the mass recruitment in Nature is restrictive since, as a result of the long persistence of the pheromone, it is difficult that ants forget a path with a high level of pheromone, although they have found a shorter one. Notice that, if this behavior is directly translated into the computer to design a search algorithm, we can get an algorithm quickly getting stuck in local optima. We will come back to this issue later.

3 From natural ants to the Ant Colony Optimization metaheuristic

ACO algorithms take inspiration from the behavior of real ant colonies to solve combinatorial optimization problems. They are based on a colony of *artificial ants*, that is, simple computational agents that work cooperatively and communicate through *artificial pheromone trails*.

ACO algorithms are essentially construction algorithms: in each algorithm iteration, every ant constructs a solution to the problem by traveling on a construction graph. Each edge of the graph, representing the possible steps the ant can make, has associated two kinds of information that guide the ant movement:

- *Heuristic information*, which measures the heuristic preference of moving from node r to node s, i.e., of traveling the edge a_{rs}. It is denoted by η_{rs}. This information is not modified by the ants during the algorithm run.

- *(Artificial) pheromone trail information*, which measures the "learned desirability" of the movement and mimics the real pheromone that natural ants deposit. This information is modified during the algorithm run depending on the solutions found by the ants. It is denoted by τ_{rs}.

This section introduces the steps leading from real ants to ACO. It should be noted for the following that ACO algorithms present a double perspective:

- On the one hand, they are an abstraction of some behavioral patterns of natural ants related to the shortest path searching behavior.
- On the other hand, they include several features that do not have a natural counterpart, but that allow to develop algorithms for obtaining good solutions to the problem tackled (for example, the use of heuristic information to guide the ant movement).

3.1 Kinds of problems solved by ACO

The type of problems being solved by artificial ants belongs to the group of (constrained) shortest path problems that can be characterized by the following aspects (we follow mainly the presentation in [25] and [31]):

- There is a *set of constraints* Ω defined for the problem under solution.
- There is a finite set of components $N = \{n_1, n_2, \ldots, n_l\}$.
- The problem presents several *states* defined upon ordered component sequences $\delta =< n_r, n_s, \ldots, n_u, \ldots >$ ($< r, s, \ldots, u, \ldots >$ to simplify) over the elements of N. If Δ is the set of all possible sequences, we denote by $\tilde{\Delta}$ the set of feasible (sub)sequences with respect to the constraints Ω. The elements in $\tilde{\Delta}$ define the *feasible states*. $|\delta|$ is the length of a sequence δ, i.e., the number of components in the sequence.
- There is a *neighborhood structure* defined as follows: δ_2 is a neighbor of δ_1 if (i) both δ_1 and δ_2 belong to Δ, (ii) the state δ_2 can be reached from δ_1 in one logical movement, i.e., if r is the last component of the sequence δ_1, there must exist a component $s \in N$ such that $\delta_2 =< \delta_1, s >$, i.e., there exists a valid transition between r and s. The *feasible neighborhood* of δ_1 is the set containing all sequences $\delta_2 \in \tilde{\Delta}$; if $\delta_2 \notin \tilde{\Delta}$, we say that δ_2 is in the *infeasible neighborhood* of δ_1.
- A solution S is an element of $\tilde{\Delta}$ verifying all the problem requirements.
- There is a cost $C(S)$ associated to each solution S.
- In some cases, a cost or an estimate of the cost may be associated to states.

As said, all the previous characteristics hold in combinatorial optimization problems that can be represented in the form of a weighted graph $G = (N, A)$, where A is the set of edges that connects the set of components N. The graph G is also called construction graph G.[2] Hence, we have that

- the components n_r are the nodes of the graph,
- the states δ (and hence the solutions S) correspond to paths in the graph, i.e., sequences of nodes or edges,
- the edges of the graph, a_{rs}, are connections/transitions defining the neighborhood structure. $\delta_2 =< \delta_1, s >$ is a neighbor of δ_1 if node r is the last component of δ_1 and edge a_{rs} exists in the graph,
- there may be explicit transition costs c_{rs} associated to each edge, and
- the components and connections may have associated pheromone trails τ, which represent some form of indirect, long term memory of the search process, and heuristic values η, which represent some heuristic information available on the problem under solution.

3.2 The artificial ant

The artificial ant is a simple, computational agent that tries to build feasible solutions to the problem tackled exploiting the available pheromone trails and heuristic information. However, if necessary, it may also build infeasible solutions that may be penalized depending on the amount of infeasibility. It has the following properties [25, 31]:

- It searches minimum cost feasible solutions for the problem being solved.
- It has a memory L storing information about the path followed until that moment, i.e., L stores the generated sequence. This memory can be used to: (i) build feasible solutions, (ii) evaluate the generated solution, and (iii) to retrace the path the ant has followed.
- It has an *initial state* $\delta_{initial}$, that usually corresponds to a unitary sequence, and one or more termination conditions t associated.
- It starts in the *initial state* and moves towards feasible states, building its associated solution incrementally.
- When it is in a state $\delta_r =< \delta_{r-1}, r >$ (i.e., it is located in node r and has previously followed the sequence δ_{r-1}), it can move to any node s of its *feasible neighborhood* $\mathcal{N}(r)$, defined as $\mathcal{N}(r) = \{s \mid (a_{rs} \in A) \textit{ and } (< \delta_r, s > \in \tilde{\Delta})\}$.

[2] As said in [31], the set of edges may fully connect the components. In this case, the implementation of the constraints is fully integrated into the construction policy of the ants.

- The movement is made by applying a transition rule, which is a function of the locally available pheromone trails and heuristic values, the ants private memory, and the problem constraints.

- When, during the construction procedure, an ant moves from node r to s, it can update the pheromone trail τ_{rs} associated to the edge a_{rs}. This process is called *online step-by-step pheromone trail update.*

- The construction procedure ends when any termination condition is satisfied, usually when an *objective state* is reached.

- Once the solution has been built, the ant can retrace the traveled path and update the pheromone trails on the visited edges/components by means of a process called *online delayed pheromone trail update.*

  This way, the only communication mechanism among the ants is the data structure storing the pheromone levels of each edge/component (shared memory).

3.3 Similarities and differences between natural and artificial ants

Real and artificial ant colonies share a number of characteristics. The most important ones can be summarized as follows (see [26] for a more detailed discussion):

- Use of a colony of individuals that interact and collaborate to solve a given task.

- Both, natural and artificial ants modify their "environment" through stigmergic communication based on pheromones. In the case of artificial ants, the *(artificial) pheromone trail* is a numeric information which is only locally available.

- Both, natural and artificial ants share a common task: the search of the shortest path (iterative construction of a minimum cost solution) from an origin, the ant nest (initial decision), to some goal state, the food (last decision).

- Artificial ants build the solutions iteratively by applying a local stochastic transition policy to move between adjacent states, as real ants do.

However, these characteristics alone do not allow to develop efficient algorithms for hard combinatorial problems. Therefore, artificial ants live in a discrete world and have additional capabilities:

- Artificial ants can make use of heuristic information (and not only pheromone trail information) in the stochastic transition policy they apply.

- They have a memory that stores the path followed by the ant.

- The amount of pheromone deposited by the artificial ants is a function of the *quality of the solution found* by each of them.[3] A major difference also concerns the timing of the pheromone deposit. Artificial ants usually only deposit pheromone after generating a complete solution.[4]

- As said in Section 2, pheromone evaporation in ACO algorithms is different than in nature, since the inclusion of an evaporation mechanism is a key question to avoid the algorithm getting stuck in local optima. Pheromone evaporation allows the artificial ant colony to softly forget its past history and to direct its search towards new space regions. This avoids a premature convergence of the algorithm to local optima.

- In order to improve the efficiency and efficacy of the system, ACO algorithms can be enriched with *additional capabilities*. Examples are the ability to look further than the next transition (*"lookahead"*) [68], *local optimization* [27, 87] and *"backtracking"* (whose use is not very extended), or so-called *candidate list* which contain a set of the most promising neighbor states [27, 25] to improve the efficiency of the algorithm.

