# SWARM-BASED OPTIMIZATION ALGORITHMS

## 1. INTRODUCTION

Nature has always inspired us. From arts to science and engineering, nature has provided an apparently endless list of objects of interest. Out of these, the human brain is probably the most enigmatic object of study. Efforts to understand it have ranged from introspection to anatomic descriptions of its physical structure. Both of these efforts have resulted in models that we have simulated in digital computers as an attempt to create intelligent systems.

The former approach gave us systems that use existing knowledge of a domain and some form of logic in order to derive new knowledge. Nevertheless, their practical success varies considerably. The modeling of the physical structure of natural systems has resulted in some of the most successful attempts at mimicking human intelligence. As an example, consider the remarkable set of applications that neural networks, and deep learning in particular, have exhibited in recent years (1,2).

Swarm intelligence (3,4), in particular the swarm-based optimization algorithms, share with neural networks the key aspect of being composed of a large set of processing units that, individually, have only limited computational power. However, put together, these units can form powerful information processing systems. Thus, putting it simply, we can say that a kind of collective intelligence emerges from the interactions of many non-intelligent units. This point is the overarching unifying theme behind swarm intelligence techniques.

After decades of research on artificial intelligence, it has become evident that intelligence is often associated with search and/or optimization (5). Therefore, it is not surprising that at the core of diverse artificial intelligence techniques lies an optimization problem or an optimization algorithm, which emphasizes the importance of search and optimization in the field.

There is a rich variety of optimization techniques available. Swarm-based algorithms are part of the family of algorithms inspired by nature. Different metaphors have been used as a basis for the development of nature-inspired search and optimization algorithms. One such metaphor is based on what we know about natural evolution, and the corresponding class of algorithms comprises the so-called evolutionary algorithms (EAs) (6). In essence, EAs employ populations of search agents that encode candidate solutions of the problem at hand. The population is evolved by applying operators that imitate the effects of biological DNA procedures, namely, crossover, mutation, and selection, in order to produce new populations. Selection pressure and the imposed competition among members of the population offer the necessary guidance toward improved solutions.

EAs have proved to be very effective in a large number of applications (7,8). Their ability to cope with uncertainty, as well as their minor requirements with respect to the form and mathematical properties of the objective function, has rendered EAs a valuable part of the state-of-the-art in modern optimization literature. Moreover, due to their ability to concurrently evolve many search points, their application in parallel computation environments becomes extremely convenient (9).

The success of EAs has increased interest in different natural systems where traits of computational efficiency have been observed. Swarms, herds, and other socially organized groups of living organisms exhibit clear evidence of problem-solving capabilities (4,10). Typical examples are ants with their remarkable ability of solving shortest path problems during foraging and bees with the ability to guide the search of their mates by waggle dancing according to their discoveries. In such systems, patterns of intelligent behavior emerge as a result of simple local interactions between their members (11).

The aforementioned properties have offered fertile ground for the development of population-based optimization algorithms based on models of natural swarms. Although intimately related with EAs, this type of algorithms promotes cooperation and collaboration rather than competition among the members of the swarm. Thus, it has been distinguished as a unique, promising new category of algorithms called *swarm intelligence* (SI) (4,10,11). Today, SI accounts for a variety of computational models with significant applications. A large number of practical applications, including traveling salesman problem, vehicle routing, flow shop scheduling, and robotics, among others, are reported in (7).

This article presents the principles of swarm-based optimization as well as representative algorithms for continuous and discrete optimization problems. The rest of the article is organized as follows: Section 2 exposes the basic principles that govern SI algorithms. In Sections 3 and 4, selected swarm-based algorithms for discrete and continuous optimization problems are presented. These include the following popular algorithms:

1. Ant colony optimization
2. Particle swarm optimization
3. Artificial bee colony
4. Cuckoo search
5. Bacterial foraging optimization
6. Gravitational search algorithm

Finally, Section 5 concludes the article.

## 2. ESSENTIALS OF SWARM INTELLIGENCE

Swarm behavior (a.k.a. swarming) is regularly observed in natural systems where socially organized living organisms exist. Insects that live in colonies such as ants, bees, and locusts constitute representative examples of such systems where, although each individual has limited sensing and response qualities, the colony as a whole exhibits highly coordinated behavior (12). Migrating birds and fish schools attain similar behaviors, offering fascinating choreographies (13). Bacteria populations perform coordinated translocations, a procedure also known as swarming motility (14). White blood cells exhibit swarm behavior when attacking parasites (15). Humans also exhibit swarm behaviors both in the physical world (16) as well as in collective decision-making procedures such as social swarming (17).

In all these systems, cooperation is the originating source of the emergent collective behavior. It constitutes the cornerstone quality that imposes the self-organization without central control, rendering the colony capable of solving problems of high complexity. The resultant coordinated behavior is typically defined as SI (4,10). The problem-solving efficiency of the observed SI in natural systems has triggered the development of a number of swarm-based approaches for optimization, simulation, and robotics.

SI algorithms employ artificial swarms of autonomous agents that follow simple rules (relative to the system's complexity) to update their state through time. In the optimization framework, a swarm consists of a number of search agents whose state represents a candidate solution of the optimization problem at hand. The agents adhere to basic rules that promote their cooperation during the search for better solutions. The search is typically defined as an iterative, highly distributed procedure where the agents explore the given search space of the problem while communicating their findings to their mates. Information sharing is a key issue for the efficiency of such computational schemes.

