Ant Colony Optimization: A Component-Wise Overview

Manuel López-Ibáñez, Thomas Stützle, and Marco Dorigo

Contents

M. López-Ibáñez (✉)
Alliance Manchester Business School, University of Manchester, Manchester, UK
e-mail: manuel.lopez-ibanez@manchester.ac.uk

T. Stützle • M. Dorigo
IRIDIA, Université libre de Bruxelles(ULB), CoDE, CP 194/6, Brussels, Belgium
e-mail: stuetzle@ulb.ac.be; mdorigo@ulb.ac.be

R. Martí et al. (eds.), *Handbook of Heuristics*,
DOI 10.1007/978-3-319-07153-4_21-1

Abstract

The indirect communication and foraging behavior of certain species of ants have inspired a number of optimization algorithms for NP-hard problems. These algorithms are nowadays collectively known as the ant colony optimization (ACO) metaheuristic. This chapter gives an overview of the history of ACO, explains in detail its algorithmic components, and summarizes its key characteristics. In addition, the chapter introduces a software framework that unifies the implementation of these ACO algorithms for two example problems, the traveling salesman problem and the quadratic assignment problem. By configuring the parameters of the framework, one can combine features from various ACO algorithms in novel ways. Examples on how to find a good configuration automatically are given in the chapter. The chapter closes with a review of combinations of ACO with other techniques and extensions of the ACO metaheuristic to other problem classes.

Keywords

Ant colony optimization • Automatic configuration • Combinatorial optimization • Metaheuristics

Introduction

Ant colony optimization (ACO) [31, 33, 35] is a metaheuristic that generates candidate solutions by repeated applications of a probabilistic solution construction procedure. In ACO, the probabilities that bias the solution construction are computed from two types of numerical information associated with construction decisions: *heuristic information*, which is derived from the problem instance being tackled, and *artificial pheromone trails*, which are adapted based on the search performance to bias the solution construction toward high-quality solutions. Pheromone trails are the result of a learning mechanism that tries to identify the solution components that, when appropriately combined, lead to high-quality solutions.

As the name suggests, ACO was inspired by entomological experiments on real ant colonies. These experiments demonstrated that some ant species can find the shortest path between their nest and a food source and showed how this mechanism is enabled by the ants' pheromone trail laying and following behavior [22]. The equations used in computer simulations that mimicked the probabilistic behavior of real ant colonies inspired computer scientists to define the way artificial ants take decisions when solving combinatorial optimization problems [30, 36, 38]. Nonetheless, rather than faithfully simulating the behavior of natural ants, the focus of ACO research, since its early days, has been the computational performance of ACO algorithms and the quality of the solutions that can be generated. To this

aim, additional algorithmic techniques have been quickly added to the first ACO algorithms to make them competitive with other metaheuristics. One example of such algorithmic techniques is the exploitation of heuristic information for guiding the construction of solutions [30, 36, 38]; another example is the use of local search algorithms for improving the solutions constructed by the ants [34, 35, 47, 123, 125].

The first ACO algorithm, *ant system*, was proposed by Dorigo et al. [30, 36–38]. After the publication of the first journal paper on ant system in 1996 [38], the research on ACO has gained momentum, and a large number of algorithmic improvements have been proposed. A first improvement over the basic form of ant system was elitist ant system (EAS) [30]. Among the most successful of these successors have been ant colony system (ACS) [34, 45] and max-min ant system (MMAS) [122, 123, 125]. Generally speaking, the main features of these improvements over the basic ant system include mechanisms to intensify the search around high-quality solutions and mechanisms to maintain a sufficient level of search space exploration.

When viewing algorithms from a component-wise perspective, that is, when considering that an algorithm is composed of algorithmic components that fulfill specific tasks for which alternative procedures may exist, there are similarities between various components of MMAS, ACS, and other ACO algorithms such as EAS [30, 38], rank-based ant system (RAS) [19], best-worst ant system (BWAS) [21], and others. In this chapter, we present the available ACO algorithms for combinatorial optimization from such a component-wise perspective. This perspective makes the contributions of each of the ACO algorithms more easily identifiable, which allows a more flexible approach toward their implementation. One additional contribution of this chapter is to provide such an implementation. The main advantage of this framework is that it enables the composition of new ACO variants. While one possible approach would be to manually explore different ACO algorithm designs and test their effectiveness, this exploration may also be done using automatic methods [65]. For illustration purposes, we demonstrate here how to find the best configuration of such a framework for the traveling salesman problem (TSP) and the quadratic assignment problem (QAP) using irace, an automatic algorithm configuration method [83].

This chapter is structured as follows. In section "Combinatorial Optimization Problems and Constructive Heuristics" we introduce some basic notions of combinatorial optimization and construction heuristics. Next, in section "The ACO Algorithmic Framework", we present the ACO metaheuristic as an algorithmic framework for single-objective combinatorial optimization problems, explaining ACO algorithms from a component-wise perspective. Section "ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP" describes how historical ACO algorithms are instantiations of this framework and how ACO algorithms can be automatically configured. Section "Applications of ACO to Other Problem Types" describes recent developments in ACO applied to problems with particular characteristics, including continuous optimization and mixed-variable, multi-objective, and dynamic problems. Section "ACO in Combination with Other Methods" reviews hybridizations of ACO with other methods, such as tree search methods, exact methods, and surrogate models. We also briefly discuss recent

experiments in dynamically changing the parameters of ACO algorithms during execution. Finally, section "Conclusions" concludes with some perspectives for future research in ACO.

Combinatorial Optimization Problems and Constructive Heuristics

Many problems of enormous practical and theoretical importance can be modeled as combinatorial optimization problems (COPs) [106]. Although practical problems usually have a large number of constraints to be satisfied and uncertainties or multiple objectives to be considered, even simpler COPs without such complicating factors may be very difficult to solve.

A COP can be defined as consisting of a set of instances [106]. An instance of a combinatorial optimization problem $(\mathscr{S}, \Omega, f)$ is defined by

- A search space $\mathscr{S}$ given by the possible value assignments to discrete decision variables x_i, with $i = 1, \ldots, n$;
- A set of constraints Ω among the decision variables;
- An objective function $f: \mathscr{S} \to \mathbb{R}$, which may have to be minimized or maximized.

The notion of *problem* refers to the general abstract task to be solved, while the *instance* is defined as a concrete instantiation of this task with fully specified data.

One of the most studied combinatorial optimization problems is the traveling salesman problem (TSP). In the TSP, a graph $G = (N, E)$ is given with $n = |N|$ nodes, a set E of edges fully connecting the nodes, and distances associated with the edges $d_{ij}, \forall (i, j) \in E$. The goal is to find a Hamiltonian tour of minimal length. Such a tour can be represented as a permutation $\pi = (\pi_1, \ldots, \pi_n)^t$ of the n nodes, where π_i is the node index at position i. Thus, $\mathscr{S}$ is the space of such permutations. The objective function in the TSP is

$$\min_{\pi \in \mathscr{S}} d_{\pi_n \pi_1} + \sum_{i=1}^{n-1} d_{\pi_i \pi_{i+1}} \tag{1}$$

It is important to note that the permutations in the TSP are cyclic, that is, for a TSP solution, the absolute position of a node in the permutation is not relevant.

Another well-known combinatorial problem is the quadratic assignment problem (QAP), where two matrices are given describing the distances between n locations and the flows (of persons, goods, etc.) between n facilities. The goal is to find an assignment of facilities to locations that minimizes the sum of the products between

distances and flows. More formally, given two $n \times n$ matrices $[d_{ij}]$ and $[f_{ij}]$ with $i, j = 1, \ldots, n$, a solution to the QAP is an assignment of facilities to locations (or, equivalently, of locations to facilities). Because the number of facilities is the same as the number of locations, such an assignment can be represented by a permutation π, where π_i is the facility assigned to location i. The objective function in the QAP is

$$\min_{\pi \in \mathscr{S}} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \cdot f_{\pi_i \pi_j} \tag{2}$$

Solving a combinatorial optimization problem can often be described as choosing a subset from a finite set of *solution components* $\mathscr{C} = \{c_1, c_2, \ldots\}$ and determining their precise combination that results in a solution that satisfies the problem constraints, i.e., a *feasible* solution, and that optimizes the objective function, i.e., an optimal solution. (For what follows, we assume without loss of generality that the objective function is to be minimized.) In the TSP, for example, the solution components c_{ij} are typically taken to be the edges (i, j) of the given graph, whereas in the QAP, they are the possible individual assignments of a facility j to a location i. Commonly, problems are solved by either searching only in the feasible candidate solution space or, if this is deemed to be too difficult, by allowing the evaluation of infeasible candidate solutions but biasing the search in some way toward feasible ones. For both the TSP and the QAP, a search in the feasible space can be easily enforced, as any permutation is a feasible candidate solution.

Many COPs with practical applications belong to the class of NP-hard problems [49], and, hence, they are considered hard to solve. The NP-hardness of many of these problems, including the TSP and the QAP, implies that, in the worst case, the time needed to find the optimal solution for a given instance grows exponentially with instance size. Thus, instead of searching for an optimal solution and proving its optimality, which may require an infeasible amount of computation time, heuristic algorithms are used to generate good solutions within a reasonable time.

Perhaps the simplest heuristic methods are constructive heuristics. A constructive heuristic starts from an empty or partial candidate solution and then iteratively extends it by adding solution components to it. In many cases, a heuristic estimation of the quality of solution components is available and can be used to guide the choice of which solution components to add. Greedy constructive heuristics rank solution components according to their heuristic value and, at each construction step, they add the best-ranked solution component to the current partial solution. If more than one solution component has the best heuristic value, tiebreaking can be done either randomly or by a secondary heuristic.

An example of constructive heuristic for the TSP is the *nearest neighbor* heuristic. Starting from a random node i and an empty candidate solution $\pi = \langle\rangle$, it selects the solution component c_{ij}, $j \in N \setminus \{i\}$ with the smallest distance d_{ij} and adds it to π such that $\pi_1 = i$ and $\pi_2 = j$. The next solution component c_{jk} chosen

is the one that minimizes the distance $d_{jk}, k \in N \setminus \{i, j\}$, and so on until a complete permutation π of the nodes in N is obtained.

A constructive heuristic for the QAP would be guided by the fact that facilities with a high total flow to other facilities should be placed at locations that are central and, thus, have a small sum of distances to other locations. Thus, one may precompute values $f = (f_1, \ldots, f_n)^t$, where $f_i = \sum_{j=1}^{n} f_{ij}$, and analogously $d = (d_1, \ldots, d_n)^t$, where $d_k = \sum_{l=1}^{n} d_{kl}$, and then assign the facility with the largest f_i to the locations with the smallest d_k and iterate these steps.

If the heuristic values of solution components remain constant during the solution construction, the construction heuristic is called *static*. Otherwise, when the heuristic values are a function of the partial solution generated so far, one talks of an *adaptive* construction heuristic. In the QAP case, one may obtain an adaptive heuristic by using the intuition that facilities with a high interaction with already assigned ones should be put on a location that is as close as possible to already chosen locations. Adaptive heuristics generally require higher computation times as the heuristic values need to be updated or even recomputed after each construction step; however, making decisions dependent on a partial solution may lead to higher-quality complete candidate solutions.

Constructive heuristics are basic ingredients of many metaheuristics. As an example, semi-greedy heuristics [64] are a randomized form of constructive heuristics, i.e., they make randomized decisions during the constructive search, thus generating many different solutions. The GRASP metaheuristic [42, 43] extends upon semi-greedy heuristics by combining randomized adaptive construction heuristics with local search algorithms. Another example is iterated greedy algorithms [112], which consist in the repeated application of constructive heuristics that start from partial solutions. A destruction mechanism generates these partial solutions by removing solution components from a complete candidate solution.

The ACO Algorithmic Framework

ACO algorithms can also be seen as an extension of construction heuristics. The main characteristics that distinguish an ACO algorithm from other metaheuristics and, in particular, other constructive metaheuristics are the following: (i) it is a population-based algorithm, where m solutions are generated at each iteration, (ii) solutions are generated by a probabilistic constructive mechanism that is biased by numerical information called pheromones (and possibly by heuristic information), and (iii) the pheromones are updated during the run of the algorithm using the quality of generated solutions as feedback.

