Available online at www.sciencedirect.com
SCIENCE DIRECT®
Physics of Life Reviews 2 (2005) 353–373
PHYSICS of LIFE reviews
www.elsevier.com/locate/plrev

Review

Ant colony optimization: Introduction and recent trends

Christian Blum [1]

ALBCOM, LSI, Universitat Politècnica de Catalunya, Jordi Girona 1-3, Campus Nord, 08034 Barcelona, Spain

Accepted 11 October 2005

Communicated by L. Perlovsky

Abstract

Ant colony optimization is a technique for optimization that was introduced in the early 1990's. The inspiring source of ant colony optimization is the foraging behavior of real ant colonies. This behavior is exploited in artificial ant colonies for the search of approximate solutions to discrete optimization problems, to continuous optimization problems, and to important problems in telecommunications, such as routing and load balancing. First, we deal with the biological inspiration of ant colony optimization algorithms. We show how this biological inspiration can be transfered into an algorithm for discrete optimization. Then, we outline ant colony optimization in more general terms in the context of discrete optimization, and present some of the nowadays best-performing ant colony optimization variants. After summarizing some important theoretical results, we demonstrate how ant colony optimization can be applied to continuous optimization problems. Finally, we provide examples of an interesting recent research direction: The hybridization with more classical techniques from artificial intelligence and operations research.

Keywords: Ant colony optimization; Discrete optimization; Hybridization

Contents

E-mail address: cblum@lsi.upc.edu (C. Blum).

[1] Christian Blum acknowledges support from the "Juan de la Cierva" program of the Spanish Ministry of Science and Technology of which he is a post-doctoral research fellow, and from the Spanish CICYT project TRACER (Grant TIC-2002-04498-C05-03).

1571-0645/$ – see front matter
doi:10.1016/j.plrev.2005.10.001

1. Introduction

Optimization problems are of high importance both for the industrial world as well as for the scientific world. Examples of practical optimization problems include train scheduling, time tabling, shape optimization, telecommunication network design, or problems from computational biology. The research community has simplified many of these problems in order to obtain scientific test cases such as the well-known traveling salesman problem (TSP) [61]. The TSP models the situation of a travelling salesman who is required to pass through a number of cities. The goal of the travelling salesman is to traverse these cities (visiting each city exactly once) so that the total travelling distance is minimal. Another example is the problem of protein folding, which is one of the most challenging problems in computational biology, molecular biology, biochemistry and physics. It consists of finding the functional shape or conformation of a protein in two- or three-dimensional space, for example, under simplified lattice models such as the hydrophobic-polar model [92]. The TSP and the protein folding problem under lattice models belong to an important class of optimization problems known as combinatorial optimization (CO).

According to Papadimitriou and Steiglitz [79], a CO problem $\mathcal{P} = (\mathcal{S}, f)$ is an optimization problem in which are given a finite set of objects $\mathcal{S}$ (also called the search space) and an objective function $f : \mathcal{S} \to \mathbb{R}^+$ that assigns a positive cost value to each of the objects $s \in \mathcal{S}$. The goal is to find an object of minimal cost value.[2] The objects are typically integer numbers, subsets of a set of items, permutations of a set of items, or graph structures. CO problems can be modelled as discrete optimization problems in which the search space is defined over a set of decision variables X_i, $i = 1, \ldots, n$, with discrete domains. Therefore, we will henceforth use the terms CO problem and discrete optimization problem interchangeably.

Due to the practical importance of CO problems, many algorithms to tackle them have been developed. These algorithms can be classified as either *complete* or *approximate* algorithms. Complete algorithms are guaranteed to find for every finite size instance of a CO problem an optimal solution in bounded time (see [77,79]). Yet, for CO problems that are $\mathcal{NP}$-hard [44], no polynomial time algorithm exists, assuming that $\mathcal{P} \neq \mathcal{NP}$. Therefore, complete methods might need exponential computation time in the worst-case. This often leads to computation times too high for practical purposes. Thus, the development of approximate methods—in which we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time—has received more and more attention in the last 30 years.

Ant colony optimization (ACO) [36] is one of the most recent techniques for approximate optimization. The inspiring source of ACO algorithms are real ant colonies. More specifically, ACO is inspired by the ants' foraging behavior. At the core of this behavior is the indirect communication between the ants by means of chemical pheromone trails, which enables them to find short paths between their nest and food sources. This characteristic of real ant colonies is exploited in ACO algorithms in order to solve, for example, discrete optimization problems.[3]

Depending on the point of view, ACO algorithms may belong to different classes of approximate algorithms. Seen from the artificial intelligence (AI) perspective, ACO algorithms are one of the most successful strands of swarm intelligence [16,17]. The goal of swarm intelligence is the design of intelligent multi-agent systems by taking inspiration from the collective behavior of social insects such as ants, termites, bees, wasps, and other animal societies such as flocks of birds or fish schools. Examples of "swarm intelligent" algorithms other than ACO are those for clustering

[2] Note that minimizing over an objective function f is the same as maximizing over $-f$. Therefore, every CO problem can be described as a minimization problem.

[3] Even though ACO algorithms were originally introduced for the application to discrete optimization problems, the class of ACO algorithms also comprises methods for the application to problems arising in networks, such as routing and load balancing (see, for example, [28]), and for the application to continuous optimization problems (see, for example, [86]). In Section 5 we will shortly deal with ACO algorithms for continuous optimization.

and data mining inspired by ants' cemetery building behavior [55,63], those for dynamic task allocation inspired by the behavior of wasp colonies [22], and particle swarm optimization [58].

Seen from the operations research (OR) perspective, ACO algorithms belong to the class of metaheuristics [13, 47,56]. The term *metaheuristic*, first introduced in [46], derives from the composition of two Greek words. *Heuristic* derives from the verb *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$) which means "to find", while the suffix *meta* means "beyond, in an upper level". Before this term was widely adopted, metaheuristics were often called *modern heuristics* [81]. In addition to ACO, other algorithms such as evolutionary computation, iterated local search, simulated annealing, and tabu search, are often regarded as metaheuristics. For books and surveys on metaheuristics see [13,47,56,81].

This review is organized as follows. In Section 2 we outline the origins of ACO algorithms. In particular, we present the foraging behavior of real ant colonies and show how this behavior can be transfered into a technical algorithm for discrete optimization. In Section 3 we provide a description of the ACO metaheuristic in more general terms, outline some of the most successful ACO variants nowadays, and list some representative examples of ACO applications. In Section 4, we discuss some important theoretical results. In Section 5, how ACO algorithms can be adapted to continuous optimization. Finally, Section 6 will give examples of a recent successful strand of ACO research, namely the hybridization of ACO algorithms with more classical AI and OR methods. In Section 7 we offer conclusions and an outlook to the future.

2. The origins of ant colony optimization

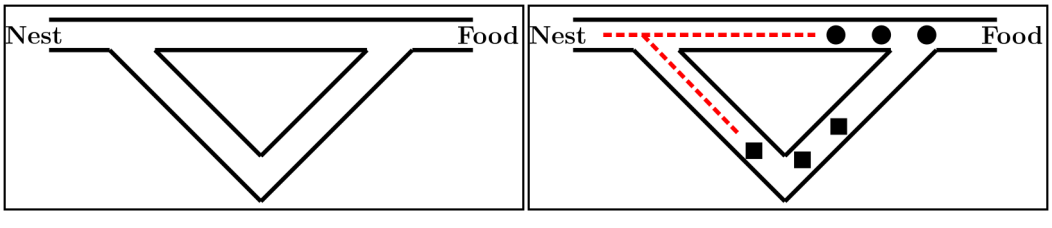
Marco Dorigo and colleagues introduced the first ACO algorithms in the early 1990's [30,34,35]. The development of these algorithms was inspired by the observation of ant colonies. Ants are social insects. They live in colonies and their behavior is governed by the goal of colony survival rather than being focused on the survival of individuals. The behavior that provided the inspiration for ACO is the ants' foraging behavior, and in particular, how ants can find shortest paths between food sources and their nest. When searching for food, ants initially explore the area surrounding their nest in a random manner. While moving, ants leave a chemical pheromone trail on the ground. Ants can smell pheromone. When choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromone that an ant leaves on the ground may depend on the quantity and quality of the food. The pheromone trails will guide other ants to the food source. It has been shown in [27] that the indirect communication between the ants via pheromone trails—known as *stigmergy* [49]—enables them to find shortest paths between their nest and food sources. This is explained in an idealized setting in Fig. 1.

As a first step towards an algorithm for discrete optimization we present in the following a discretized and simplified model of the phenomenon explained in Fig. 1. After presenting the model we will outline the differences between the model and the behavior of real ants. Our model consists of a graph $G = (V, E)$, where V consists of two nodes, namely v_s (representing the nest of the ants), and v_d (representing the food source). Furthermore, E consists of two links, namely e_1 and e_2, between v_s and v_d. To e_1 we assign a length of l_1, and to e_2 a length of l_2 such that $l_2 > l_1$. In other words, e_1 represents the short path between v_s and v_d, and e_2 represents the long path. Real ants deposit pheromone on the paths on which they move. Thus, the chemical pheromone trails are modeled as follows. We introduce an artificial pheromone value τ_i for each of the two links e_i, $i = 1, 2$. Such a value indicates the strength of the pheromone trail on the corresponding path. Finally, we introduce n_a artificial ants. Each ant behaves as follows: Starting from v_s (i.e., the nest), an ant chooses with probability

$$\mathbf{p}_i = \frac{\tau_i}{\tau_1 + \tau_2}, \quad i = 1, 2, \tag{1}$$

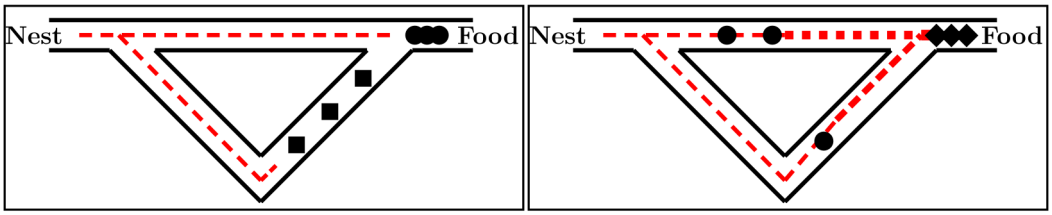
between path e_1 and path e_2 for reaching the food source v_d. Obviously, if $\tau_1 > \tau_2$, the probability of choosing e_1 is higher, and vice versa. For returning from v_d to v_s, an ant uses the same path as it chose to reach v_d,[4] and it changes the artificial pheromone value associated to the used edge. More in detail, having chosen edge e_i an ant changes the

[4] Note that this can be enforced because the setting is symmetric, i.e., the choice of a path for moving from v_s to v_d is equivalent to the choice of a path for moving from v_d to v_s.



(a) All ants are in the nest. There is no pheromone in the environment.