A number of interesting properties habitually appear in swarm-based optimization algorithms. First, the algorithm does not assume any kind of centralized control. Instead, the search agents independently follow fixed or adaptive rules to make decisions on their forthcoming moves. Thus, computation becomes completely distributed, promoting individuality within the swarm. This inherent parallelization property of swarm-based algorithms renders them perfectly suited for contemporary high-performance computation environments such as supercomputers, clusters, and multicore systems.

Second, despite of the special role of each individual agent, the swarm retains robustness due to the lack of explicit reliance on individual agents for its overall operation. Therefore, failure or replacement of fractions of the swarm is not detrimental for its operation. In fact, it has been shown that mild loss of information can be even beneficial for the swarm by enhancing its exploration properties and promoting efficient management of the available computational resources (18).

The basic mechanism that induces the emergence of complex swarm behaviors lies in the exchange of information. The agents can communicate their findings either directly or indirectly. Direct communication implies that the findings of the agents are immediately shared with their mates. On the other hand, indirect communication requires the use of external means where each agent modifies the environment in such a way that it affects other agents' behavior. In natural swarms the external mean is usually the environment, while in artificial swarms it can be an external data structure. For instance, it may come in the form of lookup tables where useful information is stored and regularly accessed by the agents. This mechanism of indirect communication in swarms is also called *stigmergy* and it is tightly related to the self-organization properties in natural swarms (10).

The exploitation of the available information among the search agents is based either on fixed or adaptive strategies. Obviously, adaptability can be highly beneficial for the swarm especially when it takes into consideration the search progress. In this framework, stochasticity is an essential property typically incorporated in the decision-making procedures of swarm-based optimization algorithms, in order to avoid the containment of the agents in narrow response patterns. Learning procedures can further enhance performance. In any case, cooperation among the search agents is the key issue for the emergence of intelligent behavior of the swarm, triggering the formation of behavioral patterns that are not hardwired in the search agents nor directly dictated by external decision centers.

## 3. ALGORITHMS FOR DISCRETE PROBLEMS

The first swarm-based optimization algorithms were designed to solve discrete optimization problems (19). In these kinds of problems, the set of possible solutions to the optimization problem at hand is finite, although typically very large. Examples of search spaces with these properties are the permutations of integers and binary strings.

All problems of practical relevance are defined on spaces where solutions can be thought of comprising a number of components. For example, a permutation of length $n$ over the first $n$ natural numbers can be seen as an array of $n$ natural numbers with the extra condition, or constraint, that no number appears more than once in the array. Formally, a discrete optimization problem consists of a search space $\mathcal{S}$ of feasible solutions,

$$s = (s_1, s_2, \ldots, s_n) \in \mathcal{S}$$

which are vectors of length $n$ where each component $s_i$, $i \in \{1, 2, \ldots, n\}$, can take a value $v_i^j$ from a discrete set $V_i$, also known as the component's domain. A solution $s^* \in \mathcal{S}$ is optimal if,

$$f(s^*) \leqslant f(s), \quad \forall s \in \mathcal{S}$$

where $f : \mathcal{S} \to \mathbb{R}$, is the objective function to be minimized.

Over the years, various swarm-based algorithms for discrete optimization problems have been proposed. The most prominent family consists of algorithms inspired by the behavior of ants. The source of inspiration for these kinds of algorithms can be traced back to an experiment with Argentine ants performed by Goss *et al.* (20). This experiment consisted in restricting the ants' way from their nest to a food source to a path crossing a bridge with two branches of unequal length. The amazing observation was that, over time, ants would choose the branch of shortest length.

Such a phenomenon is remarkable when one considers that ants are not measuring distance or time. The emergence of the intelligent choice is the result of ants laying and following pheromones while traveling back and forth through the bridge. The basic mechanism works as follows: ants lay pheromones while moving. At the same time, ants prefer to walk over areas with higher concentration of pheromones. Since ants walk at about the same speed, the ants that (by chance) initially take the shorter path, start laying pheromones earlier than the rest on their way back to the nest. Thus, they reinforce the trail they initially laid on their way to the food source. When these ants are already on their way back to the nest, reinforcing the shorter path, the ants that took longer paths may have not reached their initial destination yet.

The imbalance of pheromones after the first few trips is amplified because ants prefer areas with higher concentration of pheromones. This imbalance makes the majority of the ants to eventually use the shorter path, even if the probability of choosing each branch is initially the same. Note that this mechanism would cause a single ant to keep choosing the path that it initially selected regardless of its length. Thus, the group is smart, not the individual.

## 3.1. Ant Colony Optimization

Ant colony optimization (ACO) (21) is an algorithmic approach to discrete optimization inspired by the ants' ability to find shortest paths. The basic idea in ACO algorithms is the use of artificial ants, traversing a path on a graph whose vertices typically (but not always) represent a problem's solution components. By moving from one vertex to the next, ants construct solutions to the problem at hand.

The edges of the graph are associated with numerical values that represent pheromone concentration. Additionally, edges may carry extra information that represents domain knowledge. This extra information is heuristic in the sense that it is used to guide the solution-construction process but it does not fully determine it.

In ACO algorithms, the pheromones and the heuristic variables bias the probabilistic solution-building process of the artificial ants. The edges that are part of the paths associated with higher quality solutions receive extra pheromone reinforcement. The combination of pheromone attraction and reinforcement produces a positive feedback process that, over time, biases the search around the most promising solutions.

