Swarm Intelligence in Optimization

Christian Blum[1] and Xiaodong Li[2]

[1] ALBCOM Research Group
Universitat Politècnica de Catalunya, Barcelona, Spain
cblum@lsi.upc.edu

[2] School of Computer Science and Information Technology
RMIT University, Melbourne, Australia
xiaodong@cs.rmit.edu.au

Summary. Optimization techniques inspired by swarm intelligence have become increasingly popular during the last decade. They are characterized by a decentralized way of working that mimics the behavior of swarms of social insects, flocks of birds, or schools of fish. The advantage of these approaches over traditional techniques is their robustness and flexibility. These properties make swarm intelligence a successful design paradigm for algorithms that deal with increasingly complex problems. In this chapter we focus on two of the most successful examples of optimization techniques inspired by swarm intelligence: ant colony optimization and particle swarm optimization. Ant colony optimization was introduced as a technique for combinatorial optimization in the early 1990s. The inspiring source of ant colony optimization is the foraging behavior of real ant colonies. In addition, particle swarm optimization was introduced for continuous optimization in the mid-1990s, inspired by bird flocking.

1 Introduction

Swarm intelligence (SI), which is an artificial intelligence (AI) discipline, is concerned with the design of intelligent multi-agent systems by taking inspiration from the collective behavior of social insects such as ants, termites, bees, and wasps, as well as from other animal societies such as flocks of birds or schools of fish. Colonies of social insects have fascinated researchers for many years, and the mechanisms that govern their behavior remained unknown for a long time. Even though the single members of these colonies are non-sophisticated individuals, they are able to achieve complex tasks in cooperation. Coordinated colony behavior emerges from relatively simple actions or interactions between the colonies' individual members. Many aspects of the collective activities of social insects are self-organized and work without a central control. For example, leafcutter ants cut pieces from leaves, bring them back to their nest, and grow fungi used as food for their larvae. Weaver

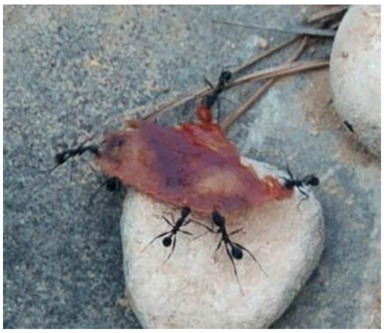


Fig. 1. Ants cooperate for retrieving a heavy prey. (Photographer: Christian Blum)

ant workers build chains with their bodies in order to cross gaps between two leaves. The edges of the two leaves are then pulled together, and successively connected by silk that is emitted by a mature larva held by a worker. Another example concerns the recruitment of other colony members for prey retrieval (see, for example, Fig. 1).

Other examples include the capabilities of termites and wasps to build sophisticated nests, or the ability of bees and ants to orient themselves in their environment. For more examples and a more detailed description see Chap. 1 of this book, as well as [21, 92]. The term swarm intelligence was first used by Beni in the context of cellular robotic systems where simple agents organize themselves through nearest-neighbor interaction [4]. Meanwhile, the term swarm intelligence is used for a much broader research field [21]. Swarm intelligence methods have been very successful in the area of optimization, which is of great importance for industry and science. This chapter aims at giving an introduction to swarm intelligence methods in optimization.

Optimization problems are of high importance both for the industrial world as well as for the scientific world. Examples of practical optimization problems include train scheduling, timetabling, shape optimization, telecommunication network design, and problems from computational biology. The research community has simplified many of these problems in order to obtain scientific test cases such as the well-known traveling salesman problem (TSP) [99]. The TSP models the situation of a traveling salesman who is required to pass through a number of cities. The goal of the traveling salesman is to traverse these cities (visiting each city exactly once) so that the total traveling distance is minimal. Another example is the problem of protein folding, which is one of the most challenging problems in computational biology, molecular biology, biochemistry, and physics. It consists of finding the functional shape or conformation of a protein in two- or three-dimensional space, for example, under simplified lattice models such as the hydrophobic-polar model [169]. The TSP and the protein folding problem under lattice models belong to an important class of optimization problems known as combinatorial optimization (CO).

In general, any optimization problem $\mathcal{P}$ can be described as a triple $(\mathcal{S}, \Omega, f)$, where

1. $\mathcal{S}$ is the search space defined over a finite set of decision variables X_i, $i = 1, \ldots, n$. In the case where these variables have discrete domains we deal with discrete optimization (or combinatorial optimization), and in the case of continuous domains $\mathcal{P}$ is called a continuous optimization problem. Mixed variable problems also exist. Ω is a set of constraints among the variables;
2. $f : \mathcal{S} \rightarrow \mathbb{R}^+$ is the objective function that assigns a positive cost value to each element (or solution) of $\mathcal{S}$.

The goal is to find a solution $s \in \mathcal{S}$ such that $f(s) \leq f(s')$, $\forall\ s' \in \mathcal{S}$ (in case we want to minimize the objective function), or $f(s) \geq f(s')$, $\forall\ s' \in \mathcal{S}$ (in case the objective function must be maximized). In real-life problems the goal is often to optimize several objective functions at the same time. This form of optimization is labelled multiobjective optimization.

Due to the practical importance of optimization problems, many algorithms to tackle them have been developed. In the context of combinatorial optimization (CO), these algorithms can be classified as either *complete* or *approximate* algorithms. Complete algorithms are guaranteed to find for every finite size instance of a CO problem an optimal solution in bounded time (see [133, 128]). Yet, for CO problems that are NP-hard [65], no polynomial time algorithm exists, assuming that $\mathcal{P} \neq \mathcal{NP}$. Therefore, complete methods might need exponential computation time in the worstcase. This often leads to computation times too high for practical purposes. In approximate methods such as SI-based algorithms we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time. Thus, the use of approximate methods has received more and more attention in the last 30 years. This was also the case in continuous optimization, due to other reasons: Approximate methods are usually easier to implement than classical gradient-based techniques. Moreover, generally they do not require gradient information. This is convenient for optimization problems where the objective function is only implicitly given (e.g., when objective function values are obtained by simulation), or where the objective function is not differentiable.

Two of the most notable swarm intelligence techniques for obtaining approximate solutions to optimization problems in a reasonable amount of computation time are ant colony optimization (ACO) and particle swarm optimization (PSO). These optimization methods will be explained in Sects. 2

and 3 respectively. In Sect. 4 we will give some further examples of algorithms for which swarm intelligence was the inspiring source.

2 Ant Colony Optimization

Ant colony optimization (ACO) [52] was one of the first techniques for approximate optimization inspired by swarm intelligence. More specifically, ACO is inspired by the foraging behavior of ant colonies. At the core of this behavior is the indirect communication between the ants by means of chemical pheromone trails, which enables them to find short paths between their nest and food sources. This characteristic of real ant colonies is exploited in ACO algorithms in order to solve, for example, discrete optimization problems.[3]

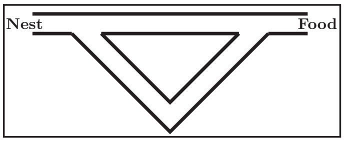
Seen from the operations research (OR) perspective, ACO algorithms belong to the class of metaheuristics [18, 68, 80]. The term *metaheuristic*, first introduced in [67], derives from the composition of two Greek words. *Heuristic* derives from the verb *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$) which means "to find", while the suffix *meta* means "beyond, in an upper level". Before this term was widely adopted, metaheuristics were often called *modern heuristics* [144]. In addition to ACO, other algorithms, such as evolutionary computation, iterated local search, simulated annealing, and tabu search, are often regarded as metaheuristics. For books and surveys on metaheuristics see [144, 68, 18, 80].

This section on ACO is organized as follows. First, in Sect. 2.1 we outline the origins of ACO algorithms. In particular, we present the foraging behavior of real ant colonies and show how this behavior can be transfered into a technical algorithm for discrete optimization. In Sect. 2.2 we provide a description of ACO in more general terms, outline some of the most successful current ACO variants, and list some representative examples of ACO applications. In Sect. 2.3, we shortly describe some recent trends in ACO.

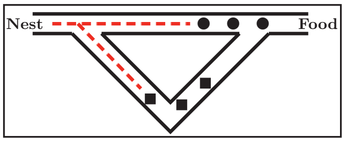
2.1 The Origins of Ant Colony Optimization

Marco Dorigo and colleagues introduced the first ACO algorithms in the early 1990s [46, 50, 51]. The development of these algorithms was inspired by the observation of ant colonies. Ants are social insects. They live in colonies and their behavior is governed by the goal of colony survival rather than being focused on the survival of individuals. The behavior that provided the inspiration for ACO is the ants' foraging behavior, and in particular, how ants

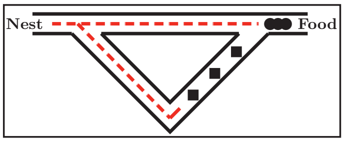
[3] Even though ACO algorithms were originally introduced for the application to discrete optimization problems, the class of ACO algorithms also comprises methods for the application to problems arising in networks, such as routing and load balancing (see, for example, [44]), and continuous optimization problems (see, for example, [159]). In Sect. 2.3 we will shortly deal with ACO algorithms for continuous optimization.



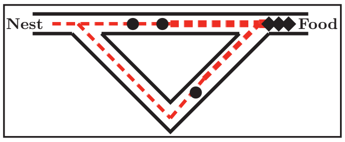
(a) All ants are in the nest. There is no pheromone in the environment



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



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



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

Fig. 2. An experimental setting that demonstrates the shortest path finding capability of ant colonies. Between the ants' nest and the only food source exist two paths of different lengths. In the four graphics, the pheromone trails are shown as dashed lines whose thickness indicates the trails' strength

can find shortest paths between food sources and their nest. When searching for food, ants initially explore the area surrounding their nest in a random manner. While moving, ants leave a chemical pheromone trail on the ground. Ants can smell pheromone. When choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromone that an ant leaves on the ground may depend on the quantity and quality of the food. The pheromone trails will guide other ants to the food source. It has been shown in [42] that the indirect communication between the ants via pheromone trails—known as *stigmergy* [70]—enables them to find shortest paths between their nest and food sources. This is explained in an idealized setting in Fig. 2.

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

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

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

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

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

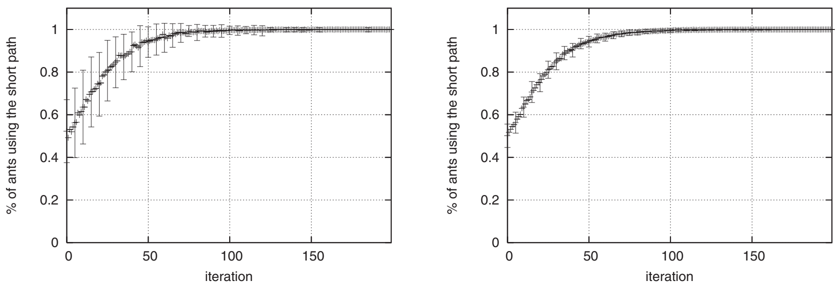
The foraging of an ant colony is in this model iteratively simulated as follows: At each step (or iteration) all the ants are initially placed in node v_s. Then, each ant moves from v_s to v_d as outlined above. As mentioned in the caption of Fig. 2(d), in nature the deposited pheromone is subject to an evaporation over time. We simulate this pheromone evaporation in the artificial model as follows:

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

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

We implemented this system and conducted simulations with the following settings: $l_1 = 1$, $l_2 = 2$, $Q = 1$. The two pheromone values were initialized to 0.5 each. Note that in our artificial system we cannot start with artificial pheromone values of 0. This would lead to a division by 0 in Eq. 1. The results

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



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

Fig. 3. Results of 100 independent runs (error bars show the standard deviation for each 5th iteration). The x-axis shows the iterations, and the y-axis the percentage of the ants using the short path

of our simulations are shown in Fig. 3. They clearly show that over time the artificial colony of ants converges to the short path, i.e., after some time all ants use the short path. In the case of 10 ants (i.e., $n_a = 10$, Fig. 3(a)) the random fluctuations are bigger than in the case of 100 ants (Fig. 3(b)). This indicates that the shortest path finding capability of ant colonies results from a cooperation between the ants.