(b) The foraging starts. In probability, 50% of the ants take the short path (symbolized by circles), and 50% take the long path to the food source (symbolized by rhombs).



(c) The ants that have taken the short path have arrived earlier at the food source. Therefore, when returning, the probability to take again the short path is higher.

(d) The pheromone trail on the short path receives, in probability, a stronger reinforcement, and the probability to take this path grows. Finally, due to the evaporation of the pheromone on the long path, the whole colony will, in probability, use the short path.

Fig. 1. An experimental setting that demonstrates the shortest path finding capability of ant colonies. Between the ants' nest and the only food source exist two paths of different lengths. In the four graphics, the pheromone trails are shown as dashed lines whose thickness indicates the trails' strength.

artificial pheromone value τ_i as follows:

$$\tau_i \leftarrow \tau_i + \frac{Q}{l_i}, \tag{2}$$

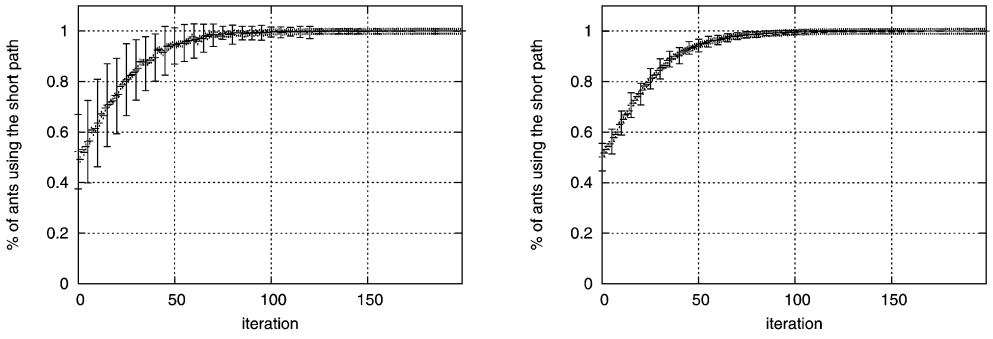
where the positive constant Q is a parameter of the model. In other words, the amount of artificial pheromone that is added depends on the length of the chosen path: the shorter the path, the higher the amount of added pheromone.

The foraging of an ant colony is in this model iteratively simulated as follows: At each step (or iteration) all the ants are initially placed in node v_s. Then, each ant moves from v_s to v_d as outlined above. As mentioned in the caption of Fig. 1(d), in nature the deposited pheromone is subject to an evaporation over time. We simulate this pheromone evaporation in the artificial model as follows:

$$\tau_i \leftarrow (1-\rho)\cdot\tau_i, \quad i = 1, 2. \tag{3}$$

The parameter $\rho \in (0, 1]$ is a parameter that regulates the pheromone evaporation. Finally, all ants conduct their return trip and reinforce their chosen path as outlined above.

We implemented this system and conducted simulations with the following settings: $l_1 = 1$, $l_2 = 2$, $Q = 1$. The two pheromone values were initialized to 0.5 each. Note that in our artificial system we cannot start with artificial pheromone values of 0. This would lead to a division by 0 in Eq. (1). The results of our simulations are shown in Fig. 2. They clearly show that over time the artificial colony of ants converges to the short path, i.e., after some time all ants use the short path. In the case of 10 ants (i.e., $n_a = 10$, Fig. 2(a)) the random fluctuations are bigger than in the case of 100 ants (Fig. 2(b)). This indicates that the shortest path finding capability of ant colonies results from a cooperation between the ants.



(a) Colony size: 10 ants (b) Colony size: 100 ants

Fig. 2. Results of 100 independent runs (error bars show the standard deviation for each 5th iteration). The x-axis shows the iterations, and the y-axis the percentage of the ants using the short path.

The main differences between the behavior of the real ants and the behavior of the artificial ants in our model are as follows:

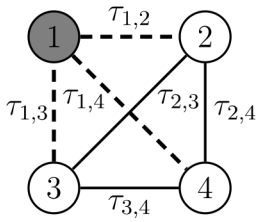
(1) While real ants move in their environment in an asynchronous way, the artificial ants are synchronized, i.e., at each iteration of the simulated system, each of the artificial ants moves from the nest to the food source and follows the same path back.
(2) While real ants leave pheromone on the ground whenever they move, artificial ants only deposit artificial pheromone on their way back to the nest.
(3) The foraging behavior of real ants is based on an implicit evaluation of a solution (i.e., a path from the nest to the food source). By implicit solution evaluation we mean the fact that shorter paths will be completed earlier than longer ones, and therefore they will receive pheromone reinforcement more quickly. In contrast, the artificial ants evaluate a solution with respect to some quality measure which is used to determine the strength of the pheromone reinforcement that the ants perform during their return trip to the nest.

2.1. Ant System for the TSP: The first ACO algorithm

The model that we used in the previous section to simulate the foraging behavior of real ants in the setting of Fig. 1 cannot directly be applied to CO problems. This is because we associated pheromone values directly to solutions to the problem (i.e., one parameter for the short path, and one parameter for the long path). This way of modeling implies that the solutions to the considered problem are already known. However, in combinatorial optimization we intend to *find* an unknown optimal solution. Thus, when CO problems are considered, pheromone values are associated to solution components instead. Solution components are the units from which solutions to the tackled problem are assembled. Generally, the set of solution components is expected to be finite and of moderate size. As an example we present the first ACO algorithm, called Ant System (AS) [30,35], applied to the TSP, which we mentioned in the introduction and which we define in more detail in the following:

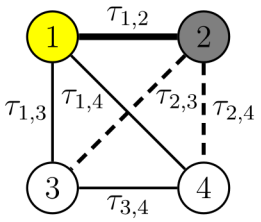
Definition 1. In the TSP is given a completely connected, undirected graph $G = (V, E)$ with edge-weights. The nodes V of this graph represent the cities, and the edge weights represent the distances between the cities. The goal is to find a closed path in G that contains each node exactly once (henceforth called a tour) and whose length is minimal. Thus, the search space $\mathcal{S}$ consists of all tours in G. The objective function value $f(s)$ of a tour $s \in \mathcal{S}$ is defined as the sum of the edge-weights of the edges that are in s. The TSP can be modelled in many different ways as a discrete optimization problem. The most common model consists of a binary decision variable X_e for each edge in G. If in a solution $X_e = 1$, then edge e is part of the tour that is defined by the solution.

Concerning the AS approach, the edges of the given TSP graph can be considered solution components, i.e., for each $e_{i,j}$ is introduced a pheromone value $\tau_{i,j}$. The task of each ant consists in the construction of a feasible TSP



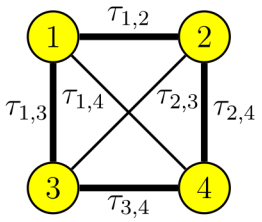
$$\mathbf{p}(e_{1,j}) = \frac{\tau_{1,j}}{\tau_{1,2} + \tau_{1,3} + \tau_{1,4}}$$

(a) First step of the solution construction.



$$\mathbf{p}(e_{2,j}) = \frac{\tau_{2,j}}{\tau_{2,3} + \tau_{2,4}}$$

(b) Second step of the solution construction.



(c) The complete solution after the final construction step.

Fig. 3. Example of the solution construction for a TSP problem consisting of 4 cities (modelled by a graph with 4 nodes; see Definition 1). The solution construction starts by randomly choosing a start node for the ant; in this case node 1. Figures (a) and (b) show the choices of the first, respectively the second, construction step. Note that in both cases the current node (i.e., location) of the ant is marked by dark gray color, and the already visited nodes are marked by light gray color (respectively yellow color, in the online version of this article). The choices of the ant (i.e., the edges she may traverse) are marked by dashed lines. The probabilities for the different choices (according to Eq. (4)) are given underneath the graphics. Note that after the second construction step, in which we exemplary assume the ant to have selected node 4, the ant can only move to node 3, and then back to node 1 in order to close the tour.

solution, i.e., a feasible tour. In other words, the notion of *task of an ant* changes from *"choosing a path from the nest to the food source"* to *"constructing a feasible solution to the tackled optimization problem"*. Note that with this change of task, the notions of nest and food source loose their meaning.

Each ant constructs a solution as follows. First, one of the nodes of the TSP graph is randomly chosen as start node. Then, the ant builds a tour in the TSP graph by moving in each construction step from its current node (i.e., the city in which she is located) to another node which she has not visited yet. At each step the traversed edge is added to the solution under construction. When no unvisited nodes are left the ant closes the tour by moving from her current node to the node in which she started the solution construction. This way of constructing a solution implies that an ant has a memory T to store the already visited nodes. Each solution construction step is performed as follows. Assuming the ant to be in node v_i, the subsequent construction step is done with probability

$$\mathbf{p}(e_{i,j}) = \frac{\tau_{i,j}}{\sum_{\{k \in \{1,\ldots,|V|\} | v_k \notin T\}} \tau_{i,k}}, \quad \forall j \in \{1,\ldots,|V|\},\ v_j \notin T. \tag{4}$$

For an example of such a solution construction see Fig. 3.

Once all ants of the colony have completed the construction of their solution, pheromone evaporation is performed as follows:

$$\tau_{i,j} \leftarrow (1-\rho) \cdot \tau_{i,j}, \quad \forall \tau_{i,j} \in \mathcal{T}, \tag{5}$$

where $\mathcal{T}$ is the set of all pheromone values. Then the ants perform their return trip. Hereby, an ant—having constructed a solution s—performs for each $e_{i,j} \in s$ the following pheromone deposit:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \frac{Q}{f(s)}, \tag{6}$$

where Q is again a positive constant and $f(s)$ is the objective function value of the solution s. As explained in the previous section, the system is iterated—applying n_a ants per iteration—until a stopping condition (e.g., a time limit) is satisfied.

Even though the AS algorithm has proved that the ants foraging behavior can be transferred into an algorithm for discrete optimization, it was generally found to be inferior to state-of-the-art algorithms. Therefore, over the years several extensions and improvements of the original AS algorithm were introduced. They are all covered by the definition of the ACO metaheuristic, which we will outline in the following section.

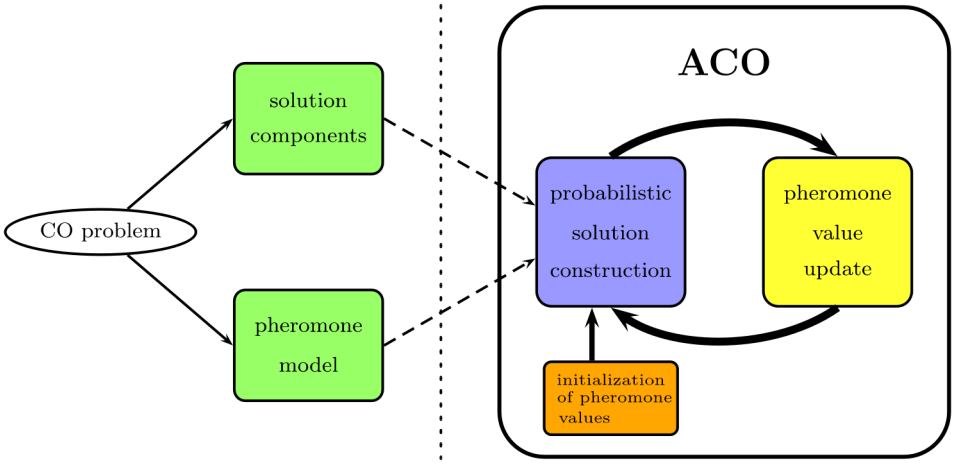


Fig. 4. The working of the ACO metaheuristic.