The general outline of an ACO algorithm is given in Fig. 1. After initializing data structures and parameters, an ACO algorithm generally iterates through two procedures: solution construction (procedure ConstructSolutions) and pheromone update (procedure UpdatePheromones). Additionally, a local search algorithm may be used to improve some or all the solutions constructed in the current iteration.

```
procedure ACO_Metaheuristic
    repeat
        for each ant do
            repeat
                ExtendPartialSolutionProbabilistically()
            until solution is complete
        for each ant ∈ SelectAntsForLocalSearch() do    // optional
            ApplyLocalSearch(ant)                       // optional
        EvaporatePheromones()
        DepositPheromones()
    until termination criteria met
end
```

Fig. 1 General pseudo-code of the ACO metaheuristic for NP-hard problems. First, each ant constructs a complete solution by probabilistically choosing solution components to extend their current partial solution (section "Solution Construction"). Some of these solutions are sometimes further improved by means of local search (section "Local Search"). Finally, pheromone values are updated according to the solutions constructed by the ants (section "Global Pheromone Update"). The update usually consist of two complementary steps: evaporation decreases pheromone values and deposit increases them

Clearly, each of these procedures may in turn consist of different phases and, depending on the particular ACO algorithm implemented, the particular choices taken within these phases may differ quite strongly. These differences, however, mainly concern specific building blocks or parameter settings that are used within the ACO algorithm phases. These building blocks and parameters are referred as algorithmic components.

In the following, we present the algorithmic components that have been proposed in typical ACO algorithms for combinatorial optimization and discuss alternative options proposed in the literature for implementing these components.

Choice of Pheromone Trails and Heuristic Information

In ACO, the solution construction is probabilistically biased by two types of information: pheromone trails and heuristic information.

The pheromone trails are a set $\mathscr{T}$ of numerical values associated, in principle, to all solution components, that is, $\forall c \in \mathscr{C}$, $\exists \tau_c \in \mathscr{T}$. Pheromone trails are updated during the run of an ACO algorithm according to the quality of previously generated solutions through the mechanisms of pheromone deposit and pheromone evaporation. During the solution construction procedure, pheromone trails bias the choice of solution components toward constructing high-quality solutions. A high value of τ_c represents a higher probability of adding solution component c to the solution being constructed. It is therefore important to appropriately define the set of

solution components $\mathscr{C}$, and the corresponding pheromone trails $\mathscr{T}$, when applying an ACO algorithm to a particular problem.

For many problems, the appropriate definition of solution component and, thus, of pheromone trail is rather straightforward. In the TSP, the adjacency information is relevant, that is, which node is the direct successor or predecessor to another one, and therefore a solution component $c_{ij} = (i, j)$ typically refers to the edge connecting nodes i and j. Thus, τ_{ij} represents the pheromone trail associated with the edge. This definition of solution component is also related to the choices made in the nearest neighbor heuristic, which underlies the construction mechanism used in almost all ACO applications to the TSP. Differently, in the QAP, the individual assignments of facilities to locations are of crucial importance. Therefore, a solution component c_{ij} denotes that facility j is assigned to location i in a candidate solution π, that is, $\pi_i = j$ and the pheromone trails τ_{ij} then represent the desirability of assigning facility j to location i. When constructing solutions for the QAP, the order in which facilities are chosen for assigning them to locations may also have an impact on the solution that is eventually constructed. Such an order may also be determined during solution construction by the ants through pheromone trails that indicate which facility should be chosen next for an assignment to a location [120]. Hence, one could have two types of pheromones, the first type τ_{ij} would refer to the desirability of assigning facility j to location i and the second one τ'_{kl} would refer to the desirability of generating an assignment for facility l directly after having assigned facility k. However, no effective implementation of such a second type of pheromone trails seems to be available.

For more complex problems than the TSP and the QAP, alternative definitions of solution components, and the corresponding pheromone trails, may have to be considered [101]. How to define solution components and their associated pheromone trails is itself an ongoing research topic [17, 99]. However, a basic guideline would be to consider effective greedy constructive heuristics and then define the pheromone trails to bias the most important decisions done by this heuristic. Using this guideline, standard definitions of solution components and pheromone trails will be available for many known problems; however, for more complex problems, some research effort may be required to determine the most appropriate solution components and pheromone trails.

The heuristic information $\mathscr{H}$ is also a set of numerical values associated with solution components ($\forall c \in \mathscr{C}, \exists \eta_c \in \mathscr{H}$). However, in contrast to the pheromone trails, heuristic information is not updated during the run of the algorithm in dependence of the quality of the solutions generated. Instead, η_c is a value that is either constant for each solution component, when static heuristic information is used, or a value that is a function of the partial solution generated so far, when adaptive heuristic information is used. The actual formulation of the heuristic information is specific to each problem. However, its computation should be fast as the value of η_c is used at every step of the solution construction by each ant. In some problems there is no readily available heuristic information that effectively guides solution construction, and thus, no heuristic information is used. This is, for example, the case in most ACO applications to the QAP.

Solution Construction

Solution construction is the process by which a new set of solutions is generated at each iteration of an ACO algorithms. In ACO terms, a solution is constructed by an ant; in optimization terms, each ant corresponds to the execution of a probabilistic solution construction procedure. In particular, each ant constructs one solution by starting from an empty solution $s = \langle\rangle$ and adding, at each construction step i, one solution component c_j from a set of candidate components N_i. The probabilistic choice at each step of the solution construction can be modeled by a probability distribution that assigns to each solution component the probability with which it is chosen. Thus, for each c_j there is a probability $Pr(c_j)$ of being chosen. This probability depends on the pheromone trails, the heuristic information, and the current partial candidate solution s, that is, $Pr(c_j|\mathscr{T}, \mathscr{H}, s)$. There are many different ways of computing this probability. Many ACO algorithms use the probabilistic rule that was introduced for ant system (AS) to choose one element from N_i as follows:

$$\Pr(c_j) = \frac{{\tau_j}^{\alpha} \cdot {\eta_j}^{\beta}}{\sum_{c_k \in N_i} {\tau_k}^{\alpha} \cdot {\eta_k}^{\beta}} \quad \forall\, c_j \in N_i, \tag{3}$$

where α and β are parameters that control the relative importance of pheromone trails and heuristic information on the decision probabilities. If α tends to zero, the solution construction becomes a probabilistic multi-start greedy algorithm; if β tends to zero, the solution construction is biased only by the pheromone trails, and heuristic information is neglected. This rule assigns a probability in a fashion similar to the well-known roulette wheel selection of evolutionary algorithms [51], where the value ${\tau_j}^{\alpha} \cdot {\eta_j}^{\beta}$ plays the role of the fitness assigned to a candidate. Clearly, many different ways can be used to define probability distributions for the solution components. For example, Maniezzo [84, 85] uses in his ANTS algorithm an additive way of combining pheromone trails and heuristic information:

$$\Pr(c_j) = \frac{\alpha \cdot \tau_j + (1-\alpha) \cdot \eta_j}{\sum_{c_k \in N_i} \alpha \cdot \tau_k + (1-\alpha) \cdot \eta_k} \quad \forall c_j \in N_i. \tag{4}$$

The above equation has the advantage over the most common one (Eq. 3) that the operations used (multiplication instead of exponentiation and sum instead of multiplication) are measurably faster in current CPUs. However, one needs to ensure that the range of values of the pheromone trails and of the heuristic information is similar to avoid undesired biases.

A method for making the construction more deterministic was proposed by Dorigo and Gambardella [34] with their ant colony system (ACS) algorithm. They introduced the *pseudorandom proportional rule*, which is controlled by a parameter q_0. At each construction step i, a random value q is drawn uniformly from $[0, 1)$; if

$q > q_0$, the probability of c_j is computed as in Eq. 3; otherwise (i.e., when $q \leq q_0$) the choice is made as

$$c_j = \arg\max_{c_k \in N_i} {\tau_k}^{\alpha} \cdot {\eta_k}^{\beta}, \tag{5}$$

that is, when $q \leq q_0$, the solution component with maximum probability is chosen deterministically. A larger value of q_0 is equivalent to a more "greedy" construction procedure, which usually results in faster convergence, whereas smaller values of q_0 lead to more varied solutions and, thus, to more exploration.

Solution construction is one of the critical components of ACO, thus techniques have been proposed to improve its efficacy and efficiency. Efficacy is sometimes improved by using *lookahead* [96], that is, by considering more than one component at a time when choosing which component should be added to the current partial solution. Efficiency can be greatly improved, particularly in large problem instances, by restricting the choice of solution components to *candidate lists* [33, 34]. Candidate lists contain the most promising solution components, and their definition depends on the instance data and/or on the current partial solution. The most efficient candidate lists depend only on the instance data and are typically computed at the start of the algorithm. One example of candidate lists is the nearest neighbor lists in the TSP, which store the *cl* nearest neighbors to each city ordered according to nondecreasing distances. Even a small value of *cl*, such as 20, is sufficient to generate very-high-quality (or even optimal) solutions to the TSP. Such candidate lists are crucial when applying ACO algorithms to large TSP instances.

Candidate lists may also save memory if the algorithm only stores the pheromone values of those solution components that are part of the candidate lists. Another approach to save memory used in some ACO variants is to avoid explicitly storing the pheromone trails of all solution components by using, for example, a hash table [3] to store only the pheromone values associated with solution components that have appeared in previouslyconstructed solutions. This might be beneficial when the fraction of pheromone values explicitly stored is small enough (relative to the total size of $\mathscr{T}$) to compensate for the fact that insertion, deletion and look-up in the hash table require more than constant time.

Independent of the usage of candidate lists, solution construction may be speeded up when using static heuristic information by precomputing the values of ${\tau_j}^{\alpha} \cdot {\eta_j}^{\beta}$, $\forall c_j \in \mathscr{C}$, as all ants use the same total values in Eqs. 3 and 5, and each ant's partial solution only affects which precomputed values are considered at each construction step. If an adaptive heuristic information is used, the total values would depend on the partial solution of each ant; hence, they cannot be precomputed in this way.

Most ACO algorithms construct solutions starting from an empty solution. However, a few proposals have studied the possibility of starting from partially constructed solutions with the goal of partially destroying a very good solution and reconstructing from it a, hopefully better, new one. This is the same concept applied by iterated greedy [112]. Examples of this kind of ACO algorithm are ACO algorithms using external memory [1], iterated ants [133], and cunning ants [132].

Considering the effectiveness of the solution construction, these methods are useful as they avoid the most expensive construction steps that happen when solutions are empty or almost empty. In the Enhanced ACO [47], solution construction is guided by the global-best solution, the best solution found since the start of a run of the algorithm, by adopting the choice done for that solution with a fixed probability at each construction step. This approach somehow introduces a guided perturbation into the best solution found so far and, thus, a straightforward way of increasing the exploitation of good solutions in ACO. It has also the advantage of resulting in faster solution construction as no probabilistic choices among several candidate components need to be done; thus, it is deemed to be useful particularly for tackling large instance sizes.

Typically, construction rules take into account the pheromone trails associated with only single solution components. However, for specific problems it may be useful to include also the pheromone associated with other solution components. An example are scheduling problems, for which the *pheromone summation rule* was proposed [92]. Its goal is to avoid that jobs are scheduled in a position too far away from positions where they occur in high-quality solutions.

Global Pheromone Update

As mentioned above, the pheromone trails are modified during the run of an ACO algorithm in order to bias the construction of new solutions. Pheromone update usually consists of two complementary steps: pheromone evaporation and pheromone deposition. The general idea behind these two steps is to bias the pheromone trails so as to favor the construction of high-quality solutions. This general idea can be implemented in a number of different ways.