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

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

Ant System for the TSP: The First ACO Algorithm

The model that we used in the previous section to simulate the foraging behavior of real ants in the setting of Fig. 2 cannot directly be applied to CO problems. This is because we associated pheromone values directly with solutions to the problem (i.e., one parameter for the short path, and one parameter for the long path). This way of modeling implies that the solutions to the considered problem are already known. However, in combinatorial optimization we intend to *find* an unknown optimal solution. Thus, when CO problems are considered, pheromone values are associated with solution components instead. Solution components are the units from which solutions to the tackled problem are assembled. Generally, the set of solution components is expected to be finite and of moderate size. As an example we present the first ACO algorithm, called Ant System (AS) [46, 51], applied to the TSP, which we mentioned in the introduction and which we define in more detail in the following:

Definition 1. *In the TSP is given a completely connected, undirected graph $G = (V, E)$ with edge weights. The nodes V of this graph represent the cities, and the edge weights represent the distances between the cities. The goal is to find a closed path in G that contains each node exactly once (henceforth called a tour) and whose length is minimal. Thus, the search space $\mathcal{S}$ consists of all tours in G. The objective function value $f(s)$ of a tour $s \in \mathcal{S}$ is defined as the sum of the edge weights of the edges that are in s.*

Concerning the AS approach, the edges of the given TSP graph can be considered solution components, i.e., for each $e_{i,j}$ is introduced a pheromone value $\tau_{i,j}$. The task of each ant consists in the construction of a feasible TSP solution, i.e., a feasible tour. In other words, the notion of *task of an ant* changes from *"choosing a path from the nest to the food source"* to *"constructing a feasible solution to the tackled optimization problem"*. Note that with this change of task, the notions of nest and food source lose their meaning.

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

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

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

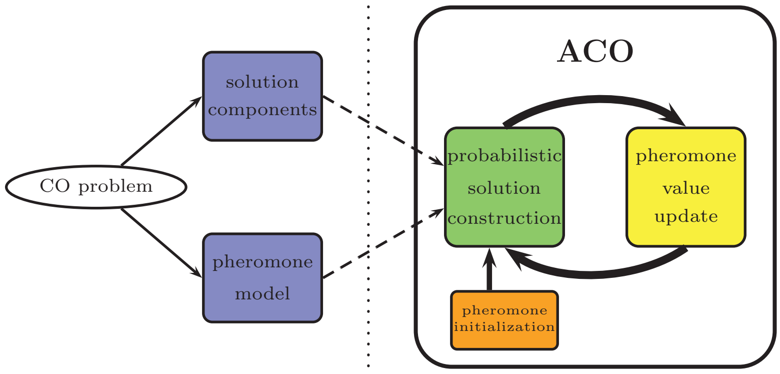


Fig. 4. The ACO framework

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

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

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

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

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

2.2 Ant Colony Optimization: A General Description

The ACO framework, as we know it today, was first defined by Dorigo and colleagues in 1999 [48]. The recent book by Dorigo and Stützle gives a more comprehensive description [52]. The definition of the ACO framework covers most—if not all—existing ACO variants for discrete optimization problems. In the following, we give a general description of this framework.

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

- candidate solutions are constructed using a pheromone model, that is, a parameterized probability distribution over the solution space;
- the candidate solutions are used to modify the pheromone values in a way that is deemed to bias future sampling towards high-quality solutions. The pheromone update aims to concentrate the search in regions of the search space containing high-quality solutions. It implicitly assumes that good solutions consist of good solution components.

In the following we give a more detailed description of solution construction and pheromone update.

Solution Construction

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

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

[5] Note that the description of ACO as given for example in [48] allows pheromone values also to be associated with links between solution components. However, for the purpose of this introduction it is sufficient to assume pheromone values associated with components.

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

where η is an optional weighting function, that is, a function that, sometimes depending on the current sequence, assigns at each construction step a heuristic value $\eta(c_j)$ to each feasible solution component $c_j \in \mathcal{N}(s)$. The values that are given by the weighting function are commonly called the *heuristic information*. Furthermore, the exponents α and β are positive parameters whose values determine the relation between pheromone information and heuristic information. In the previous section's TSP example, we chose not to use any weighting function η, and we set α to 1.

Pheromone Update

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

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

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

Instantiations of this update rule are obtained by different specifications of $\mathcal{S}_{upd}$ and by different weight settings. In most cases, $\mathcal{S}_{upd}$ is composed of some of the solutions generated in the respective iteration (henceforth denoted by $\mathcal{S}_{iter}$) and the best solution found since the start of the algorithm (henceforth denoted by s_{bs}). Solution s_{bs} is often called the best-so-far solution. A well-known example is the *AS-update* rule, that is, the update rule of AS (see also Sect. 2.1). The AS-update rule, which is well known due to the fact that AS was the first ACO algorithm to be proposed in the literature, is obtained from update rule (8) by setting $\mathcal{S}_{upd} \leftarrow \mathcal{S}_{iter}$ and $w_s = 1$, $\forall s \in \mathcal{S}_{upd}$. An example of a pheromone update rule that is more used in practice is the *IB-update* rule (where IB stands for *iteration-best*). The IB-update rule is

Table 1. A selection of ACO variants

ACO variant	Authors	Main reference
Elitist AS (EAS)	Dorigo	[46]
	Dorigo, Maniezzo, and Colorni	[51]
Rank-based AS (RAS)	Bullnheimer, Hartl, and Strauss	[26]
$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS)	Stützle and Hoos	[164]
Ant Colony System (ACS)	Dorigo and Gambardella	[49]
Hyper-Cube Framework (HCF)	Blum and Dorigo	[16]

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

Well-Performing ACO Variants

Even though the original AS algorithm achieved encouraging results for the TSP problem, it was found to be inferior to state-of-the-art algorithms for the TSP as well as for other CO problems. Therefore, several extensions and improvements of the original AS algorithm were introduced over the years. An overview is provided in Table 1. These ACO variants mostly differ in the pheromone update rule that is applied.

In addition to these ACO variants, the ACO community has developed additional algorithmic features for improving the search process performed by ACO algorithms. A prominent example is the so-called candidate list strategy, which is a mechanism to restrict the number of available choices at each solution construction step. Usually, this restriction applies to a number of the best choices with respect to their transition probabilities (see Eq. 7). For example, in the case of the application of ACS (see Table 1) to the TSP, the restriction to the closest cities at each construction step both improved the final solution quality and led to a significant speedup of the algorithm (see [61]). The reasons for this are as follows: First, in order to construct high-quality solutions it is often enough to consider only the "promising" choices at each construction step. Second, to consider fewer choices at each construction step speeds up the solution construction process, because the reduced number of choices reduces the computation time needed to make a choice.

Applications of ACO Algorithms

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

The bioinformatics and biomedical fields in particular show an increasing interest in ACO. Recent applications of ACO to problems arising in these areas include the applications to protein folding [153, 154], to multiple sequence alignment [127], to DNA sequencing by hybridization [20], and to the prediction of major histocompatibility complex (MHC) class II binders [86]. ACO algorithms are currently among the state-of-the-art methods for solving, for example, the sequential ordering problem [62], the resource constraint project scheduling problem [120], the open shop scheduling problem [14], assembly line balancing [15], and the 2D and 3D hydrophobic polar protein folding problem [154]. In Table 2 we provide a list of representative ACO applications. For a more comprehensive overview that also covers the application of ant-based algorithms to routing in telecommunication networks we refer the interested reader to [52].

2.3 Recent Trends

Theoretical Work on ACO

The first theoretical works on ACO algorithms appeared in 2002. They deal with the question of algorithm convergence [75, 76, 163]. In other words: will a given ACO algorithm find an optimal solution when given enough resources? This is an interesting question, because ACO algorithms are stochastic search procedures in which the pheromone update could prevent them from ever reaching an optimum.

Recently, researchers have been dealing with the relation of ACO algorithms to other methods for learning and optimization. The work presented in [7] relates ACO to the fields of optimal control and reinforcement learning, whereas [183] describes the common aspects of ACO algorithms and probabilistic learning algorithms such as stochastic gradient ascent (SGA) and the

Table 2. A representative selection of ACO applications

Problem	Authors	Reference
Traveling salesman problem	Dorigo, Maniezzo, and Colorni	[46, 50, 51]
	Dorigo and Gambardella	[49]
	Stützle and Hoos	[164]
Quadratic assignment problem	Maniezzo	[109]
	Maniezzo and Colorni	[111]
	Stützle and Hoos	[164]
Scheduling problems	Stützle	[162]
	den Besten, Stützle, and Dorigo	[41]
	Gagné, Price, and Gravel	[59]
	Merkle, Middendorf, and Schenk	[120]
	Blum (resp., Blum and Sampels)	[14, 19]
Vehicle routing problems	Gambardella, Taillard, and Agazzi	[63]
	Reimann, Doerner, and Hartl	[145]
Timetabling	Socha, Sampels, and Manfrin	[160]
Set packing	Gandibleux, Delorme, and T'Kindt	[64]
Graph coloring	Costa and Hertz	[38]
Shortest supersequence problem	Michel and Middendorf	[123]
Sequential ordering	Gambardella and Dorigo	[62]
Constraint satisfaction problems	Solnon	[161]
Data mining	Parpinelli, Lopes, and Freitas	[134]
Maximum clique problem	Bui and Rizzo Jr	[25]
Edge-disjoint paths problem	Blesa and Blum	[13]
Cell placement in circuit design	Alupoaei and Katkoori	[2]
Communication network design	Maniezzo, Boschetti, and Jelasity	[110]
Bioinformatics problems	Shmygelska, Aguirre-Hernández, and Hoos	[153]
	Moss and Johnson	[127]
	Karpenko, Shi, and Dai	[86]
	Shmygelska and Hoos	[154]
	Korb, Stützle, and Exner	[93]
	Blum and Yábar Vallès	[20]
Industrial problems	Bautista and Pereira	[3]
	Blum, Bautista, and Pereira	[15]
	Silva, Runkler, Sousa, and Palm	[156]
	Gottlieb, Puchta, and Solnon	[69]
	Corry and Kozan	[37]
Continuous optimization	Bilchev and Parmee	[6]
	Monmarché, Venturini, and Slimane	[125]
	Dréo and Siarry	[54]
	Socha and Dorigo	[159]
	Socha and Blum	[158]
Multiobjective problems	Guntsch and Middendorf	[74]
	Lopéz-Ibáñez, Paquete, and Stützle	[106]
	Doerner, Gutjahr, Hartl, Strauss, and Stummer	[45]
Dynamic (or stochastic) problems	Guntsch and Middendorf	[73]
	Bianchi, Gambardella, and Dorigo	[5]
Music	Guéret, Monmarché, and Slimane	[72]

cross-entropy (CE) method. Meuleau and Dorigo have shown in [121] that ACO's pheromone update is very similar to stochastic gradient ascent in the space of pheromone values.