3. The ant colony optimization metaheuristic

The ACO metaheuristic, as we know it today, was first formalized by Dorigo and colleagues in 1999 [32]. The recent book by Dorigo and Stützle gives a more comprehensive description [36]. The definition of the ACO metaheuristic covers most—if not all—existing ACO variants for discrete optimization problems. In the following, we give a general description of the framework of the ACO metaheuristic.

The basic way of working of an ACO algorithm is graphically shown in Fig. 4. Given a CO problem to be solved, one first has to derive a finite set $\mathcal{C}$ of solution components which are used to assemble solutions to the CO problem. Second, one has to define a set of *pheromone values* $\mathcal{T}$. This set of values is commonly called the *pheromone model*, which is—seen from a technical point of view—a parameterized probabilistic model. The pheromone model is one of the central components of the ACO metaheuristic. The pheromone values $\tau_i \in \mathcal{T}$ are usually associated to solution components.[5] The pheromone model is used to probabilistically generate solutions to the problem under consideration by assembling them from the set of solution components. In general, the ACO approach attempts to solve an optimization problem by iterating the following two steps:

- candidate solutions are constructed using a pheromone model, that is, a parameterized probability distribution over the solution space;
- the candidate solutions are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solutions.

The pheromone update aims to concentrate the search in regions of the search space containing high quality solutions. In particular, the reinforcement of solution components depending on the solution quality is an important ingredient of ACO algorithms. It implicitly assumes that good solutions consist of good solution components. To learn which components contribute to good solutions can help assembling them into better solutions.

Algorithm 1. *Ant colony optimization (ACO)*

while termination conditions not met **do**
  ScheduleActivities
    AntBasedSolutionConstruction() {see Algorithm 2}
    PheromoneUpdate()
    DaemonActions() {optional}

[5] Note that the description of the ACO metaheuristic as given for example in [32] allows pheromone values also to be associated with links between solution components. However, for the purpose of this introduction it is sufficient to assume pheromone values associated to components.

end ScheduleActivities
end while

In the following, we give a more technical description of the general ACO metaheuristic whose framework is shown in Algorithm 1. ACO is an iterative algorithm whose run time is controlled by the principal while-loop of Algorithm 1. In each iteration the three algorithmic components AntBasedSolutionConstruction(), PheromoneUpdate(), and DaemonActions()—gathered in the ScheduleActivities construct—must be scheduled. The ScheduleActivities construct does not specify how these three activities are scheduled and synchronized. This is up to the algorithm designer. In the following we outline these three algorithmic components in detail.

Algorithm 2. *Procedure AntBasedSolutionConstruction() of Algorithm 1*

$s = \langle\rangle$
Determine $\mathcal{N}(s)$
while $\mathcal{N}(s) \neq \emptyset$ **do**
  $c \leftarrow$ ChooseFrom($\mathcal{N}(s)$)
  $s \leftarrow$ extend s by appending solution component c
  Determine $\mathcal{N}(s)$
end while

AntBasedSolutionConstruction() (see also Algorithm 2): Artificial ants can be regarded as probabilistic constructive heuristics that assemble solutions as sequences of solution components. The finite set of solution components $\mathcal{C} = \{c_1, \ldots, c_n\}$ is hereby derived from the discrete optimization problem under consideration. For example, in the case of AS applied to the TSP (see previous section) each edge of the TSP graph was considered a solution component. Each solution construction starts with an empty sequence $s = \langle\rangle$. Then, the current sequence s is at each construction step extended by adding a feasible solution component from the set $\mathcal{N}(s) \subseteq \mathcal{C} \setminus s$.[6] The specification of $\mathcal{N}(s)$ depends on the solution construction mechanism. In the example of AS applied to the TSP (see previous section) the solution construction mechanism restricted the set of traversable edges to the ones that connected the ants' current node to unvisited nodes. The choice of a solution component from $\mathcal{N}(s)$ (see function ChooseFrom($\mathcal{N}(s)$) in Algorithm 2) is at each construction step performed probabilistically with respect to the pheromone model. In most ACO algorithms the respective probabilities—also called the *transition probabilities*—are defined as follows:

$$\mathbf{p}(c_i \mid s) = \frac{[\tau_i]^\alpha \cdot [\eta(c_i)]^\beta}{\sum_{c_j \in \mathcal{N}(s)} [\tau_j]^\alpha \cdot [\eta(c_j)]^\beta}, \quad \forall c_i \in \mathcal{N}(s), \tag{7}$$

where η is an optional weighting function, that is, a function that, sometimes depending on the current sequence, assigns at each construction step a heuristic value $\eta(c_j)$ to each feasible solution component $c_j \in \mathcal{N}(s)$. The values that are given by the weighting function are commonly called the *heuristic information*. Furthermore, the exponents α and β are positive parameters whose values determine the relation between pheromone information and heuristic information. In the previous sections' TSP example, we chose not to use any weighting function η, and we have set α to 1. It is interesting to note that by implementing the function ChooseFrom($\mathcal{N}(s)$) in Algorithm 2 such that the solution component that maximizes Eq. (7) is chosen deterministically (i.e., $c \leftarrow \text{argmax}\{\eta(c_i) \mid c_i \in \mathcal{N}(s)\}$), we obtain a deterministic greedy algorithm.

PheromoneUpdate(): Different ACO variants mainly differ in the update of the pheromone values they apply. In the following, we outline a general pheromone update rule in order to provide the basic idea. This pheromone update rule consists of two parts. First, a *pheromone evaporation*, which uniformly decreases all the pheromone values, is performed. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a sub-optimal region. It implements a useful form of *forgetting*, favoring the exploration of new areas in the search space. Second, one or more solutions from the current and/or from earlier iterations are used to

[6] Note that for this set-operation the sequence s is regarded as an ordered set.

increase the values of pheromone trail parameters on solution components that are part of these solutions:

$$\tau_i \leftarrow (1-\rho)\cdot\tau_i + \rho\cdot\sum_{\{s\in\mathcal{S}_{\text{upd}}|c_i\in s\}} w_s\cdot F(s), \tag{8}$$

for $i=1,\ldots,n$. Hereby, $\mathcal{S}_{\text{upd}}$ denotes the set of solutions that are used for the update. Furthermore, $\rho\in(0,1]$ is a parameter called evaporation rate, and $F:\mathcal{S}\mapsto\mathbb{R}^+$ is a so-called quality function such that $f(s)<f(s')\Longrightarrow F(s)\geqslant F(s'),\forall s\neq s'\in\mathcal{S}$. In other words, if the objective function value of a solution s is better than the objective function value of a solution s', the quality of solution s will be at least as high as the quality of solution s'. Eq. (8) also allows an additional weighting of the quality function, i.e., $w_s\in\mathbb{R}^+$ denotes the weight of a solution s.

Instantiations of this update rule are obtained by different specifications of $\mathcal{S}_{\text{upd}}$ and by different weight settings. In many cases, $\mathcal{S}_{\text{upd}}$ is composed of some of the solutions generated in the respective iteration (henceforth denoted by $\mathcal{S}_{\text{iter}}$) and the best solution found since the start of the algorithm (henceforth denoted by s_{bs}). Solution s_{bs} is often called the best-so-far solution. A well-known example is the *AS-update* rule, that is, the update rule of AS (see also Section 2.1). The AS-update rule, which is well-known due to the fact that AS was the first ACO algorithm to be proposed in the literature, is obtained from update rule (8) by setting

$$\mathcal{S}_{\text{upd}}\leftarrow\mathcal{S}_{\text{iter}}\quad\text{and}\quad w_s=1\quad\forall s\in\mathcal{S}_{\text{upd}}, \tag{9}$$

that is, by using all the solutions that were generated in the respective iteration for the pheromone update, and by setting the weight of each of these solutions to 1. An example of a pheromone update rule that is more used in practice is the *IB-update* rule (where IB stands for *iteration-best*). The IB-update rule is given by:

$$\mathcal{S}_{\text{upd}}\leftarrow\left\{s_{\text{ib}}=\text{argmax}\left\{F(s)\mid s\in\mathcal{S}_{\text{iter}}\right\}\right\}\quad\text{with } w_{s_{\text{ib}}}=1, \tag{10}$$

that is, by choosing only the best solution generated in the respective iteration for updating the pheromone values. This solution, denoted by s_{ib}, is weighted by 1. The IB-update rule introduces a much stronger bias towards the good solutions found than the AS-update rule. However, this increases the danger of premature convergence. An even stronger bias is introduced by the *BS-update* rule, where BS refers to the use of the best-so-far solution s_{bs}. In this case, $\mathcal{S}_{\text{upd}}$ is set to $\{s_{\text{bs}}\}$ and s_{bs} is weighted by 1, that is, $w_{s_{\text{bs}}}=1$. In practice, ACO algorithms that use variations of the IB-update or the BS-update rule and that additionally include mechanisms to avoid premature convergence achieve better results than algorithms that use the AS-update rule. Examples are given in the following section.

DaemonActions(): Daemon actions can be used to implement centralized actions which cannot be performed by single ants. Examples are the application of local search methods to the constructed solutions, 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 may decide to deposit extra pheromone on the solution components that belong to the best solution found so far.

3.1. Successful ACO variants

Even though the original AS algorithm achieved encouraging results for the TSP problem, it was found to be inferior to state-of-the-art algorithms for the TSP as well as for other CO problems. Therefore, several extensions and improvements of the original AS algorithm were introduced over the years. In the following we outline the basic features of the ACO variants listed in Table 1.

Table 1
A selection of ACO variants

ACO variant	Authors	Main reference
Elitist AS (EAS)	Dorigo	[30]
	Dorigo, Maniezzo, and Colorni	[35]
Rank-based AS (RAS)	Bullnheimer, Hartl, and Strauss	[21]
$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS)	Stützle and Hoos	[91]
Ant Colony System (ACS)	Dorigo and Gambardella	[33]
Hyper-Cube Framework (HCF)	Blum and Dorigo	[11]

A first improvement over AS was obtained by the Elitist AS (EAS) [30,35], which is obtained by instantiating pheromone update rule (8) by setting $\mathcal{S}_{\text{upd}} \leftarrow \mathcal{S}_{\text{iter}} \cup \{s_{\text{bs}}\}$, that is, by using all the solutions that were generated in the respective iteration, and in addition the best-so-far solution, for updating the pheromone values. The solution weights are defined as $w_s = 1\ \forall s \in \mathcal{S}_{\text{iter}}$. Only the weight of the best-so-far solution may be higher: $w_{s_{\text{bs}}} \geqslant 1$. The idea is hereby to increase the exploitation of the best-so-far solution by introducing a strong bias towards the solution components it contains.