Since ACO is a general framework rather than a specific algorithm, it needs to be tailored to the specific problem under consideration. This characteristic makes it very flexible and amenable for hybridization with problem-specific heuristics and improvement procedures. In the following paragraphs, two examples are used to present sample instantiations of the ACO framework.

***ACO in Permutation Problems.*** As a first example, consider an optimization problem where the solution space is the set of permutations of the first $n$ positive integers. Examples of problems whose solutions can be modeled as permutations are the traveling salesperson problem (TSP) and job shop scheduling problems.

The space of all possible permutations of the first $n$ positive integers can be modeled as a fully connected graph,

$$G = (V, E, W)$$

where $V$ is the set of vertices $v_i$ with $i \in \{1, 2, \ldots, n\}$; $E$ is the set of edges connecting every pair of distinct vertices (i.e., $E$ does not have edges that directly link vertices with themselves); and $W$ is a matrix of weights (costs) associated with every edge in $E$. A permutation is then represented by a path in $G$ that connects the $n$ vertices as illustrated in Figure 1.

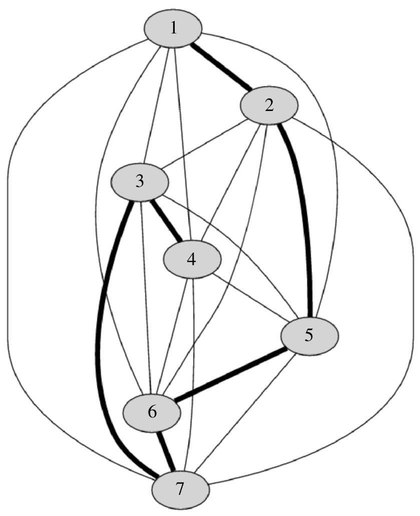

**Figure 1.** A fully connected graph representing the search space of all permutations of the first seven positive integers. The highlighted path represents the permutation $\{1, 2, 5, 6, 7, 3, 4\}$ or the reverse permutation, depending on the traversing direction.

For a better understanding of the search process performed by an ACO algorithm on a permutation search space, it is helpful to borrow language that would be used to describe a real biological system. In this context, the environment is the graph $G$. An ant is a solution-construction procedure that builds new solutions component by component. When an ant moves from one vertex to another, the solution associated with that ant is extended with an extra component, namely, the destination vertex.

Pheromones are numerical values associated with the edges in $E$. The values of the pheromone levels on the edges of the graph change over time as a result of two processes: *(i)* reinforcement, and *(ii)* evaporation. The specific operations associated with these processes differ in different ACO algorithms, but they all try to simulate the natural phenomena of pheromone-trail reinforcement of real ants (recall the shorter path discovery process enabled by pheromone laying and following), and evaporation, which is the process whereby the colony forgets previously visited areas. Thus, at any point in time, the pheromone distribution in the environment reflects the colony's learned features of the search space.

Thus, the basic process proceeds as follows. For each ant, a starting vertex is chosen at random. From there, ants move to another vertex not previously visited. Each ant probabilistically chooses the next vertex to visit. The probabilistic rule used in this step biases the choice toward edges with higher pheromone levels and higher heuristic values. When all the ants complete the process of creating full-length permutations, the pheromone-update phase begins by first decreasing all pheromone values on all the edges, and then providing higher reinforcement on edges that belong to superior solutions. Thus, when the next iteration begins, the choices of the ants tend to favor components that contribute toward better solutions.

***ACO in Binary Search Spaces.*** As a second example, consider an optimization problem where the solution space is better represented by a vector of binary components. Problems where the order of the components in a solution is irrelevant, like assignment and scheduling, fall in this category.

ACO algorithms can solve these kinds of problems by associating with each component of the solution vector a pheromone and a heuristic value. The solution-construction process consists in each ant starting with an empty solution and, at each step, deciding whether to add a solution component or not. This amounts to setting a decision variable to 1 or 0. In some cases, there are ways to determine when to stop testing whether to add components or not. In other cases, the ants need to test as many times as there are solution components. Thus, in binary optimization problems, it is possible that different ants may take different number of steps. As in the previous example, pheromones first evaporate and then being reinforced based on the components' association with higher quality solutions.

***Algorithmic Framework.*** A schematic view of the processes described above is shown in Algorithm 1. The two examples presented above explain the *Construct* and *Update* steps. This is because those two steps are necessary for the framework to work. However, a third optional step, identified as *Improve*, is typically performed in order to enhance the performance of ACO algorithms.

The daemon actions usually take the form of local-search procedures applied to the solutions produced by the ants. Local search exploits domain knowledge and its implementation departs from the basic SI principles, and particularly decentralization. The combined use of the basic ACO steps with local search produces highly performing, full-fledged ACO algorithms.

***ACO Instantiations.*** ACO defines a general framework and, therefore, it needs to be instantiated in order to be of practical use. An ACO instantiation is an actual implementation choice of at least two of the three steps in Algorithm 1. The two most commonly used ACO algorithms are described below.