While convergence proofs can provide insight into the working of an algorithm, they are usually not very useful to the practitioner who wants to implement efficient algorithms. More relevant for practical applications might be the research efforts that were aimed at a better understanding of the behavior of ACO algorithms. Representative works are the ones on negative search bias [17] and the study of models of ACO algorithms [117, 118]. For a recent survey on theoretical work on ACO see [47].

Applying ACO to Continuous Optimization Problems

Many practical optimization problems can be formulated as continuous optimization problems, that is, problems in which the decision variables have continuous domains. While ACO algorithms were originally introduced to solve discrete problems, their adaptation to solve continuous optimization problems enjoys increasing attention. Early applications of ant-based algorithms to continuous optimization include algorithms such as Continuous ACO (CACO) [6], API [125], and Continuous Interacting Ant Colony (CIAC) [54]. However, all these approaches are conceptually quite different from ACO for discrete problems. The latest approach called $\mathrm{ACO}_{\mathbb{R}}$, which was proposed by Socha in [157, 159], is closest to the spirit of ACO for discrete problems. While ACO algorithms for discrete optimization problems construct solutions by sampling at each construction step a discrete probability distribution that is derived from the pheromone information, $\mathrm{ACO}_{\mathbb{R}}$ utilizes a continuous probability density function (PDF) for generating solutions. This density function is produced, for each solution construction, from an archive of solutions that the algorithm keeps and updates at all times. The archive update corresponds to the pheromone update in ACO algorithms for discrete optimization problems. Recently, $\mathrm{ACO}_{\mathbb{R}}$ was applied to neural network training [158].

Hybridizing ACO with Branch & Bound Derivatives

Beam search (BS) is a classical tree search method that was introduced in the context of scheduling [131], but has since then been successfully applied to many other CO problems (e.g., see [40]). BS algorithms are incomplete derivatives of branch & bound algorithms, and are therefore approximate methods. The central idea behind BS is to construct a number of k_{bw} (the so-called beam width) solutions in parallel and non-independently. At each construction step the algorithm selects at most k_{bw} partial solutions by utilizing bounding information. Even though both ACO and BS have the common feature that they are based on the idea of constructing candidate solutions step-by-step, the ways by which the two methods explore the search space are quite different. While BS is a deterministic algorithm that uses a lower bound for guiding the search process, ACO algorithms are adaptive and probabilistic procedures. Furthermore, BS algorithms reduce the search space in the hope of not excluding all optimal solutions, while ACO algorithms consider the whole search space. Based on these observations Blum introduced a hybrid between ACO and BS which was labelled *Beam-ACO* [14, 15]. Beam-ACO is an ACO algorithm in which the standard ACO solution construction mechanism is replaced by a probabilistic beam search procedure. Work that is in a similar vein can be found in [109, 112].

ACO and Constraint Programming

Another interesting hybridization example concerns the use of constraint programming (CP) techniques (see [114]) for restricting the search performed by an ACO algorithm to promising regions of the search space. The motivation for this type of hybridization is as follows: Generally, ACO algorithms are competitive with other optimization techniques when applied to problems that are not overly constrained. However, when highly constrained problems such as scheduling or timetabling are concerned, the performance of ACO algorithms generally degrades. Note that this is also the case for other metaheuristics. The reason is to be found in the structure of the search space: When a problem is not overly constrained, it is usually not difficult to find feasible solutions. The difficulty rather lies in the optimization part, namely the search for good feasible solutions. On the other hand, when a problem is highly constrained the difficulty is rather in finding any feasible solution. This is where CP comes into play, because these problems are the target problems for CP applications. The idea of hybridizing ACO with CP is simple [122]. At each iteration, first constraint propagation is applied in order to reduce the remaining search tree. Then, solutions are constructed in the standard ACO way with respect to the reduced search tree. After the pheromone update, additional constraints might be added to the system.

Applying ACO in a Multilevel Framework

Multilevel techniques have been employed for quite a long time, especially in the area of multigrid methods (see [23] for an overview). More recently, they have been brought into focus by Walshaw for the application to CO. Walshaw and coworkers applied multilevel techniques to graph-based problems such as mesh partitioning [177]. The basic idea of a multilevel scheme is simple. Starting from the original problem instance, smaller and smaller problem instances are obtained by successive coarsening until some stopping criteria are satisfied. This creates a hierarchy of problem instances in which the problem instance of a given level is always smaller (or of equal size) to the problem instance of the next lower level. Then, a solution is computed to the smallest problem instance and successively transformed into a solution of the next higher level until a solution for the original problem instance is obtained. At each level, the obtained solution might be subject to a refinement process, for example, an ACO algorithm. Applications of ACO in multilevel frameworks include [94, 95, 20].

3 Particle Swarm Optimization

Particle swarm optimization (PSO) is a population-based stochastic optimization technique modelled on the social behaviors observed in animals or insects, e.g., bird flocking, fish schooling, and animal herding [92]. It was originally proposed by James Kennedy and Russell Eberhart in 1995 [91]. Since its inception, PSO has gained increasing popularity among researchers and practitioners as a robust and efficient technique for solving difficult optimization problems. In PSO, individual *particles* of a swarm represent potential solutions, which move through the problem search space seeking an optimal, or good enough, solution. The particles broadcast their current positions to neighboring particles. The position of each particle is adjusted according to its *velocity* (i.e., rate of change) and the difference between its current position, respectively the best position found by its neighbors, and the best position it has found so far. As the model is iterated, the swarm focuses more and more on an area of the search space containing high-quality solutions.

PSO has close ties to *artificial life* models. Early works by Reynolds on a flocking model *Boids* [146], and Heppner's studies on rules governing large numbers of birds flocking synchronously [78], indicated that the emergent group dynamics such as the bird flocking behavior are based on local interactions. These studies were the foundation for the subsequent development of PSO for the application to optimization. PSO is in some way similar to *cellular automata* (CA), which are often used for generating interesting self-replicating patterns based on very simple rules, e.g., John Conway's *Game of Life*. CAs have three main attributes: (1) individual cells are updated in parallel; (2) the value of each new cell depends only on the old values of the cell and its neighbors; and (3) all cells are updated using the same rules [149]. Particles in a swarm are analogous to CA cells, whose states are updated in many dimensions simultaneously.

The term *particle swarm* was coined by James Kennedy and Russell Eberhart, who were responsible for inventing the original PSO. Initially they intended to model the movements of flocks of birds and schools of fish. As their model further evolved to handle optimization, the visual plots they used started to display something more like *swarms* of mosquitoes. The term *particle* was used simply because the notion of velocity was adopted in PSO and *particle* seemed to be the most appropriate term in this context.

This section on PSO is organized as follows. In Sect. 3.1 we first present the original PSO developed by Kennedy and Eberhart. This is followed by descriptions of a number of key improvements and generalizations to the basic PSO algorithm. We then give an overview of several PSO variants that represent important progress made in this area, and a list of representative examples of PSO applications. In Sect. 3.2 we outline some recent trends in PSO research, including its theoretical works and its application in the areas of multiobjective optimization, dynamic optimization, and constraint handling.

3.1 Particle Swarm Optimization: An Introduction

In PSO, the velocity of each particle is modified iteratively by its *personal best* position (i.e., the best position found by the particle so far), and the best position found by particles in its neighborhood. As a result, each particle searches around a region defined by its personal best position and the best position from its neighborhood. Henceforth we use $\mathbf{v}_i$ to denote the velocity of the ith particle in the swarm, $\mathbf{x}_i$ to denote its position, $\mathbf{p}_i$ to denote the *personal best* position and $\mathbf{p}_g$ the best position found by particles in its neighborhood. In the original PSO algorithm, $\mathbf{v}_i$ and $\mathbf{x}_i$, for $i = 1, \ldots, n$, are updated according to the following two equations [91]:

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \boldsymbol{\varphi}_1 \otimes (\mathbf{p}_i - \mathbf{x}_i) + \boldsymbol{\varphi}_2 \otimes (\mathbf{p}_g - \mathbf{x}_i), \tag{9}$$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i, \tag{10}$$

where $\boldsymbol{\varphi}_1 = c_1\mathbf{R_1}$ and $\boldsymbol{\varphi}_2 = c_2\mathbf{R_2}$. $\mathbf{R}_1$ and $\mathbf{R}_2$ are two separate functions each returning a vector comprising random values uniformly generated in the range [0,1]. c_1 and c_2 are acceleration coefficients. The symbol $\otimes$ denotes point-wise vector multiplication. Equation (9) shows that the velocity term $\mathbf{v}_i$ of a particle is determined by three parts, the "momentum", the "cognitive", and the "social" part. The "momentum" term $\mathbf{v}_i$ represents the previous velocity term which is used to carry the particle in the direction it has travelled so far; the "cognitive" part, $\boldsymbol{\varphi}_1 \otimes (\mathbf{p}_i - \mathbf{x}_i)$, represents the tendency of the particle to return to the best position it has visited so far; the "social" part, $\boldsymbol{\varphi}_2 \otimes (\mathbf{p}_g - \mathbf{x}_i)$, represents the tendency of the particle to be attracted towards the position of the best position found by the entire swarm.

Position $\mathbf{p}_g$ in the "social" part is the best position found by particles in the neighborhood of the ith particle. Different neighborhood topologies can be used to control information propagation between particles. Examples of neighborhood topologies include ring, star, and von Neumann. Constricted information propagation as a result of using small neighborhood topologies such as von Neumann has been shown to perform better on complex problems, whereas larger neighborhoods generally perform better on simpler problems [116]. Generally speaking, a PSO implementation that chooses $\mathbf{p}_g$ from within a restricted local neighborhood is referred to as *lbest* PSO, whereas choosing $\mathbf{p}_g$ without any restriction (hence from the entire swarm) results in a *gbest* PSO. Algorithm 1 summarizes a basic PSO algorithm.

Figure 3.1 shows each component of the velocity term $\mathbf{v}_i$ in vector form, and the resulting position, $\mathbf{x}_i$ (updated), for the ith particle. Note that the inertia coefficient w is used to scale the previous velocity term, normally to reduce the "momentum" of the particle. More discussion on w will be provided in the next section.