Another improvement over AS is the Rank-based AS (RAS) proposed in [21]. The pheromone update of RAS is obtained from update rule (8) by filling $\mathcal{S}_{\text{upd}}$ with the best $m-1$ (where $m-1 \leqslant n_a$) solutions from $\mathcal{S}_{\text{iter}}$, and by additionally adding the best-so-far solution s_{bs} to $\mathcal{S}_{\text{upd}}$. The weights of the solutions are set as $w_s = m - r_s$ $\forall s \in \mathcal{S}_{\text{upd}} \setminus \{s_{\text{bs}}\}$, where r_s is the rank of solution s. Finally, the weight $w_{s_{\text{bs}}}$ of solution s_{bs} is set to m. This means that at each iteration the best-so-far solution has the highest influence on the pheromone update, while a selection of the best solutions constructed at that current iteration influences the update depending on their ranks.

One of the most successful ACO variants today is $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [91], which is characterized as follows. Depending on some convergence measure, at each iteration either the IB-update or the BS-update rule (both as explained in the previous section) are used for updating the pheromone values. At the start of the algorithm the IB-update rule is used more often, while during the run of the algorithm the frequency with which the BS-update rule is used increases. $\mathcal{MM}$AS algorithms use an explicit lower bound $\tau_{\min} > 0$ for the pheromone values. In addition to this lower bound, $\mathcal{MM}$AS algorithms use $F(s_{\text{bs}})/\rho$ as an upper bound to the pheromone values. The value of this bound is updated each time a new best-so-far solution is found by the algorithm.

Ant Colony System (ACS), which was introduced in [33], differs from the original AS algorithm in more aspects than just in the pheromone update. First, instead of choosing at each step during a solution construction the next solution component according to Eq. (7), an artificial ant chooses, with probability q_0, the solution component that maximizes $[\tau_i]^\alpha \cdot [\eta(c_i)]^\beta$, or it performs, with probability $1-q_0$, a probabilistic construction step according to Eq. (7). This type of solution construction is called *pseudo-random proportional*. Second, ACS uses the BS-update rule with the additional particularity that the pheromone evaporation is only applied to values of pheromone trail parameters that belong to solution components that are in s_{bs}. Third, after each solution construction step, the following additional pheromone update is applied to the pheromone value τ_i whose corresponding solution component c_i was added to the solution under construction:

$$\tau_i \leftarrow (1-\xi)\cdot\tau_i + \xi\cdot\tau_0, \tag{11}$$

where τ_0 is a small positive constant such that $F_{\min} \geqslant \tau_0 \geqslant c$, $F_{\min} \leftarrow \min\{F(s) \mid s \in \mathcal{S}\}$, and c is the initial value of the pheromone values. In practice, the effect of this local pheromone update is to decrease the pheromone values on the visited solution components, making in this way these components less desirable for the following ants. This mechanism increases the exploration of the search space within each iteration.

One of the most recent developments is the Hyper-Cube Framework (HCF) for ACO algorithms [11]. Rather than being an ACO variant, the HCF is a framework for implementing ACO algorithms which is characterized by a pheromone update that is obtained from update rule (8) by defining the weight of each solution in $\mathcal{S}_{\text{upd}}$ to be $(\sum_{\{s\in\mathcal{S}_{\text{upd}}\}} F(s))^{-1}$. Remember that in Eq. (8) solutions are weighted. The set $\mathcal{S}_{\text{upd}}$ can be composed in any possible way. This means that ACO variants such as AS, ACS, or $\mathcal{MM}$AS can be implemented in the HCF. The HCF comes with several benefits. On the practical side, the new framework automatically handles the scaling of the objective function values and limits the pheromone values to the interval $[0, 1]$.[7] On the theoretical side, the new framework allows to prove that in the case of an AS algorithm applied to unconstrained problems, the average quality of the solutions produced continuously increases in expectation over time. The name Hyper-Cube Framework stems from the fact that with the weight setting as outlined above, the pheromone update can be interpreted as a shift in a hypercube (see [11]).

In addition to the ACO variants outlined above, the ACO community has developed additional algorithmic features for improving the search process performed by ACO algorithms. A prominent example are so-called candidate list strategies. A candidate list strategy is a mechanism to restrict the number of available choices at each solution

[7] Note that in standard ACO variants the upper bound of the pheromone values depends on the pheromone update and on the problem instance that is tackled.

construction step. Usually, this restriction applies to a number of the best choices with respect to their transition probabilities (see Eq. (7)). For example, in the case of the application of ACS to the TSP, the restriction to the closest cities at each construction step both improved the final solution quality and led to a significant speedup of the algorithm (see [40]). The reasons for this are as follows: First, in order to construct high quality solutions it is often enough to consider only the "promising" choices at each construction step. Second, to consider fewer choices at each construction step speeds up the solution construction process, because the reduced number of choices reduces the computation time needed to make a choice.

3.2. Applications of ACO algorithms to discrete optimization problems

As mentioned before, ACO was introduced by means of the proof-of-concept application to the TSP. Since then, ACO algorithms have been applied to many discrete optimization problems. First, classical problems other than the TSP, such as assignment problems, scheduling problems, graph coloring, the maximum clique problem, or vehicle routing problems were tackled. More recent applications include, for example, cell placement problems arising in circuit design, the design of communication networks, or bioinformatics problems. In recent years some researchers have also focused on the application of ACO algorithms to multi-objective problems and to dynamic or stochastic problems.

Especially the bioinformatics and biomedical fields shows an increasing interest in ACO. Recent applications of ACO to problems arising in these areas include the applications to protein folding [83,84], to multiple sequence alignment [76], and to the prediction of major histocompatibility complex (MHC) class II binders [57]. The protein folding problem is one of the most challenging problems in computational biology, molecular biology, biochemistry and physics. It consists of finding the functional shape or conformation of a protein in two- or three-dimensional space, for example, under simplified lattice models such as the hydrophobic-polar model. The ACO algorithm proposed in [84] is reported to perform better than existing state-of-the-art algorithms when proteins are concerned whose native conformations do not contain structural nuclei at the ends, but rather in the middle of the sequence. Multiple sequence alignment concerns the alignment of several protein or DNA sequences in order to find similarities among them. This is done, for example, in order to determine the differences in the same protein coming from different species. This information might, for example, support the inference of phylogenetic trees. The ACO algorithm proposed in [76] is reported to scale well with growing sequence sizes. Finally, the prediction of the binding ability of antigen peptides to major MHC class II molecules are important in vaccine development. The performance of the ACO algorithm proposed in [57] for tackling this problem is reported to be comparable to the well-known Gibbs approach.

ACO algorithms are currently among the state-of-the-art methods for solving, for example, the sequential ordering problem [41], the resource constraint project scheduling problem [71], the open shop scheduling problem [9], and the 2D and 3D hydrophobic polar protein folding problem [84]. In Table 2 we provide a list of representative ACO applications. For a more comprehensive overview that also covers the application of ant-based algorithms to routing in telecommunication networks we refer the interested reader to [36].

4. Theoretical results

The first theoretical problem considered was the one concerning convergence. The question is: will a given ACO algorithm find an optimal solution when given enough resources? This is an interesting question, because ACO algorithms are stochastic search procedures in which the pheromone update could prevent them from ever reaching an optimum. Two different types of convergence were considered: *convergence in value* and *convergence in solution*. Convergence in value concerns the probability of the algorithm generating an optimal solution at least once. On the contrary, convergence in solution concerns the evaluation of the probability that the algorithm reaches a state which keeps generating the same optimal solution. The first convergence proofs concerning an algorithm called graph-based ant system (GBAS) were presented by Gutjahr in [53,54]. A second strand of work on convergence focused on a class of ACO algorithms that are among the best-performing in practice, namely, algorithms that apply a positive lower bound $\tau_{\min}$ to all pheromone values. The lower bound prevents that the probability to generate any solution becomes zero. This class of algorithms includes ACO variants such as ACS and $\mathcal{MM}$AS. Dorigo and Stützle, first in [90] and later in [36], presented a proof for the convergence in value, as well as a proof for the convergence in solution, for algorithms from this class.

Table 2
A representative selection of ACO applications

Problem	Authors	Reference
Traveling salesman problem	Dorigo, Maniezzo, and Colorni	[30,34,35]
	Dorigo and Gambardella	[33]
	Stützle and Hoos	[91]
Quadratic assignment problem	Maniezzo	[64]
	Maniezzo and Colorni	[66]
	Stützle and Hoos	[91]
Scheduling problems	Stützle	[89]
	den Besten, Stützle, and Dorigo	[26]
	Gagné, Price, and Gravel	[39]
	Merkle, Middendorf, and Schmeck	[71]
	Blum (respectively, Blum and Sampels)	[9,14]
Vehicle routing problems	Gambardella, Taillard, and Agazzi	[42]
	Reimann, Doerner, and Hartl	[82]
Timetabling	Socha, Sampels, and Manfrin	[87]
Set packing	Gandibleux, Delorme, and T'Kindt	[43]
Graph coloring	Costa and Hertz	[24]
Shortest supersequence problem	Michel and Middendorf	[74]
Sequential ordering	Gambardella and Dorigo	[41]
Constraint satisfaction problems	Solnon	[88]
Data mining	Parpinelli, Lopes, and Freitas	[80]
Maximum clique problem	Bui and Rizzo Jr	[20]
Edge-disjoint paths problem	Blesa and Blum	[7]
Cell placement in circuit design	Alupoaei and Katkoori	[1]
Communication network design	Maniezzo, Boschetti, and Jelasity	[65]
Bioinformatics problems	Shmygelska, Aguirre-Hernández, and Hoos	[83]
	Moss and Johnson	[76]
	Karpenko, Shi, and Dai	[57]
	Shmygelska and Hoos	[84]
Industrial problems	Bautista and Pereira	[2]
	Silva, Runkler, Sousa, and Palm	[85]
	Gottlieb, Puchta, and Solnon	[48]
	Corry and Kozan	[23]
Multi-objective problems	Guntsch and Middendorf	[52]
	Lopéz-Ibáñez, Paquete, and Stützle	[62]
	Doerner, Gutjahr, Hartl, Strauss, and Stummer	[29]
Dynamic (respectively, stochastic) problems	Guntsch and Middendorf	[51]
	Bianchi, Gambardella, and Dorigo	[4]
Music	Guéret, Monmarché, and Slimane	[50]

Recently, researchers have been dealing with the relation of ACO algorithms to other methods for learning and optimization. One example is the work presented in [6] that relates ACO to the fields of optimal control and reinforcement learning. A more prominent example is the work that aimed at finding similarities between ACO algorithms and other probabilistic learning algorithms such as stochastic gradient ascent (SGA), and the cross-entropy (CE) method. Zlochin et al. [96] proposed a unifying framework called *model-based search* (MBS) for this type of algorithms. Meuleau and Dorigo have shown in [72] that the pheromone update as outlined in the proof-of-concept application

to the TSP [34,35] is very similar to a stochastic gradient ascent in the space of pheromone values. Based on this observation, the authors developed an SGA-based type of ACO algorithm whose pheromone update describes a stochastic gradient ascent. This algorithm can be shown to converge to a local optimum with probability 1. In practice, this SGA-based pheromone update has not been much studied so far. The first implementation of SGA-based ACO algorithms was proposed in [8] where it was shown that SGA-based pheromone updates avoid certain types of search bias.