3.4 Operation mode and generic structure of an ACO algorithm

As seen in the previous sections, the basic operation mode of an ACO algorithm is as follows: the m (artificial) ants of the colony move, concurrently and asynchronously, through adjacent states of a problem (that can be represented in the form of a weighted graph). This movement is made according to a *transition rule* which is based on local information available at the components (nodes). This local information comprises heuristic and memoristic (pheromone trails) information to guide the search. By moving on the construction graph, ants incrementally build solutions. Optionally, ants can release pheromone each time they cross an edge (connection) while constructing solutions (*online step-by-step pheromone trail update*). Once every ant has generated a solution, it is evaluated and it can deposit an amount of pheromone which is a function of the quality of the ant's solution (*online delayed pheromone trail update*). This information will guide the search of the other ants of the colony in the future.

Moreover, the generic operation mode of the ACO algorithm also includes two additional procedures, *pheromone trail evaporation* and *daemon actions*. The pheromone evaporation is triggered by the environment and it is used as a mechanism to avoid search stagnation and to allow the ants to explore new space regions. Daemon actions are optional actions —without a natural counterpart— to implement tasks from a global perspective that is lacking to the local perspective of the ants. The additional capabilities mentioned in Section 3.3 are included in

[3] However, this difference is relative as some natural ant species deposit a higher quantity of pheromone when they found a richer food source [9].

[4] Nevertheless, as we will see in the following, few ACO algorithms also modify the pheromone trails while constructing a solution.

these actions. Examples are observing the quality of all the solutions generated and releasing an additional pheromone amount only on the transitions/components associated to some of the solutions, or applying a local search procedure to the solutions generated by the ants before updating the pheromone trails. In both cases, the daemon replaces the online delayed pheromone update and the process is called *offline pheromone trail update*.

The structure of a generic ACO algorithm for is as follows [25, 26].

```
Procedure ACO_Metaheuristic
    parameter_initialization
    while (termination_criterion_not_satisfied)
        schedule_activities
                ants_generation_and_activity()
                pheromone_evaporation()
                daemon_actions() {optional}
        end schedule_activities
    end while
end Procedure
```

```
Procedure ants_generation_and_activity()
   repeat in parallel for k=1 to m (number_of_ants)
      new_ant(k)
   end repeat in parallel
end Procedure
```

```
Procedure new_ant(ant_id)
    initialize_ant(ant_id)
    L = update_ant_memory()
    while (current_state ≠ target_state)
        P = compute_transition_probabilities(A,L,Ω)
        next_state = apply_ant_decision_policy(P,Ω)
        move_to_next_state(next_state)
        if (on_line_step-by-step_pheromone_update)
            deposit_pheromone_on_the_visited_edge()
        end if
    L = update_internal_state()
    end while
    if (online_delayed_pheromone_update)
        for each visited edge
            deposit_pheromone_on_the_visited_edge()
        end for
    end if
    release_ant_resources(ant_id)
end Procedure
```

The first step involves the initialization of the parameter values considered by the algorithm. Among others, the initial pheromone trail value associated to each transition, τ_0, which is a small positive value that is typically the same for all connections/components, the number of ants in the colony, m, and the weights defining the balance between the heuristic and memoristic information in the probabilistic transition rule have to be set.[5]

The main procedure of the ACO metaheuristic manages, by means of the **schedule_activities** construct, the scheduling of the three components mentioned in this section: (i) the generation and operation of the artificial ants, (ii) the pheromone evaporation, and (iii) the daemon actions. The implementation of this construct will define the existing syncronism between the three components. While the application to "classical" $\mathcal{NP}$-hard (non distributed) problems typically uses rather a sequential schedule, in distributed problems like network routing parallelism can be easily and efficiently exploited.

As said, several components are either optional, such as the daemon actions, or strictly dependent on the specific ACO algorithm, e.g., when and where the pheromone is deposited. Generally, the online step-by-step pheromone trail update and the online delayed pheromone trail update are mutually exclusive and they both are not usually present or missing at the same time (if both are missing, typically the daemon updates the pheromone trails).

On the other hand, notice that the procedure `update_ant_memory` involves specifying the initial state from which the ant starts its path and storing the corresponding component in the ant memory L. The decision on which will be that node (it can be a random choice or a fixed one for the whole colony, a random choice or a fixed one for each ant, etc.) depends on the specific application.

Finally, note that the procedures `compute_transition_probabilities` and `apply_ant_decision_policy` consider the current state of the ant, the current values of the pheromones visible in that node and the problem constraints Ω to establish the probabilistic transition process to other feasible states.

3.5 Relation between ACO and ant algorithms

It is important to notice that the term *ACO metaheuristic* stands for the generic operation mode of ACO. The name *ACO algorithm* is used to refer to any specific instance of the generic algorithm shown in Section 3.4, such as those that are analyzed in the following Section 4. It should be noted that the ACO metaheuristic comprises a very wide class of algorithms that can have very different shapes. This is mainly due to the rather complex types of interactions possible through the **schedule_activities** construct among the activities `ants_generation_and_activity()`, `pheromone_evaporation()`, and `daemon_actions()`. It should be noted, however, that in many applications ants typically move in a synchronized way and the algorithmic outline of actual ACO algorithms follows a much simpler flow of activities [84, 85]. The main reason for the greater generality of the ACO metaheuristic is that it was defined *a posteriori* as a common framework to already

[5] This aspect will be analyzed in depth in the next section when introducing specific ACO algorithms.

existing applications to $\mathcal{NP}$-hard optimization problems and routing in telecommunications networks, an inherently dynamic problem (see also Section 5 for a short overview). While these two types of applications are similar from a high-level perspective, due to the very different application domains, the algorithms and, in particular, the interactions among the three activities `ants_generation_and_activity()`, `pheromone_evaporation()`, and `daemon_actions()`, are very different from a low-level perspective.

However, the ACO metaheuristic is not general enough to cover the full family of ant algorithms, which can loosely be defined as approximate methods to solve combinatorial problems based on characteristics of the generic behavior of natural ants.[6] Examples of ant algorithms not covered by ACO are *Fast Ant System* [92] and *Hybrid Ant System* [39]. While the former is a construction algorithm based on the operation of a single ant without using explicit pheromone evaporation, the latter is a local search procedure that makes use of pheromone trail information to generate the neighbor solutions. In [90], an experimental study of these two ant algorithms for the TSP is presented.

3.6 Steps to solve a problem by ACO

From the currently known ACO applications, we can identify some guidelines of how to attack problems by ACO. These guidelines can be summarized by the following six design tasks:

1. Represent the problem in the form of sets of components and transitions or by means of a weighted graph (see Section 3.1), that is traveled by the ants to build solutions.

2. Appropriately define the meaning of the pheromone trails τ_{rs}, i.e., the type of decision they bias. This is a crucial step in the implementation of an ACO algorithm and often, a good definition of the pheromone trails is not a trivial task and it typically requires insight into the problem under the solution.

3. Appropriately define the heuristic preference to each decision that an ant has to take while constructing a solution, i.e., define the heuristic information η_{rs} associated to each component or transition. Notice that heuristic information is crucial for good performance if local search algorithms are not available or can not be applied.

4. If possible, implement an efficient local search algorithm for the problem under solution, because the results of many ACO applications to $\mathcal{NP}$-hard combinatorial optimization problems show that the best performance is achieved when coupling ACO with local optimizers [25, 31].

5. Choose a specific ACO algorithm (some of the available ones are described in the next section) and apply it to the problem being solved, taking the previous aspects into account.

[6] Note that every ACO algorithm is also an ant algorithm but the opposite is not true.

6. Tune the parameters of the ACO algorithm. A good starting point for parameter tuning is to use parameter settings that were found to be good when applying the ACO algorithm to similar problems or to a variety of other problems. An alternative to time-consuming personal involvement in the tuning task is to use automatic procedures for parameter tuning [6].