Pheromone evaporation decreases the pheromone values by some factor with the goal of reducing the effect of previous pheromone depositions and, in this way, help to forget previous poor decisions. Pheromone evaporation itself can be applied, in principle, to some or all the pheromone trails. It can be described as

$$\mathscr{T}_{\text{new}} = \mathsf{evaporation}(\mathscr{T}_{\text{old}}, \rho, S^{\text{eva}}), \tag{6}$$

where $\rho \in (0, 1)$ is a parameter called evaporation rate and S^{eva} denotes the set of solutions that are selected for evaporating the pheromone trails. Most of the existing ACO algorithms do not make use of the solutions in S^{eva} to implement the pheromone evaporation and simply reduce the pheromone trail value of all solution components by the same factor:

$$\tau_j = (1 - \rho) \cdot \tau_j \qquad \forall \tau_j \in \mathscr{T}. \tag{7}$$

A value of $\rho = 1$ would mean that pheromone trails are reset at each iteration of the algorithm and, thus, no learning occurs. A value of $\rho = 0$ would mean that no evaporation occurs, which would result in an unlimited accumulation of

pheromones. For intermediate values of ρ, the amount of pheromone decreases geometrically as a function of this parameter.

In some ACO algorithms, the global pheromone evaporation only applies to some specific solution components. For example, in ACS pheromone evaporation affects only the solution for which pheromone is deposited in the current iteration:

$$\tau_j = (1 - \rho) \cdot \tau_j \qquad \forall \tau_j \mid c_j \in s^{\text{eva}}, \tag{8}$$

where $\forall \tau_j | c_j \in s^{\text{eva}}$ are all the pheromone trails associated with solution components that occur in the solution s^{eva} chosen for pheromone deposition (see also below). Equation 6 is also general enough to include the pheromone evaporation applied in population-based ACO [55], where the pheromone trails are defined as a function of the solution components occurring in a set of candidate solutions and where pheromone evaporation corresponds to the reduction of the amount of pheromone that is associated with components of the solutions that leave this set of candidate solutions.

Evaporation has the effect of slowing down the convergence of the algorithm as it reduces the probability of constructing again solutions previously constructed. However, while pheromone deposition selectively reinforces the pheromone trails associated with some solution components, pheromone evaporation has the effect of decreasing the probability of selecting those solution components less recently reinforced, thus allowing the algorithm to focus on the most recently found solutions and to "forget" previous pheromone depositions.

Pheromone deposition consists in increasing the pheromone values of a few selected solution components. These solution components belong to one or more solutions previously constructed by the ACO algorithm. In a general form, the pheromone deposition can be written as

$$\tau_j = \tau_j + \sum_{s_k \in S^{\text{upd}} | c_j \in s_k} w_k \cdot F(s_k), \tag{9}$$

where $S^{\text{upd}} \subseteq S^{\text{eva}}$ is the set of solutions chosen to deposit pheromones, w_k is a weight that is associated with solution $s_k \in S^{\text{upd}}$, and $F(s_k)$ is a function that is nondecreasing with respect to the quality of the solution s_k – that is, whenever $f(s_i) < f(s_l)$ in the minimization case, then it follows that $F(s_i) \geq F(s_l)$. The amount $w_k \cdot F(s_k)$ therefore corresponds to the amount of pheromone that is deposited by a solution s_k.

The solutions used for updating the pheromone trails (S^{upd}) have a strong influence in the behavior of an ACO algorithm. Ant system (AS) selects all solutions constructed in the latest iteration. In contrast, there are alternative methods that consider one single solution for the pheromone deposition, e.g., the iteration-best update (ib-update), which uses the best solution from those generated in the most recent algorithm iteration; the global-best update (gb-update), which uses the best solution found since the start of the algorithm; and the restart-best update

(rb-update), which uses the best solution found since the pheromones were reinitialized (see also below on pheromone reinitialization). Intermediate alternative methods would define a set of candidate solutions that may comprise a number of the best solutions of the latest algorithm iteration and solutions such as the global-best or the restart-best ones and use these to deposit pheromone. The particular set of solutions that is chosen to deposit pheromone has a direct effect on the speed of convergence and possible stagnation behavior of the algorithm, which occur when the pheromone trails have converged and the same solutions are constructed over and over again. The gb-update provides the fastest convergence but may more easily lead to such stagnation, while allowing all candidate solutions to deposit pheromones may delay the convergence of the algorithm [35].

In addition to the choice of solutions that form S^{upd}, the amount of pheromone deposited by these solutions also has a direct impact on the search behavior of ACO algorithms. A typical setting in ACO algorithms is to make the amount of pheromone deposited inversely proportional to the objective function value (in the minimization case), that is, to set $w_k \cdot F(s_k) = 1/f(s_k)$. This is the case, for example, for MMAS [125]. (Instead of making the amount of pheromone deposited a function of the quality of the solution, one may also deposit a constant amount of pheromone if the bias toward good solutions is ensured by choosing ib-update, gb-update, or similar biases toward the best candidate solutions.) Various other ACO algorithms add additional weighting factors that may be dependent or not on the quality of the solution relative to others. ACS uses a pheromone update by setting $w_k \cdot F(s_k) = \rho/f(s_k)$, where ρ is the evaporation rate. AS initially used a setting of $w_k \cdot F(s_k) = Q/f(s_k)$, where Q is a parameter; however, in many recent implementations, one simply uses $Q = 1$. Various ACO algorithms use an unequal weighting for the amount of pheromone deposited. For example, RAS [19] makes the weight dependent on the rank of a solution in the current algorithm iteration by choosing $w_k \cdot F(s_k) = \max\{0, w - r\}/f(s_k)$, where $w = |S^{\text{upd}}|$ is the number of solutions that deposit pheromone after each iteration (this parameter is called *rasrank* in section "ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP") and r is a solutions' rank in the current iteration. The largest amount of pheromone ($w/f(s_k)$) in RAS is assigned to the global-best solution s_{gb}. This corresponds to a choice where S^{upd} comprises the global-best solution and the $w - 1$ best-quality solutions of the current iteration. In EAS [30, 36, 38], the usual AS deposition rule is followed, that is, all solutions generated in the current iteration deposit pheromone; in addition, the global-best solution deposits an amount of pheromone $w_{\text{gb}} \cdot F(s_{\text{gb}}) = Q \cdot m_{\text{elite}}/f(s_{\text{gb}})$, where Q is the same parameter as for AS and m_{elite} is the multiplicative factor that increases the weight of the global-best solution. In BWAS [21], not only the global-best solution deposits pheromone, but also further evaporation is applied, following Eq. 7, to the pheromone trails associated with the solution components that appear in the worst solution of the current iteration and that do not appear in the global-best one.

Pheromone Update Schedule

As said above, the choice of the solutions that deposit pheromone has a strong impact on the search behavior of ACO algorithms. The main goal of pheromone update schedules is to adapt the choice of the solutions that deposit pheromone during the ACO algorithm run. In the simplest case, this is done following a predefined schedule. The first algorithm to make such a predefined schedule explicit is MMAS in its application to the TSP [118,125] (related ideas are also discussed in [90]). In particular, when local search is applied, the central idea is to shift from a frequent use of the ib-update, which is used by default, toward an increasing frequency of the gb-update (or an rb-update). This shift has the effect of changing the search behavior from a more explorative one toward a more exploitative one that searches for new, improving solutions around the best-so-far ones. When used after the reinitialization of the pheromone trails (see section "Pheromone Reinitialization"), the schedule can also switch in the pheromone deposit between ib-update, rb-update, and gb-update. In particular, if the restart-best solution is used every iteration and no new restart-best solution is found for a number of iterations, the schedule switches to gb-update. As a result, the pheromone trails will implicitly interpolate between the restart-best solution and the global-best solution in a way that resembles the ideas underlying path relinking [50].

The update schedule may have a critical impact on the performance of an ACO algorithm, because it determines the balance between speed of convergence and exploration capabilities. It is also very sensitive to the termination criteria: an update schedule that performs well for long computation times may converge too slowly if the algorithm is terminated much earlier.

Initialization of Pheromones

Two alternative ways of initializing the pheromone trails have been proposed: either using a very small initial value (ACS and BWAS) or using a rather large value (MMAS), where small and large are relative to the amount of pheromone deposited in the global pheromone update. A small value results in a rather quick bias toward the best solutions, while a large value results in a more explorative initial search phase (though depending also on other parameters such as the evaporation rate). Neither the original AS (nor EAS and RAS) specify how the pheromones are initialized and leave it open as a parameter τ_0 to be specified by the user.

Pheromone Reinitialization

It has been shown that resetting the pheromone values back to their initial value may help in long runs by increasing the exploration of the search space [125]. This procedure is often called *restart*, since it is equivalent to restarting the run,

although the information about the global-best solution found is kept. While most ACO algorithms do not implement such restarts, restarting has been shown to be very effective for problems such as the TSP and the QAP where the use of strong local search algorithms leads to fast convergence.

MMAS was the first ACO algorithm to employ pheromone reinitialization to avoid stagnation. In particular, MMAS computes a measure, called branching factor, of the potential alternatives encoded in the pheromone trails. When the branching factor goes under a certain threshold value (close to 1), pheromone values are reset to their maximum value ($\tau_{\max}$).

Another ACO algorithm that uses pheromone reinitialization is best-worst ant system (BWAS) [21], which reinitializes the pheromone values to τ_0 whenever the distance, in the decision space, between the global-best solution and the worst solution found in the last iteration falls under a threshold value.

In order to avoid very frequent restarts, there is often a "grace period" when no restarts are allowed; for example, a certain number of iterations since the last restart-best solution was found.

Local Pheromone Update

Local pheromone update is a mechanism that works during solution construction. It modifies the amount of pheromone associated with solution components that have just been chosen by the ants. The first such mechanisms were studied during the design of AS in the so-called ant-density and the ant-quantity models [20,30,37] but were abandoned due to their poor performance. In later research, local pheromone update to diminish the pheromone trails associated with chosen solution components was explored in the design of the ACS algorithm [34]. In particular, ACS uses local pheromone update following

$$\tau_j = (1 - \xi) \cdot \tau_j + \xi \cdot \tau_0 \qquad \forall \tau_j \mid c_j, \tag{10}$$

where c_j is the solution component just selected by an ant during solution construction, $\xi \in (0, 1)$ is a parameter controlling the strength of the local update, and τ_0 is the initial pheromone value, which is set to a very small value, much lower than the expected amount of pheromone deposited in the global pheromone update. The effect of this update rule in ACS is to increase the search exploration during construction making used solution components less attractive.

Although the local pheromone update is very effective when combined with the other characteristics of ACS, it is difficult to incorporate it on its own to other ACO algorithms, because it relies on a strong gb-update and a lack of pheromone evaporation.

An important implementation detail is whether the solutions are constructed in *parallel*, that is, each ant chooses one solution component at a time, or *sequentially*, that is, each ant completes its own solution before the next one starts constructing

their own. The different construction schemes may introduce differences in the effect of the local pheromone update.

Pheromone Limits

MMAS introduced the concept of explicit pheromone limits that restrict the possible values the pheromone strength can take to the range $[\tau_{\min}, \tau_{\max}]$. The goal is to prevent stagnation, which occurs when the pheromone values of some solution components are so low that there is no possibility the component will ever be selected (or alternatively, the pheromone values of some solution components are so high that there is no possibility that any other component will ever be selected).

The original proposal showed how to adapt the limits within a run of MMAS. In practice, however, the upper pheromone limit seems to be less important as the maximum amount of pheromone possible on any solution component is already limited by pheromone evaporation.

The definition of $\tau_{\min}$ and $\tau_{\max}$ is problem specific. In general, $\tau_{\max}$ should be an estimation of the pheromone added by the optimal solution. In the TSP, for instance, it corresponds to $\tau_{\max} = \frac{1}{\rho \cdot f(s_{\text{opt}})}$; however, since the optimal solution s_{opt} is not known, the global-best solution s_{gb} is used instead.

The main parameter controlling the pheromone limits is p_{dec}, which is the probability that an ant chooses exactly the sequence of solution components that reproduces the best solution found so far. The value $\tau_{\min}$ is given by

$$\tau_{\min} = \frac{\tau_{\max} \cdot (1 - \sqrt[n]{p_{\text{dec}}})}{n' \cdot \sqrt[n]{p_{\text{dec}}}} \tag{11}$$

where n' is an estimation of the number of solution components available to each ant at each construction step. This value often corresponds to $n/2$ (or to half the size of the candidate lists if they are used). Nonetheless, in some implementations of MMAS, and specially when local search is used to improve the solutions constructed by the ants, the above computation may be simplified to $\tau_{\min} = \frac{\tau_{\max}}{2 \cdot n}$.