While convergence proofs can provide insight into the working of an algorithm, they are usually not very useful to the practitioner that wants to implement efficient algorithms. This is because, generally, either infinite time or infinite space are required for an optimization algorithm to converge to an optimal solution (or to the optimal solution value). The existing convergence proofs for particular ACO algorithms are no exception. As more relevant for practical applications might be considered the research efforts that were aimed at a better understanding of the behavior of ACO algorithms. Of particular interest is hereby the understanding of negative search bias that might cause the failure of an ACO algorithm. For example, when applied to the job shop scheduling problem, the average quality of the solutions produced by some ACO algorithms decreases over time. This is clearly undesirable, because instead of successively finding better solutions, the algorithm finds successively worse solutions over time. As one of the principal causes for this search bias were identified situations in which some solution components on average receive update from more solutions than solution components they compete with [12]. Merkle and Middendorf [69,70] were the first to study the behavior of a simple ACO algorithm by analyzing the dynamics of its *model*, which is obtained by applying the expected pheromone update. Their work deals with the application of ACO to idealized permutation problems. When applied to constrained problems such as permutation problems, the solution construction process of ACO algorithms consists of a sequence of local decisions in which later decisions depend on earlier decisions. Therefore, the later decisions of the construction process are inherently biased by the earlier ones. The work of Merkle and Middendorf shows that this leads to a bias which they call *selection bias*. Furthermore, the competition between the ants was identified as the main driving force of the algorithm.

For a recent survey on theoretical work on ACO see [31].

5. Applying ACO to continuous optimization

Many practical optimization problems can be formulated as continuous optimization problems. These problems are characterized by the fact that the decision variables have continuous domains, in contrast to the discrete domains of the variables in discrete optimization. While ACO algorithms were originally introduced to solve discrete problems, their adaptation to solve continuous optimization problems enjoys an increasing attention. Early applications of ant-based algorithms to continuous optimization include algorithms such as Continuous ACO (CACO) [5], the API algorithm [75], and Continuous Interacting Ant Colony (CIAC) [37]. However, all these approaches are conceptually quite different from ACO for discrete problems. The latest approach, which was proposed by Socha in [86], is closest to the spirit of ACO for discrete problems. In the following we shortly outline this algorithm. For the sake of simplicity, we assume the continuous domains of the decision variables $X_i, i = 1, \ldots, n$, to be unconstrained.

As outlined before, in ACO algorithms for discrete optimization problems solutions are constructed by sampling at each construction step a discrete probability distribution that is derived from the pheromone information. In a way, the pheromone information represents the stored search experience of the algorithm. In contrast, ACO for continuous optimization—in the literature denoted by $\text{ACO}_{\mathbb{R}}$—utilizes a continuous probability density function (PDF). This density function is produced, for each solution construction, from a population of solutions that the algorithm keeps at all times. The management of this population works as follows. Before the start of the algorithm, the population—whose cardinality k is a parameter of the algorithm—is filled with random solutions. This corresponds to the pheromone value initialization in ACO algorithms for discrete optimization problems. Then, at each iteration the set of generated solutions is added to the population and the same number of the worst solutions are removed from it. This action corresponds to the pheromone update in discrete ACO. The aim is to bias the search process towards the best solutions found during the search.

For constructing a solution, an ant chooses at each construction step $i = 1, \ldots, n$, a value for decision variable X_i. In other words, if the given optimization problem has n dimensions, an ant chooses in each of n construction steps a value for exactly one of the dimensions. In the following we explain the choice of a value for dimension i. For performing this choice an ant uses a Gaussian kernel, which is a weighted superposition of several Gaussian functions,

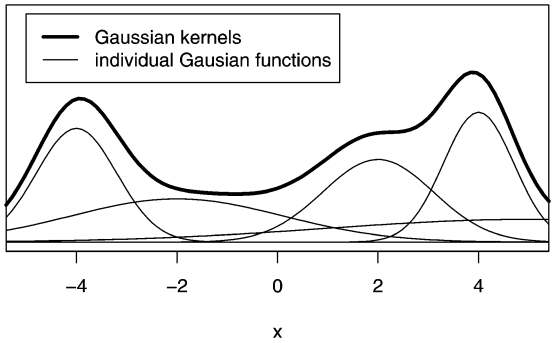


Fig. 5. An example of a Gaussian kernel PDF consisting of five separate Gaussian functions. © Springer-Verlag, Berlin, Germany. The author is grateful to K. Socha for providing this graphic.

as PDF. Concerning decision variable X_i (i.e., dimension i) the Gaussian kernel G_i is given as follows:

$$G_i(x) = \sum_{j=1}^{k} \omega_j \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}}, \quad \forall x \in \mathbb{R}, \tag{12}$$

where the jth Gaussian function is derived from the jth member of the population, whose cardinality is at all times k. Note that $\vec{\omega}$, $\vec{\mu}$, and $\vec{\sigma}$ are vectors of size k. Hereby, $\vec{\omega}$ is the vector of weights, whereas $\vec{\mu}$ and $\vec{\sigma}$ are the vectors of means and standard deviations respectively. Fig. 5 presents an example of a Gaussian kernel PDF consisting of five separate Gaussian functions.

The question is how to sample a Gaussian kernel G_i, which is problematic: While G_i describes very precisely the probability density function that must be sampled, there are no straightforward methods for sampling G_i. In $\text{ACO}_{\mathbb{R}}$ this is accomplished as follows. Each ant, before starting a solution construction, that is, before choosing a value for the first dimension, chooses exactly one of the Gaussian functions j, which is then used for all n construction steps. The choice of this Gaussian function, in the following denoted by j^*, is performed with probability

$$\mathbf{p}_j = \frac{\omega_j}{\sum_{l=1}^{k} \omega_l}, \quad \forall j = 1, \ldots, k, \tag{13}$$

where ω_j is the weight of Gaussian function j, which is obtained as follows. All solutions in the population are ranked according to their quality (e.g., the inverse of the objective function value in the case of minimization) with the best solution having rank 1. Assuming the rank of the jth solution in the population to be r, the weight ω_j of the jth Gaussian function is calculated according to the following formula:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(r-1)^2}{2q^2k^2}}, \tag{14}$$

which essentially defines the weight to be a value of the Gaussian function with argument r, with a mean of 1.0, and a standard deviation of qk. Note that q is a parameter of the algorithm. In case the value of q is small, the best-ranked solutions are strongly preferred, and in case it is larger, the probability becomes more uniform. Due to using the ranks instead of the actual fitness function values, the algorithm is not sensitive to the scaling of the fitness function.

The sampling of the chosen Gaussian function j^* may be done using a random number generator that is able to generate random numbers according to a parameterized normal distribution, or by using a uniform random generator in conjunction with (for instance) the Box–Muller method [18]. However, before performing the sampling, the mean and the standard deviation of the j^*th Gaussian function must be specified. First, the value of the ith decision variable in solution j^* is chosen as mean, denoted by μ_{j^*}, of the Gaussian function. Second, the average distance of the other population members from the j^*th solution multiplied by a parameter ρ is chosen as the standard deviation, denoted by σ_{j^*}, of the Gaussian function:

$$\sigma_{j^*} = \frac{1}{k} \rho \sum_{l=1}^{k} \sqrt{\left(x_i^l - x_i^{j^*}\right)^2}. \tag{15}$$

Parameter ρ, which regulates the speed of convergence, has a role similar to the pheromone evaporation rate ρ in ACO for discrete problems. The higher the value of $\rho \in (0, 1)$, the lower the convergence speed of the algorithm, and hence the lower the learning rate. Since this whole process is done for each dimension (i.e., decision variable) in turn, each time the distance is calculated only with the use of one single dimension (the rest of them are discarded). This allows the handling of problems that are scaled differently in different directions.

$ACO_{\mathbb{R}}$ was successfully applied both to scientific test cases as well as to real world problems such as feedforward neural network training [15,86].

6. A new trend: Hybridization with AI and OR techniques

Hybridization is nowadays recognized to be an essential aspect of high performing algorithms. Pure algorithms are almost always inferior to hybridizations. In fact, many of the current state-of-the-art ACO algorithms include components and ideas originating from other optimization techniques. The earliest type of hybridization was the incorporation of local search based methods such as local search, tabu search, or iterated local search, into ACO. However, these hybridizations often reach their limits when either large-scale problem instances with huge search spaces or highly constrained problems for which it is difficult to find feasible solutions are concerned. Therefore, some researchers recently started investigating the incorporation of more classical AI and OR methods into ACO algorithms. One reason why ACO algorithms are especially suited for this type of hybridization is their constructive nature. Constructive methods can be considered from the viewpoint of tree search [45]. The solution construction mechanism of ACO algorithms maps the search space to a tree structure in which a path from the root node to a leaf corresponds to the process of constructing a solution (see Fig. 6). Examples for tree search methods from AI and OR are greedy algorithms [79], backtracking techniques [45], rollout and pilot techniques [3,38], beam search [78], or constraint programming (CP) [68].

The main idea of the existing ACO hybrids is the use of techniques for shrinking or changing in some way the search space that has to be explored by ACO. In the following we present several successful examples.