It should be clear that the above steps can only give a very rough guide to the implementation of ACO algorithms. In addition, often the implementation is an iterative process, where with some further insight into the problem and the behavior of the algorithm, some initially taken choices need to be revised. Finally, we want to insist in the fact that probably the most important of these steps are the first four, because a poor choice at this stage typically can not be made up with pure parameter fine-tuning.

4 Ant Colony Optimization models

Several algorithms have been proposed in the literature following the ACO metaheuristic. Among the available ACO algorithms for $\mathcal{NP}$-hard combinatorial optimization problems are *Ant System* [28], *Ant Colony System* [27], *Max-Min Ant System* [88], *Rank-based Ant System* [12], and *Best-Worst Ant System* [20]. In the following, we give a short description of these algorithms.[7] While Ant System is mainly of historical interest because it was the first ACO algorithm, the other four typically achieve much better computational results. A major omission in our description is AntNet, a successful ACO algorithm for network routing. However, this algorithm is rather application specific and we refer to [24] for a detailed description.

Notice that in the following we consider the case, where pheromones and heuristic information are only attached to the connections, which is the case for many applications of ACO to sequencing or assignment problems. It is straightforward to extend the description to the case in which pheromones are associated to components.

4.1 Ant System

Ant System (AS) [28], developed by Dorigo, Maniezzo and Colorni in 1991, was the first ACO algorithm. Initially, three different variants, *AS-density*, *AS-quantity* and *AS-cycle*, differing in the way in which the pheromone trails are updated, were proposed. In the former two ones, ants release pheromone while building their solutions (i.e., they apply an *online step-by-step pheromone update*), with the difference that the amount deposited in *AS-density* is constant while the one released in *AS-quantity* directly depends on the heuristic desirability of the transition η_{ij}. Finally, in *AS-cycle*, the pheromone deposit is done once the solution is completed (*online delayed pheromone update*). This latter variant was the one performing

[7]For a more detailed description of these algorithms, including some comparisons of their performance when applied to the traveling salesman problem, we refer to [25, 26, 30, 85].

best and actually this is the variant that is now referred to as AS in the literature (and in the remainder of this paper).

AS is characterized by the fact that the pheromone update is triggered once all ants have completed their solutions and it is done as follows. First, all pheromone trails are reduced by a constant factor, implementing in this way the pheromone evaporation. Second, every ant of the colony deposits an amount of pheromone which is a function of the quality of its solution. Initially, AS did not use any daemon actions, but it is very straightforward to, for example, add a local search procedure to refine the solutions generated by the ants.

Solutions in AS are constructed as follows. At each construction step, an ant k in AS chooses to go to a next node with a probability that is computed as

$$p_{rs}^{k} = \begin{cases} \frac{[\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta}}{\sum_{u \in \mathcal{N}_r^k} [\tau_{ru}]^{\alpha} \cdot [\eta_{ru}]^{\beta}}, & \text{if } s \in \mathcal{N}_k(r) \\ 0, & \text{otherwise} \end{cases},$$

where $\mathcal{N}_k(r)$ is the feasible neighborhood of ant k when located at node r, and $\alpha, \beta \in \mathbb{R}$ are two parameters that weight the relative importance of the pheromone trail and the heuristic information. Each ant k stores the sequence it has followed so far and this memory L_k is, as explained before, exploited to determine $\mathcal{N}_k(r)$ in each construction step.

As regards parameters α and β, their role is as follows: if $\alpha = 0$, those nodes with better heuristic preference have a higher probability of being selected, thus making the algorithm close to a classical *probabilistic greedy algorithm* (with multiple starting points in case ants are located in different nodes at the beginning of each iteration). However, if $\beta = 0$, only the pheromone trails are considered to guide the constructive process, which can cause a quick *stagnation*, i.e., a situation where the pheromone trails associated to some transitions are significantly higher than the remainder, thus making the ants always build the same solutions, usually local optima. Hence, there is a need to establish a proper balance between the importance of heuristic and pheromone trail information.

As said, the pheromone deposit is made once all ants have finished to construct their solutions. First, the pheromone trail associated to every arc is evaporated by reducing all pheromones by a constant factor:

$$\tau_{rs} \leftarrow (1 - \rho) \cdot \tau_{rs},$$

where $\rho \in (0, 1]$ is the evaporation rate. Next, each ant retraces the path it has followed (this path is stored in its local memory L_k) and deposits an amount of pheromone $\Delta\tau_{rs}^{k}$ on each traversed connection:

$$\tau_{rs} \leftarrow \tau_{rs} + \Delta\tau_{rs}^{k}, \quad \forall a_{rs} \in S_k,$$

where $\Delta\tau_{rs}^{k} = f(C(S_k))$, i.e., the amount of pheromone released depends on the quality $C(S_k)$ of the solution S_k of ant k.

To summarize the description of the AS, we will show the composition of procedure `new_ant` for this particular ACO algorithm:

1 **Procedure** `new_ant(ant_id)`
2 `k = ant_id; r = generate_initial_state;` $S_k = r$
3 $L_k = r$
4 **while** `(current_state` $\neq$ `target_state)`
5 **for** each $s \in \mathcal{N}_k(r)$ **do** $p^k_{rs} = \frac{[\tau_{rs}]^\alpha \cdot [\eta_{rs}]^\beta}{\sum_{u \in \mathcal{N}^k_r} [\tau_{ru}]^\alpha \cdot [\eta_{ru}]^\beta}$
6 `next_state = apply_ant_decision_policy(P,`$\mathcal{N}_k(r)$`)`
7 r `= next_state;` $S_k = < S_k, r >$
8 ---
9 $L_k = L_k \cup r$
10 **end while**
{`the pheromone_evaporation() procedure triggers and evaporates pheromone in every edge` a_{rs}: $\tau_{rs} = (1-\rho) \cdot \tau_{rs}$}
11 **for** each edge $a_{rs} \in S_k$ **do**
12 $\tau_{rs} = \tau_{rs} + f(C(S_k))$
13 **end for**
14 `release_ant_resources(ant_id)`
15 **end Procedure**

Notice that the empty line *8* is included to remark that no online step-by-step pheromone update is made and that *before the line 12, the pheromone evaporation must have been applied by the daemon.* In fact, this is one example, where the **schedule_activities** construct interferes with the functioning of the single main procedures of the ACO metaheuristic, as indicated on page 9.

Before concluding this section, it is important to notice that the creators of the AS also proposed a typically better performing, extended version of this algorithm called *elitist AS* [28]. In elitist AS, once the ants have released pheromone on the connections associated to their generated solutions, the daemon performs an additional pheromone deposit on the edges belonging to the best solution found until that moment in the search process (this solution is also called global-best solution in the following). The amount of pheromone deposited, which depends on the quality of that global best solution, is weighted by the number of elitist ants considered, e, as follows:

$$\tau_{rs} \leftarrow \tau_{rs} + e \cdot f(C(S_{global-best})), \quad \forall a_{rs} \in S_{global-best}$$

4.2 Ant Colony System

Ant Colony System (ACS) [27] is one of the first successors of AS. It introduces three major modifications into AS:

1. ACS uses a different transition rule, which is called *pseudo-random proportional rule*: Let k be an ant located at a node r, $q_0 \in [0,1]$ be a parameter, and q a random value in $[0,1]$. The next node s is randomly chosen according to the following probability distribution

If $q \leq q_0$:

$$p^k_{rs} = \begin{cases} 1, & \text{if } s = arg \max\limits_{u \in \mathcal{N}_k(r)} \{\tau_{ru} \cdot \eta^{\beta}_{ru}\} \\ 0, & \text{otherwise} \end{cases} ,$$

else ($q > q_0$):

$$p^k_{rs} = \begin{cases} \frac{[\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta}}{\sum_{u \in \mathcal{N}^k_r} [\tau_{ru}]^{\alpha} \cdot [\eta_{ru}]^{\beta}}, & \text{if } s \in \mathcal{N}_k(r) \\ 0, & \text{otherwise} \end{cases}$$

As can be seen, the rule has a double aim: when $q \leq q_0$, it exploits the available knowledge, choosing the best option with respect to the heuristic information and the pheromone trail. However, if $q > q_0$, it applies a controlled exploration, as done in AS. In summary, the rule establishes a trade-off between the exploration of new connections and the exploitation of the information available at that moment.