ACS also implicitly uses pheromone trail limits. Due to the way its local pheromone update rule is defined, the pheromone trails can never fall under the value τ_0, and they cannot grow above $1/F(s_{\text{gb}})$, thus limiting implicitly the pheromone trails to the range $[\tau_0, 1/F(s_{\text{gb}})]$.

Local Search

In many combinatorial problems, the use of local search algorithms is essential for obtaining competitive results. Local search algorithms start from an initial solution and iteratively apply small changes to it, as defined by a *neighborhood operator*, in order to find an improved solution. Two of the most basic local search algorithms

are best improvement and first improvement, which replace the current solution with the best- and first-improving solution found in the neighborhood, respectively. These algorithms stop if, after examining the whole neighborhood of the current solution, no improved solution can be found, thus stopping at a local optimum. Other local search algorithms that do not necessarily stop at a local optima, such as tabu search or simulated annealing, have also been combined with ACO algorithms [82, 120].

When combined with an ACO algorithm, local search is typically applied to the solutions constructed by the ants. The decision about which ant applies local search to its solution depends on the effectiveness and efficiency of the specific local search [47]. If the local search is relatively fast and effective, it may be worth applying it to all ants' solutions. Otherwise, restricting it either to the iteration-best ant or to a candidate set of promising solutions may save computation time, while still giving a chance to further improve the best solution found by the ants.

ACO Algorithms as Instantiations of the ACO Metaheuristic

Given the algorithmic components discussed above, then ACO algorithms from the literature can be described as combinations of specific components. (This list includes only a small subset of the ACO algorithms and algorithmic components proposed in the literature. A more comprehensive component view of ACO is left for further work.)

In particular, AS uses Eq. 3 for solution construction for all ants, Eq. 7 for pheromone evaporation, and Eq. 9 for the pheromone deposition with all ants using the same weight $w_1 = \cdots = w_m$, where m is the number of ants. EAS, RAS, BWAS, and MMAS all use the same solution construction rules and the same evaporation mechanism as AS, but they differ from AS in the way pheromone is deposited; also, they add new algorithmic components. In particular, the pheromone deposition of EAS uses the global-best solutions with a weight m_{elite}. Similarly, the only difference between AS and RAS is the way pheromone is deposited, which also uses the global-best solution and weights defined according to a parameter called *rasrank*, already explained in section "Global Pheromone Update". The pheromone deposition of BWAS also uses the global-best solution and, in addition, applies further evaporation to pheromone trails associated with solution components that appear in the worst solution of each iteration and not in the global-best one. In addition, BWAS performs pheromone reinitialization depending on the average distance between the iteration-worst solution and the global-best one. MMAS adds several features: it enforces minimum and maximum limits to pheromone values and uses an update schedule to switch between iteration-best, global-best, and restart-best deposition and pheromone reinitialization according to branching factor, and pheromones are initialized to a high initial value. MMAS variants that use the pseudorandom proportional rule (Eq. 5) have also been considered in the literature [124]. Finally, ACS is the ACO algorithm that structurally differs the most from AS: it adds the pseudorandom proportional rule to AS solution construction, pheromone is deposited using the global-best solution, and evaporation is performed

at the same time on those values updated during pheromone deposition (Eq. 8). Moreover, further evaporation is performed during solution construction (local pheromone update, section "Local Pheromone Update") by each ant.

ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP

We present in this section a software framework that unifies the implementation of several ACO algorithms found in the literature. The code is publicly available at http://iridia.ulb.ac.be/aco-tsp-qap/. The implementation aims at generality, by clearly separating the general components of the ACO metaheuristic from the problem-specific components that vary from one problem to another. In this chapter, we present an implementation for the TSP and the QAP. However, it is possible to extend the framework to other combinatorial optimization problems with far less effort than an implementation from scratch.

In its current form, the software implements AS, MMAS, EAS, RAS, ACS, and BWAS. In addition to the usual ACO parameters (α, β, ρ, m, ξ, etc), it also allows combining various algorithmic components by setting specific parameters. In particular:

- Pheromone limits as defined by MMAS can be enabled or disabled independently of the algorithm chosen (except for ACS).
- Pheromone reinitialization (restart) can also be enabled or disabled independently of the algorithm. In particular, when more than res_{it} iterations have passed since the last restart-best solution was found, a restart condition is tested. This restart condition may be the branching factor going under a threshold value (res_{bf}), as in MMAS, or the distance between the global-best and the iteration-worst ants going under a threshold value (res_{dist}). In addition, a setting of *always* (*never*) means that the restart condition is always (never) satisfied.
- Any algorithm may use the pseudorandom proportional rule of ACS (Eq. 5) by simply using a value $q_0 > 0$.

In the problem-specific part, the implementation for the TSP follows what is provided by the ACOTSP software [119], that is, heuristic information, candidate lists, and various types of local search, including 3-opt first improvement with don't-look-back bits (*dlb-bits*) and nearest neighborhood lists (*nnls*). The part corresponding to the QAP does not implement heuristic information nor candidate lists, since these techniques have not proved useful for the QAP. It does implement, however, various local search algorithms, such as the fast 2-exchange best- and first-improvement local searches proposed by Taillard [128] and the robust tabu search proposed in the same paper.

The update schedule (section "Pheromone Update Schedule") is implemented as specific to each problem and ACO algorithm; however, it would be straightforward to extend the update schedule to all ACO algorithms. In the case when MMAS is

Table 1 Default parameter settings for each ACO algorithm from the literature. Parameters that are available only for particular ACO algorithms are described under the table. The default parameter configuration used in the experiments is based on MMAS (see Table 2)

	m	ρ	q_0	*ph-limits*	*Restart*	Restart parameters
	TSP / QAP	TSP / QAP				
AS, EAS, RAS	25 / 5	0.5 / 0.2	0.0	No	Never	
BWAS	25 / 5	0.5 / 0.2	0.0	No	Distance	$res_{\text{dist}} = 0.05$
MMAS	25 / 5	0.2 / 0.2	0.0	Yes	Branch-factor	$res_{\text{bf}} = \begin{cases} 1.00001 & \text{(TSP)} \\ 1.1 & \text{(QAP)} \end{cases}$
ACS	10 / 5	0.1 / 0.2	0.98	No	Never	

$m_{\text{elite}} = n$ when using EAS, $rasranks = 6$ when using RAS, $\xi = 0.1$ when using ACS, $slen = 250$ (TSP) or 20 (QAP) when using MMAS

used together with local search, the user can control the frequency with which the restart-best (or global-best) solution, instead of the iteration-best one, is used to update the pheromone trails, by means of a parameter called *schedule length* (*slen*). Higher values mean longer emphasis on the iteration-best solution, thus possibly increasing exploration at the expense of faster convergence.

In the two following sections, we show how to find a good performing ACO algorithm for a specific class of instances of the TSP and the QAP by automatically configuring the proposed ACO framework. In particular, there is, for each problem, a set of training instances and another set of testing instances. The automatic configuration tool irace [83] is used here to find a parameter configuration given the set of training instances. Finally, this parameter configuration is compared with the default configuration for each problem by evaluating both configurations on the set of test instances.

The implementation of the proposed ACO framework assigns default values, consistent with the literature, to certain parameters depending on the particular ACO algorithm chosen. These default values are documented in Table 1 for completeness. For the purpose of automatic configuration, we consider a single default configuration based on MMAS. Table 2 summarizes, for each problem, the parameters of the proposed ACO framework, their domain, and default values considered in the automatic configuration experiments described below. Although some parameters are still exclusive to a particular ACO algorithm (e.g., ξ can only be used together with ACS), the framework allows using components from one ACO algorithm in others (e.g., the restart mechanism of BWAS can be enabled for any algorithm by setting parameter *restart* to the value "*distance*").

Finding a Better ACO Configuration for the TSP

In the case of the TSP, we consider random Euclidean TSP instances, in particular 50 instances of each size $\{1000, 1500, 2000, 2500, 3000\}$ for training and other 50

Table 2 Domains and default values of the parameter settings of the ACO framework considered in the automatic configuration experiments for the TSP and the QAP. A value of n/a means that the parameter does not exist for the corresponding problem. A value of "–" means that the parameter has no value in the default configuration (because it depends on another parameter setting that is not enabled by default). The list of parameters enabled only for certain values of other parameters is given under the table

	TSP		QAP	
Parameter	Domain	Default	Domain	Default
algorithm	{AS, EAS, RAS, ACS, MMAS, BWAS}	MMAS	{AS, EAS, RAS, ACS, MMAS, BWAS}	MMAS
m	$[1, 500]$	25	$[1, 10]$	5
α	$(0.0, 5.0)$	1.0	$(0.0, 5.0)$	1.0.
β	$(0.0, 10.0)$	2.0	n/a	n/a
ρ	$(0.01, 1.0)$	0.2	$(0.01, 1.0)$	0.2
q_0	$(0.0, 1.0)$	0.0	$(0.0, 1.0)$	0.0
cl	$[5, 50]$	20	n/a	n/a
ξ	$(0.01, 1.0)$	–	$(0.01, 1.0)$	–
rasrank	$[1, 100]$	–	$[1, 100]$	–
m_{elite}	$[1, 750]$	–	$[1, 750]$	–
p_{dec}	$(0.001, 0.5)$	–	$(0.001, 1)$	0.005
ph-limits	{Yes, no}	Yes	{Yes, no}	Yes
slen	$[20, 500]$	250	$[5, 250]$	20
restart	{Never, branch-factor, distance, always}	Branch-factor	{Never, branch-factor, distance, always}	Branch-factor
res_{bf}	$(1.0, 2.0)$	1.00001	$(1.0, 2.0)$	1.1
res_{dist}	$(0.01, 0.1)$	–	$(0.01, 0.1)$	–
res_{it}	$[1, 500]$	250	$[1, 50]$	5
localsearch	{None, 2-opt, $2.5-opt, 3-opt$}	3-opt	{None, best-2-opt, short-tabu-search, long-tabu-search}	Best-2-opt
dlb-bits	{Yes, no}	Yes	{Yes, no}	No
nnls	$[5, 50]$	20	n/a	n/a
rasrank	When $algo =$ RAS			
m_{elite}	When $algo =$ EAS			
ξ	When $algo =$ ACS			
p_{dec}	when *ph-limits* = yes (and for the TSP only if also *restart* = never)			
ph-limits	When $algo \neq$ ACS			
slen	When $algo =$ MMAS			
res_{bf}	When *restart* = branch-factor			
res_{dist}	When *restart* = distance			
res_{it}	When *restart* ≠ never			
dlb-bits, *nnls*	When $localsearch \neq none$			

Table 3 Best configurations found by irace for the TSP

algo	*m*	α	β	ρ	q_0	ξ	*cl*	*nnls*	*ph-limits*	*slen*	*restart*	res_{it}
ACS	28	3.07	5.09	0.32	0.53	0.21	22	9	–	–	Branch-factor ($res_{\text{bf}} = 1.74$)	212
MMAS	40	0.94	4.11	0.72	0.14	–	18	12	Yes	191	Branch-factor ($res_{\text{bf}} = 1.91$)	367

Common parameters: *localsearch* = 3-opt + dlb-bits

instances of each size for testing. A single run of irace has a maximum budget of 25,000 runs of the ACO software, and each run is stopped after 60 CPU seconds. Since the goal is to investigate whether it is possible at all to find a better ACO algorithm than the default one from the literature and in order to speed up the process, irace starts from the default configuration for the TSP (Table 2) as an initial solution. After irace finishes, the configuration found is evaluated on the test instances. In particular, we select two configurations found by irace that improve over the default (see Table 3): one is a tuned variant of MMAS and the other is a tuned variant of ACS. These two configurations use a higher value of β and a stronger evaporation ρ than the default configuration. Since $q_0 > 0$, this MMAS variant uses the pseudorandom proportional rule from ACS, which is a MMAS variant that has also been explored previously in the literature [124]. It is also interesting that the restart strategy is typically much more aggressive than the default, with a much larger threshold branching factor, which results in more frequent restarts, only limited by the grace period (res_{it}). Other parameters, such as the size of the candidate list (*cl*), remain mostly unchanged from the default configuration.