6.1. *Beam-ACO: Hybridizing ACO with beam search*

Beam search (BS) is a classical tree search method that was introduced in the context of scheduling [78], but has since then been successfully applied to many other CO problems (e.g., see [25]). BS algorithms are incomplete derivatives of branch & bound algorithms, and are therefore approximate methods. The central idea behind BS is to allow the extension of sequences in several possible ways. At each step the algorithm extends each sequence from a set $\mathcal{B}$, which is called the *beam*, in at most k_{ext} possible ways. Each extension is performed in a deterministic greedy

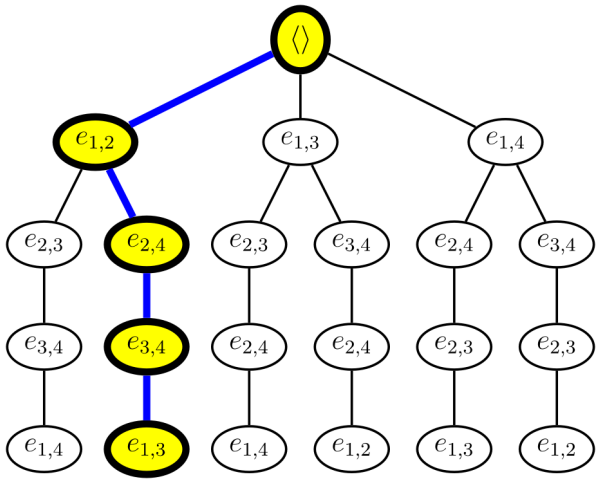


Fig. 6. This search tree corresponds to the solution construction mechanism for the small TSP instance as shown and outlined in Fig. 3. The bold path in this search tree corresponds to the construction of solution $s = \langle e_{1,2}, e_{2,4}, e_{3,4}, e_{1,3} \rangle$ (as shown in Fig. 3(c)).

fashion by using a scoring function. Each newly obtained sequence is either stored in the set of complete solutions $\mathcal{B}_c$ (in case it corresponds to a complete solution), or in the set $\mathcal{B}_{\text{ext}}$ (in case it is a further extensible sequence). At the end of each step, the algorithm creates a new beam $\mathcal{B}$ by selecting up to k_{bw} (called the *beam width*) sequences from the set of further extensible sequences $\mathcal{B}_{\text{ext}}$. In order to select sequences from $\mathcal{B}_{\text{ext}}$, BS algorithms use a mechanism to evaluate sequences. An example of such a mechanism is a lower bound. Given a sequence s, a lower bound computes the minimum objective function value for any complete solution that can be constructed starting from s. The existence of an accurate—and computationally inexpensive—lower bound is crucial for the success of beam search.[8]

Even though both ACO and BS have the common feature that they are based on the idea of constructing candidate solutions step-by-step, the ways by which the two methods explore the search space are quite different. While BS is a deterministic algorithm that uses a lower bound for guiding the search process, ACO algorithms are adaptive and probabilistic procedures. Furthermore, BS algorithms reduce the search space in the hope of not excluding all optimal solutions, while ACO algorithms consider the whole search space. Based on these observations Blum introduced a hybrid between ACO and BS which was labelled *Beam-ACO* [9]. The basic algorithmic framework of Beam-ACO is the framework of ant colony optimization. However, the standard ACO solution construction mechanism is replaced by a solution construction mechanism in which each artificial ant performs a probabilistic BS in which the extension of partial solutions is done in the ACO fashion rather than deterministically. As the transition probabilities depend on the pheromone values, which change over time, the probabilistic beam searches that are performed by this algorithm are adaptive.

Beam-ACO was applied to the NP-hard open shop scheduling (OSS) problem, for which it currently is a state-of-the-art method. However, Beam-ACO is a general approach that can be applied to any CO problem. A crucial point of any Beam-ACO application is the use of an efficient and accurate lower bound. Work that is related to Beam-ACO can be found in [64,67]. For example in [64], the author describes an ACO algorithm for the quadratic assignment problem as an approximate non-deterministic tree search procedure. The results of this approach are compared to both exact algorithms and beam search techniques. An ACO approach to set partitioning that allowed the extension of partial solutions in several possible ways was presented in [67].

6.2. *ACO and constraint programming*

Another successful hybridization example concerns the use of constraint programming (CP) techniques (see [68]) for restricting the search performed by an ACO algorithm to promising regions of the search space. The motivation for this type of hybridization is as follows: Generally, ACO algorithms are competitive with other optimization techniques when applied to problems that are not overly constrained. However, when highly constrained problems such as scheduling or timetabling are concerned, the performance of ACO algorithms generally degrades. Note that this is also the case for other metaheuristics. The reason is to be found in the structure of the search space: When a problem is not overly constrained, it is usually not difficult to find feasible solutions. The difficulty rather lies in the optimization part, namely the search for good feasible solutions. On the other side, when a problem is highly constrained the difficulty is rather in finding any feasible solution. This is where CP comes into play, because these problems are the target problems for CP applications.

CP is a programming paradigm in which a combinatorial optimization problem is modelled as a discrete optimization problem. In that way, CP specifies the constraints a feasible solution must meet. The CP approach to search for a feasible solution often works by the iteration of constraint propagation and the addition of additional constraints. Constraint propagation is the mechanism that reduces the domains of the decision variables with respect to the given set of constraints. Let us consider the following example: Given are two decision variables, X_1 and X_2, both having the same domain $\{0, 1, 2\}$. Furthermore, given is the constraint $X_1 < X_2$. Based on this constraint the application of constraint propagation would remove the value 2 from the domain of X_1. From a general point of view, the constraint propagation mechanism reduces the size of the search tree that must be explored by optimization techniques on the search for good feasible solutions.

The idea of hybridizing ACO with CP is graphically shown in Fig. 7. At each iteration, first constraint propagation is applied in order to reduce the remaining search tree. Then, solutions are constructed in the standard ACO way with

[8] Note that an inaccurate lower bound might bias the search towards bad areas of the search space.

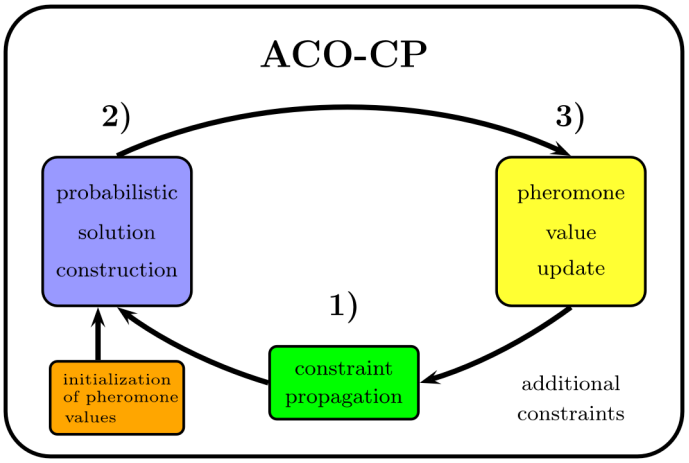


Fig. 7. ACO-CP: A hybrid between ACO and CP.

respect to the reduced search tree. After the pheromone update, additional constraints might be added to the system (i.e., *posted* in CP language). As an example, consider an iteration in which the best-so-far solution is improved. In this case a constraint might be posted that requires a feasible solution to be at least as good as the new best-so-far solution. Meyer and Ernst introduced and applied the idea described above in an application to the single machine job scheduling problem [73]. The results are especially promising for problem instances where the search space of feasible solutions is too large for complete methods, but already sufficiently fragmented to cause difficulties for ACO.

6.3. Applying ACO in a multilevel framework

The idea that is presented in the following is not really an ACO hybrid. It rather promotes the application of ACO within a general problem solving framework known as the multilevel framework. Optimization techniques that are based on this framework, i.e., multilevel techniques, have been in use since quite a long time, especially in the area of multigrid methods (see [19] for an overview). More recently, they have been brought into focus by Walshaw for the application to CO. Walshaw and co-workers applied multilevel techniques to graph-based problems such as mesh partitioning [95], the traveling salesman problem [93], and graph coloring [94]. The basic idea of a multilevel scheme is simple. Starting from the original problem instance, smaller and smaller problem instances are obtained by successive coarsening until some stopping criteria are satisfied. This creates a hierarchy of problem instances in which the problem instance of a given level is always smaller (or of equal size) to the problem instance of the next lower level. Then, a solution is computed to the smallest problem instance and successively transformed into a solution of the next higher level until a solution for the original problem instance is obtained. At each level, the obtained solution might be subject to a refinement process. This idea is illustrated with respect to the application of ACO as refinement process in Fig. 8.

In [59,60] the authors presented the first application of an ACO algorithm in a multilevel framework for mesh partitioning. The resulting algorithm outperforms the classical k-METIS and Chaco algorithms for graph partitioning on several benchmark instances. Furthermore, its performance is comparable to the performance of the evolutionary multilevel algorithm provided by the popular JOSTLE library, and obtains for some of the benchmark instances new best solutions. In general, applying ACO in a multilevel framework is only possible if an efficient and sensible way of contracting a problem instance, and expanding solutions to higher levels, can be found. So far, the multilevel framework is mainly used in graph-based problems, where the coarsening of a problem instance is obtained by edge contractions. The application of the multilevel framework to problems that are not graph-based is one of the research subjects for the near future.

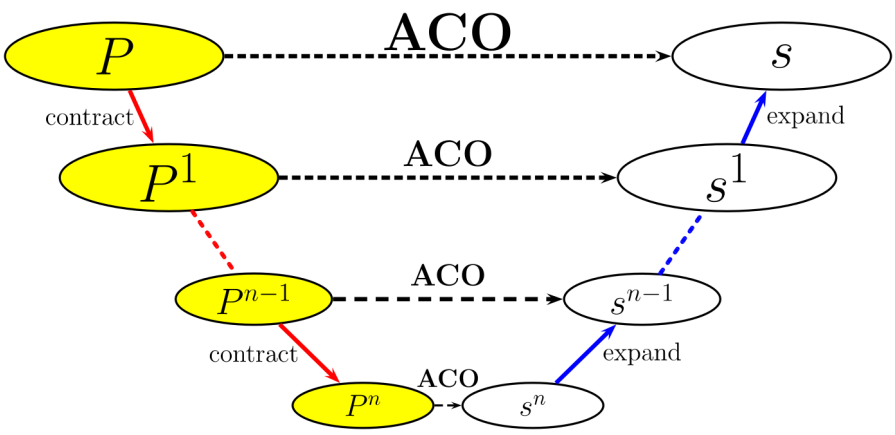


Fig. 8. ML-ACO: Applying ACO in the multilevel framework. The original problem instance is P. In an iterative process this problem instance is simplified (i.e., contracted) until the lowest level instance P^n is reached. Then, an ACO algorithm (or any other optimization technique) can be used to tackle problem P^n. The obtained best solution s^n is expanded into a solution $s^{n-1'}$ of the next bigger problem instance P^{n-1}. With this solution as the first best-so-far solution, the same ACO algorithm might be applied to tackle problem instance P^{n-1} resulting in a best obtained solution s^{n-1}. This process goes on until the original problem instance was tackled.

6.4. Applying ACO to an auxiliary search space

The idea that we present in this section is based on replacing the original search space of the tackled optimization problem with an auxiliary search space to which ACO is then applied. A precondition for this technique is a function that maps each object from the auxiliary search space to a solution to the tackled optimization problem. This technique can be beneficial in case (1) the generation of objects from the auxiliary search space is more efficient than the construction of solutions to the optimization problem at hand, and/or (2) the mapping function is such that objects from the auxiliary search space are mapped to high quality solutions of the original search space.