Earlier studies showed that the velocity as defined in Eq. (9) has a tendency to explode to a large value, resulting in particles exceeding the boundaries of the search space. This is more likely to happen especially when a particle is far from $\mathbf{p}_g$ or $\mathbf{p}_i$. To overcome this problem, a velocity clamping method can be adopted where the maximum allowed velocity value is set to V_{max} in each dimension of $\mathbf{v}_i$. This method does not necessarily prevent particles

Algorithm 1 The PSO algorithm, assuming maximization

```
Randomly generate an initial swarm
repeat
  for each particle $i$ do
    if $f(\mathbf{x}_i) > f(\mathbf{p}_i)$ then $\mathbf{p}_i \leftarrow \mathbf{x}_i$
    $\mathbf{p}_g = \max(\mathbf{p}_{neighbours})$
    Update velocity (see Eq. (9))
    Update position (see Eq. (10))
  end for
until termination criterion is met
```

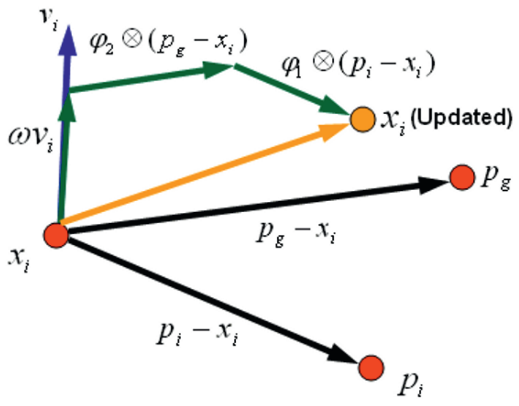


Fig. 5. Visualizing PSO components as vectors

from leaving the search space nor from converging. However, it does limit the particle step size, thereby preventing further divergence of particles.

Inertia Weight

Observe that the positions $\mathbf{p}_i$ and $\mathbf{p}_g$ in Eq. (9) can be collapsed into a single term $\mathbf{p}$ without losing any information:

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \varphi \otimes (\mathbf{p} - \mathbf{x}_i), \tag{11}$$
$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i, \tag{12}$$

where $\mathbf{p} = \frac{\varphi_1 \mathbf{p}_i + \varphi_2 \mathbf{p}_g}{\varphi_1 + \varphi_2}$, and $\varphi = \varphi_1 + \varphi_2$. Note that $\mathbf{p}$ represents the weighted average of the $\mathbf{p}_i$ and $\mathbf{p}_g$. It can be seen that the previous velocity term in Eq. (11) tends to keep the particle moving in the current direction. A coefficient *inertia weight*, w, can be used to control this influence on the new velocity. The velocity update (see Eq. (9)) can be now revised as:

$$\mathbf{v}_i \leftarrow w\mathbf{v}_i + \boldsymbol{\varphi}_1 \otimes (\mathbf{p}_i - \mathbf{x}_i) + \boldsymbol{\varphi}_2 \otimes (\mathbf{p}_g - \mathbf{x}_i) \tag{13}$$

The inertia-weighted PSO can converge under certain conditions even without using V_{max} [33]. For $w > 1$, velocities increase over time, causing particles to diverge eventually beyond the boundaries of the search space. For $w < 0$, velocities decrease over time, eventually reaching 0, resulting in convergence behavior. Eberhart and Shi suggested the use of a time-varying inertia weight, gradually decreasing its value typically from 0.9 to 0.4 (with $\boldsymbol{\varphi} = 4.0$) [55].

Clerc described a general PSO algorithm that uses a *constriction coefficient*. Among the models suggested, the Constriction Type 1 PSO is equivalent to the inertia-weighted PSO [33]. The velocity update in Eq. (13) can be rewritten as:

$$\mathbf{v}_i \leftarrow \chi(\mathbf{v}_i + \boldsymbol{\varphi}_1 \otimes (\mathbf{p}_i - \mathbf{x}_i) + \boldsymbol{\varphi}_2 \otimes (\mathbf{p}_g - \mathbf{x}_i)), \tag{14}$$

where $\chi = \frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|}$, and $\varphi = c_1 + c_2, \varphi > 4$. If φ is set to 4.1, and $c_1 = c_2 = 2.05$, then the constriction coefficient χ will be 0.7298. Applying χ in Eq. (14) results in the previous velocity scaled by 0.7298, and the "cognitive" and "social" parts multiplied by 1.496 (i.e., 0.7298 times 2.05). Both theoretical and empirical results suggested that the above configuration using a constant constriction coefficient $\chi = 0.7298$ ensures convergent behavior [55] without using V_{max}. However, early empirical studies by Eberhart and Shi suggested that it may be still a good idea to use velocity clamping together with the constriction coefficient, which showed improved performance on certain problems.

Fully Informed Particle Swarm

Equation (11) indicates that a particle tends to converge towards a point determined by $\mathbf{p} = \frac{\boldsymbol{\varphi}_1\mathbf{p}_i + \boldsymbol{\varphi}_2\mathbf{p}_g}{\boldsymbol{\varphi}_1+\boldsymbol{\varphi}_2}$, where $\boldsymbol{\varphi} = \boldsymbol{\varphi}_1 + \boldsymbol{\varphi}_2$. In the fully informed particle swarm (FIPS) as proposed by Mendes [116], $\mathbf{p}$ can be further generalized to any number of terms:

$$\mathbf{p} = \frac{\sum_{k\in\mathcal{N}} \mathbf{r}[0, \frac{c_{max}}{|\mathcal{N}|}] \otimes \mathbf{p}_k}{\sum_{k\in\mathcal{N}} \boldsymbol{\varphi}_k}, \tag{15}$$

where $\mathbf{p}_k$ denotes the best previous position found by the kth particle in $\mathcal{N}$, which is a set of neighbors including the current particle itself. Note again that the division is a point wise operator here. If we set $k = 2$, $\mathbf{p}_1 = \mathbf{p}_i$, and $\mathbf{p}_2 = \mathbf{p}_g$, with both $\mathbf{p}_i, \mathbf{p}_g \in \mathcal{N}$, then the Constriction Type 1 PSO is just a special case of the more general PSO defined in Eq. (11). A significant implication of Eq. (15) is that it allows us to think more freely about employing terms of influence other than just $\mathbf{p}_i$ and $\mathbf{p}_g$ [116] [90].

PSO Variants

Although the canonical PSO was designed for continuous optimization, it can be extended to operate on binary search spaces. Kennedy and Eberhart developed a simple binary PSO by altering the velocity term in the canonical PSO into a probability threshold to determine if $\mathbf{x_i}$ is 0 or 1 [92]. PSO can be also extended to solve discrete or mixed (continuous and discrete) optimization problems [180, 32]. PSO can be adapted to work with discrete variables by simply discretizing the values after using them in the velocity and position update equations. Clerc provided several examples of PSO applied to combinatorial problems such as the knapsack, the traveling salesman, and the quadratic assignment problems [32].

An adaptive PSO version, *tribes*, was developed by Clerc [32], where the swarm size is determined by strategies for generating new particles as well as for removing poorly performing particles. The concept of a tribe is used to group particles that inform each other. Clerc's goal was to develop a PSO which can find the parameters on its own (e.g., swarm size), and still maintain a relatively good performance.

Kennedy proposed a PSO variant, bare-bones PSO, which does not use the velocity term [89]. In the bare-bones PSO each dimension of the new position of a particle is randomly selected from a Gaussian distribution with the mean being the average of $\mathbf{p_i}$ and $\mathbf{p_g}$ and the standard deviation being the distance between $\mathbf{p_i}$ and $\mathbf{p_g}$:

$$\mathbf{x_i} \leftarrow \mathcal{N}\left(\frac{\mathbf{p_i}+\mathbf{p_g}}{2}, ||\mathbf{p_i}-\mathbf{p_g}||\right) \tag{16}$$

Note that there is no velocity term used in Eq. (16). The new particle position is simply generated via the Gaussian distribution. Sampling distributions other than Gaussian can also be employed [32, 147].

It has been observed that the canonical PSO tends to prematurely converge to local optima. To combat this problem, several PSO variants have incorporated a diversity maintenance mechanism. For example, ARPSO (attractive and repulsive PSO) was proposed to use a diversity measure to trigger an alternation between phases of attraction and repulsion [148]. A dissipative PSO was described in [179] to increase randomness. Similarly, a PSO with self-organized criticality was introduced in [107]. A PSO variant based on fitness-distance-ratio (FDR-PSO) was proposed in [173], to encourage interactions among particles that are both fit and close to each other. FDR-PSO was shown to give superior performance to the canonical PSO. FDR-PSO can be seen as using a dynamically defined neighborhood topology. Various neighborhood topologies have been adopted to restrict particle interactions [165, 116]. In particular, the von Neumann neighborhood topology has been shown to provide good performance across a range of test functions [116]. In [83], an H-PSO (Hierarchical PSO) was proposed, where a hierarchical tree structure is adopted to restrict the interactions among particles. Each particle is influenced only by its own personal best position and by the best position of the particle that is directly above it in the hierarchy. Another recently proposed PSO, CLPSO [105], which is in some way similar to FDR-PSO, allows incorporation of learning from more previous personal best positions. Gaussian distribution was employed in a PSO variant as a mutation operator in [79]. A cooperative PSO, similar to those previously developed coevolutionary algorithms, was also proposed in [171].

PSO variants have also been developed for solving multimodal optimization problems, where multiple equally good optima are sought. Niching methods such as crowding and fitness sharing that have been developed for evolutionary algorithms can be easily incorporated into PSO algorithms. Some representative PSO niching variants include NichePSO [142], SPSO (Speciation-based PSO) [101, 135, 8], and a PSO algorithm using a stretching function [138].

Applications of PSO Algorithms

PSO algorithms have been applied to optimization problems ranging from classical problems such as scheduling, the traveling salesman problem, neural network training, and task assignment, to highly specialized applications such as reactive power and voltage control [180], biomedical image registration [176], and even music composition [10]. In recent years, PSO is also a popular choice of many researchers for handling multiobjective optimization [155] and dynamic optimization problems [102].

One of the earliest applications of PSO was the evolution of neural network structures. Eberhart et al. used PSO to replace the traditional back-propagation learning algorithm in a multilayer perceptron [57]. Because of its fast convergence behavior, using PSO for neural network training can sometimes save a considerable amount of computation time compared with other optimization methods.

Table 3 shows a list of examples of PSO applications that can be found in the literature. For more information on PSO applications we refer the interested reader to [32].

3.2 Recent Trends

Theoretical Work on PSO

Since PSO was first introduced by Kennedy and Eberhart in 1995 [91], several studies have been carried out on understanding the convergence properties of PSO [33, 132, 170, 168]. Since an analysis of the convergence behavior of a swarm of multiple interactive particles is difficult, many of these works focus on studying the convergence behaviors of a simplified PSO system.