Figure 2 compares the results obtained by these two tuned configurations with those obtained using the default configuration on the test instances. We ran each configuration once on each test instance up to 60 CPU seconds, and we computed the relative percentage deviation (RPD), with respect to the optimal solution, of the best solution found throughout the run. Both plots show that the solutions obtained by the tuned configurations are, in most instances, better than those obtained by the default configuration, and the differences are specially large for the largest instances. These observations are further confirmed by comparing the mean and standard deviation of the values obtained by each configuration (Table 4). In particular, a 95 % confidence interval around the mean difference between the default and the tuned configurations does not contain the value zero, which indicates that the observed differences are statistically significant.

Finding a Better ACO Configuration for the QAP

In the case of the QAP, we consider two different instance sets: RR, where the distance matrix entries correspond to the Euclidean distances between points

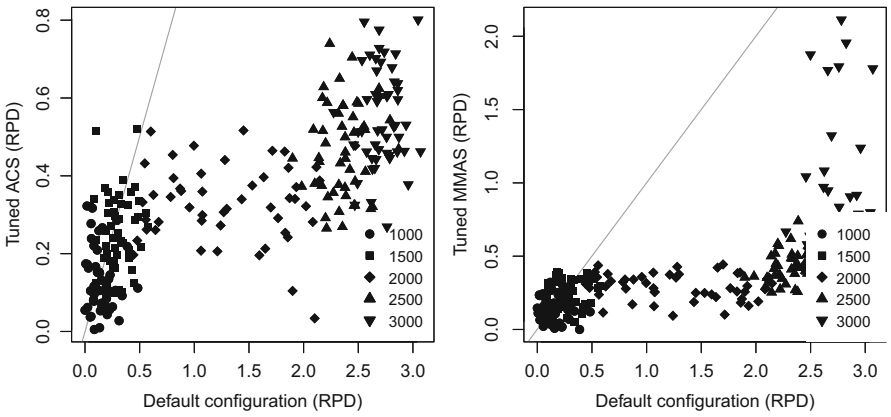

Fig. 2 Comparison of tuned vs. default configurations of ACOTSP on the test TSP instances. Tuned ACS configuration (*left*) and tuned MMAS configuration (*right*). Each point gives the relative percentage deviation from the optimal solution on one particular instance

Table 4 Results on the test TSP instances obtained by the default configuration and two tuned configurations found by irace. Objective function values are given as relative percentage deviation from the optimal solution. ΔCI denotes the 95 % confidence interval around the mean difference between the default configuration minus the configuration in that column (thus, positive values would denote that the default configuration is worse)

	Default	Tuned ACS	Tuned MMAS
mean	1.37	0.35	0.38
sd	1.08	0.18	0.32
ΔCI		[0.904, 1.14]	[0.875, 1.1]

randomly generated in a square of side length 300 and the flow matrices are randomly generated according to a uniform distribution, and RS, where the distance matrix is generated as in the previous case and the flow matrix shows a structure similar to that found in real-world QAP instances. Within each set, we consider 100 instances, and we use half of them for training and the other half for testing the ACO configurations generated. For tuning the ACO framework, we consider a similar setup as in the TSP case, that is, each run of irace has a maximum budget of 25,000 runs of the ACO software, and each run stops after 60 CPU seconds.

For each of the two training sets, RR and RS, Table 5 reports the best configuration found in three independent runs of irace. These two configurations are fairly similar but quite different from the default one. First of all, it is somehow surprising that none of them uses pheromone limits, despite this being one of the key characteristics of MMAS. This could be due to the stronger restart strategies, which do not allow pheromone values to reach the limits and, thus, enforcing the limits adds an overhead that never pays off. Also, the schedule-length (*slen*) value is more than five times higher than the default, which implies a much higher exploitation of the iteration-best solution.

Table 5 Best configurations found by irace for the QAP

	algo	*m*	α	ρ	q_0	*dlb-bits*	*ph-limits*	*slen*	*restart*	res_{it}
RR	MMAS	6	0.324	0.29	0.062	Yes	No	153	Distance ($res_{dist} = 0.051$)	22
RS	MMAS	4	0.164	0.342	0.284	Yes	No	170	Branch-factor ($res_{bf} = 1.822$)	40

Common parameters: *localsearch* = best-2-opt

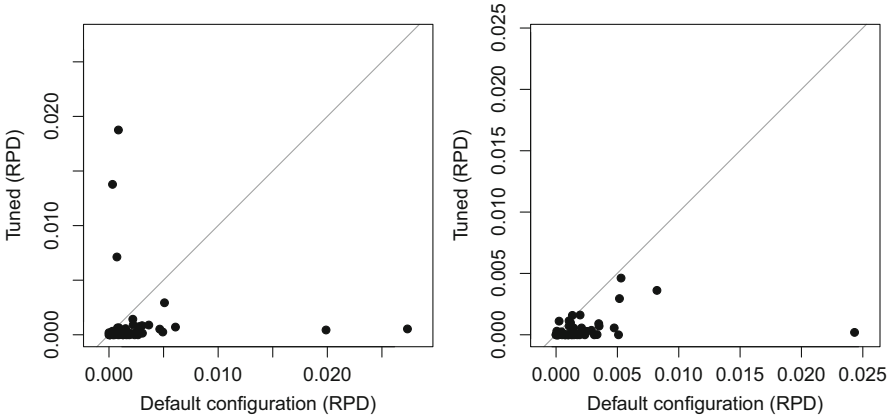

Fig. 3 Comparison of tuned vs. default configurations on the QAP test instances. RR instances (*left*) and RS instances (*right*). Each point gives the relative percentage deviation from the optimal solution on one particular instance

The two configurations of Table 5 are run on each test instance for 60 CPU seconds. Since for these instances the optimal objective values are not known, the relative percentage deviation (RPD) is computed with respect to a reference solution, for each instance, obtained by running the default configuration ten times with a time limit of 600 CPU seconds and keeping the best solution found.

Figure 3 compares the above configurations with the default on the test instances. The improvements of the tuned configurations over the default are not so clear for the QAP as they were for the TSP. In general, most RPD values are extremely small, which indicates that the variations over the best-known solutions are very small for all configurations. In both cases, there are a few instances where the default configuration obtains better RPD values than the tuned one. The statistical analysis in Table 6 shows that the mean difference between the default and tuned configuration for the RS instances is very small but still statistically significant (since the confidence interval does not contain zero), whereas for the RR instances, the null hypothesis of zero difference cannot be rejected (since the corresponding CI contains the value zero). The latter result is likely caused by the relatively large RPD values obtained by the tuned configuration on three particular instances, as shown on the left side of Fig. 3. These results could provide the motivation for

Table 6 Results on the QAP test instances obtained by the default configuration and the tuned configurations found by irace. CI denotes the 95 % confidence interval around the mean difference between the default configuration minus the configuration in that column (thus, positive values would denote that the default configuration is worse)

RR instances		
	Default	Tuned ACO
Mean	0.00241	0.00109
sd	0.00466	0.00335
ΔCI		[−0.000352, 0.00299]

RS instances		
	Default	Tuned ACO
Mean	0.00210	0.000485
sd	0.00364	0.000941
ΔCI		[0.00062, 0.00261]

exposing additional algorithmic components to automatic configuration, such as the precise update schedule of MMAS, which is known to be very sensitive to the instance characteristics [125]. Extending the proposed framework with new algorithmic components from other ACO algorithms may also further improve the results on the QAP.

Applications of ACO to Other Problem Types

Although the ACO metaheuristic was primarily designed for tackling single-objective combinatorial optimization problems, its main ideas have been extended to tackle other types of problems. In this section, we provide an overview of such problems and how ACO has been adapted to deal with their particular characteristics. Comprehensive reviews of ACO applications are available in the literature [35, 126].

Continuous Optimization Problems

In continuous optimization problems, the domain of the decision variables is the set of real numbers or a subset thereof. In some problems, the simplest approach is to discretize the real-valued domain, which would allow the direct application of ACO. This was, for example, the approach followed by [70] when applying ACO to the protein–ligand docking problem, where a discrete ACO algorithm was combined with a local search for continuous domains.

In many problems, this discretization approach is not feasible, and the problem must be tackled in the continuous domain. A first group of algorithms for continuous optimization is inspired by the behavior of some ant species [14, 39, 97]. However, these algorithms diverge from the basic framework of the ACO metaheuristic, for example, by requiring the direct communication among ants instead of a pheromone structure. Therefore, they should rather be considered a separate class of algorithms.

A second group of algorithms directly translates the ideas underlying the ACO metaheuristic to the continuous domain. For example, Socha and Dorigo [116] replace the discrete pheromone distributions by Gaussian kernel functions

that take the role of the pheromone model that is updated by the ants. Other similar approaches are found in [114, 131]. More recently, Liao et al. [74, 76] proposed a unified model of various ACO algorithms for continuous optimization problems. Their model allows the automatic configuration of new ACO algorithms for continuous optimization very much in the line of what is presented in section "ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP". They show that their final configured continuous ACO algorithms are competitive to other state-of-the-art algorithms for continuous optimization.

These approaches have also been extended to mixed-variable (continuous and discrete) problems [75, 115], by using appropriate ways of representing pheromone trails when handling discrete and continuous variables.

Multi-objective Problems

In many real-world problems, a solution is evaluated according to multiple, conflicting criteria (objectives). If there is a priori information about the importance of each objective, the objectives can be aggregated according to a preference model, and solutions are compared in this way. In the absence of such preference model, one can only say that a solution is better than another if the former is not worse in all objectives and better in at least one of them. Thus, the goal becomes to find (an approximation of) the Pareto set, i.e., the set of solutions not dominated by any other feasible solution [40, 117].

There are several proposals in the literature on how to apply the ACO metaheuristic to multi-objective combinatorial optimization problems. A few of these proposals assume an order of importance among the objectives [46]; however, most proposals attempt to approximate the Pareto set [5, 48, 80, 113]. As pointed out by López-Ibáñez and Stützle [79], many of these multi-objective ACO (MOACO) algorithms share similar algorithmic components combined in different ways. For example, Pareto ACO [27], BicriterionAnt [67], and COMPETants [26] use a different pheromone set $\mathscr{T}$ (a matrix, in their case) per objective, whereas MACS [10] and mACO-3 [2] use a single $\mathscr{T}$. Some papers have specifically compared both design choices [2, 82]. Moreover, there are basically two ways of updating the pheromone trails in MOACOs: either selecting the best solution for each objective, as done in COMPETants, Pareto ACO, and mACO-4 [2], or selecting (a subset of) all nondominated solutions, as done in BicriterionAnt, MACS, and mACO-3. Finally, a few MOACO algorithms make use of multiple colonies, where a colony is understood as a group of ants that construct solutions only according to the pheromone trails associated with their colony. The pheromone trails of each colony may correspond to multiple pheromone matrices (which are aggregated during solution construction), and colonies may exchange solutions [67]. Nonetheless, it is straightforward to define multi-colony variants of most MOACO algorithms [79].

Most of the MOACO algorithms were proposed for bi-objective problems. In the bi-objective case, the results presented in the literature indicate that the use of one pheromone matrix per objective and of multiple colonies, each of them with

their own pheromone matrices, is necessary for MOACO algorithms to perform well on different regions of the objective space [48, 79, 80]. When applied to problems with more than two objectives, this approach quickly becomes too computationally expensive. Notable exceptions are MOACO algorithms based on population-based ACO [4, 56]. However, a thorough comparison with other MOACO approaches on problems with more than two objectives is missing in the literature.

Dynamic Problems

Dynamic problems are those in which some information about the problem changes or becomes available after the start of the algorithm. This is the typical case in network routing, where ACO algorithms specifically designed for these problems have achieved notable results [23, 24, 130]. These algorithms differ significantly from the classical ACO algorithms for combinatorial optimization problems: ants are launched asynchronously in a distributed way, and no global pheromone update takes place. However, there are also in the literature dynamic variants of classical combinatorial problems, such as the dynamic TSP and the dynamic vehicle routing problem (VRP), where cities may appear or disappear and the distances between them may change. ACO algorithms for the dynamic TSP [41, 53], the dynamic QAP [54], and the dynamic VRP [29, 98] follow more closely the general outline of ACO algorithms discussed in section "The ACO Algorithmic Framework". In addition, they use specific routines to modify the pheromone trails after detecting changes in the problem data or structure. A real-world application of ACO to the scheduling of hospital nurses in Austria is described by Gutjahr and Rauner [63]. More recently, Lissovoi and Witt [77] formally analyze which type of changes in the dynamic shortest path can be tracked by a constant number of ants using a simplified MMAS variant. A recent overview of ACO algorithms for dynamic optimization problems is given by Leguizamón and Alba [73].