2. Only the daemon (and not the individual ants) trigger the pheromone update, i.e., an offline pheromone trail update is done. To do so, ACS only considers one single ant, the one who generated the global best solution, $S_{global-best}$ (although in early papers, an update based on the iteration-best ant was considered as well [27], ACS almost always applies a global-best update).

   The pheromone update is done by first evaporating the pheromone trails on all the connections used by the global-best ant (it is important to notice that *in ACS, pheromone evaporation is only applied to the connections of the solution that is also used to deposit pheromone*) as follows:

   $$\tau_{rs} \leftarrow (1 - \rho) \cdot \tau_{rs}, \quad \forall a_{rs} \in S_{global-best}$$

   Next, the daemon deposits pheromone by the rule:

   $$\tau_{rs} \leftarrow \tau_{rs} + \rho \cdot f(C(S_{global-best})), \quad \forall a_{rs} \in S_{global-best}$$

   Additionally, the daemon can apply a local search algorithm to improve the ants' solutions before updating the pheromone trails.

3. Ants apply an *online step-by-step pheromone trail update* that encourages the generation of different solutions to those yet found. Each time an ant travels an edge a_{rs}, it applies the rule:

   $$\tau_{rs} \leftarrow (1 - \varphi) \cdot \tau_{rs} + \varphi \cdot \tau_0,$$

   where $\varphi \in (0, 1]$ is a second pheromone decay parameter. As can be seen, the *online step-by-step update rule* includes both, pheromone evaporation and deposit. Because the amount of pheromone deposited is very small (in fact, τ_0 is the initial pheromone trail value which is chosen in such a way that, in practice, it corresponds to a lower pheromone trail limit, i.e., by the choice of the ACS pheromone update rules, no pheromone trail value can fall below τ_0), the application of this rule makes the pheromone trail on the connections traversed by an ant decrease.[8]. Hence, this results in an additional exploration technique of ACS by making the connections traversed by an ant less attractive to following ants and helps to avoid that every ant follows the same path.

The procedures `new_ant` and `daemon_actions` (which in this case interacts with the `pheromone_evaporation` procedure) for ACS are as follows:

```
Procedure new_ant(ant_id)
    k = ant_id; r = generate_initial_state; $S_k = r$
    $L_k = r$
    while (current_state $\neq$ target_state)
        for each $s \in \mathcal{N}_k(r)$ do compute $b_{rs} = \tau_{rs} \cdot \eta_{rs}^{\beta}$
        q = generate_random_value_in_[0,1]
        if ($q <= q_0$)
            next_state = max($b_{rs}, \mathcal{N}_k(r)$)
        else
            for each $s \in \mathcal{N}_k(r)$ do
                $p_{rs}^k = \frac{b_{rs}}{\sum_{u \in \mathcal{N}_k(r)} b_{ru}}$
            next_state = apply_ant_decision_policy(P,$\mathcal{N}_k(r)$)
        end if
        $r$ = next_state; $S_k = < S_k, r >$
        $\tau_{rs} = (1 - \varphi) \cdot \tau_{rs} + \varphi \cdot \tau_0$
        $L_k = L_k \cup r$
    end while
    ---
    ---
    ---
    release_ant_resources(ant_id)
end Procedure
```

```
Procedure daemon_actions
    for each $S_k$ do local_search($S_k$) {optional}
    $S_{current-best}$ = best_solution ($S_k$)
    if (better($S_{current-best}, S_{global-best}$))
        $S_{global-best} = S_{current-best}$
    end if
```

[8] ACS is actually based on Ant-Q, an earlier algorithm proposed by Gambardella and Dorigo [36] The only difference between ACS and Ant-Q is in the definition of the term τ_0 in the online step-by-step update rule, which in Ant-Q is the discounted evaluation of the next state, set to $\gamma \cdot \max_{s \in \mathcal{N}_k(r)} \{\tau_{rs}\}$. However, experimental results suggested that ACS results in the same level of performance and, because of its greater simplicity, it was preferred.

for each edge $a_{rs} \in S_{global-best}$ **do**
  {the pheromone_evaporation() procedure triggers and evaporates pheromone in edge a_{rs}: $\tau_{rs} = (1-\rho) \cdot \tau_{rs}$}
  $\tau_{rs} = \tau_{rs} + \rho \cdot f(C(S_{global-best}))$
end for
end Procedure

4.3 Max-Min Ant System

Max-Min Ant System ($\mathcal{MM}$AS) [89, 84, 88], developed by Stützle and Hoos in 1996, is one of the best performing extensions of AS. It extends the basic AS in the following aspects:

1. An offline pheromone trail update is applied, similar to ACS. After all ants have constructed a solution, first every pheromone trail is evaporated:

$$\tau_{rs} \leftarrow (1-\rho) \cdot \tau_{rs},$$

   and next pheromone is deposited according to:

$$\tau_{rs} \leftarrow \tau_{rs} + f(C(S_{best})), \quad \forall a_{rs} \in S_{best}$$

   The *best ant* that is allowed to add pheromone may be the *iteration-best* or the *global-best* solution. Experimental results have shown that the best performance is obtained by gradually increasing the frequency of choosing the global-best solution for the pheromone trail update [84, 88].

   In addition, in $\mathcal{MM}$AS typically the ants' solutions are improved using local optimizers before the pheromone update.

2. The possible values for the pheromone trails are limited to the range [τ_{min}, τ_{max}]. The chance of algorithm stagnation is thus decreased by giving each connection some, although very small, probability of being chosen. In practice, heuristics exist for setting τ_{min} and τ_{max}. First, it can be shown that, because of the pheromone evaporation, the maximal possible pheromone trail level is limited to $\tau^*_{max} = 1/(\rho \cdot C(S^*))$, where S^* is the optimal solution. Based on this result, the global-best solution can be used to estimate τ_{max} by replacing S^* with $S_{global-best}$ in the equation for τ^*_{max}. For τ_{min} it is often enough to choose it as some constant factor lower than τ_{max} (see also [88] for details on possible ways to define τ_{min}).

   As a means for further increasing the exploration of solutions, $\mathcal{MM}$AS also uses the occasional re-initialization of the pheromone trails [87, 84, 88].

3. Instead of initializing the pheromones to a small amount, in $\mathcal{MM}$AS the pheromone trails are initialized to an estimate of the maximum allowed pheromone trail value (the estimate can be obtained by first generating some solution S' by a greedy construction heuristic and then replacing S' in the equation for τ^*_{max}). This leads to an additional diversification component

in the algorithm, because at the beginning the relative differences of the pheromone trails will not be very marked, which is different when initializing the pheromone trails to some very small value.

The structure of procedure `daemon_actions` in $\mathcal{MM}$AS is shown below:

```
Procedure daemon_actions
    for each S_k do local_search(S_k)
    S_current-best = best_solution (S_k)
    if (best(S_current-best, S_global-best))
        S_global-best = S_current-best
    end if
    S_best = decision(S_global-best, S_current-best)
    for each edge a_rs ∈ S_best do
        τ_rs = τ_rs + ·f(C(S_best))
        if (τ_rs < τ_min) τ_rs = τ_min
    end for
    if (stagnation_condition)
        for each edge a_rs do τ_rs = τ_max
    end if
end Procedure
```

4.4 Rank-based Ant System

The *rank-based Ant System* (AS_{rank}) [12] is another extension of the AS proposed by Bullnheimer, Hartl and Strauss in 1997. It incorporates the idea of ranking into the pheromone update, which is again developed offline by the daemon as follows:

1. The m ants are ranked according to decreasing quality of their solutions: $(S_1^{'}, \ldots, S_m^{'})$, with $S_1^{'}$ being the best solution built in the current generation.
2. The daemon deposits pheromone on the connections passed by the $\sigma - 1$ best ants (*elitist ants*). The amount of pheromone deposited directly depends on the ant's rank and on the quality of its solution.
3. The connections crossed by the global-best solution receive an additional amount of pheromone which depends on the quality of that solution. This pheromone deposit is considered to be the most important, hence, it receives a weight of σ.