Kennedy provided the first analysis of a simplified particle behavior in [88], where particle trajectories for a range of variable choices were given. In [132],

Table 3. A representative selection of PSO applications

Problem	Authors	Reference
Traveling salesman problem	Onwubolu and Clerc	[130]
Flowshop scheduling	Rameshkumar, Suresh and Mohanasundaram	[143]
Task assignment	Salman, Imtiaz and Al-Madani	[150]
Neural networks	Kennedy, Eberhart, and Shi	[92]
	Mendes, Cortez, Rocha, and Neves	[115]
	Conradie, Miikkulaninen and Aldrich	[35]
	Gudisz and Venayagamoorthy	[71]
	Settles, Rodebaugh and Soule	[152]
Bioinformatics	Correa, Freitas and Johnson	[36]
	Georgiou, Pavlidis, Parsopoulos and Vrahatis	[66]
Industrial applications	Katare, Kalos and West	[87]
	Marinke, Matiko, Araujo and Coelho	[113]
Reactive power and voltage control	Yoshida, Kawata, et. al	[180]
PID controller	Gaing	[60]
Biomedical image registration	Wachowiak et. al	[176]
Floor planning	Sun, Hsieh, Wang and Lin	[166]
Quantizer design	Zha and Venayagamoorthy	[182]
Power systems	Venayagamoorthy	[174]
Clustering analysis	Chen and Ye	[30]
Constraint handling	Pulido and Coello	[140]
	Liang and Suganthan	[104]
Electromagnetic applications	Mikki and Kishk	[124]
Multiobjective problems	Moore and Chapman	[126]
	Coello and Lechuga	[34]
	Fieldsend and Singh	[58]
	Hu and Eberhart	[81]
	Parsopoulos and Vrahatis	[137]
	Li	[100]
Dynamic problems	Carlisle and Dozier	[28]
	Hu and Eberhart	[82]
	Eberhart and Shi	[56]
	Carlisle and Dozier	[29]
	Blackwell and Branke	[11, 12]
	Jason and Middendorf	[84]
	Parrott and Li	[135]
	Li, Blackwell, and Branke	[102]
Music	Blackwell and Bentley	[10]

Ozcan and Mohan showed that a particle in a one-dimensional PSO system, with its $\mathbf{p_i}$, $\mathbf{p_g}$, φ_1, and φ_2 kept constant, follows the path of a sinusoidal wave, where the amplitude and frequency of the wave are randomly decided.

A formal theoretical analysis of the convergence properties of a simplified PSO was provided by Clerc [33]. Clerc [33] represented the PSO system as defined in equations (11) and (12) as a dynamic system in state-space form. By simplifying the PSO to a deterministic dynamic system, its convergence can be shown based on the eigenvalues of the state matrix. A similar work was also carried out by Bergh and Engelbrecht [170], where regions of parameter space that guarantee convergence are identified. The conditions for convergence derived from both studies [33, 170] are: $w < 1$ and $w > \frac{1}{2}(c_1 + c_2) - 1$.

In a more recent work [172], Bergh and Engelbrecht generalized the above analysis by including the inertia weight w, and also provided a formal convergence proof of particles in this representation. Furthermore, they studied the particle trajectory with a relaxed assumption to allow stochastic values for φ_1 and φ_2. They demonstrated that a particle can exhibit a combination of divergent and convergent behaviors with certain probabilities when different values of φ_1 and φ_2 are used.

In [85], Kadirkamanathan et al. recently provided a new approach to the convergence analysis of PSO without the assumption of non-random PSO. The analysis of stochastic particle dynamics was made feasible by representing particle dynamics as a nonlinear feedback controlled system as formulated by Lure [53]. The convergence analysis was carried out using the concept of passive systems and Lyapunov stability [175]. Some conservative conditions for convergence were derived in this study.

PSO for Multiobjective Optimization

Multiobjective optimization (MO) problems represent an important class of real-world problems. Typically such problems involve trade-offs. For example, a car manufacturer may wish to maximize its profit, but meanwhile also to minimize its production cost. These objectives are typically conflicting to each other. For example, a higher profit could increase the production cost. Generally, there is no single optimal solution. Often the manufacturer needs to consider many possible "trade-off" solutions before choosing the one that suits its need. The curve or surface (for more than two objectives) describing the optimal trade-off solutions between objectives is known as the *Pareto front*. A multiobjective optimization algorithm is required to find solutions as close as possible to the Pareto front, while maintaining a good solution diversity along the Pareto front.

To apply PSO to multiobjective optimization problems, several issues have to be taken into consideration:

1. How to choose $\mathbf{p_g}$ (i.e., a leader) for each particle? The PSO needs to favor non-dominated particles over dominated ones, and drive the population towards different parts of the Pareto front, not just towards a single point. This requires that particles be allocated to different leaders.
2. How to identify non-dominated particles with respect to all particles' current positions and personal best positions? And how to retain these solutions during the search process? One strategy is to combine all particles' personal best positions and current positions, and then extract the non-dominated solutions from the combined population.
3. How to maintain particle diversity so that a set of well-distributed solutions can be found along the Pareto front? Some classic niching methods (e.g., crowding or sharing) can be adopted for this purpose.

The first PSO for solving multiobjective optimization was proposed by Moore and Chapman in 1999 [126]. An *lbest* PSO was used, and $\mathbf{p_g}$ was chosen from a local neighborhood using a ring topology. All personal best positions were kept in an archive. At each particle update, the current position is compared with solutions in this archive to see if the current position represents a non-dominated solution. The archive is updated at each iteration to ensure it contains only non-dominated solutions.

Interestingly it was not until 2002 that the next publication on PSO for multiobjective optimization appeared. In [34], Coello and Lechuga proposed MOPSO (Multiobjective PSO) which uses an external archive to store non-dominated solutions. The diversity of solutions is maintained by keeping only one solution within each hypercube which is predefined by a user in the objective space. In [137], Parsopoulos and Vrahatis adopted a more traditional weighted-sum approach. However, by using gradually changing weights, their approach was able to find a diverse set of solutions along the Pareto front. In [58], Fieldsend and Singh proposed a PSO using a *dominated tree* structure to store non-dominated solutions found. The selection of leaders was also based on this structure. To maintain a better diversity, a *turbulence* operator was adopted to function as a 'mutation' operator in order to perturb the velocity value of a particle.

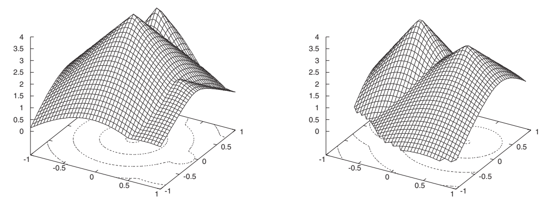
With the aim of increasing the efficiency of extracting non-dominated solutions from a swarm, Li proposed NSPSO (Non-dominated Sorting PSO) [100], which follows the principal idea of the well-known NSGA II algorithm [39]. In NSPSO, instead of comparing solely a particle's personal best with its potential offspring, all particles' personal best positions and offspring are first combined to form a temporary population. After this, domination comparisons for all individuals in this temporary population are carried out. This approach will ensure more non-dominated solutions can be discovered through the domination comparison operations than the above-mentioned multiobjective PSO algorithms.

Many more multiobjective PSO variants have been proposed in recent years. A survey conducted by Sierra and Coello in 2006 shows that there are currently 25 different PSO algorithms for handling multiobjective optimization problems. Interested readers should refer to [155] for more information on these different approaches.

PSO for Dynamic Optimization

Many real-world optimization problems are dynamic and require optimization algorithms capable of adapting to the changing optima over time. For example, traffic conditions in a city change dynamically and continuously. What might be regarded as an optimal route at one time might not be optimal the next minute. In contrast to optimization towards a static optimum, in a dynamic environment the goal is to track as closely as possible the dynamically changing optima. Figure 6 shows an example of a three-peak dynamic environment.

A defining characteristic of PSO is its fast convergent behavior and inherent adaptability [92]. Particles can adaptively adjust their positions based



(a) Before (b) After

Fig. 6. Three-peak dynamic environment, before and after movement of optima. Note that the small peak to the right of the figure becomes hidden and that the highest point switches optimum

on their dynamic interactions with other particles in the population. This makes PSO especially appealing as a potential solution to dynamic optimization problems. Several studies have suggested various approaches to applying PSO to solve dynamic optimization problems [28, 29, 56, 82, 103, 11, 12, 135]. These studies showed that the original PSO must be adapted to meet the additional challenges presented by dynamic optimization problems. In particular, the following questions need to be addressed:

1. How do we detect a change that has actually occurred?
2. Which response strategies are appropriate to use once a change is detected?
3. How do we handle the issue of 'out-of-date' memory as particles' personal best positions become invalid once the environment has changed?
4. How do we handle the trade-off issue between convergence (in order to locate optima) and diversity (in order to relocate changed optima)?

One of the early works on using PSO for dynamic optimization was by Eberhart and Shi in [56], where they used an inertia-weighted PSO to track the optimum of a three-dimensional unimodal parabolic function which changes its maxima every 100 iterations. It was found under certain circumstances that the PSO's performance was comparable to or better than that of previously published evolutionary algorithms.

For detection, Carlisle and Dozier used a *sentry* particle which is randomly chosen at each iteration [28]. The sentry particle gets evaluated before each iteration and compares its fitness with its previous fitness value. If the two values are different, indicating the environment has changed, then the whole population gets alerted and several possible responses can then be triggered. A simple strategy was also proposed by Hu and Eberhart to re-evaluate $\mathbf{p_g}$ and a second-best particle to detect if a change has occurred [82].

Various response strategies have been proposed. To deal with the issue of 'out-of-date' memory as the environment changes, Carlisle and Dozier proposed to periodically replace all personal best positions by their corresponding current positions when a change has been detected [29]. This allows particles to forget their past experience and use only up-to-date knowledge about the new environment. Hu and Eberhart studied the effects of re-randomizing various proportions of the swarm to maintain some degree of diversity in order to better track the optima after a change [82]. However, this approach suffers from possible information loss since the re-randomized portion of the population does not retain any information that might be useful from the past iterations.

In order to maintain better particle diversity throughout a run, Blackwell and Bentley introduced charged swarms where mutually repelling *charged* particles orbit a nucleus of neutral particles (conventional PSO particles) [9]. Whereas the charged particles allow the swarm to better adapt to changes in the environment, the neutral particles play the role of continuing to converge towards the optimum.

Inspired by multi-population EA approaches such as the self-organizing scouts [24], Blackwell and Branke proposed an interacting multi-swarm PSO as a further improvement to the charged swarms [11]. The multi-swarm PSO aims at maintaining multiple swarm populations on different peaks. Multiple swarms are prevented from converging to the same optimum by randomizing the worse of two swarms that come too close. The multi-swarm PSO also replaces the charged particles with quantum particles whose position is solely based on a probability function centered around the swarm attractor. The resulting multi-quantum swarms outperform charged and standard PSOs on the moving peaks function. This multi-swarm approach is particularly attractive because of its improved adaptability in a more complex multimodal dynamic environment where multiple peaks exist and need to be tracked.

With a similar aim to locate and track multiple peaks in a dynamic environment, Parrott and Li in [101, 135] proposed a species-based PSO (SPSO) incorporating a speciation algorithm first proposed by Pétrowski [139]. The SPSO uses a local "species seed" which provides the local $\mathbf{p_g}$ to particles whose positions are within a user-specified radius of the seed. This encourages swarms to converge onto multiple local optima instead of a single global optimum, hence developing multiple sub-populations in parallel. In addition, the dynamic SPSO uses a parameter p_{max} to limit the number of particles allowed in a species (or swarm), with the excess particles reinitialized at random positions in the total search space. In [102], Li et al. also demonstrated that the quantum particle model in [11] can be incorporated into SPSO to improve its optima-tracking performance for the moving peaks problem [24].

In another work [84], Janson and Middendorf proposed a PSO using a dynamic and hierarchical neighborhood structure to handle dynamic optimization problems. They demonstrated that such a structure is useful for maintaining some particle diversity in a dynamic environment.