**Algorithm 1.** *Ant Colony Optimization*

```
1: Initialize Pheromones.
2: while (not stopping condition) do
3:     Construct solutions // One solution per ant
4:     Improve solutions via daemon actions // Optional operations
5:     Update pheromones
6: end while
7: Report overall best solution
```

***Ant System.*** The ACO algorithm called ant system (AS) (22) actually predates the definition of the ACO framework itself, and therefore, while it is not a state-of-the-art algorithm, it is of historical importance. In AS, there are $m$ ants that, at each iteration, update the problem's pheromone values. The rule to update the $i$th pheromone value is

$$\tau_i = (1-\rho)\tau_i + \sum_{k=1}^{m} \Delta\tau_i^k \tag{1}$$

where $\tau_i$ is the pheromone value associated with the $i$th solution component; $\rho \in [0,1]$ is a parameter called pheromone evaporation rate; and $\Delta\tau_i^k$ is the quantity of pheromone laid on the $i$th component by the $k$th ant. The value of $\Delta\tau_i^k$ is a function of the quality of the solution that contains the $i$th component. If no solution contains the $i$th component then $\Delta\tau_i^k = 0$. Typically, the better a solution is, the higher is the amount of the deposited pheromone.

During solution construction, ants choose components via Monte Carlo simulation using the following rule,

$$p_{i|s} = \frac{\tau_i^{\alpha}\eta_i^{\beta}}{\sum_{j\in N(s)} \tau_j^{\alpha}\eta_j^{\beta}} \tag{2}$$

where $p_{i|s}$ is the probability of choosing the $i$th component given the partial solution $s$; $\alpha$ and $\beta$ are parameters that control the relative influence of pheromones and the heuristic information, $\eta$, during the solution-construction process; and $N(s)$ is a function that returns the indices of all the valid components that could follow from the current partial solution $s$.

***Max–Min Ant System.*** Max–min ant system (MMAS) (23) is an ACO algorithm that follows the original natural inspiration closer than AS. The first distinctive feature is the aggressive reinforcement of higher quality solutions by allowing only one of the best ants to deposit pheromone. The second distinctive feature is the explicit control of diversity by bounding the pheromone levels within minimum and maximum values (hence the name). Thus, the pheromone update takes the form

$$\tau_i = \begin{cases} \tau_{\min}, & \text{if } \tau_i \leqslant \tau_{\min}, \\ \tau_{\max}, & \text{if } \tau_i \geqslant \tau_{\max}, \\ (1-\rho)\tau_i + \Delta\tau_i^{\text{best}}, & \text{otherwise}, \end{cases} \tag{3}$$

where $\tau_{\min}$ and $\tau_{\max}$ are the minimum and maximum allowed pheromone values, while $\tau_i^{\text{best}}$ is the reinforcement produced by the ant associated with the best solution. The definition of the best solution may be subject also to design decisions. Common options are the iteration-best-solution, the best-so-far solution, and combinations of these two.

In MMAS, pheromone values are initialized to $\tau_{\max}$ to encourage exploration during the first iterations of the algorithm. MMAS also features extra algorithmic components such as reinitialization and pheromone-value smoothing in order to boost exploration. These techniques are typically used whenever the algorithm fails to make any progress for a prespecified number of iterations.

## 4. ALGORITHMS FOR CONTINUOUS PROBLEMS

The ample literature on swarm-based algorithms for real-valued problems impels us to make a selection among the most popular and widely used algorithms. The selection is based on essential aspects of the algorithms, namely, novelty, efficiency, and popularity. Thus, the Particle swarm optimization, artificial bee colony, cuckoo search, bacteria foraging optimization, and gravitational search algorithm are discussed in the following paragraphs. Although the main field of application of these algorithms is continuous optimization, a number of variants and modifications have been proposed also for discrete or semi-continuous problems. However, such approaches are not discussed in this article.

In the forthcoming sections, the optimization problem is assumed to be of the form,

$$\min_{x \in \mathcal{S} \subset \mathbb{R}^n} f(x) \tag{4}$$

where, for simplicity reasons, the feasible set $\mathcal{S}$ is defined as an $n$-dimensional hyperbox,

$$\mathcal{S} = \left[s_1^{\min}, s_1^{\max}\right] \times \cdots \times \left[s_n^{\min}, s_n^{\max}\right]$$

with $s_i^{\min}, s_i^{\max} \in \mathbb{R}$, being the lower and upper bound of the $i$th directional component, respectively.

### 4.1. Particle Swarm Optimization

Particle swarm optimization (PSO) is the most recognizable swarm-based optimization algorithm for continuous problems, originally introduced by Eberhart and Kennedy (24). Its development originated in simulations of swarming search agents. The resultant models exhibited similarities with models from the field of particle physics from which, basic ideas and relevant notation were borrowed (4,25).

The main rules implemented by PSO are the neighbor velocity matching and acceleration by distance, which were shown to produce swarming behavior in early simulations (4). The first versions of the algorithm were described through simple difference equations. These equations promoted exploration based on randomized position shifts of the search agents toward aggregated directions defined through difference vectors. These vectors constitute a form of cooperation among the agents in terms of information exchange. In the following paragraphs, the so called canonical particle swarm (26) model is presented. This model has constituted the origin for numerous other PSO variants and occupies a salient position in the state-of-the-art of swarm-based optimization methods.

Let us consider the global optimization problem of equation 4 and assume a group of $N$ search points (agents) that iteratively probe the search space $\mathcal{S}$. This group is called a *swarm* and henceforth denoted as

$$P^{(t)} = \left\{x_1^{(t)}, \ldots, x_N^{(t)}\right\}$$

where $t$ stands for the iteration index. For simplicity reasons, let us also define as

$$I = \{1, 2, \ldots, N\}, \qquad J = \{1, 2, \ldots, n\} \tag{5}$$

the sets of particles' and components' indices, respectively. Each search point is a candidate solution of the problem, i.e., it constitutes an element of $\mathcal{S}$,

$$x_i^{(t)} = \left(x_{i1}^{(t)}, \ldots, x_{in}^{(t)}\right) \in \mathcal{S}, \quad i \in I$$

and it is called a particle. The number $N$ of particles is called the swarm size and it is usually a user-defined parameter.