This operation mode is put into effect by means of the following pheromone update rule, which is applied to every edge once all the pheromone trails have been evaporated:

$$\tau_{rs} \leftarrow \tau_{rs} + \sigma \cdot \Delta\tau_{rs}^{gb} + \Delta\tau_{rs}^{rank},$$

$$\text{where} \quad \Delta\tau_{rs}^{gb} = \begin{cases} f(C(S_{global-best})), & \text{if } a_{rs} \in S_{global-best} \\ 0, & \text{otherwise} \end{cases},$$

$$\Delta\tau_{rs}^{rank} = \begin{cases} \sum\limits_{\mu=1}^{\sigma-1} (\sigma-\mu)\cdot f(C(S_{\mu}^{'})), & \text{if } a_{rs} \in S_{\mu}^{'} \\ 0, & \text{otherwise} \end{cases}$$

Hence, the AS$_{rank}$ daemon procedure presents the following structure:

Procedure `daemon_actions`
    for `each` S_k **do** `local_search(`S_k`)` {`optional`}
    `rank` $(S_1, \ldots, S_m)$ `in decreasing order of solution quality into` $(S_1^{'}, \ldots, S_m^{'})$
    if `(best(`$S_1^{'}, S_{global-best}$`))`
        $S_{global-best} = S_1^{'}$
    end if
    for $\mu = 1$ `to` $(\sigma - 1)$ **do**
        for `each edge` $a_{rs} \in S_{\mu}^{'}$ **do**
            $\tau_{rs} = \tau_{rs} + (\sigma - \mu) \cdot f(C(S_{\mu}^{'}))$
        end for
    end for
    for `each edge` $a_{rs} \in S_{global-best}$ **do**
        $\tau_{rs} = \tau_{rs} + \sigma \cdot f(C(S_{global-best}))$
    end for
end Procedure

4.5 Best-Worst Ant System

Best-Worst Ant System (*BWAS*) [20], proposed by Cordón et al. in 1999, is an ACO algorithm which incorporates evolutionary computation concepts.[9] It constitutes another extension of AS, which uses its transition rule and pheromone evaporation mechanism (the evaporation is applied to every transition as in AS, AS$_{rank}$ and $\mathcal{MM}$AS). Besides, as done in $\mathcal{MM}$AS, BWAS always considers the systematic exploitation of local optimizers to improve the ants' solutions. At the core of BWAS, the three following daemon actions are found:

1. The *best-worst pheromone trail update rule*, which reinforces the edges contained in the global best solution. In addition, the update rule penalizes every connection of the worst solution generated in the current iteration, $S_{current-worst}$, that are not present in the global-best one through an additional evaporation of the pheromone trails. Hence, BWAS update rule becomes:

$$\tau_{rs} \leftarrow \tau_{rs} + \rho \cdot f(C(S_{global-best})), \quad \forall a_{rs} \in S_{global-best},$$

$$\tau_{rs} \leftarrow (1-\rho)\cdot\tau_{rs}, \quad \forall a_{rs} \in S_{current-worst} \text{ and } a_{rs} \notin S_{global-best}$$

[9] The relation between these two metaheuristics will be analyzed in Section 6.1.

2. A *pheromone trail mutation* is performed to introduce diversity in the search process. To do so, the pheromone trail associated to one of the transitions starting from each node (i.e., each row of the pheromone trail matrix) is mutated with probability P_m by considering any real-coded mutation operator.

   The original BWAS proposal applied an operator altering the pheromone trail of every mutated transition by adding or subtracting the same amount in each iteration. The mutation range $mut(it, \tau_{threshold})$, which depends on the average of the pheromone trails in the transitions of the global best solution, $\tau_{threshold}$, is less strong in the early stages of the algorithm—when there is no risk of stagnation—and stronger in the latter ones, when the danger of stagnation is stronger:

$$\tau'_{rs} \leftarrow \begin{cases} \tau_{rs} + mut(it, \tau_{threshold}), & \text{if } a = 0 \\ \tau_{rs} - mut(it, \tau_{threshold}), & \text{if } a = 1 \end{cases}$$

   with a being a random value in $\{0, 1\}$ and it being the current iteration.

3. As other ACO models, BWAS considers the re-initialization of the pheromone trails when it gets stuck, which is done by setting every pheromone trail to τ_0.

The BWAS `daemon_actions` procedure is as follows:

```
Procedure daemon_actions
    for each S_k do local_search(S_k)
    S_current-best = best_solution (S_k)
    if (best(S_current-best, S_global-best))
        S_global-best = S_current-best
    end if
    for each edge a_rs ∈ S_global-best do
        τ_rs = τ_rs + ρ · f(C(S_global-best))
        sum = sum + τ_rs
    end for
    τ_threshold = sum / |S_global-best|
    S_current-worst = worst_solution (S_k)
    for each edge a_rs ∈ S_current-worst and a_rs ∉ S_global-best do
        τ_rs = (1 − ρ) · τ_rs
    end for
    mut = mut(it,τ_threshold)
    for each node/component r ∈ {1,...,l} do
        z = generate_random_value_in_[0,1]
        if (z <= P_m)
                s = generate_random_value_in_[1,...,l]
                a = generate_random_value_in_[0,1]
                if (a = 0) τ_rs = τ_rs + mut
                else τ_rs = τ_rs - mut
        end if
```

```
    end for
    if (stagnation_condition)
        for each edge a_rs do τ_rs = τ_0
    end if
end Procedure
```

The interested reader can refer to the paper [19] for an analysis of the individual performance of each of the three BWAS components and the different combinations of them on the traveling salesman problem. In that study, it is shown that the version of BWAS that includes all three components performs better than other variants that include only a single component or a combination of two of the three components in the most of the cases. A similar study was done for BWAS applied to the quadratic assignment problem in [18]. Moreover, a new, well-performing ACO model called *Best-Worst Ant Colony System*, which is based on introducing the three BWAS components into the ACS, is proposed in [18].

5 Applications of Ant Colony Optimization

ACO algorithms have been applied to a large number of different combinatorial optimization problems. Current ACO applications follow into two classes of applications. The first class of problems comprises $\mathcal{NP}$-hard combinatorial optimization problems, for which classical techniques often show poor behavior. Characteristic for almost all successful ACO applications to these problems is that ants are coupled with local search algorithms that fine-tune the ants' solutions. The second class of applications comprises dynamic shortest path problems, where the problem instance under solution changes at algorithm run-time. These changes may affect the topology of the problem such as the availability of links etc. or, if the problem topology is fixed, characteristics such as edge costs, vary with time. In this case the algorithm has to adapt to the problem's dynamics. This latter class comprises applications of ACO to routing in telecommunication networks.

Instead of giving a detailed account of the various ACO applications, we shortly describe the (early) historical development of the applications; for more extensive overviews on applications of ACO we refer to [25, 26, 30, 31].

The first combinatorial problem tackled by an ACO algorithm was the traveling salesman problem (TSP), because this problem is probably the best known instance of an $\mathcal{NP}$-hard, constrained shortest path problem, thus, making it easy to adapt the real ant's behavior to solve it. Since the first application of AS in Dorigo's PhD dissertation in 1991, it became a common test-bed of several contributions proposing better performing ACO models than AS [27, 88, 12, 20].