PSO for Constraint Handling

Many real-world problems require an optimization algorithm to find solutions that satisfy a certain number of constraints. The most common approach for solving constrained problems is the use of a penalty function, where the constrained problem is transformed into an unconstrained one, by penalizing the constraints and creating a single objective function. Parsopoulos and Vrahatis proposed a PSO where non-stationary penalty functions are used [136]. The penalty value is dynamically modified during a run. This method is problem dependent; however its results are generally superior to those obtained through stationary functions. In Toscano and Coello's PSO algorithm [141], if both particles compared are infeasible, then the particle that has the lowest value in its total violation of constraints wins. One major disadvantage of using penalty functions, in which case all constraints must be combined into a single objective function (this is also called the weighted-sum approach), is that a user must specify a weight coefficient for each constraint. However, finding optimal weight coefficients is no easy task. A preferred approach is a multiobjective one where the concept of "dominance" can be used to identify better solutions which are non-dominated solutions with respect to the current population. The merit of this multiobjective approach is that the user is no longer required to specify any weight coefficient.

Another useful technique as described by Clerc is "confinement by dichotomy" [32], which makes use of an iterative procedure to find points that are close to the boundaries defined by constraints. Both the dichotomy and multiobjective methods are general enough that they are applicable to most constrained optimization problems.

4 Further Examples of Swarm Intelligence in Optimization

ACO and PSO are two very successful examples of swarm intelligence, yet there are many more applications based on SI principles. Some representative examples are given in the following.

4.1 Applications Inspired by the Division of Labour

Algorithms based on the division of labour in ant colonies and wasp colonies are an important example of the use of swarm intelligence principles in technical applications. Much of the relevant works go back to the study of Wilson [178], who showed that the concept of division of labour in colonies of ants from the *Pheidole* genus allows the colony to adapt to changing demands. Workers in these colonies are generally divided into two groups: Small minors and larger majors. The minors are mostly doing quotidian tasks, whereas the majors do seed milling, storing of abdominal food, or defense tasks. By experimentally reducing the number of minors, Wilson observed that some of the majors switched to tasks usually fulfilled by minors. The division of labour was later modelled by Theraulaz et al. [167] and Bonabeau et al. [22] by means of response threshold models. The model permits a set of threshold values for each individual, one threshold value for each type of task. A threshold value, which may be fixed or dynamically changing over time, can be interpreted as the degree of specialization for the respective task. Furthermore, each task emits a stimulus in order to attract the attention of the individuals, which—depending on the stimulus and the corresponding threshold value—decide whether to accept or to decline the task.

The above-mentioned response threshold models inspired several technical applications. Campos et al. [27], Cicirello and Smith [31], and Nouyan et al. [129] deal with a static, or a dynamic task allocation problem where trucks have to be painted in a number of painting booths. Another application concerns media streaming in peer-to-peer networks. Here, a peer must adapt to changes in the supply and demand of media streams. For this purpose, Sasabe et al. [151] propose a novel caching algorithm based on a response threshold model. In [181], Yu and Ram propose a multi-agent system for the scheduling of dynamic job shops with flexible routing and sequence-dependent setups based on the division of labour in social insects. Finally, Merkle et al. [119] use a response threshold model for the self-organized task allocation for computing systems with reconfigurable components.

4.2 Ant-Based Clustering and Sorting

In 1991 Deneubourg et al. [43] proposed a model to describe the clustering as well as the sorting behavior of ants. Here, clustering refers to the gathering of items in order to form heaps. This happens, for example, when ants of the species *Pheidole pallidula* cluster the bodies of dead nest mates (also known as cemetery formation). Sorting, on the other hand, refers to the spatial arrangement of different objects according to their properties, a behavior which can be observed, for example, in nests of the species *Leptothorax unifasciatus.* Ants of this species compactly cluster eggs and microlarvae at the center of the brood area, whereas the largest larvae are placed at the periphery of the brood area. In computer simulations in [43] ants were modelled as agents randomly moving in their environment in which items were initially scattered. Agents were able to pick up items, to transport them, and to drop them. The probabilities for these actions were derived from the distribution of items in the agents' local neighborhood. For example, items that are isolated had a higher probability of being picked up. As a result, a clustering and sorting of items in the agents environment was obtained.

Mostly based on the above mentioned model by Deneubourg et al., several algorithms for clustering and sorting appeared in the literature. The first one was an algorithm proposed in [108] that extended the original model in order to be able to handle numerical data. Later papers deal with the models' use for the two-dimensional visualization of document collections such as Web data (see, for example, [77]) and for graph partitioning (see, for example, [97]).

4.3 Other Applications

Recently, research on swarm robotics has taken much inspiration from swarm intelligence. For example, the path finding and orientation skills of the desert ant *Cataglyphis* were used as an archetype for building a robot orientation unit [98]. Models for the division of labor between members of an ant colony were used to regulate the joint work of robots (see, for example, [1]). In [96] the collective transport of ants inspired the design of controllers of robots for doing coordinated work. More detailed and up-to-date information can be found in Chap. 3 of this book.

Acknowledgements

This work was supported by grants TIN2005-08818 (OPLINK) and TIN2007-66523 (FORMALISM) of the Spanish government, and by the EU project FRONTS (FP7-ICT-2007-1) funded by the European Comission under the FET Proactive Initiative *Pervasive Adaptation*. In addition, Christian Blum acknowledges support from the *Ramón y Cajal* program of the Spanish Ministry of Science and Technology of which he is a research fellow.

References

1. W. Agassounoun, A. Martinoli, and R. Goodman. A scalable, distributed algorithm for allocating workers in embedded systems. In Terry Bahill, editor, *Proceedings of the 2001 IEEE Systems, Man and Cybernetics Conference*, pages 3367–3373. IEEE Press, 2001.
2. S. Alupoaei and S. Katkoori. Ant colony system application to macrocell overlap removal. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(10):1118–1122, 2004.
3. J. Bautista and J. Pereira. Ant algorithms for a time and space constrained assembly line balancing problem. *European Journal of Operational Research*, 177(3), 2007.
4. G. Beni. The concept of cellular robotic systems. In *Proceedings of the IEEE International Symposium on Intelligent Systems*, pages 57–62. IEEE Press, Piscataway, NJ, 1988.

5. L. Bianchi, L. M. Gambardella, and M. Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In J. J. Merelo, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacanas, and H.-P. Schwefel, editors, *Proceedings of PPSN VII, Seventh International Conference on Parallel Problem Solving from Nature*, volume 2439 of *Lecture Notes in Computer Science*, pages 883–892. Springer, Berlin, Germany, 2002.
6. B. Bilchev and I. C. Parmee. The ant colony metaphor for searching continuous design spaces. In T. C. Fogarty, editor, *Proceedings of the AISB Workshop on Evolutionary Computation*, volume 993 of *Lecture Notes in Computer Science*, pages 25–39. Springer, Berlin, Germany, 1995.
7. M. Birattari, G. Di Caro, and M. Dorigo. Toward the formal foundation of ant programming. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms – Proceedings of ANTS 2002 – Third International Workshop*, volume 2463 of *Lecture Notes in Computer Science*, pages 188–201. Springer, Berlin, Germany, 2002.
8. S. Bird and X. Li. Adaptively choosing niching parameters in a PSO. In Mike Cattolico, editor, *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006*, pages 3–10. ACM, 2006.
9. T. Blackwell and P. J. Bentley. Dynamic search with charged swarms. In *Proc. the Workshop on Evolutionary Algorithms Dynamic Optimization Problems (EvoDOP 2003)*, pages 19–26, 2002.
10. T. Blackwell and P. J. Bentley. Improvised music with swarms. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC 2002*, pages 1462–1467. IEEE Press, 2002.
11. T. Blackwell and J. Branke. Multi-swarm optimization in dynamic environments. In *EvoWorkshops*, volume 3005 of *Lecture Notes in Computer Science*, pages 489–500. Springer, 2004.
12. T. Blackwell and J. Branke. Multi-swarms, exclusion and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472, 2006.
13. M. Blesa and C. Blum. Ant colony optimization for the maximum edge-disjoint paths problem. In G. R. Raidl, S. Cagnoni, J. Branke, D. W. Corne, R. Drechsler, Y. Jin, C. G. Johnson, P. Machado, E. Marchiori, R. Rothlauf, G. D. Smith, and G. Squillero, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2004*, volume 3005 of *Lecture Notes in Computer Science*, pages 160–169. Springer, Berlin, Germany, 2004.
14. C. Blum. Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.
15. C. Blum, J. Bautista, and J. Pereira. Beam-ACO applied to assembly line balancing. In M. Dorigo, L. M. Gambardella, A. Martinoli, R. Poli, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence – Proceedings of ANTS 2006 – Fifth International Workshop*, volume 4150 of *Lecture Notes in Computer Science*, pages 96–107. Springer, Berlin, Germany, 2006.
16. C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 34(2):1161–1172, 2004.

17. C. Blum and M. Dorigo. Search bias in ant colony optimization: On the role of competition-balanced systems. *IEEE Transactions on Evolutionary Computation*, 9(2):159–174, 2005.
18. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
19. C. Blum and M. Sampels. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3(3):285–308, 2004.
20. C. Blum and M. Yábar Vallès. Multi-level ant colony optimization for DNA sequencing by hybridization. In F. Almeida, M. Blesa, C. Blum, J. M. Moreno, M. Pérez, A. Roli, and M. Sampels, editors, *Proceedings of HM 2006 – 3rd International Workshop on Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 94–109. Springer-Verlag, Berlin, Germany, 2006.
21. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY, 1999.
22. E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg. Fixed response thresholds and the regulation of division of labor in social societies. *Bulletin of Mathematical Biology*, 60:753–807, 1998.
23. A. Brandt. Multilevel computations: Review and recent developments. In S. F. McCormick, editor, *Multigrid Methods: Theory, Applications, and Supercomputing, Proceedings of the 3rd Copper Mountain Conference on Multigrid Methods*, volume 110 of *Lecture Notes in Pure and Applied Mathematics*, pages 35–62. Marcel Dekker, New York, 1988.
24. J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Norwell, MA, 2002.
25. T. N. Bui and J. R. Rizzo Jr. Finding maximum cliques with distributed ants. In K. Deb et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, volume 3102 of *Lecture Notes in Computer Science*, pages 24–35. Springer, Berlin, Germany, 2004.
26. B. Bullnheimer, R. Hartl, and C. Strauss. A new rank-based version of the Ant System: A computational study. *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999.
27. M. Campos, E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg. Dynamic scheduling and division of labor in social insects. *Adaptive Behavior*, 8(3):83–96, 2000.
28. A. Carlisle and G. Dozier. Adapting particle swarm optimization to dynamic environments. In the *Proceedings of the International Conference on Artificial Intelligence (ICAI 2000)*, pages 429–434, Las Vegas, Nevada, USA, 2000.
29. A. Carlisle and G. Dozier. Tracking changing extrema with adaptive particle swarm optimizer. In *Proceedings of the 5th Biannual World Automation Congress*, pages 265–270, Orlando FL, USA, 2002.
30. C.-Y. Chen and F. Ye. Particle swarm optimization algorithm and its application to clustering analysis. In *IEEE International Conference on Networking, Sensing and Control*, volume 2, pages 789–794, 2004.
31. V. A. Cicirello and S. S. Smith. Wasp-like agents for distributed factory coordination. *Journal of Autonomous Agents and Multi-Agent Systems*, 8:237–266, 2004.
32. M. Clerc. *Particle Swarm Optimization*. ISTE Ltd, UK, 2006.