Each particle is randomly initialized in $\mathcal{S}$ according to a probability distribution over the whole search space. The uniform distribution is regularly adopted for this purpose, that is,

$$x_{ij}^{(0)} \sim \mathcal{U}\left(\left[s_j^{\min}, s_j^{\max}\right]\right), \quad i \in I, \quad j \in J \tag{6}$$

After initialization, the particles are allowed to probe the search space $\mathcal{S}$ by iteratively updating their positions. For this purpose, appropriate position shifts,

$$v_i^{(t)} = \left(v_{i1}^{(t)}, \ldots, v_{in}^{(t)}\right), \quad i \in I$$

are used at each iteration. These position shifts are called the velocities and they are also randomly and uniformly initialized,

$$v_{ij}^{(0)} \sim \mathcal{U}\left(\left[-v_j^{\max}, v_j^{\max}\right]\right), \quad i \in I, \quad j \in J$$

where $v_j^{\max}$ is a maximum value, usually defined as a fraction of the search space in the specific direction, that is,

$$v_j^{\max} = \alpha_j \left(s_j^{\max} - s_j^{\min}\right), \quad \alpha_j \in (0, 1], \quad j \in J$$

Higher values of $\alpha_j$ allow bigger steps to be taken by the particles, thereby promoting global exploration. On the other hand, smaller values are beneficial for local exploration. Of course, longer steps can make particles to leave the feasible set and, therefore, it is necessary to enforce bound constraints in an appropriate way (27).

During its journey, each particle retains in external memory the best position it has ever visited,

$$b_i^{(t)} = \left(b_{i1}^{(t)}, \ldots, b_{in}^{(t)}\right) \in \mathcal{S}, \quad i \in I$$

with,

$$b_i^{(t)} = x_i^{(\tau^*)}$$

$$\tau^* = \underset{\tau \in \{0,1,\ldots,t\}}{\arg\min}\, f\left(x_i^{(\tau)}\right)$$

This information is exploited to bias the particle's move toward promising regions of the search space that it has already visited, aiming at discovering even better candidate solutions.

In addition to its findings, each particle cooperates with other particles that form its neighborhood. In essence, a neighborhood of a particle consists of a number of other particles of the swarm that apprise it of their best positions. Neighborhoods can be arbitrarily defined in different sets, such as the search space $\mathcal{S}$ or the particles' indices set $I$. In order to promote diversity in the swarm and avoid the rapid formation of particle clusters, the set $I$ is more

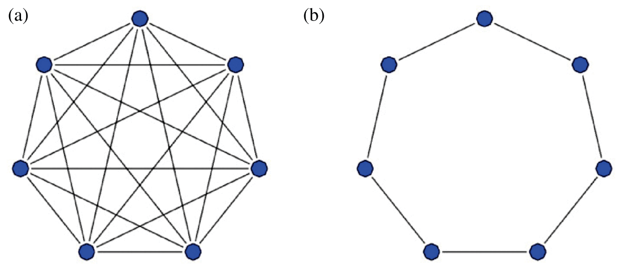

**Figure 2.** The fully connected (a) and the ring (b) neighborhood topologies.

commonly used for this purpose. Thus, neighborhoods are defined as subsets of $I$, which determine the communication channels in the swarm.

It is highly convenient to represent neighborhoods as graphs consisting of nodes (particles) and edges (communication channels). Such graphs define different neighborhood topologies. Two popular neighborhood topologies, namely, the fully connected and the ring, are depicted in Figure 2. Based on the description above, the neighborhood of the $i$th particle according to the ring topology is defined as

$$\mathcal{N}_i = \{i-r,\ldots,i-1,i,i+1,\ldots,i+r\} \subseteq I$$

where $r \leqslant N/2$ is a user-defined parameter called the neighborhood's radius, which determines the influence of the swarm on each particle. The indices of the particles in cyclic topologies such as the described ones are assumed to recycle at both ends, that is, index 1 follows after $N$. Obviously, the fully connected topology can be considered as a special case of the ring topology with $\mathcal{N}_i = I$ for all $i \in I$.

Let $g_{it}$ be the index of the best position with the lowest function value in the neighborhood $\mathcal{N}_i$ at iteration $t$, that is,

$$g_{it} = \arg\min_{j\in\mathcal{N}_i} f\left(b_j^{(t)}\right) \tag{7}$$

Then, the velocity and position update of the $i$th particle is given as follows,

$$v_{ij}^{(t+1)} = w\, v_{ij}^{(t)} + c_1\,\mathcal{R}_1\left(b_{ij}^{(t)} - x_{ij}^{(t)}\right) + c_2\,\mathcal{R}_2\left(b_{g_{it},j}^{(t)} - x_{ij}^{(t)}\right) \tag{8}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)} \tag{9}$$

where $i \in I$, $j \in J$; $\mathcal{R}_1$ and $\mathcal{R}_2$ are samples drawn from a uniform distribution in the range $[0,1]$ (different for each $i$, $j$, and $t$); $c_1, c_2 \geqslant 0$, are acceleration parameters; and $w \geqslant 0$ is a parameter called the inertia weight, which regulates the contribution of the previous velocity to the current one.