Stochastic Problems

In some optimization problems, either the objective function, the decision variables, or the constraints are not deterministic but subject to uncertainty or noise, and they are specified only within certain bounds or in terms of a probability distribution. The first stochastic problem to which ACO algorithms have been applied is the probabilistic TSP (PTSP), where one is given for each city a probability that it requires a visit. The goal in the PTSP is to find a tour of minimal expected length over all the cities. The first ACO algorithm for the PTSP was proposed by Bianchi et al. [11], who were using ACS. This algorithm was later improved upon by Guntsch and Branke [52] and further by Balaprakash et al. [7]. The ACO algorithms developed in that latter paper were then shown to match or even surpass in some cases state-of-the-art performance for the PTSP [8], and extensions thereof are state-of-the-art for the vehicle routing problem with stochastic customers and

demands [9]. Another early ACO proposal for problems under uncertainty is the S-ACO algorithm [59], which has been the basis of later applications, for example, to the selection of optimal screening policies for diabetic retinopathy [18]. The S-ACO algorithm was later extended by Birattari et al. [15] who integrated F-Race for determining the global-best solution in an ACO algorithm for stochastic optimization and have shown positive results for this latter integration. Another notable example is the application of ACO to the VRP with stochastic demands [12]. A survey of various metaheuristics, including ACO, for stochastic combinatorial optimization problems is provided by Bianchi et al. [13].

ACO in Combination with Other Methods

In this section, we briefly overview research on combining ACO with other methods.

ACO and Tree Search Methods

Ants in ACO perform a probabilistic solution construction that can be seen as a stochastic exploration of a search tree. Thus, a natural extension of ACO is to make use of tree search techniques such as branch-and-bound. An example is the approximate nondeterministic tree search (ANTS) algorithm by Maniezzo [84, 85], which uses lower bounds in three different ways. First, it incorporates a lower bound estimate as the heuristic information. Second, it prunes feasible solution components during construction if the estimated solution cost is larger than a threshold (e.g., the cost of the best solution found so far). Third, the order of assignment of solution components during solution construction is influenced by lower bound computations.

A different alternative is to integrate concepts from ACO into tree search methods. An example of this approach is Beam-ACO [16], which incorporates the use of pheromone trails into beam search. Beam search is an incomplete tree search method that keeps a set of partial solutions (the beam) and, at each iteration, selects a number of potential extensions of each of them by adding an additional solution component. These potential extensions are selected based on heuristics [105]. The number of extensions is typically larger than the size of the beam; thus, only the best extensions, according to a lower bound estimate, are kept for the following iteration. Beam-ACO executes beam search several times, replacing the deterministic solution extension of beam search with the probabilistic construction of ACO. In this way, the extension of partial solutions is biased by pheromone trails that are updated with the best solutions found in previous executions of beam search [16]. If lower bound estimates are not reliable or computationally feasible for the problem at hand, an alternative is to perform *stochastic sampling* [78], that is, to complete partial solutions by means of the construction procedure of ACO and use the cost of the complete solution (or the best of several samples) as the cost estimate of the partial solution.

ACO and Exact Methods

The combination of heuristic and exact methods, sometimes called *matheuristics*, is a promising trend in optimization. Roughly speaking there are two types of matheuristics: those that use exact methods to solve simpler versions of a problem in order to improve the solutions generated by a heuristic or provide better estimates of solution quality and those that use heuristic methods to provide initial solutions or to constrain the search space of exact methods in order to make their application feasible to large problems. (The exploitation of tree search techniques such as branch-and-bound or beam search discussed before, can actually also be seen as such a matheuristic approach.)

An example of the first type of matheuristic is the use of constraint programming (CP) techniques [86] to help ACO focus on feasible solutions. This is particularly useful in highly constrained problems, such as scheduling or timetabling, where a major challenge is to find feasible solutions among many infeasible ones. Classical ACO algorithms do not perform satisfactorily on such overly constrained problems. Meyer and Ernst [95] use CP techniques to identify in the ants' construction procedure whether specific solution components lead to infeasible solutions. An example of the second type of matheuristic is the work by Massen et al. [88, 89]. Their pheromone-based column generation algorithm uses an ACO algorithm to generate a heuristic set of feasible routes, from which an optimal subset that is a solution to a vehicle routing problem is selected by an exact solver. The combined algorithm is still a heuristic, because the exact solver does not consider all possible routes, but it allows applying the exact solver to problems with many feasible routes and black-box constraints.

ACO and Surrogate Models

When the evaluation of candidate solutions is very costly in terms of computation time, usually only a small number of candidates can actually be evaluated. Costly solution evaluation arises in the case of simulation-based optimization [6] and, frequently, in industrial settings [44, 129]. Applications with very small evaluation budgets have been rarely studied in the ACO literature, and preliminary results suggest that typical ACO parameter settings are not appropriate in such context [109]. In cases with very small evaluation budget, also the use of surrogate models during optimization may prove useful [68, 100, 134]. Surrogate modeling approaches build prediction models that estimate the quality of candidate solutions, such that only the most promising candidate solutions are actually evaluated. A first approach combining surrogate models and ACO algorithms for tackling combinatorial optimization problems was studied by Pérez Cáceres et al. [109]; further research is needed to generalize the results to other problems.

Parameter Adaptation

Good parameter settings may not only be found "offline," as done in section "ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP", but modified or adapted "online" while the algorithm is running. Good parameter settings may also be predicted according to instance features [102]. In the context of ACO, several parameters have a strong effect on the balance between search space exploration and exploitation of the best found solutions, and their proper values may additionally depend strongly on how much computational time is available to the algorithm. Parameters β and ρ are the earliest and most frequently modified parameters in the literature [90, 93]. A significant effort has been done to define adaptive parameter choices, where the algorithm parameters are modified as a predefined function of the ACO search behavior [72, 111], or to define self-adaptive parameter adaptation schemes, where the algorithm modifies the parameters itself during execution [69, 87, 110]. A recent paper [127] critically reviews the works that have applied parameter adaptation to ACO algorithms. The same paper also shows that the convergence speed of MMAS can be significantly improved, without leading to early stagnation, by means of very simple prescheduled parameter variations, such as starting with a very high value of β and reducing it to a much smaller value after a number of iterations. López-Ibáñez and Stützle [81] demonstrate how such prescheduled variations can be automatically tuned for the purpose of improving the convergence speed of ACO algorithms and show that fast increments in the number of ants, fast decrements of β, and, in particular, slow decrements of q_0 produce remarkable improvements on the TSP. (The software described in section "ACOTSP/ACOQAP: A Unified Framework of ACO Algorithms for the TSP and QAP" and published alongside this chapter is also able to replicate these parameter variation schemes.) Finally, Pellegrini et al. [108] empirically showed that parameter adaptation methods proposed in the literature (excluding prescheduled variations) may often worsen the performance of state-of-the-art ACO algorithms and only improve over static parameter settings when the latter produce very poor results and when only one parameter is carefully adapted.

Conclusions

In this chapter, we have reviewed ACO algorithms from a component-wise perspective, and we have provided an example implementation of ACO algorithms according to this perspective. We have also concisely reviewed trends in ACO research on applications to problems with challenging characteristics such as time-varying problem data, multiple objectives, and stochastic data as well as algorithmic developments that combine ACO algorithms with other techniques. We did not cover in this chapter research on the parallelization of ACO algorithms, which has the goal of either speeding up the execution of a single run of an ACO algorithm or of increasing the quality of the final solution obtained within the same wall-clock

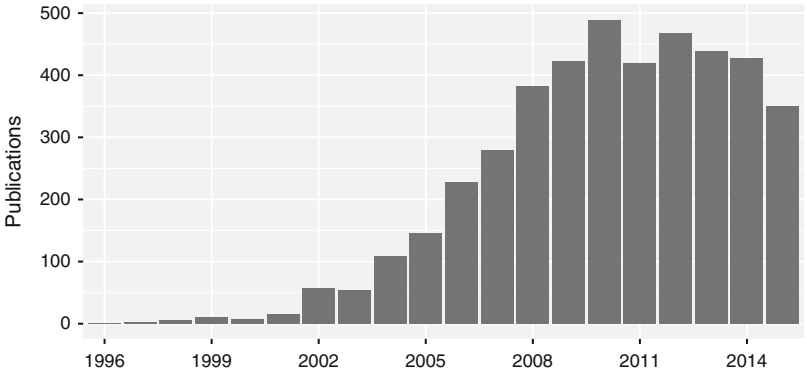

Fig. 4 Number of publications per year in the Scopus database for which their titles contain the keywords "ant system," "ant colony system," or "ant colony optimization"

time (by evaluating more solutions than would be possible without parallelization) [107]. Another important area not covered in this chapter is the ongoing work on the theoretical understanding of ACO algorithms. Early theoretical work has focused on the convergence behavior of ACO algorithms [57, 58, 121]. Later work has analyzed the dynamics of ACO behavior [91] and its relationship to other algorithms [94]. An overview of early theoretical work on ACO is given by Dorigo and Blum [32]. More recent works theoretically analyze the convergence speed of ACO algorithms [28, 60–62, 66, 71, 103, 104].

These developments together with the large number of ACO algorithm variants that have been proposed in the literature and the large number of applications to a very diverse set of problems show that ACO has become a widely accepted and well-established metaheuristic. This fact is confirmed also by the number of publications that have as central topic some aspect related to ACO. Figure 4 shows the number of publications in the Scopus database that have one of the three terms "ant system," "ant colony system," or "ant colony optimization" in the article title. There is a very strong increase from the period 2000–2010, while after 2010 the number of publications remained at a high level. The success of ACO is also witnessed by a number of industrial applications. For example, AntOptima (www.antoptima.com) is a small company using the ACO methodology for tackling industrial problems in distribution and production management, while ArcelorMittal is using ACO algorithms in various areas relevant to steel production to improve operative production performance [25, 44]. ACO is also a central topic of journals and conferences specializing in the area of swarm intelligence such as the ANTS conference series started in 1998 (http://iridia.ulb.ac.be/ants/) and the journal "*Swarm Intelligence*." This success of ACO is due to (i) a truly original algorithmic idea inspired by a natural phenomenon, (ii) a strong versatility of the resulting algorithmic method, and (iii) a focus of ACO research on performance and a pragmatic approach trying to make it a useful technique. In the future, the ideas underlying the ACO metaheuristic promise to be of crucial importance when

tackling challenging problems where aspects of constructive search, distributed information, and dynamic domains match well the inherent characteristics of ACO.

Acknowledgements The research leading to the results presented in this chapter has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC Grant Agreement nº246939. Thomas Stützle and Marco Dorigo acknowledge support of the F.R.S.-FNRS of which they are a senior research associate and a research director, respectively.