33. M. Clerc and J. Kennedy. The particle swarm—explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6:58–73, 2002.
34. C. Coello Coello and M. Salazar Lechuga. MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. In *Congress on Evolutionary Computation (CEC 2002)*, volume 2, pages 1051–1056, Piscataway, New Jersey, May 2002. IEEE Service Center.
35. A.V.E. Conradie, R. Miikkulaninen, and C. Aldrich. Adaptive control utilizing neural swarming. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, New York, USA, 2002.
36. E.S. Correa, A. Freitas, and C.G. Johnson. A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set. In *GECCO 2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, volume 1, pages 35–42, Seattle, Washington, USA, 2006. ACM Press.
37. P. Corry and E. Kozan. Ant colony optimization for machine layout problems. *Computational Optimization and Applications*, 28(3):287–310, 2004.
38. D. Costa and A. Hertz. Ants can color graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.
39. K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
40. F. Della Croce, M. Ghirardi, and R. Tadei. Recovering beam search: enhancing the beam search approach for combinatorial optimisation problems. In *Proceedings of PLANSIG 2002 – 21st workshop of the UK Planning and Scheduling Special Interest Group*, pages 149–169, 2002.
41. M. L. den Besten, T. Stützle, and M. Dorigo. Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Proceedings of PPSN VI, Sixth International Conference on Parallel Problem Solving from Nature*, volume 1917 of *Lecture Notes in Computer Science*, pages 611–620. Springer, Berlin, Germany, 2000.
42. J.-L. Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels. The self-organizing exploratory pattern of the *Argentine* ant. *Journal of Insect Behaviour*, 3:159–168, 1990.
43. J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting: Robot-like ants and ant-like robots. In *Proceedings of the First International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 1*, pages 356–365. MIT Press, Cambridge, MA, 1991.
44. G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.
45. K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. *Annals of Operations Research*, 131:79–99, 2004.
46. M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.
47. M. Dorigo and C. Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2-3):243–278, 2005.

48. M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.
49. M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
50. M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.
51. M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–41, 1996.
52. M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, MA, 2004.
53. C. A. Dosoer and M. Vidyasagar. *Feedback Systems: Input–Ouput Properties.* Academics, New York, 1975.
54. J. Dréo and P. Siarry. A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms – Proceedings of ANTS 2002 – Third International Workshop*, volume 2463 of *Lecture Notes in Computer Science*, pages 216–221. Springer, Berlin, Germany, 2002.
55. R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proc. of IEEE Int. Conf. Evolutionary Computation*, pages 84–88, 2000.
56. R. C. Eberhart and Y. Shi. Tracking and optimizing dynamic systems with particle swarms. In *Proc. the 2001 Congress on Evolutionary Computation (CEC 2001)*, pages 94–100. IEEE Press, 2001.
57. R. C. Eberhart, P. K. Simpson, and R. W. Dobbins. *Computational Intelligence PC Tools.* Academic Press, Boston, 1996.
58. J. E. Fieldsend and S. Singh. A multiobjective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence. In *Proceedings of the 2002 UK Workshop on Computational Intelligence*, pages 37–44, Birmingham, UK, September 2002.
59. C. Gagné, W. L. Price, and M. Gravel. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. *Journal of the Operational Research Society*, 53:895–906, 2002.
60. Z. L. Gaing. A particle swarm optimization approach for optimum design of PID controller in AVR system. *IEEE Transactions on Energy Conversion*, 19(2):384–391, June 2004.
61. L. M. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In T. Baeck, T. Fukuda, and Z. Michalewicz, editors, *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, pages 622–627. IEEE Press, Piscataway, NJ, 1996.
62. L. M. Gambardella and M. Dorigo. Ant Colony System hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.
63. L. M. Gambardella, É. D. Taillard, and G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw-Hill, London, UK, 1999.

64. X. Gandibleux, X. Delorme, and V. T'Kindt. An ant colony optimisation algorithm for the set packing problem. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 49–60. Springer, Berlin, Germany, 2004.
65. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
66. V. Georgiou, N. Pavlidis, K. Parsopoulos, and M. Vrahatis. Optimizing the performance of probabilistic neural networks in a bioinformatics task. In *Proceedings of the EUNITE 2004 Conference*, pages 34–40, 2004.
67. F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533–549, 1986.
68. F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, 2002.
69. J. Gottlieb, M. Puchta, and C. Solnon. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In S. Cagnoni, J. J. Romero Cardalda, D. W. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, E. Marchiori, J.-A. Meyer, M. Middendorf, and G. R. Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2003*, volume 2611 of *Lecture Notes in Computer Science*, pages 246–257. Springer, Berlin, Germany, 2003.
70. P.-P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez *bellicositermes natalensis et cubitermes sp.* La théorie de la stigmergie: Essai d'interprétation des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.
71. V. G. Gudise and G. K. Venayagamoorthy. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pages 110–117, Indianapolis, Indiana, USA, 2003.
72. C. Guéret, N. Monmarché, and M. Slimane. Ants can play music. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 310–317. Springer, Berlin, Germany, 2004.
73. M. Guntsch and M. Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, volume 2037 of *Lecture Notes in Computer Science*, pages 213–222. Springer, Berlin, Germany, 2001.
74. M. Guntsch and M. Middendorf. Solving multiobjective permutation problems with population based ACO. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2636 of *Lecture Notes in Computer Science*, pages 464–478. Springer, Berlin, Germany, 2003.
75. W. J. Gutjahr. A graph-based ant system and its convergence. *Future Generation Computer Systems*, 16(9):873–888, 2000.

76. W. J. Gutjahr. ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82(3):145–153, 2002.
77. J. Handl and B. Meyer. Improved ant-based clustering and sorting in a document retrieval interface. In J. J. Merelo, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacanas, and H.-P. Schwefel, editors, *Proceedings of PPSN VII, Seventh International Conference on Parallel Problem Solving from Nature*, volume 2439 of *Lecture Notes in Computer Science*, pages 913–923. Springer, Berlin, Germany, 2002.
78. F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. In S. Krasner, editor, *The Ubiquity of Chaos*, Washington, DC, 1990. AAAS Publications.
79. N. Higashi and H. Iba. Particle swarm optimization with Gaussian mutation. In *Proc. of the 2003 IEEE Swarm Intelligence Symposium (SIS'03)*, pages 72–79, 2003.
80. H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications.* Elsevier, Amsterdam, The Netherlands, 2004.
81. X. Hu and R. Eberhart. Multiobjective optimization using dynamic neighborhood particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC 2002*, pages 1677–1681. IEEE Press, 2002.
82. X. Hu and R. C. Eberhart. Adaptive particle swarm optimisation: detection and response to dynamic systems. In *Proc. Congress on Evolutionary Computation*, pages 1666–1670, 2002.
83. S. Janson and M. Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(6):1272–1282, 2005.
84. S. Janson and M. Middendorf. A hierarchical particle swarm optimizer for noisy and dynamic environments. *Genetic Programming and Evolvable Machines*, 7(4):329–354, 2006.
85. V. Kadirkamanathan, K. Selvarajah, and P. Fleming. Stability analysis of the particle dynamics in particle swarm optimizer. *IEEE Transactions on Evolutionary Computation*, 10(3):245–255, June 2006.
86. O. Karpenko, J. Shi, and Y. Dai. Prediction of MHC class II binders using the ant colony search strategy. *Artificial Intelligence in Medicine*, 35(1-2):147–156, 2005.
87. S. Katare, A. Kalos, and D. West. A hybrid swarm optimizer for efficient parameter estimation. In *Proceedings of the 2004 Congress on Evolutionary Computation CEC 2004)*, pages 309–315. IEEE Press, 2004.
88. J. Kennedy. The behaviour of particles. In *Evolutionary Programming VII: Proceedings of the 7th Annual Conference*, volume 1447 of *Lecture Notes in Computer Science*, pages 581–589, San Diego, CA, 1998. Springer, Berlin, Germany.
89. J. Kennedy. Bare bones particle swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003)*, pages 80–87, Indianapolis, Indiana, USA, 2003.
90. J. Kennedy. In search of the essential particle swarm. In *Proc. of the 2006 IEEE Congress on Evolutionary Computation*, pages 6158–6165. IEEE Press, 2006.
91. J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE Press, Piscataway, NJ, 1995.

92. J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, CA, 2004.
93. O. Korb, T. Stützle, and T. E. Exner. PLANTS: Application of ant colony optimization to structure-based drug design. In M. Dorigo, L. M. Gambardella, A. Martinoli, R. Poli, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence – Proceedings of ANTS 2006 – Fifth International Workshop*, volume 4150 of *Lecture Notes in Computer Science*, pages 247–258. Springer, Berlin, Germany, 2006.
94. P. Korošec, J. Šilc, and B. Robič. Mesh-partitioning with the multiple ant-colony algorithm. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 430–431. Springer, Berlin, Germany, 2004.
95. P. Korošec, J. Šilc, and B. Robič. Solving the mesh-partitioning problem with an ant-colony algorithm. *Parallel Computing*, 30:785–801, 2004.
96. C. R. Kube and E. Bonabeau. Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30:85–101, 2000.
97. P. Kuntz, D. Snyers, and P. Layzell. A stochastic heuristic for visualizing graph clusters in a bi-dimensional space prior to partitioning. *Journal of Heuristics*, 5(3):327–351, 1998.
98. D. Lambrinos, R. Möller, T. Labhart, R. Pfeifer, and R. Wehner. A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems*, 30:39–64, 2000.
99. E. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Travelling Salesman Problem*. John Wiley & Sons, New York, NY, 1985.
100. X. Li. A Non-dominated Sorting Particle Swarm Optimizer for Multiobjective Optimization. In E. Cantú-Paz et al., editor, *Genetic and Evolutionary Computation—GECCO 2003. Proceedings, Part I*, pages 37–48. Springer. Lecture Notes in Computer Science Vol. 2723, July 2003.
101. X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In K. Deb, editor, *Proceedings of Genetic and Evolutionary Computation Conference 2004 (GECCO'04) (LNCS 3102)*, pages 105–116, 2004.
102. X. Li, J. Branke, and T. Blackwell. Particle swarm with speciation and adaptation in a dynamic environment. In Mike Cattolico, editor, *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006*, pages 51–58. ACM, 2006.
103. X. Li and K.H. Dam. Comparing particle swarms for tracking extrema in dynamic environments. In *Proc. of the 2003 IEEE Congress on Evolutionary Computation*, pages 1772–1779, 2003.
104. J. J. Liang and P. N. Suganthan. Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism. In *Proc. of the 2006 IEEE Congress on Evolutionary Computation*, pages 9–16. IEEE Press, 2006.
105. J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Trans. Evol. Comput.*, 10(3):281–295, June 2006.
106. M. López-Ibáñez, L. Paquete, and T. Stützle. On the design of ACO for the biobjective quadratic assignment problem. In M. Dorigo, M. Birattari,

C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 214–225. Springer, Berlin, Germany, 2004.