A number of interesting observations can be made in equations 8 and 9. First, we see that PSO updates the velocities and the particles component-wisely, taking into consideration the previous moving direction (inertia term). In order to avoid the uncontrollable increase of the magnitude of the velocities, also known as the swarm explosion effect, the inertia weight $w$ is used. Appropriate fixed or dynamic values of $w$, typically in the range $[0,1]$, modulate the magnitude of the velocities and gradually lead the swarm to convergence. The stability analysis of PSO derived in (28) suggested the following algebraically equivalent form of equation 8,

$$v_{ij}^{(t+1)} = \chi\left[v_{ij}^{(t)} + c_1\,\mathcal{R}_1\left(b_{ij}^{(t)} - x_{ij}^{(t)}\right) + c_2\,\mathcal{R}_2\left(b_{g_{it},j}^{(t)} - x_{ij}^{(t)}\right)\right]$$

where $\chi$ is a parameter called the constriction coefficient, which plays the role of the inertia weight. For this case, the theoretical analysis suggests the default parameter values,

$$\chi = 0.729, \qquad c_1 = c_2 = 2.05$$

which correspond to $w = 0.729$, $c_1 = c_2 = 1.494$, in equation 8. Nevertheless, dynamic adaptation of the parameters is frequently used (29).

The update of the best positions succeeds the update and evaluation of the particles as follows:

$$b_i^{(t+1)} = \begin{cases} x_i^{(t+1)}, & \text{if } f_i^{(t+1)} \leqslant f_{b_i}^{(t)}, \\ b_i^{(t)}, & \text{otherwise,} \end{cases} \tag{10}$$

where $i \in I$ and

$$f_i^{(t+1)} = f\left(x_i^{(t+1)}\right), \qquad f_{b_i}^{(t)} = f\left(b_i^{(t)}\right)$$

Then, the best particles in the neighborhoods are determined anew using equation 7, and the algorithm proceeds to the next iteration. Execution stops as soon as user-defined stopping conditions, such as reaching a prespecified error goal or a maximum number of functions evaluations, are fulfilled. The algorithm is summarized in Algorithm 2.

PSO inherently integrates the basic properties of swarm-based optimization algorithms described in Section 2. The swarm does not assume any central control.

**Algorithm 2.** *Canonical Particle Swarm Optimization*

```
1: Initialize swarm, velocities, and best positions
2: while (not stopping condition) do
3:     for ($i$ = 1 … $N$) do
4:         Find neighborhood's $\mathcal{N}_i$ best as in equation 7
5:         Update velocity and position with equations 8 and 9
6:     end for
7:     for ($i$ = 1 … $N$) do
8:         Update best positions with equation 10
9:     end for
10: end while
11: Report overall best solution
```

Computation is completely distributed and can be parallelized either in swarm, particle, or component level (relevant software can be found in (30)). Moreover, there is no crucial dependence of the swarm on specific particles, which is essential for retaining robustness. In fact, recent developments suggested that deliberate loss of information can be even beneficial for the swarm (18).

The emergent behavior of the swarm is attributed to the cooperation among the particles in terms of information exchange. In PSO, communication is direct since the particles share their findings (best positions) directly with other particles. Adaptability can be achieved through parameter adaptation. For instance, decreasing the inertia weight from higher values toward zero changes the dynamic of the algorithm from exploration (global search) to exploitation (local search). Finally, the stochasticity induced by the stochastic terms in velocity update ensures diverse response patterns of the particles.

Comprehensive presentations of PSO as well as discussions on its crucial aspects such as parameter tuning, constraint handling, performance enhancement techniques, and sophisticated variants can be found in dedicated books such as (29,31–36), as well as in survey articles such as (37–42). PSO has been imitated to great extent in other swarm-based algorithms such as the firefly algorithm and the bat algorithm (43), and the glowworm swarm optimization (44).

## 4.2. Artificial Bee Colony

Artificial bee colony (ABC) is a swarm-based algorithm that follows the physical analogue of honey bees. It was originally introduced by Karaboga (45). The algorithm is based on relevant models that describe the foraging behavior of honey bees, which is characterized by colony division. Specifically, the swarm consists of three types of honey bees (search agents), namely, the employed bees, the onlooker bees, and the scout bees. The first type defines the current food sources of the bees, that is, positions in the search space. The second type assesses the discovered food sources and probabilistically selects some of them to intensify search. The last type performs random search.

Putting it formally, a swarm of $N$ scout bees is randomly initialized according to equation 6 within the search space of the general global optimization problem of equation 4. Also, let $I$ and $J$ be the two sets defined in equation 5 (our notation deliberately imitates PSO to emphasize the resemblance among the two algorithms). A fitness value is computed for each position as follows:

$$F_{x_i}^{(t)} = \begin{cases} \frac{1}{\left(1+f_i^{(t)}\right)}, & \text{if } f_i^{(t)} \geqslant 0, \\ 1 + \left|f_i^{(t)}\right|, & \text{otherwise,} \end{cases} \tag{11}$$

where $f_i^{(t)} = f\left(x_i^{(t)}\right)$ (initially, $t = 0$). Naturally, fitness assignment can be appropriately modified if necessary.