Chronologically, the next two applications were the quadratic assignment problem (QAP) [60, 28] (the best performing ACO algorithms for the QAP are described in [88, 59]) and the job-shop scheduling problem [17] in 1994. Among the next applications are the first network routing applications, starting in 1996 with the work of Schoonderwoerd et al. [81] and the work on AntNet by Di Caro and Dorigo [24]. Already in 1997, one year after the publication of the first journal article on ACO in 1996 [28], the number of ACO applications started to increase strongly. Early applications from 1997 (although sometimes published later) include classical vehicle routing problems [11], sequential ordering [37], flow shop scheduling [82], and graph coloring [21] problems. Since then, many different authors have used the ACO meta-heuristic to solve a large number of combinatorial optimization problems such as shortest common supersequence, generalized assignment, set covering, multiple knapsack and constraint satisfaction problems, among others. The interested reader can find a summary of available applications as of end 2000 in [31]. Apart from the previous applications, ACO recently was used also for machine learning purposes, concretely to the design of learning algorithms for knowledge representation structures such as classical logic rules [76, 77], fuzzy logic rules [15, 3, 16] and Bayesian networks [23, 22], showing very promising results.

Nowadays, ACO reaches state-of-the-art results for several of the problems to which it was applied: QAP, sequential ordering, vehicle routing, scheduling, and packet-switched network routing, among others. The available computational results for many other problems are often very good and close to state-of-the-art, which is noteworthy, because many of these problems have already attracted a huge amount of research effort. Besides, the ACO meta-heuristic is being applied to novel real-world problems with very promising results (for an example, see the application to design combinatorial logic circuits in this issue [62]).

6 Relation of ACO to other metaheuristics

ACO algorithms have several similarities with other general optimization, learning and modeling approaches such as heuristic graph search, Monte Carlo simulation methods and neural networks that are analyzed in [26]. Moreover, the ACO meta-heuristic also shares several common aspects with other metaheuristics such as evolutionary computation and Estimation of Distribution Algorithms, on the one hand, and GRASP and multi-start local search, on the other hand.

6.1 Relation between ACO, Evolutionary Computation and Estimation of Distribution Algorithms

As already noted in [26], there are similarities between the operation mode of ACO algorithms and evolutionary algorithms such as the use of a population of individuals encoding problem solutions that are stochastically generated.

One main difference is that in evolutionary computation, the knowledge about the problem is contained in the current population, whilst in ACO it is stored in the memoristic structure collecting the pheromone trails. However, there exists also a specific class of evolutionary algorithms, the so-called Estimation of Distribution Algorithms (EDAs) [56], which, as ACO, are based on maintaining a memoristic structure that represents a probability distribution defined on the problem variables. This probability distribution is adapted at algorithm run-time by generating solutions based on the current distribution that are in a second step used to update the probability distribution. This process is repeated iteratively.

The best known EDA algorithm, the Population-Based Incremental Learning (PBIL) [5], is also the most similar to ACO algorithms [20], and specifically to the ACS. PBIL deals with a probability array $P = (p_1, \ldots, p_n)$ of dimension n equal to the number of problem variables, which encodes a probability distribution representing a prototype for good quality solutions and which is used to generate a population of possible solutions (binary arrays) in each iteration. This probability array undergoes adaption during the algorithm run. In the basic PBIL model, it is updated depending on the quality of the best solution generated in the current iteration using an ACS-like offline pheromone update rule. On the other hand, the components of P also suffer random mutations to avoid the chance of premature convergence.

The similarities between ACO algorithms and PBIL were also analyzed by Monmarché et al. [71]. They presented a common framework, which they called *Probabilistic Search Metaheuristic*, which also includes one further EDA model, the Bit-Simulated Crossover [91]. Besides, Dorigo et al. have also studied similarities between ACO and other techniques and they identified a larger group of algorithms with common characteristics that they called *Model-based Search* [94] (see also Section 7).

Finally, we should remind that the similarities between ACO and evolutionary computation-EDAs have motivated the integration of some aspects of the latter into ACO algorithms to improve their performance. This is the case, in particular, for BWAS (see Section 4.5). In fact, BWAS incorporates several EDA concepts, mainly from PBIL, like the mutation of pheromone trails and the punishment (removal of pheromone) by the worst solution of an iteration.

6.2 Relation between ACO, GRASP, and multi-start

Many metaheuristics for $\mathcal{NP}$-hard combinatorial optimization problems implement some form of a multi-start local search, where iteratively starting solutions for a local search are generated. These metaheuristics include iterated local search (ILS) [57], variable neighborhood search (VNS) [51], GRASP [34, 35], and memetic algorithms (MA) [72]. In fact, many ACO algorithms also belong to this class of metaheuristics.[10] In this case, the operation mode of ACO algorithms can be represented as follows:

While *(not stopping condition)* **do**

1. *Probabilistic construction of solutions by a colony of ants.*
2. *Local optimization of these solutions.*
3. *Offline pheromone trail update.*

A main difference between ACO and ILS, VNS, or MA is the way solutions are generated. While ACO constructs solutions from scratch, ILS, VNS, and MA

[10] Notice that this is true for most of the best performing ACO algorithms when applied to $\mathcal{NP}$-hard problems, but not for ACO algorithms applied, for example, to routing in telecommunication networks.

modify existing solutions by the application of appropriate operators that, when applied to one or several solutions, return a solution that represents a perturbation of the initial solution(s). Additionally, ACO uses an explizit form of memory (in the form of pheromone trails), which is typically not the case for the other three metaheuristics.

However, ACO shares the particularity of solution construction with GRASP that probabilistically constructs solutions and then applies local search to them. In GRASP, this two-phase process is repeated many times and the best solution found is returned. In the GRASP construction phase, at each construction step a candidate set is built by first ordering solution components according to some heuristic information, and then including the most promising components (greedy component). Next, a random decision is made among the components of the candidate set, typically with all members of the candidate set having the same probability of being selected (randomized component). A particularity of GRASP is that the heuristic information at each step takes into account the already available partial solution (adaptive component). From this description, it is clear that GRASP and ACO are different. While ACO uses memoristic information on the algorithm history provided by the pheromone trails, GRASP does not use such a type of information to bias the construction process. On the other side, ACO does not necessarily use heuristic information which is dependent on the partial solution, which is always done by GRASP. However, using adaptive (dynamic) heuristic information was shown to improve ACO performance for several applications; we refer to [31] for a discussion of this issue.

7 Theoretical developments

Current theoretical results on ACO concern mainly two aspects, (i) proofs on the convergence behavior of ACO algorithms and (ii) the establishment of formally well founded links between ACO algorithms and related algorithmic techniques.

Stützle and Dorigo have proven convergence properties for a class of ACO algorithms called $\mathrm{ACO}_{\tau_{min}}$ [86]. Characteristic of algorithms in $\mathrm{ACO}_{\tau_{min}}$ is that they use an elitist pheromone update strategy and lower limits on the values of any pheromone trail. For $\mathrm{ACO}_{\tau_{min}}$ they proved that for any small constant $\epsilon > 0$ and for a sufficiently large number of algorithm iterations t, the probability of finding at least once an optimal solution is $P^*(t) \geq 1 - \epsilon$ and that this probability tends to 1 for $t \to \infty$. The major importance of this result is that it applies to at least two of the (experimentally) most successful ACO algorithms: ACS [27] and $\mathcal{MM}$AS [88].

However, the first to proof convergence of a particular ACO algorithm, the so called graph-based Ant System (GBAS), was Gutjahr [48]. He proved that (i) for each $\epsilon > 0$, for a fixed ρ, and for a sufficiently large number of ants, the probability P that a fixed ant constructs the optimal solution at iteration t is $P \geq 1 - \epsilon$ for all $t \geq t_0$, with $t_0 = t_0(\epsilon)$ and (ii) for each $\epsilon > 0$, for a fixed number of ants, and for an evaporation rate ρ sufficiently close to zero, the probability P that a fixed ant constructs the optimal solution at iteration t is $P \geq 1 - \epsilon$ for all $t \geq t_0$, with $t_0 = t_0(\epsilon)$.