An example of such a hybrid ACO algorithm was presented in [10] for the application to the k-cardinality tree (KCT) problem. In this problem is given an edge-weighted graph and a cardinality $k > 0$. The original search space consists of all trees in the given graph with exactly k edges, i.e., all k-cardinality trees. The objective function value of a given tree is computed as the sum of the weights of its edges. The authors of [10] chose the set of all l-cardinality trees (where $l > k$, and l fixed) in the given graph as auxiliary search space. The mapping between the auxiliary search space and the original search space was performed by a polynomial-time dynamic programming algorithm for finding the optimal k-cardinality tree that is contained in an l-cardinality tree. The experimental results show that the ACO algorithm working on the auxiliary search space significantly improves over the same ACO algorithm working on the original search space.

7. Conclusions

In this work we first gave a detailed description of the origins and the basics of ACO algorithms. Then we outlined the general framework of the ACO metaheuristic and presented some of the most successful ACO variants today. After listing some representative applications of ACO, we summarized the existing theoretical results and outlined the latest developments concerning the adaptation of ACO algorithms to continuous optimization. Finally, we provided a survey on a very interesting recent research direction: The hybridization of ACO algorithms with more classical artificial intelligence and operations research methods. As examples we presented the hybridization with beam search and with constraint programming. The central idea behind these two approaches is the reduction of the search space that has to be explored by ACO. This can be especially useful when large scale problem instances are considered. Other hybridization examples are the application of ACO for solution refinement in multilevel frameworks, and the application of ACO to auxiliary search spaces. In the opinion of the author, this research direction offers many possibilities for valuable future research.

Acknowledgements

Many thanks to the anonymous referees for helping immensely to improve this paper.

References

[1] Alupoaei S, Katkoori S. Ant colony system application to marcocell overlap removal. IEEE Trans Very Large Scale Integr (VLSI) Syst 2004;12(10):1118–22.
[2] Bautista J, Pereira J. Ant algorithms for assembly line balancing. In: Dorigo M, Di Caro G, Sampels M, editors. Ant algorithms—Proceedings of ANTS 2002—Third international workshop. Lecture Notes in Comput Sci, vol. 2463. Berlin: Springer; 2002. p. 65–75.
[3] Bertsekas DP, Tsitsiklis JN, Wu C. Rollout algorithms for combinatorial optimization. J Heuristics 1997;3:245–62.
[4] Bianchi L, Gambardella LM, Dorigo M. An ant colony optimization approach to the probabilistic traveling salesman problem. In: Merelo JJ, Adamidis P, Beyer H-G, Fernández-Villacanas J-L, Schwefel H-P, editors. Proceedings of PPSN-VII, seventh international conference on parallel problem solving from nature. Lecture Notes in Comput Sci, vol. 2439. Berlin: Springer; 2002. p. 883–92.
[5] Bilchev B, Parmee IC. The ant colony metaphor for searching continuous design spaces. In: Fogarty TC, editor. Proceedings of the AISB workshop on evolutionary computation. Lecture Notes in Comput Sci, vol. 993. Berlin: Springer; 1995. p. 25–39.
[6] Birattari M, Di Caro G, Dorigo M. Toward the formal foundation of ant programming. In: Dorigo M, Di Caro G, Sampels M, editors. Ant algorithms—Proceedings of ANTS 2002—Third international workshop. Lecture Notes in Comput Sci, vol. 2463. Berlin: Springer; 2002. p. 188–201.
[7] Blesa M, Blum C. Ant colony optimization for the maximum edge-disjoint paths problem. In: Raidl GR, Cagnoni S, Branke J, Corne DW, Drechsler R, Jin Y, Johnson CG, Machado P, Marchiori E, Rothlauf R, Smith GD, Squillero G, editors. Applications of evolutionary computing, proceedings of EvoWorkshops 2004. Lecture Notes in Comput Sci, vol. 3005. Berlin: Springer; 2004. p. 160–9.
[8] Blum C. Theoretical and practical aspects of Ant colony optimization. Dissertations in Artificial Intelligence. Berlin: Akademische Verlagsgesellschaft Aka GmbH; 2004.
[9] Blum C. Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. Computers & Operations Res 2005;32(6):1565–91.
[10] Blum C, Blesa MJ. Combining ant colony optimization with dynamic programming for solving the k-cardinality tree problem. In: Cabestany J, Prieto A, Sandoval F, editors. 8th international work-conference on artificial neural networks, computational intelligence and bioinspired systems (IWANN'05). Lecture Notes in Comput Sci, vol. 3512. Berlin: Springer; 2005. p. 25–33.
[11] Blum C, Dorigo M. The hyper-cube framework for ant colony optimization. IEEE Trans Syst Man Cybernet Part B 2004;34(2):1161–72.
[12] Blum C, Dorigo M. Search bias in ant colony optimization: On the role of competition-balanced systems. IEEE Trans Evolutionary Comput 2005;9(2):159–74.
[13] Blum C, Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput Surveys 2003;35(3):268–308.
[14] Blum C, Sampels M. An ant colony optimization algorithm for shop scheduling problems. J Math Modelling Algorithms 2004;3(3):285–308.
[15] Blum C, Socha K. Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In: Proceedings of the 5th international conference on hybrid intelligent systems (HIS). Piscataway, NJ: IEEE Press; 2005. p. 233–8.
[16] Bonabeau E, Dorigo M, Theraulaz G. Swarm intelligence: From natural to artificial systems. New York: Oxford University Press; 1999.
[17] Bonabeau E, Dorigo M, Theraulaz G. Inspiration for optimization from social insect behavior. Nature 2000;406:39–42.
[18] Box GEP, Muller ME. A note on the generation of random normal deviates. Ann Math Statist 1958;29(2):610–1.
[19] Brandt A. Multilevel computations: Review and recent developments. In: McCormick SF, editor. Multigrid methods: Theory, applications, and supercomputing, proceedings of the 3rd Copper Mountain conference on multigrid methods. Lecture Notes in Pure and Appl Math, vol. 110. New York: Marcel Dekker; 1988. p. 35–62.
[20] Bui TN, Rizzo JR. Finding maximum cliques with distributed ants. In: Deb K, et al., editors. Proceedings of the genetic and evolutionary computation conference (GECCO 2004). Lecture Notes in Comput Sci, vol. 3102. Berlin: Springer; 2004. p. 24–35.
[21] Bullnheimer B, Hartl R, Strauss C. A new rank-based version of the Ant System: A computational study. Central European J Operations Res Econom 1999;7(1):25–38.
[22] Campos M, Bonabeau E, Theraulaz G, Deneubourg J-L. Dynamic scheduling and division of labor in social insects. Adapt Behavior 2000;8(3):83–96.
[23] Corry P, Kozan E. Ant colony optimisation for machine layout problems. Comput Optim Appl 2004;28(3):287–310.
[24] Costa D, Hertz A. Ants can color graphs. J Oper Res Soc 1997;48:295–305.
[25] Della Croce F, Ghirardi M, Tadei R, Recovering beam search: enhancing the beam search approach for combinatorial optimisation problems. In: Proceedings of PLANSIG 2002—21th workshop of the UK planning and scheduling special interest group; 2002. p. 149–69.
[26] den Besten ML, Stützle T, Dorigo M. Ant colony optimization for the total weighted tardiness problem. In: Schoenauer M, Deb K, Rudolph G, Yao X, Lutton E, Merelo JJ, Schwefel H-P, editors. Proceedings of PPSN-VI, sixth international conference on parallel problem solving from nature. Lecture Notes in Comput Sci, vol. 1917. Berlin: Springer; 2000. p. 611–20.
[27] Deneubourg J-L, Aron S, Goss S, Pasteels J-M. The self-organizing exploratory pattern of the Argentine ant. J Insect Behaviour 1990;3:159–68.
[28] Di Caro G, Dorigo M. AntNet: Distributed stigmergetic control for communications networks. J Artificial Intelligence Res 1998;9:317–65.
[29] Doerner K, Gutjahr WJ, Hartl RF, Strauss C, Stummer C. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. Ann Oper Res 2004;131:79–99.
[30] Dorigo M, Optimization, learning and natural algorithms. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992 [in Italian].