Then, the employed bees exploit the information provided by the scout bees and perform local search around the memorized food sources by producing new vectors through differences of the form,

$$v_{ij}^{(t)} = x_{ij}^{(t)} + \mathcal{R}\left(x_{ij}^{(t)} - x_{kj}^{(t)}\right) \tag{12}$$

where $k \neq i$ is a randomly selected index from $I$; $j$ is randomly selected from $J$; and $\mathcal{R} \sim \mathcal{U}([-\alpha, \alpha])$ is a random variable with $\alpha > 0$ being a user-defined parameter (typically, $\alpha = 1$).

For each new position $v_i$, its fitness value is computed according to equation 11. Then, it competes with the corresponding $x_i$ as follows:

$$x_i^{(t)} = \begin{cases} v_i^{(t)}, & \text{if } F_{v_i}^{(t)} > F_{x_i}^{(t)}, \\ x_i^{(t)}, & \text{otherwise.} \end{cases} \tag{13}$$

After that, onlooker bees take action. Each one adopts one of the available positions $x_i$ by conducting fitness-based probabilistic selection with replacement, also known as roulette wheel selection. The selection probabilities of the positions $x_i$ are defined as

$$\rho_i^{(t)} = \frac{F_{x_i}^{(t)}}{\sum_{k=1}^{N} F_{x_i}^{(t)}}$$

Obviously, positions of higher fitness are selected more frequently by the onlooker bees.

Each onlooker bee conducts local search around its selected position according to equation 12 and, again, the produced new positions compete with the existing ones according to equation 13. This way, the ABC algorithm intensifies search around the best discovered solutions, thereby promoting exploitation.

Each existing position $x_i$ that could not be improved for a maximum (user-defined) number $t_{\text{ab}}$ of iterations is abandoned. The corresponding employed bee becomes a scout bee and randomly reinitializes its position in the search space. This completes a full iteration of the ABC algorithm. The best solution found by the algorithm is stored separately and updated at the end of each iteration.

Stripping the algorithm from its natural analogue, it can be described in terms of a multistart random local search approach, where the collective behavior emerges from the implied promotion of intense local search around the most promising positions. The main prerequisite for this behavior is the cooperation in terms of information sharing among the bees (agents).

ABC has been studied in various works. A comprehensive presentation of the algorithm can be found in (46), while a recent survey can be found in (47). ABC is also closely related to other approaches based on the bee colony analogue, such as the bees algorithm (48,49).

## 4.3. Cuckoo Search

Cuckoo search (CS) was introduced by Yang and Deb (50). Inspiration sprang from the parasitic breeding behavior of cuckoo birds, which lay their eggs in nests of other bird species. Although host birds often unmask the fraud and destroy the outlander's eggs or abandon their nests, some special cuckoo species can deceive the host birds by imitating their egg color and texture. The hatched cuckoo birds instinctively destroy the rest of the eggs in the nest of the host birds, thereby claiming larger portions of the provided food.

The natural paradigm of cuckoo's breeding behavior is roughly modeled in the CS algorithm. Let the global optimization problem and quantities defined in equations 4 and 5. Initially, $N$ host nests (positions) are randomly generated in the search space and their objective values,

$$f_i^{(t)} = f\left(x_i^{(t)}\right), \quad i \in I$$

are computed (initially, $t = 0$). Then, a new position (nest) is produced using the following scheme (51),

$$x_i^{(t+1)} = \begin{cases} x_i^{(t)} + \alpha\, s\left(x_j^{(t)} - x_k^{(t)}\right), & \text{if } \mathcal{R} \leqslant \rho_\alpha, \\ x_i^{(t)} + \alpha\, L(s, \lambda), & \text{otherwise}, \end{cases}$$

where $s > 0$ is the step size; $\alpha > 0$ is a scaling factor; $x_j^{(t)}$ and $x_k^{(t)}$ are two positions randomly selected among the existing ones; $\rho_\alpha \in [0, 1]$ is a user-defined probability of switching from local to global random walk; and,

$$L(s, \lambda) = \frac{\lambda\, \Gamma(\lambda)\, \sin(\pi\lambda/2)}{\pi\, s^{1+\lambda}}$$

where $\Gamma(.)$ is the Gamma function. The scaling factor $\alpha$ usually assumes values between 1% and 10% of the characteristic scale of the problem.

The new position $x_i^{(t+1)}$ is assessed and an existing position $x_j^{(t)}$ is randomly selected to compete with it. If the new position has better value than the selected one, it replaces it in the group of nests. Also, a user-defined percentage, $\beta \in [0, 1]$, of the worst positions is abandoned, replaced by new randomly selected positions. This phase completes the algorithm's iteration and imitates the corresponding nest dereliction in the physical analogue.

CS has gained increasing popularity but it has also received criticism regarding its actual relevance with the physical analogue, which seems somehow unjustified. Indeed, CS seems to be based solely on the concept of random walk, producing new positions according to a Markov chain type procedure with special transition probabilities. The cuckoo breeding paradigm seems to be useful only to fit these mathematical procedures in the modern trend of nature-inspired algorithms. Nevertheless, it appears that CS is a useful swarm-based method with interesting applications. Recent developments, implementation details, as well as a brief literature review of the algorithm can be found in (52).

### 4.4. Bacteria Foraging Optimization

Bacteria foraging optimization (BFO) was originally introduced by Passino (53). It was inspired by the collective foraging behavior of bacteria such as *Escherichia coli*, which is guided by chemotaxis based on nutrient gradients in their surrounding environment. Besides the direction of the highest nutrients concentration, bacteria also take into consideration information communicated by other bacteria through chemical signal molecules. This form of cooperation is highly responsible for their swarming behavior.