Yet, GBAS is an ACO algorithm which was never applied to any combinatorial optimization problem so far. Nevertheless, from the description of the algorithm and some calculations on parameter settings for low values of ϵ, which are implied by the theorem, the algorithm would result in an extremely slow progress towards good solutions and therefore, most likely, it will not be relevant to ACO practice.

In a very recent paper, Gutjahr [49] extended the convergence results for GBAS [48] to two variants of GBAS obtaining the same type of convergence results as for Simulated Annealing [50]: convergence of the current solution to an optimal solution with probability one. This result is established for two GBAS variants with (i) time-dependent pheromone evaporation and (ii) time-dependent lower pheromone bounds.

There are a number of differences between the proofs for $\mathrm{ACO}_{\tau_{min}}$ and GBAS. The most important concerns the type of convergence proved. While for $\mathrm{ACO}_{\tau_{min}}$ *convergence in value* is proven (that is, the algorithm will eventually find the optimal solution), for GBAS Gutjahr proves *convergence in solution* (that is, the algorithm will converge to a situation in which it generates the optimal solution over and over again). It is clear that for practical purposes it is enough to proof convergence in value, because basically all metaheuristics return the best solution found during the search process and therefore it is enough to generate the optimal solution at least once.

Formal relationships between ACO algorithms and stochastic gradient descent (SGD), a technique which has been extensively used in machine learning [80, 70], are established by Meuleau and Dorigo [66]. They show that a variant of AS can be interpreted as a SGD algorithm, which performs a gradient descent in the space of the pheromone trails. As mentioned in Section 6.1, Dorigo et al. [32] give an interpretation of ACO as a model-based search method. These kinds of techniques are characterized by the fact that candidate solutions are generated by a parameterized probabilistic model that is updated after each iteration by previously seen solutions. This feedback is used to bias the probability distribution for the following iterations. In this framework, links between ACO and EDAs, SGD, the cross-entropy method, and model-based genetic algorithms [73] are established.

8 New trends in Ant Colony Optimization

Two still very active research directions in ACO are the the application of ACO algorithms to challenging combinatorial optimization problems and the development of new, better performing algorithmic variants of ACO algorithms. In particular, the application of ACO to a steadily increasing variety of (in most cases $\mathcal{NP}$-hard) application problems accounts for the largest part of new contributions. Most noteworthy regarding the various applications are papers concerned with multi-objective problems and those to dynamic optimization problems. In multi-objective optimization problems, several competing objectives have to be optimized. First approaches to multi-objective optimization include the two-colony approach by Gambardella, Taillard and Agazzi to vehicle routing problems with time windows, where two hierarchically ordered objectives are considered [38]. Mariano and

Morales considered the application of ACO to a multi-objective optimization problem arising in the design of water irrigation networks. In their approach one colony is assigned to each objective and the objectives are ranked by importance [61]. Iredi, Merkle and Middendorf explored the application of ACO for finding Pareto-optimal solutions to a bi-criterion single-machine scheduling problem [53].

Some of the greatest successes of ACO algorithms so far were obtained in applications to highly dynamic problems like those to telecommunications routing with AntNet by Di Caro and Dorigo being the best performing variant [24]. For such applications, ACO algorithms are well suited, because they match very well the problem characteristics like stochastic state transitions and that only local information is available. Another research direction in dynamic problems is the application of ACO to time-varying variants of classical $\mathcal{NP}$-hard optimization problems like the TSP or the QAP. Examples are the papers by Guntsch and Middendorf [46, 45, 47] and Eyckelhof and Snoek [33].

With respect to algorithmic techniques in general, a hot recent trend is the development of hybrid techniques combining ideas from exact algorithms and metaheuristics. In ACO, one particular algorithms has actually be designed with as such a hybrid: the Approximate Nondeterministic Tree-Search (ANTS) procedure by Maniezzo [58] combines ACO algorithms with lower bounding techniques from mathematical programming. In particular, lower bounds on the cost of completing a partial solution are computed that are used as the heuristic information on the attractiveness of adding solution components. This offers advantages like the possible elimination of extensions of partial solutions, if these would lead to a larger cost than the best solution already found.

Some research efforts in ACO were also directed towards speeding up the algorithm by using parallel processing. ACO appears to be naturally suited for parallel processing because of the use of a population of solutions. However, the communication overhead through the exchange of the pheromone evaporation makes difficult the realization of fine-grained parallelization schemes. Therefore, most parallelization schemes focus on course-grained parallelization [13, 68, 69, 83], which is implemented typically in a way similar to the island-model in evolutionary algorithms [14]. One recent exception to these (rather standard) parallelization approaches is the adaptation of an ACO algorithm to run on reconfigurable processor arrays [64].

Finally, as ACO becomes more and more widely used, several researchers also have shifted the attention for examining the reasons of ACO's success, towards a deeper understanding of the search behavior. Stützle and Hoos have linked the reasons of ACO success to results of the search space analysis of combinatorial optimization problems. In particular, ACO appears to perform particularly well on problems which show a high correlation between the fitness of solutions and the distance to global optima (measured by the so called fitness-distance correlation [54]) [88]. Studies of ACO behavior on simple problems like shortest path problems or (polynomially solvable) permutation problems shed some light on the importance of some design features of ACO such as the quality-based pheromone update or different strategies for solution construction [29, 63]. The dynamics of ACO algorithms is studied by Merkle and Middendorf [65]. Finally, Blum, Samples and Zlochin have shown that particular definitions of the pheromone trails may lead to the unexpected situation that the performance of Ant System may even degrade at run time [8]. Also when using more performing ACO algorithms than AS, Blum and Sampels show that the way pheromone trails are defined has (as it may be expected) an enormous influence on ACO algorithms' behavior [7].

9 Concluding remarks

Nowadays, ACO is a well defined and good performing metaheuristic that is more and more often applied to solve a variety of complex combinatorial problems. In this paper, we have reviewed the underlying ideas of this approach that lead from the biological inspiration to the ACO metaheuristic. Most of the existing approaches have been described and some results regarding topics such as the relationship to other metaheuristics and theoretical aspects have been summarized. Moreover, we have identified some recent trends in the field, trying to put some light on the possible future development of ACO.

References

[1] E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. Simulated annealing. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 91–120. John Wiley & Sons, Chichester, 1997.

[2] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.

[3] R. Alcalá, J. Casillas, O. Cordón, and F. Herrera. Improvement to the cooperative rules methodology by using the Ant Colony System algorithm. *Mathware & Soft Computing*, 8(3):321–335, 2001.

[4] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, NY, 1996.

[5] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, pages 38–46. Morgan Kaufmann Publishers, Palo Alto, CA, 1995.

[6] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editor, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[7] C. Blum and M. Sampels. When model bias is stronger than selection pressure. In *Proceedings of PPSN-VII, Seventh International Conference on Parallel*

Problem Solving from Nature, Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany, 2002. To appear.

[8] C. Blum, M. Sampels, and M. Zlochin. On a particularity in model-based search. In W. B. Langdon et al., editor, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 35–42. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

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

[10] G. Brassard and P. Bratley. *Fundamentals of Algorithmics.* Prentice Hall, Englewood Cliffs, NJ, 1996.

[11] 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.

[12] 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.

[13] 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, The Netherlands, 1998.

[14] E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms.* Kluwer Academic Publishers, Boston, MA, 2000.

[15] J. Casillas, O. Cordón, and F. Herrera. Learning fuzzy rules using ant colony optimization algorithms. 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 13–21. IRIDIA, Université Libre de Bruxelles, Belgium, 2000.

[16] J. Casillas, O. Cordón, and F. Herrera. Different approaches to induce cooperation in fuzzy linguistic models under the COR methodology. In B. Bouchon-Meunier, J. Gutiérrez-Rios, L. Magdalena, and R.R. Yager, editors, *Technologies for Constructing Intelligent Systems 1. Tasks*, pages 321–334. Physica-Verlag, 2002.