107. M. Lovbjerg and T. Krink. Extending particle swarm optimizers with self-organized criticality. In *Proc. of the 2002 IEEE Congr. Evol. Comput.*, pages 1588–1593. IEEE Press, 2002.
108. E. D. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 3 (SAB 94)*, pages 501–508. MIT Press, 1994.
109. V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.
110. V. Maniezzo, M. Boschetti, and M. Jelasity. An ant approach to membership overlay design. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 37–48. Springer, Berlin, Germany, 2004.
111. V. Maniezzo and A. Colorni. The Ant System applied to the quadratic assignment problem. *IEEE Transactions on Data and Knowledge Engineering*, 11(5):769–778, 1999.
112. V. Maniezzo and M. Milandri. An ant-based framework for very strongly constrained problems. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms – Proceedings of ANTS 2002 – Third International Workshop*, volume 2463 of *Lecture Notes in Computer Science*, pages 222–227. Springer, Berlin, Germany, 2002.
113. R. Marinke, I. Matiko, E. Araujo, and L. Coelho. Particle swarm optimization (PSO) applied to fuzzy modeling in a thermal-vacuum system. In *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, pages 67–72. IEEE Computer Society, 2005.
114. K. Marriott and P. Stuckey. *Programming With Constraints.* MIT Press, Cambridge, MA, 1998.
115. R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle swarms for feedforward neural networks training. In *International Joint Conference on Neural Networks*, pages 1895–1889. Honolulu (Hawaii), USA, 2002.
116. R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, June 2004.
117. D. Merkle and M. Middendorf. Modelling ACO: Composed permutation problems. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms – Proceedings of ANTS 2002 – Third International Workshop*, volume 2463 of *Lecture Notes in Computer Science*, pages 149–162. Springer, Berlin, Germany, 2002.
118. D. Merkle and M. Middendorf. Modelling the dynamics of ant colony optimization algorithms. *Evolutionary Computation*, 10(3):235–262, 2002.
119. D. Merkle, M. Middendorf, and A. Scheidler. Self-organized task allocation for computing systems with reconfigurable components. In *Proceedings of the 20th*

International Parallel and Distributed Processing Symposium (IPDPS 2006), 8 pages, IEEE press, 2006.
120. D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333–346, 2002.
121. N. Meuleau and M. Dorigo. Ant colony optimization and stochastic gradient descent. *Artificial Life*, 8(2):103–121, 2002.
122. B. Meyer and A. Ernst. Integrating ACO and constraint propagation. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 166–177. Springer, Berlin, Germany, 2004.
123. R. Michel and M. Middendorf. An island model based ant system with lookahead for the shortest supersequence problem. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 692–701. Springer, Berlin, Germany, 1998.
124. S. Mikki and A. Kishk. Investigation of the quantum particle swarm optimization technique for electromagnetic applications. In *2005 IEEE Antennas and Propagation Society International Symposium*, volume 2A, pages 45–48, 2005.
125. N. Monmarché, G. Venturini, and M. Slimane. On how *Pachycondyla apicalis* ants suggest a new search algorithm. *Future Generation Computer Systems*, 16:937–946, 2000.
126. J. Moore and R. Chapman. Application of particle swarm to multiobjective optimization. Department of Computer Science and Software Engineering, Auburn University, 1999.
127. J. D. Moss and C. G. Johnson. An ant colony algorithm for multiple sequence alignment in bioinformatics. In D. W. Pearson, N. C. Steele, and R. F. Albrecht, editors, *Artificial Neural Networks and Genetic Algorithms*, pages 182–186. Springer, Berlin, Germany, 2003.
128. G. L. Nemhauser and A. L. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, New York, 1988.
129. S. Nouyan, R. Ghizzioli, M. Birattari, and M. Dorigo. An insect-based algorithm for the dynamic task allocation problem. *Künstliche Intelligenz*, 4:25–31, 2005.
130. G. Onwubolu and M. Clerc. Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization. *International Journal of Production Research*, 42(3/01):473–491, February 2004.
131. P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.
132. E. Ozcan and C.K. Mohan. Analysis of a simple particle swarm optimization system. In *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 253–258, 1998.
133. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization—Algorithms and Complexity.* Dover Publications, Inc., New York, NY, 1982.
134. R. S. Parpinelli, H. S. Lopes, and A. A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332, 2002.

135. D. Parrott and X. Li. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation*, 10(4):440–458, August 2006.
136. K. Parsopoulos and M. Vrahatis. Particle swarm optimization method for constrained optimization problems. *Intelligent Technologies—Theory and Applications: New Trends in Intelligent Technologies*, 76:214–220, 2002.
137. K. Parsopoulos and M. Vrahatis. Particle swarm optimization method in multiobjective problems. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)*, pages 603–607. Madrid, Spain, ACM Press, 2002.
138. K. Parsopoulos and M. Vrahatis. On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):211–224, June 2004.
139. A. Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation*, pages 798–803, 1996.
140. G. Pulido and C. Coello Coello. A constraint-handling mechanism for particle swarm optimization. In *Proc. of the 2004 IEEE Congress on Evolutionary Computation*, pages 1396–1403. IEEE Press, 2004.
141. G. T. Pulido and C. Coello Coello. A constraint-handling mechanism for particle swarm optimization. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1396–1403, Portland, Oregon, 20-23 June 2004. IEEE Press.
142. A. P. Engelbrecht, R. Brits and F. van den Bergh. A niching particle swarm optimizer. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning 2002 (SEAL 2002)*, pages 692–696, 2002.
143. K. Rameshkumar, R. Suresh, and K. Mohanasundaram. Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan. In *First International Conference of Advances in Natural Computation*, pages 572–581, 2005.
144. C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Inc., New York, NY, 1993.
145. M. Reimann, K. Doerner, and R. F. Hartl. D-ants: Savings based ants divide and conquer the vehicle routing problems. *Computers & Operations Research*, 31(4):563–591, 2004.
146. C.W. Reynolds. Flocks, herds and schools: a distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.
147. T. Richer and T. Blackwell. The *Lévy* particle swarm. In *Congress on Evolutionary Computation (CEC 2006)*, pages 808– 815. IEEE press, 2006.
148. J. Riget and J. Vesterstroem. A diversity-guided particle swarm optimizer—the ARPSO. *Technical Report 2002-02, Department of Computer Science, University of Aarhus*, 2002.
149. R. Rucker. *Seek!* Four Walls Eight Windows, New York, 1999.
150. A. Salman, A. Imtiaz, and S. Al-Madani. Particle swarm optimization for task assignment problem. In *IASTED International Conference on Artificial Intelligence and Applications (AIA 2001)*, Marbella, Spain, 2001.
151. M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara. Effective methods for scalable and continuous media streaming on peer-to-peer networks. *European Transactions on Telecommunications*, 15:549–558, 2004.

152. M. Settles, B. Rodebaugh, and T. Soule. Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network. In *Genetic and Evolutionary Computation Conference 2003 (GECCO 2003)*, pages 151–152, Chicago, USA, 2003.
153. A. Shmygelska, R. Aguirre-Hernández, and H. H. Hoos. An ant colony optimization algorithm for the 2D HP protein folding problem. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms – Proceedings of ANTS 2002 – Third International Workshop*, volume 2463 of *Lecture Notes in Computer Science*, pages 40–52. Springer, Berlin, Germany, 2002.
154. A. Shmygelska and H. H. Hoos. An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. *BMC Bioinformatics*, 6(30):1–22, 2005.
155. M. Reyes Sierra and C. Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.
156. C. A. Silva, T. A. Runkler, J. M. Sousa, and R. Palm. Ant colonies as logistic processes optimizers. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms – Proceedings of ANTS 2002 – Third International Workshop*, volume 2463 of *Lecture Notes in Computer Science*, pages 76–87. Springer, Berlin, Germany, 2002.
157. K. Socha. ACO for continuous and mixed-variable optimization. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 25–36. Springer, Berlin, Germany, 2004.
158. K. Socha and C. Blum. An ant colony optimization algorithm for continuous optimization: An application to feed-forward neural network training. *Neural Computing & Applications*, 2007. In press.
159. K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *European Journal of Operational Research*, 2007. In press.
160. K. Socha, M. Sampels, and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In S. Cagnoni, J. J. Romero Cardalda, D. W. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, E. Marchiori, J.-A. Meyer, M. Middendorf, and G. R. Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2003*, volume 2611 of *Lecture Notes in Computer Science*, pages 334–345. Springer, Berlin, Germany, 2003.
161. C. Solnon. Ant can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4):347–357, 2002.
162. T. Stützle. An ant approach to the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, pages 1560–1564. Verlag Mainz, Aachen, Germany, 1998.
163. T. Stützle and M. Dorigo. A short convergence proof for a class of ACO algorithms. *IEEE Transactions on Evolutionary Computation*, 6(4):358–365, 2002.
164. T. Stützle and H. H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.
165. P.N. Suganthan. Particle swarm optimiser with neighbourhood operator. In *Congress on Evolutionary Computation (CEC 1999)*, pages 1958–1962, Washington, USA, 1999.

166. T.-Y. Sun, S.-T. Hsieh, H.-M. Wang, and C.-W. Lin. Floorplanning based on particle swarm optimization. In *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures 2006*, pages 5–10. IEEE Press, 2006.
167. G. Theraulaz, E. Bonabeau, and J.-L. Deneubourg. Response threshold reinforcement and division of labour in insect societies. *Proceedings: Biological Sciences*, 265(1393):327–332, 1998.
168. I. C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection, 2003.
169. R. Unger and J. Moult. Finding the lowest free-energy conformation of a protein is an NP-hard problem: Proofs and implications. *Bulletin of Mathematical Biology*, 55(6):1183–1198, 1993.
170. F. van den Bergh. *Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.
171. F. van den Bergh and A.P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Trans. Evol. Compu.*, 8:225–239, Jun. 2004.
172. F. van den Bergh and A.P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176:937–971, 2006.
173. K. Veeramachaneni, T. Peram, C. Mohan, and L. Osadciw. Optimization using particle swarm with near neighbor interactions. In *Proc. of Genetic and Evolutionary Computation Conference*, pages 110 – 121, Chicago, Illinois, 2003.
174. G. K. Venayagamoorthy. Optimal control parameters for a UPFC in a multimachine using PSO. In *Proceedings of the 13th International Intelligent Systems Application to Power Systems 2005*, pages 488–493, 2005.
175. M. Vidyasagar. *Nonlinear Systems Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1993.
176. M. Wachowiak, R. Smolikova, Y. Zheng, J. Zurada, and A. Elmaghraby. An approach to multimodal biomedical image registration utilizing particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):289–301, June 2004.
177. C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, 2000.
178. E. O. Wilson. The relation between caste ratios and division of labour in the ant genus *phedoile*. *Behavioral Ecology and Sociobiology*, 16(1):89–98, 1984.
179. X. Xie, W. Zhang, and Z. Yang. A dissipative particle swarm optimization. In *Proc. Congr. Evol. Comput. 2002 (CEC 2002)*, pages 1456–1461. IEEE Press, 2002.
180. H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, November 2001.
181. X. Yu and B. Ram. Bio-inspired scheduling for dynamic job shops with flexible routing and sequence-dependent setups. *International Journal of Production Research*, 44(22):4793–4813, 2006.
182. W. Zha and G. K. Venayagamoorthy. Neural networks based non-uniform scalar quantizer design with particle swarm optimization. In *Proceedings 2005 IEEE Swarm Intelligence Symposium (SIS 2005)*, pages 143–148. IEEE Press, 2005.

183. M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131(1–4):373–395, 2004.