[31] Dorigo M, Blum C. Ant colony optimization theory: A survey. Theoret Comput Sci 2005;344(2–3):243–78.
[32] Dorigo M, Di Caro G, Gambardella LM. Ant algorithms for discrete optimization. Artificial Life 1999;5(2):137–72.
[33] Dorigo M, Gambardella LM. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Trans Evolutionary Comput 1997;1(1):53–66.
[34] Dorigo M, Maniezzo V, Colorni A, Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.
[35] Dorigo M, Maniezzo V, Colorni A. Ant System: Optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybernet Part B 1996;26(1):29–41.
[36] Dorigo M, Stützle T. Ant Colony optimization. Cambridge, MA: MIT Press; 2004.
[37] Dréo J, Siarry P. A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions. In: Dorigo M, Di Caro G, Sampels M, editors. Ant algorithms—Proceedings of ANTS 2002—Third international workshop. Lecture Notes in Comput Sci, vol. 2463. Berlin: Springer; 2002. p. 216–21.
[38] Duin C, Voß S. The pilot method: A strategy for heuristic repetition with application to the steiner problem in graphs. Networks 1999;34(3):181–91.
[39] Gagné C, Price WL, Gravel M. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. J Oper Res Soc 2002;53:895–906.
[40] Gambardella LM, Dorigo M. Solving symmetric and asymmetric TSPs by ant colonies. In: Baeck T, Fukuda T, Michalewicz Z, editors. Proceedings of the 1996 IEEE international conference on evolutionary computation (ICEC'96). Piscataway, NJ: IEEE Press; 1996. p. 622–7.
[41] Gambardella LM, Dorigo M. Ant Colony System hybridized with a new local search for the sequential ordering problem. INFORMS J Comput 2000;12(3):237–55.
[42] Gambardella LM, Taillard ÉD, Agazzi G. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: Corne D, Dorigo M, Glover F, editors. New ideas in optimization. London: McGraw-Hill; 1999. p. 63–76.
[43] Gandibleux X, Delorme X, T'Kindt V. An ant colony optimisation algorithm for the set packing problem. In: Dorigo M, Birattari M, Blum C, Gambardella LM, Mondada F, Stützle T, editors. Proceedings of ANTS 2004—Fourth international workshop on Ant colony optimization and swarm intelligence. Lecture Notes in Comput Sci, vol. 3172. Berlin: Springer; 2004. p. 49–60.
[44] Garey MR, Johnson DS. Computers and intractability; A guide to the theory of NP Completeness. New York: WH Freeman; 1979.
[45] Ginsberg ML. Essentials of artificial intelligence. San Mateo, CA: Morgan Kaufmann; 1993.
[46] Glover F. Future paths for integer programming and links to artificial intelligence. Comput Oper Res 1986;13:533–49.
[47] Glover F, Kochenberger G, editors. Handbook of metaheuristics. Norwell, MA: Kluwer Academic; 2002.
[48] Gottlieb J, Puchta M, Solnon C. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In: Cagnoni S, Romero Cardalda JJ, Corne DW, Gottlieb J, Guillot A, Hart E, Johnson CG, Marchiori E, Meyer J-A, Middendorf M, Raidl GR, editors. Applications of evolutionary computing, proceedings of EvoWorkshops, 2003. Lecture Notes in Comput Sci, vol. 2611. Berlin: Springer; 2003. p. 246–57.
[49] Grassé P-P. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. La théorie de la stigmergie: Essai d'interprétation des termites constructeurs. Insectes Sociaux 1959;6:41–81.
[50] Guéret C, Monmarché N, Slimane M. Ants can play music. In: Dorigo M, Birattari M, Blum C, Gambardella LM, Mondada F, Stützle T, editors. Proceedings of ANTS 2004—Fourth international workshop on Ant colony optimization and swarm intelligence. Lecture Notes in Comput Sci, vol. 3172. Berlin: Springer; 2004. p. 310–7.
[51] Guntsch M, Middendorf M. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: Boers EJW, Gottlieb J, Lanzi PL, Smith RE, Cagnoni S, Hart E, Raidl GR, Tijink H, editors. Applications of evolutionary computing: Proceedings of EvoWorkshops, 2001. Lecture Notes in Comput Sci, vol. 2037. Berlin: Springer; 2001. p. 213–22.
[52] Guntsch M, Middendorf M. Solving multi-objective permutation problems with population based ACO. In: Fonseca CM, Fleming PJ, Zitzler E, Deb K, Thiele L, editors. Proceedings of the second international conference on evolutionary multi-criterion optimization (EMO 2003). Lecture Notes in Comput Sci, vol. 2636. Berlin: Springer; 2003. p. 464–78.
[53] Gutjahr WJ. A graph-based ant system and its convergence. Future Generat Comput Syst 2000;16(9):873–88.
[54] Gutjahr WJ. ACO algorithms with guaranteed convergence to the optimal solution. Informat Process Lett 2002;82(3):145–53.
[55] Handl J, Knowles J, Dorigo M. Ant-based clustering and topographic mapping. Artificial Life 2006;12(1). In press.
[56] Hoos HH, Stützle T. Stochastic local search: Foundations and applications. Amsterdam: Elsevier; 2004.
[57] Karpenko O, Shi J, Dai Y. Prediction of MHC class II binders using the ant colony search strategy. Artificial Intelligence in Medicine 2005;35(1–2):147–56.
[58] Kennedy J, Eberhart RC. Particle swarm optimization. In: Proceedings of the 1995 IEEE international conference on neural networks, vol. 4. Piscataway, NJ: IEEE Press; 1995. p. 1942–8.
[59] Korošec P, Šilc J, Robič B. Mesh-partitioning with the multiple ant-colony algorithm. In: Dorigo M, Birattari M, Blum C, Gambardella LM, Mondada F, Stützle T, editors. Proceedings of ANTS 2004—Fourth international workshop on Ant colony optimization and swarm intelligence. Lecture Notes in Comput Sci, vol. 3172. Berlin: Springer; 2004. p. 430–1.
[60] Korošec P, Šilc J, Robič B. Solving the mesh-partitioning problem with an ant-colony algorithm. Parallel Comput 2004;30:785–801.
[61] Lawler E, Lenstra JK, Rinnooy Kan AHG, Shmoys DB. The travelling Salesman problem. New York: John Wiley & Sons; 1985.
[62] López-Ibáñez M, Paquete L, Stützle T. On the design of ACO for the biobjective quadratic assignment problem. In: Dorigo M, Birattari M, Blum C, Gambardella LM, Mondada F, Stützle T, editors. Proceedings of ANTS 2004—Fourth international workshop on Ant colony optimization and swarm intelligence. Lecture Notes in Comput Sci, vol. 3172. Berlin: Springer; 2004. p. 214–25.
[63] Lumer E, Faieta B. Diversity and adaptation in populations of clustering ants. In: Meyer J-A, Wilson SW, editors. Proceedings of the third international conference on simulation of adaptive behavior: From animals to animats, vol. 3. Cambridge, MA: MIT Press; 1994. p. 501–8.

[64] Maniezzo V. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS J Comput 1999;11(4):358–69.
[65] Maniezzo V, Boschetti M, Jelasity M. An ant approach to membership overlay design. In: Dorigo M, Birattari M, Blum C, Gambardella LM, Mondada F, Stützle T, editors. Proceedings of ANTS 2004—Fourth international workshop on Ant colony optimization and swarm intelligence. Lecture Notes in Comput Sci, vol. 3172. Berlin: Springer; 2004. p. 37–48.
[66] Maniezzo V, Colorni A. The Ant System applied to the quadratic assignment problem. IEEE Trans Data Knowledge Engrg 1999;11(5):769–78.
[67] Maniezzo V, Milandri M. An ant-based framework for very strongly constrained problems. In: Dorigo M, Di Caro G, Sampels M, editors. Ant algorithms—Proceedings of ANTS 2002—Third international workshop. Lecture Notes in Comput Sci, vol. 2463. Berlin: Springer; 2002. p. 222–7.
[68] Marriott K, Stuckey P. Programming with constraints. Cambridge, MA: MIT Press; 1998.
[69] Merkle D, Middendorf M. Modelling ACO: Composed permutation problems. In: Dorigo M, Di Caro G, Sampels M, editors. Ant algorithms—Proceedings of ANTS 2002—Third international workshop. Lecture Notes in Comput Sci, vol. 2463. Berlin: Springer; 2002. p. 149–62.
[70] Merkle D, Middendorf M. Modelling the dynamics of ant colony optimization algorithms. Evolutionary Comput 2002;10(3):235–62.
[71] Merkle D, Middendorf M, Schmeck H. Ant colony optimization for resource-constrained project scheduling. IEEE Trans Evolutionary Comput 2002;6(4):333–46.
[72] Meuleau N, Dorigo M. Ant colony optimization and stochastic gradient descent. Artificial Life 2002;8(2):103–21.
[73] Meyer B, Ernst A. Integrating ACO and constraint propagation. In: Dorigo M, Birattari M, Blum C, Gambardella LM, Mondada F, Stützle T, editors. Proceedings of ANTS 2004—Fourth international workshop on Ant colony optimization and swarm intelligence. Lecture Notes in Comput Sci, vol. 3172. Berlin: Springer; 2004. p. 166–77.
[74] Michel R, Middendorf M. An island model based ant system with lookahead for the shortest supersequence problem. In: Eiben AE, Bäck T, Schoenauer M, Schwefel H-P, editors. Proceedings of PPSN-V, fifth international conference on parallel problem solving from nature. Lecture Notes in Comput Sci, vol. 1498. Berlin: Springer; 1998. p. 692–701.
[75] Monmarché N, Venturini G, Slimane M. On how pachycondyla apicalis ants suggest a new search algorithm. Future Generation Comput Syst 2000;16:937–46.
[76] Moss JD, Johnson CG. An ant colony algorithm for multiple sequence alignment in bioinformatics. In: Pearson DW, Steele NC, Albrecht RF, editors. Artificial neural networks and genetic algorithms. Berlin: Springer; 2003. p. 182–6.
[77] Nemhauser GL, Wolsey AL. Integer and combinatorial optimization. New York: John Wiley & Sons; 1988.
[78] Ow PS, Morton TE. Filtered beam search in scheduling. Internat J Production Res 1988;26:297–307.
[79] Papadimitriou CH, Steiglitz K. Combinatorial optimization—Algorithms and complexity. New York: Dover; 1982.
[80] Parpinelli RS, Lopes HS, Freitas AA. Data mining with an ant colony optimization algorithm. IEEE Trans Evolutionary Comput 2002;6(4):321–32.
[81] Reeves CR, editor. Modern heuristic techniques for combinatorial problems. New York: John Wiley & Sons; 1993.
[82] Reimann M, Doerner K, Hartl RF. *D*-ants: Savings based ants divide and conquer the vehicle routing problems. Comput Oper Res 2004;31(4):563–91.
[83] Shmygelska A, Aguirre-Hernández R, Hoos HH. An ant colony optimization algorithm for the 2D HP protein folding problem. In: Dorigo M, Di Caro G, Sampels M, editors. Ant algorithms—Proceedings of ANTS 2002—Third international workshop. Lecture Notes in Comput Sci, vol. 2463. Berlin: Springer; 2002. p. 40–52.
[84] Shmygelska A, Hoos HH. An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. BMC Bioinformatics 2005;6(30):1–22.
[85] Silva CA, Runkler TA, Sousa JM, Palm R. Ant colonies as logistic processes optimizers. In: Dorigo M, Di Caro G, Sampels M, editors. Ant algorithms—Proceedings of ANTS 2002—Third international workshop. Lecture Notes in Comput Sci, vol. 2463. Berlin: Springer; 2002. p. 76–87.
[86] Socha K. ACO for continuous and mixed-variable optimization. In: Dorigo M, Birattari M, Blum C, Gambardella LM, Mondada F, Stützle T, editors. Proceedings of ANTS 2004—Fourth international workshop on Ant colony optimization and swarm intelligence. Lecture Notes in Comput Sci, vol. 3172. Berlin: Springer; 2004. p. 25–36.
[87] Socha K, Sampels M, Manfrin M. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In: Cagnoni S, Romero Cardalda JJ, Corne DW, Gottlieb J, Guillot A, Hart E, Johnson CG, Marchiori E, Meyer A, Middendorf M, Raidl GR, editors. Applications of evolutionary computing, proceedings of EvoWorkshops 2003. Lecture Notes in Comput Sci, vol. 2611. Berlin: Springer; 2003. p. 334–45.
[88] Solnon C. Ant can solve constraint satisfaction problems. IEEE Trans Evolutionary Comput 2002;6(4):347–57.
[89] Stützle T. An ant approach to the flow shop problem. In: Proceedings of the 6th european congress on intelligent techniques & soft computing (EUFIT'98). Aachen: Verlag Mainz; 1998. p. 1560–4.
[90] Stützle T, Dorigo M. A short convergence proof for a class of ACO algorithms. IEEE Trans Evolutionary Comput 2002;6(4):358–65.
[91] Stützle T, Hoos HH. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant system. Future Generat Comput Syst 2000;16(8):889–914.
[92] Unger R, Moult J. Finding the lowest free-energy conformation of a protein is an *NP*-hard problem: Proofs and implications. Bull Math Biol 1993;55(6):1183–98.
[93] Walshaw C. A multilevel approach to the travelling salesman problem. Oper Res 2002;50(5):862–77.
[94] Walshaw C. Multilevel refinement for combinatorial optimization problems. Ann Oper Res 2004;131:325–72.
[95] Walshaw C, Cross M. Mesh partitioning: A multilevel balancing and refinement algorithm. SIAM J Sci Comput 2000;22(1):63–80.
[96] Zlochin M, Birattari M, Meuleau N, Dorigo M. Model-based search for combinatorial optimization: A critical survey. Ann Oper Res 2004;131(1–4):373–95.