Cross-References

- Automatic Tuning of Parameters
- GRASP
- Iterated Greedy
- Iterated Local Search
- Matheuristics
- Multiobjective Optimization

References

1. Acan A (2004) An external memory implementation in ant colony optimization. In: Dorigo M et al (eds) 4th international workshop on Ant colony optimization and swarm intelligence (ANTS 2004). Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 73–84
2. Alaya I, Solnon C, Ghédira K (2007) Ant colony optimization for multi-objective optimization problems. In: 19th IEEE international conference on tools with artificial intelligence (ICTAI 2007), vol 1. IEEE Computer Society Press, Los Alamitos, pp 450–457
3. Alba E, Chicano F (2007) ACOhg: dealing with huge graphs. In: Thierens D et al (eds) Proceedings of the genetic and evolutionary computation conference (GECCO 2007). ACM Press, New York, pp 10–17
4. Angus D (2007) Population-based ant colony optimisation for multi-objective function optimisation. In: Randall M, Abbass HA, Wiles J (eds) Progress in artificial life (ACAL). Lecture notes in computer science, vol 4828. Springer, Heidelberg, pp 232–244
5. Angus D, Woodward C (2009) Multiple objective ant colony optimisation. Swarm Intell 3(1):69–85
6. April J, Glover F, Kelly JP, Laguna M (2003) Simulation-based optimization: practical introduction to simulation optimization. In: Chick SE, Sanchez PJ, Ferrin DM, Morrice DJ (eds) Proceedings of the 35th winter simulation conference: driving innovation, vol 1. ACM Press, New York, pp 71–78
7. Balaprakash P, Birattari M, Stützle T, Yuan Z, Dorigo M (2009) Estimation-based ant colony optimization algorithms for the probabilistic travelling salesman problem. Swarm Intell 3(3):223–242
8. Balaprakash P, Birattari M, Stützle T, Dorigo M (2010) Estimation-based metaheuristics for the probabilistic travelling salesman problem. Comput Oper Res 37(11):1939–1951
9. Balaprakash P, Birattari M, Stützle T, Dorigo M (2015) Estimation-based metaheuristics for the single vehicle routing problem with stochastic demands and customers. Comput Optim Appl 61(2):463–487
10. Barán B, Schaerer M (2003) A multiobjective ant colony system for vehicle routing problem with time windows. In: Proceedings of the twenty-first IASTED international conference on

applied informatics, Insbruck, pp 97–102
11. Bianchi L, Gambardella LM, Dorigo M (2002) An ant colony optimization approach to the probabilistic traveling salesman problem. In: Merelo JJ et al (eds) Parallel problem solving from nature, PPSN VII. Lecture notes in computer science, vol 2439. Springer, Heidelberg, pp 883–892
12. Bianchi L, Birattari M, Manfrin M, Mastrolilli M, Paquete L, Rossi-Doria O, Schiavinotto T (2006) Hybrid metaheuristics for the vehicle routing problem with stochastic demands. J Math Modell Algorithms 5(1):91–110
13. Bianchi L, Dorigo M, Gambardella LM, Gutjahr WJ (2009) A survey on metaheuristics for stochastic combinatorial optimization. Nat Comput 8(2):239–287
14. Bilchev G, Parmee IC (1995) The ant colony metaphor for searching continuous design spaces. In: Fogarty TC (ed) Evolutionary computing, AISB Workshop. Lecture notes in computer science, vol 993. Springer, Heidelberg, pp 25–39
15. Birattari M, Balaprakash P, Dorigo M (2006) The ACO/F-RACE algorithm for combinatorial optimization under uncertainty. In: Doerner KF, Gendreau M, Greistorfer P, Gutjahr WJ, Hartl RF, Reimann M (eds) Metaheuristics – progress in complex systems optimization. Operations research/computer science interfaces series, vol 39. Springer, New York, pp 189–203
16. Blum C (2005) Beam-ACO – hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comput Oper Res 32(6):1565–1591
17. Blum C, Dorigo M (2005) Search bias in ant colony optimization: on the role of competition-balanced systems. IEEE Trans Evol Comput 9(2):159–174
18. Brailsford SC, Gutjahr WJ, Rauner MS, Zeppelzauer W (2006) Combined discrete-event simulation and ant colony optimisation approach for selecting optimal screening policies for diabetic retinopathy. Comput Manag Sci 4(1):59–83
19. Bullnheimer B, Hartl RF, Strauss C (1999) A new rank-based version of the ant system: a computational study. Cent Eur J Oper Res Econ 7(1):25–38
20. Colorni A, Dorigo M, Maniezzo V (1992) Distributed optimization by ant colonies. In: Varela FJ, Bourgine P (eds) Proceedings of the first European conference on artificial life. MIT Press, Cambridge, pp 134–142
21. Cordón O, de Viana IF, Herrera F, Moreno L (2000) A new ACO model integrating evolutionary computation concepts: the best-worst ant system. In: Dorigo M et al (eds) Abstract proceedings of ANTS 2000 – from ant colonies to artificial ants: second international workshop on ant algorithms. IRIDIA, Université Libre de Bruxelles, Belgium, pp 22–29
22. Deneubourg JL, Aron S, Goss S, Pasteels JM (1990) The self-organizing exploratory pattern of the Argentine ant. J Insect Behav 3(2):159–168
23. Di Caro GA, Dorigo M (1998) AntNet: distributed stigmergetic control for communications networks. J Artif Intell Res 9:317–365
24. Di Caro GA, Ducatelle F, Gambardella LM (2005) AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. Eur Trans Telecommun 16(5):443–455
25. Díaz D, Valledor P, Areces P, Rodil J, Suárez M (2014) An ACO algorithm to solve an extended cutting stock problem for scrap minimization in a bar mill. In: Dorigo M et al (eds) Swarm Intelligence, 9th International Conference, ANTS 2014. Lecture notes in computer science, vol 8667. Springer, Heidelberg, pp 13–24
26. Doerner KF, Hartl RF, Reimann M (2003) Are COMPETants more competent for problem solving? The case of a multiple objective transportation problem. Cent Eur J Oper Res Econ 11(2):115–141
27. Doerner KF, Gutjahr WJ, Hartl RF, Strauss C, Stummer C (2004) Pareto ant colony optimization: a metaheuristic approach to multiobjective portfolio selection. Ann Oper Res 131:79–99
28. Doerr B, Neumann F, Sudholt D, Witt C (2011) Runtime analysis of the 1-ANT ant colony optimizer. Theor Comput Sci 412(1):1629–1644
29. Donati AV, Montemanni R, Casagrande N, Rizzoli AE, Gambardella LM (2008) Time dependent vehicle routing problem with a multi ant colony system. Eur J Oper Res 185(3):1174–1191

30. Dorigo M (1992) Optimization, learning and natural algorithms. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy (in Italian)
31. Dorigo M (2007) Ant colony optimization. Scholarpedia 2(3):1461
32. Dorigo M, Blum C (2005) Ant colony optimization theory: a survey. Theor Comput Sci 344(2–3):243–278
33. Dorigo M, Di Caro GA (1999) The ant colony optimization meta-heuristic. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw Hill, London, pp 11–32
34. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66
35. Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge
36. Dorigo M, Maniezzo V, Colorni A (1991) The ant system: an autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy
37. Dorigo M, Maniezzo V, Colorni A (1991) Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy
38. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B 26(1):29–41
39. Dréo J, Siarry P (2004) Continuous interacting ant colony algorithm based on dense heterarchy. Future Gener Comput Syst 20(5):841–856
40. Ehrgott M (2000) Multicriteria optimization. Lecture notes in economics and mathematical systems, vol 491. Springer, Berlin
41. Eyckelhof CJ, Snoek M (2002) Ant systems for a dynamic TSP: ants caught in a traffic jam. In: Dorigo M et al (eds) Ant algorithms. Third international workshop, ANTS 2002. Lecture notes in computer science, vol 2463. Springer, Heidelberg, pp 88–99
42. Feo TA, Resende MGC (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8(2):67–71
43. Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Global Optim 6:109–113
44. Fernández S, Álvarez S, Díaz D, Iglesias M, Ena B (2014) Scheduling a galvanizing line by ant colony optimization. In: Dorigo M et al (eds) Swarm Intelligence. 9th International conference, ANTS 2014. Lecture notes in computer science, vol 8667. Springer, Heidelberg, pp 146–157
45. Gambardella LM, Dorigo M (1996) Solving symmetric and asymmetric TSPs by ant colonies. In: Bäck T, Fukuda T, Michalewicz Z (eds) Proceedings of the 1996 IEEE international conference on evolutionary computation (ICEC'96). IEEE Press, Piscataway, pp 622–627
46. Gambardella LM, Taillard ÉD, Agazzi G (1999) MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw Hill, London, pp 63–76
47. Gambardella LM, Montemanni R, Weyland D (2012) Coupling ant colony systems with strong local searches. Eur J Oper Res 220(3):831–843
48. García-Martínez C, Cordón O, Herrera F (2007) A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. Eur J Oper Res 180(1):116–148
49. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman & Co, San Francisco
50. Glover F (1998) A template for scatter search and path relinking. In: Hao JK, Lutton E, Ronald EMA, Schoenauer M, Snyers D (eds) Artificial evolution. Lecture notes in computer science, vol 1363. Springer, Heidelberg, pp 1–51
51. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Boston
52. Guntsch M, Branke J (2003) New ideas for applying ant colony optimization to the probabilistic tsp. In: Cagnoni S et al (eds) Applications of evolutionary computing. Proceedings of EvoWorkshops 2003. Lecture notes in computer science, vol 2611. Springer, Heidelberg, pp 165–175

53. Guntsch M, Middendorf M (2001) Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: Boers EJW et al (eds) Applications of evolutionary computing. Proceedings of EvoWorkshops 2001. Lecture notes in computer science, vol 2037. Springer, Heidelberg, pp 213–222
54. Guntsch M, Middendorf M (2002) Applying population based ACO to dynamic optimization problems. In: Dorigo M et al (eds) Ant algorithms. Third international workshop, ANTS 2002. Lecture notes in computer science, vol 2463. Springer, Heidelberg, pp 111–122
55. Guntsch M, Middendorf M (2002) A population based approach for ACO. In: Cagnoni S et al (eds) Applications of evolutionary computing. Proceedings of EvoWorkshops 2002. Lecture notes in computer science, vol 2279. Springer, Heidelberg, pp 71–80
56. Guntsch M, Middendorf M (2003) Solving multi-objective permutation problems with population based ACO. In: Fonseca CM, Fleming PJ, Zitzler E, Deb K, Thiele L (eds) Evolutionary multi-criterion optimization, EMO 2003. Lecture notes in computer science, vol 2632. Springer, Heidelberg, pp 464–478
57. Gutjahr WJ (2000) A Graph-based ant system and its convergence. Future Gener Comput Syst 16(8):873–888
58. Gutjahr WJ (2002) ACO algorithms with guaranteed convergence to the optimal solution. Inf Process Lett 82(3):145–153
59. Gutjahr WJ (2004) S-ACO: An ant-based approach to combinatorial optimization under uncertainty. In: Dorigo M et al (eds) 4th international workshop on Ant colony optimization and swarm intelligence (ANTS 2004). Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 238–249
60. Gutjahr WJ (2006) On the finite-time dynamics of ant colony optimization. Method Comput Appl Probab 8(1):105–133
61. Gutjahr WJ (2007) Mathematical runtime analysis of ACO algorithms: survey on an emerging issue. Swarm Intell 1(1):59–79
62. Gutjahr WJ (2008) First steps to the runtime complexity analysis of ant colony optimization. Comput Oper Res 35(9):2711–2727
63. Gutjahr WJ, Rauner MS (2007) An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria. Comput Oper Res 34(3):642–666
64. Hart JP, Shogan AW (1987) Semi-greedy heuristics: an empirical study. Oper Res Lett 6(3):107–114
65. Hoos HH (2012) Programming by optimization. Commun ACM 55(2):70–80
66. Iacopino C, Palmer P (2012) The dynamics of ant colony optimization algorithms applied to binary chains. Swarm Intell 6(4):343–377
67. Iredi S, Merkle D, Middendorf M (2001) Bi-criterion optimization with multi colony ant algorithms. In: Zitzler E, Deb K, Thiele L, Coello Coello CA, Corne D (eds) Evolutionary Multi-criterion Optimization, EMO 2001. Lecture notes in computer science, vol 1993. Springer, Heidelberg, pp 359–372
68. Jones DR, Schonlau M, Welch WJ (1998) Efficient global optimization of expensive black-box functions. J Global Optim 13(4):455–492
69. Khichane M, Albert P, Solnon C (2009) An ACO-based reactive framework for ant colony optimization: first experiments on constraint satisfaction problems. In: Stützle T (ed) Learning and intelligent optimization. Third international conference, LION 3. Lecture notes in computer science, vol 5851. Springer, Heidelberg, pp 119–133
70. Korb O, Stützle T, Exner TE (2007) An ant colony optimization approach to flexible protein–ligand docking. Swarm Intell 1(2):115–134
71. Kötzing T, Neumann F, Röglin H, Witt C (2012) Theoretical analysis of two ACO approaches for the traveling salesman problem. Swarm Intell 6(1):1–21
72. Kovářík O, Skrbek M (2008) Ant colony optimization with castes. In: Kurkova-Pohlova V, Koutnik J (eds) ICANN'08: Proceedings of the 18th international conference on artificial neural networks, Part I. Lecture notes in computer science, vol 5163. Springer, Heidelberg, pp 435–442