[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, and F. Herrera. Analysis of the Best-Worst Ant System and its variants on the QAP. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 228–234. Springer Verlag, Berlin, Germany, 2002.

[19] O. Cordón, I. Fernández de Viana, and F. Herrera. Analysis of the Best-Worst Ant System and its variants on the TSP. *Mathware & Soft Computing*, this issue, 2002.

[20] 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. IRIDIA, Université Libre de Bruxelles, Belgium, 2000.

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

[22] L.M. de Campos, J.M. Fernández-Luna, J.A. Gámez, and J.M. Puerta. Ant colony optimization for learning bayesian networks. *International Journal of Approximate Reasoning*, 2002. to appear.

[23] L.M. de Campos, J.A. Gámez, and J.M. Puerta. Learning bayesian networks by ant colony optimisation: Searching in two different spaces. *Mathware & Soft Computing*, this issue, 2002.

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

[25] 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.

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

[27] 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.

[28] 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.

[29] M. Dorigo and T. Stützle. An experimental study of the simple ant colony optimization algorithm. In N. Mastorakis, editor, *2001 WSES International Conference on Evolutionary Computation (EC'01)*, pages 253–258. WSES Press, 2001.

[30] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, in preparation.

[31] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer Academic Publishers, To appear in 2002.

[32] M. Dorigo, M. Zlochin, N. Meuleau, and M. Birattari. Updating ACO pheromones using stochastic gradient ascent and cross-entropy methods. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2002*, volume 2279 of *Lecture Notes in Computer Science*, pages 21–30. Springer Verlag, Berlin, Germany, 2002.

[33] C. J. Eyckelhof and M. Snoek. Ant systems for a dynamic TSP: Ants caught in a traffic jam. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of ANTS2002 – From Ant Colonies to Artificial Ants: Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 88–99. Springer Verlag, Berlin, Germany, 2002.

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

[35] P. Festa and M. G. C. Resende. GRASP: An annotated bibliography. In P. Hansen and C. C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*. Kluwer Academic Publishers, 2001. in press.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

[41] F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, to appear in 2002.

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

[43] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

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

[45] M. Guntsch and M. Middendorf. An ant colony optimization approach to dynamic TSP. In L. Spector et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 860–867. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[46] M. Guntsch and M. Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In E.J.W. Boers et al., editor, *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, volume 2037 of *Lecture Notes in Computer Science*, pages 213–222. Springer Verlag, Berlin, Germany, 2001.

[47] M. Guntsch and M. Middendorf. Applying population based ACO to dynamic optimization problems. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 111–122. Springer Verlag, Berlin, Germany, 2002.

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

[49] W. J. Gutjahr. ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82(3):145–153, 2002.

[50] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of OR*, 13:311–329, 1988.

[51] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-Heuristics — Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[52] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[53] S. Iredi, D. Merkle, and M. Middendorf. Bi-criterion optimization with multi colony ant algorithms. In E. Zitzler et al., editor, *Evolutionary Multi-Criterion Optimization, First International Conference (EMO'01)*, volume 1993, pages 359–372. Springer Verlag, Berlin, Germany, 2001.

[54] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L.J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufman, 1995.

[55] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[56] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.

[57] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. Technical Report AIDA-00-06, FG Intellektik, FB Informatik, TU Darmstadt, Germany, November 2000. To appear in F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, Kluwer, 2002.

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

[59] 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.

[60] 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.

[61] C. E. Mariano and E. Morales. MOAQ: An Ant-Q algorithm for multiple objective optimization problems. In W. Banzhaf et al., editor, *Genetic and Evolutionary Computation Conference (GECCO-99)*, volume 1, pages 894–901. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[62] B. Mendoza and C.A. Coello. An approach based on the use of the Ant System to design combinatorial logic circuits. *Mathware & Soft Computing*, this issue, 2002.

[63] D. Merkle and M. Middendorf. On the behaviour of ant algorithms: Studies on simple problems. In *Proceedings of the 4th Metaheuristics International Conference (MIC'2001)*, pages 573–577, Porto, Portugal, 2001.

[64] D. Merkle and M. Middendorf. Fast ant colony optimization on runtime reconfigurable processor arrays. *Genetic Programming and Evolvable Machines*, 3(3), 2002. to appear.

[65] D. Merkle and M. Middendorf. Studies on the dynamics of ant colony optimization algorithms. In W. B. Langdon et al., editor, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 105–112. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[66] N. Meuleau and M. Dorigo. Ant colony optimization and stochastic gradient descent. *Artificial Life*, 8(2):103–121, 2002.

[67] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer Verlag, Berlin, Germany, 2000.

[68] 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.

[69] M. Middendorf, F. Reischle, and H. Schmeck. Multi colony ant algorithms. *Journal of Heuristics*, 8(3):305–320, 2002.

[70] T. Mitchell. *Machine Learning*. McGraw Hill, Boston, MA, 1997.

[71] N. Monmarché, E. Ramat, G. Dromel, M. Slimane, and G. Venturini. On the similarities between AS, BSC and PBIL: Toward the birth of a new metaheuristic. Technical Report E3i, 215, Ecole d'Ingénieurs en Informatique pour l'Industrie, Université de Tours, 1999.

[72] P. Moscato. Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 219–234. McGraw Hill, London, UK, 1999.

[73] H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (BGA). *Evolutionary Computation*, 1(4):335–360, 1993.

[74] I. H. Osman and J. P. Kelly, editors. *Meta-Heuristics: Theory & Applications*. Kluwer Academic Publishers, Boston, MA, 1996.

[75] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization – Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[76] R.S. Parpinelli, H.S. Lopes, and A.A. Freitas. An ant conlony based system for data mining: application to medical data. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 791–798. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[77] R.S. Parpinelli, H.S. Lopes, and A.A. Freitas. Data mining with an Ant Colony Optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 2002, in press.

[78] J. M. Pasteels, J.-L. Deneubourg, and S. Goss. Self-organization mechanisms in ant societies (I): Trail recruitment to newly discovered food sources. *Experientia Supplementum*, 54:155–175, 1987.

[79] C. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. McGraw Hill, London, UK, 1995.

[80] H. Robbins and H. Monroe. A stochastic approximation method. *Annals of Mathematics and Statistics*, 22:400–407, 1951.

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

[82] 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, Wissenschaftsverlag, Aachen, Germany, 1998.

[83] T. Stützle. Parallelization strategies for ant colony optimization. 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 722–731. Springer Verlag, Berlin, Germany, 1998.

[84] T. Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*, volume 220 of *DISKI*. Infix, Sankt Augustin, Germany, 1999.

[85] T. Stützle and M. Dorigo. ACO algorithms for the traveling salesman problem. In K. Miettinen, M. M. Mäkelä, P. Neittaanmäki, and J. Périaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 163–183. John Wiley & Sons, Chichester, UK, 1999.

[86] T. Stützle and M. Dorigo. A short convergence proof for a class of Ant Colony Optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 2002, in press.

[87] 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.

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

[89] T. Stützle and H.H. Hoos. Improving the Ant System: A detailed report on the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. Technical Report AIDA–96–12, FG Intellektik, FB Informatik, TU Darmstadt, Germany, August 1996.

[90] T. Stützle and S. Linke. Experiments with variants of ant algorithms. *Mathware & Soft Computing*, this issue, 2002.

[91] G. Syswerda. Simulated crossover in genetic algorithms. In *Foundations of Genetic Algorithms, Second Workshop (FOGA 2)*, pages 309–314, 1993.

[92] É. D. Taillard and L. M. Gambardella. Adaptive memories for the quadratic assignment problem. Technical Report IDSIA-87-97, IDSIA, Lugano, Switzerland, 1997.

[93] S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization.* Kluwer Academic Publishers, Boston, MA, 1999.

[94] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization. Technical Report TR/IRIDIA/2001-16, IRIDIA, Université Libre de Bruxelles, Belgium, 2001.