73. Leguizamón G, Alba E (2013) Ant colony based algorithms for dynamic optimization problems. In: Alba E, Nakib A, Siarry P (eds) Metaheuristics for dynamic optimization, studies in computational intelligence, vol 433. Springer, Berlin/Heidelberg, pp 189–210
74. Liao T, Montes de Oca MA, Aydın D, Stützle T, Dorigo M (2011) An incremental ant colony algorithm with local search for continuous optimization. In: Krasnogor N, Lanzi PL (eds) Proceedings of the genetic and evolutionary computation conference, GECCO 2011. ACM Press, New York, pp 125–132
75. Liao T, Socha K, Montes de Oca MA, Stützle T, Dorigo M (2014) Ant colony optimization for mixed-variable optimization problems. IEEE Trans Evol Comput 18(4):503–518
76. Liao T, Stützle T, Montes de Oca MA, Dorigo M (2014) A unified ant colony optimization algorithm for continuous optimization. Eur J Oper Res 234(3):597–609
77. Lissovoi A, Witt C (2015) Runtime analysis of ant colony optimization on dynamic shortest path problems. Theor Comput Sci 61(Part A):73–85
78. López-Ibáñez M, Blum C (2010) Beam-ACO for the travelling salesman problem with time windows. Comput Oper Res 37(9):1570–1583
79. López-Ibáñez M, Stützle T (2012) The automatic design of multi-objective ant colony optimization algorithms. IEEE Trans Evol Comput 16(6):861–875
80. López-Ibáñez M, Stützle T (2012) An experimental analysis of design choices of multi-objective ant colony optimization algorithms. Swarm Intell 6(3):207–232
81. López-Ibáñez M, Stützle T (2014) Automatically improving the anytime behaviour of optimisation algorithms. Eur J Oper Res 235(3):569–582
82. López-Ibáñez M, Paquete L, Stützle T (2006) Hybrid population-based algorithms for the bi-objective quadratic assignment problem. J Math Modell Algorithms 5(1):111–137
83. López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Stützle T, Birattari M (2016) The irace package: iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58
84. Maniezzo V (1999) Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS J Comput 11(4):358–369
85. Maniezzo V, Carbonaro A (2000) An ANTS heuristic for the frequency assignment problem. Futur Gener Comput Syst 16(8):927–935
86. Marriott K, Stuckey P (1998) Programming with constraints. MIT Press, Cambridge
87. Martens D, Backer MD, Haesen R, Vanthienen J, Snoeck M, Baesens B (2007) Classification with ant colony optimization. IEEE Trans Evol Comput 11(5):651–665
88. Massen F, Deville Y, van Hentenryck P (2012) Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility. In: Beldiceanu N, Jussien N, Pinson E (eds) Integration of AI and OR techniques in contraint programming for combinatorial optimization problems. Lecture notes in computer science, vol 7298. Springer, Heidelberg, pp 260–274
89. Massen F, López-Ibáñez M, Stützle T, Deville Y (2013) Experimental analysis of pheromone-based heuristic column generation using irace. In: Blesa MJ, Blum C, Festa P, Roli A, Sampels M (eds) Hybrid metaheuristics. Lecture notes in computer science, vol 7919. Springer, Heidelberg, pp 92–106
90. Merkle D, Middendorf M (2001) Prospects for dynamic algorithm control: Lessons from the phase structure of ant scheduling algorithms. In: Heckendorn RB (ed) Proceedings of the 2001 genetic and evolutionary computation conference – workshop program. Workshop "The Next Ten Years of Scheduling Research". Morgan Kaufmann Publishers, San Francisco, pp 121–126
91. Merkle D, Middendorf M (2002) Modeling the dynamics of ant colony optimization. Evol Comput 10(3):235–262
92. Merkle D, Middendorf M (2003) Ant colony optimization with global pheromone evaluation for scheduling a single machine. Appl Intell 18(1):105–111
93. Merkle D, Middendorf M, Schmeck H (2002) Ant colony optimization for resource-constrained project scheduling. IEEE Trans Evol Comput 6(4):333–346
94. Meuleau N, Dorigo M (2002) Ant colony optimization and stochastic gradient descent. Artif Life 8(2):103–121

95. Meyer B, Ernst AT (2004) Integrating ACO and constraint propagation. In: Dorigo M et al (eds) Ant colony optimization and swarm intelligence. 4th international workshop, ANTS 2004. Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 166–177
96. Michel R, Middendorf M (1998) An island model based ant system with lookahead for the shortest supersequence problem. In: Eiben AE, Bäck T, Schoenauer M, Schwefel HP (eds) Parallel problem solving from nature, PPSN V. Lecture notes in computer science, vol 1498. Springer, Heidelberg, pp 692–701
97. Monmarché N, Venturini G, Slimane M (2000) On how *pachycondyla apicalis* ants suggest a new search algorithm. Futur Gener Comput Syst 16(8):937–946
98. Montemanni R, Gambardella LM, Rizzoli AE, Donati AV (2005) Ant colony system for a dynamic vehicle routing problem. J Comb Optim 10:327–343
99. Montgomery J, Randall M, Hendtlass T (2008) Solution bias in ant colony optimisation: lessons for selecting pheromone models. Comput Oper Res 35(9):2728–2749
100. Moraglio A, Kattan A (2011) Geometric generalisation of surrogate model based optimization to combinatorial spaces. In: Merz P, Hao JK (eds) Proceedings of EvoCOP 2011 – 11th European conference on evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol 6622. Springer, Heidelberg, pp 142–154
101. Morin S, Gagné C, Gravel M (2009) Ant colony optimization with a specialized pheromone trail for the car-sequencing problem. Eur J Oper Res 197(3):1185–1191
102. Nallaperuma S, Wagner M, Neumann F (2014) Parameter prediction based on features of evolved instances for ant colony optimization and the traveling salesperson problem. In: Bartz-Beielstein T, Branke J, Filipič B, Smith J (eds) PPSN 2014. Lecture notes in computer science, vol 8672. Springer, Heidelberg, pp 100–109
103. Neumann F, Witt C (2006) Runtime analysis of a simple ant colony optimization algorithm. Electronic Colloquium on Computational Complexity (ECCC) 13(084)
104. Neumann F, Sudholt D, Witt C (2009) Analysis of different MMAS ACO algorithms on unimodal functions and plateaus. Swarm Intell 3(1):35–68
105. Ow PS, Morton TE (1988) Filtered beam search in scheduling. Int J Prod Res 26:297–307
106. Papadimitriou CH, Steiglitz K (1982) Combinatorial optimization – algorithms and complexity. Prentice Hall, Englewood Cliffs
107. Pedemonte M, Nesmachnow S, Cancela H (2011) A survey on parallel ant colony optimization. Appl Soft Comput 11(8):5181–5197
108. Pellegrini P, Birattari M, Stützle T (2012) A critical analysis of parameter adaptation in ant colony optimization. Swarm Intell 6(1):23–48
109. Pérez Cáceres L, López-Ibáñez M, Stützle T (2015) Ant colony optimization on a limited budget of evaluations. Swarm Intell 9(2-3):103–124
110. Randall M (2004) Near parameter free ant colony optimisation. In: Dorigo M et al (eds) 4th international workshop on Ant colony optimization and swarm intelligence (ANTS 2004). Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 374–381
111. Randall M, Montgomery J (2002) Candidate set strategies for ant colony optimisation. In: Dorigo M et al (eds) 3rd international workshop on Ant algorithms (ANTS 2002). Lecture notes in computer science, vol 2463. Springer, Heidelberg, pp 243–249
112. Ruiz R, Stützle T (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 177(3):2033–2049
113. Schilde M, Doerner KF, Hartl RF, Kiechle G (2009) Metaheuristics for the bi-objective orienteering problem. Swarm Intell 3(3):179–201
114. Socha K (2004) ACO for continuous and mixed-variable optimization. In: Dorigo M et al (eds) 4th international workshop on Ant colony optimization and swarm intelligence (ANTS 2004). Lecture notes in computer science, vol 3172. Springer, Heidelberg, pp 25–36
115. Socha K, Dorigo M (2007) Ant colony optimization for mixed-variable optimization problems. Technical Report TR/IRIDIA/2007-019, IRIDIA, Université Libre de Bruxelles
116. Socha K, Dorigo M (2008) Ant colony optimization for continuous domains. Eur J Oper Res 185(3):1155–1173

117. Steuer RE (1986) Multiple criteria optimization: theory, computation and application. Wiley series in probability and mathematical statistics. John Wiley & Sons, New York
118. Stützle T (1998) Local search algorithms for combinatorial problems – analysis, improvements, and new applications. PhD thesis, FB Informatik, TU Darmstadt
119. Stützle T (2002) ACOTSP: a software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem. http://www.aco-metaheuristic.org/aco-code/
120. Stützle T, Dorigo M (1999) ACO algorithms for the quadratic assignment problem. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw Hill, London, pp 33–50
121. Stützle T, Dorigo M (2002) A short convergence proof for a class of ACO algorithms. IEEE Trans Evol Comput 6(4):358–365
122. Stützle T, Hoos HH (1996) Improving the ant system: a detailed report on the MAX–MIN ant system. Technical Report AIDA–96–12, FG Intellektik, FB Informatik, TU Darmstadt
123. Stützle T, Hoos HH (1997) The MAX–MIN ant system and local search for the traveling salesman problem. In: Bäck T, Michalewicz Z, Yao X (eds) Proceedings of the 1997 IEEE international conference on evolutionary computation (ICEC'97). IEEE Press, Piscataway, pp 309–314
124. Stützle T, Hoos HH (1999) MAX–MIN ant system and local search for combinatorial optimization problems. In: Voß S, Martello S, Osman IH, Roucairol C (eds) Meta-heuristics: advances and trends in local search paradigms for optimization. Kluwer Academic, Dordrecht, pp 137–154
125. Stützle T, Hoos HH (2000) MAX–MIN ant system. Future Gener Comput Syst 16(8):889–914
126. Stützle T, López-Ibáñez M, Dorigo M (2011) A concise overview of applications of ant colony optimization. In: Cochran JJ (ed) Wiley encyclopedia of operations research and management science, vol 2. John Wiley & Sons, pp 896–911
127. Stützle T, López-Ibáñez M, Pellegrini P, Maur M, Montes de Oca MA, Birattari M, Dorigo M (2012) Parameter adaptation in ant colony optimization. In: Hamadi Y, Monfroy E, Saubion F (eds) Autonomous search. Springer, Berlin, pp 191–215
128. Taillard ÉD (1991) Robust taboo search for the quadratic assignment problem. Parallel Comput 17(4-5):443–455
129. Teixeira C, Covas J, Stützle T, Gaspar-Cunha A (2012) Multi-objective ant colony optimization for solving the twin-screw extrusion configuration problem. Eng Optim 44(3):351–371
130. Torres CE, Rossi LF, Keffer J, Li K, Shen CC (2010) Modeling, analysis and simulation of ant-based network routing protocols. Swarm Intell 4(3):221–244
131. Tsutsui S (2006) An enhanced aggregation pheromone system for real-parameter optimization in the ACO metaphor. In: Dorigo M et al (eds) 5th international workshop on Ant colony optimization and swarm intelligence (ANTS 2006). Lecture notes in computer science, vol 4150. Springer, Heidelberg, pp 60–71
132. Tsutsui S (2007) Ant colony optimization with cunning ants. Trans Jpn Soc Artifi Intell 22:29–36
133. Wiesemann W, Stützle T (2006) Iterated ants: an experimental study for the quadratic assignment problem. In: Dorigo M et al (eds) 5th international workshop on Ant colony optimization and swarm intelligence (ANTS 2006). Lecture notes in computer science, vol 4150. Springer, Heidelberg, pp 179–190
134. Zaefferer M, Stork J, Friese M, Fischbach A, Naujoks B, Bartz-Beielstein T (2014) Efficient global optimization for combinatorial problems. In: Igel C, Arnold DV (eds) Proceedings of the genetic and evolutionary computation conference, GECCO 2014. ACM Press, New York, pp 871–878