Let $N$ denote the number of artificial bacteria (search agents) that are used for the solution of the $n$-dimensional global optimization problem of equation 4. According to the physical analogue, there are three nested phases of bacterial behavior, namely, chemotaxis, reproduction, and elimination dispersal, which are also simulated in the BFO algorithm. We henceforth use the iteration counters $t_c$, $t_r$, and $t_e$, for the corresponding phases above. Thus, the bacteria swarm can be represented as

$$P^{(t_c,t_r,t_e)} = \left\{x_1^{(t_c,t_r,t_e)}, \ldots, x_N^{(t_c,t_r,t_e)}\right\}$$

Similarly to the previous algorithms, the bacteria are randomly initialized in the search space as follows:

$$x_{ij}^{(0,0,0)} \sim \mathcal{U}\left(\left[s_j^{\min}, s_j^{\max}\right]\right), \quad i \in I, \quad j \in J$$

where $I$ and $J$ are the sets of indices defined in equation 5, and they are evaluated with the original objective function of equation 4,

$$f_i^{(t_c,t_r,t_e)} = f\left(x_i^{(t_c,t_r,t_e)}\right), \quad i \in I$$

Then, the main BFO procedure starts with each bacterium undergoing a chemotaxis step that involves its indirect communication with the rest of the swarm. Specifically, for the $i$th bacterium, a penalized objective value is computed (54),

$$F_i^{(t_c,t_r,t_e)} = f_i^{(t_c,t_r,t_e)} + g_i^{(t_c,t_r,t_e)} \tag{14}$$

where,

$$g_i^{(t_c,t_r,t_e)} = g\left(x_i^{(t_c,t_r,t_e)}\right) = \sum_{j=1}^{N}\left[-d_{\text{att}}\, \exp\left(-w_{\text{att}}\, \Delta_{i,j}^{(t_c,t_r,t_e)}\right)\right] + \sum_{j=1}^{N}\left[-d_{\text{rep}}\, \exp\left(-w_{\text{rep}}\, \Delta_{i,j}^{(t_c,t_r,t_e)}\right)\right] \tag{15}$$

is called the cell-to-cell attractant effect, and,

$$\Delta_{i,j}^{(t_c,t_r,t_e)} = \left\| x_i^{(t_c,t_r,t_e)} - x_j^{(t_c,t_r,t_e)} \right\|^2$$

is the squared Euclidean distance between two bacteria. The parameters $d_{\text{att}}$, $w_{\text{att}}$, $d_{\text{rep}}$, and $w_{\text{rep}}$, determine the magnitude of attraction or repulsion between the bacteria. These parameters are user-defined, although their proper setting is not an easy task (55).

After that, tumbling takes place where a new position is generated for the $i$th bacterium as follows (54):

$$x_i^{(t_c+1,t_r,t_e)} = x_i^{(t_c,t_r,t_e)} + c_i\, \frac{R_i}{\|R_i\|} \tag{16}$$

where $c_i$ is a user-defined step size (typically smaller than 1), and $R_i$ is an $n$-dimensional random vector with components uniformly distributed in the range $[-1, 1]$, that is,

$$R_i = \left(R_{i1}, R_{i2}, \ldots, R_{in}\right), \quad R_{ij} \sim \mathcal{U}([-1, 1])$$

while $\|R_i\|$ denotes the measure of $R_i$. The new position is also evaluated according to

$$F_i^{(t_c+1,t_r,t_e)} = f_i^{(t_c+1,t_r,t_e)} + g_i^{(t_c+1,t_r,t_e)}$$

If the new value is better than the one of equation 14, then the corresponding position $x_i^{(t_c+1,t_r,t_e)}$ replaces the existing one $x_i^{(t_c,t_r,t_e)}$. The algorithm continues the production of new positions according to equation 16 with the same $R_i$ for a user-defined maximum number of improving steps, $q_{\max}$, or until an inferior position is produced, always preserving the best found position and its value. This procedure is called the swim of the bacterium and completes its chemotaxis step. The algorithm continues with the next bacterium, until the whole swarm has been processed.

The chemotaxis loop is repeated for a user-defined number of iterations, $t_c^{\max}$, for each bacterium. When it finishes, reproduction takes place. Specifically, the cumulative value obtained from chemotaxis, also called *health*, is computed for each bacterium (54),

$$F_i^{\text{health}} = \sum_{j=1}^{t_c^{\max}} F_i^{(t_c+j,t_r,t_e)}, \quad i \in I \tag{17}$$

which is the equivalent of the amount of nutrients consumed by the bacterium during its lifetime in the physical analogue. The bacteria are then sorted according to their health values, with lower values denoting better bacteria (in minimization case). A user-defined number, $M \leqslant N/2$, of the worst bacteria is eliminated from the swarm, and the $M$ best ones are replicated in the swarm claiming the empty positions. This completes the $(t_r + 1)$th reproduction step and the swarm starts a new chemotaxis loop.

When $t_r$ reaches a user-defined maximum number of reproduction cycles, $t_r^{\max}$, the $(t_e + 1)$th elimination-dispersal phase takes place. In this phase, each bacterium is eliminated with a fixed probability $\rho_e$, and its position is occupied by a new, randomly positioned bacterium. Then, the whole cycle of reproduction and chemotaxis is repeated again. When a maximum number $t_e^{\max}$ of elimination-dispersal cycles is reached, the algorithm is terminated reporting the best position it has ever visited.

BFO resembles gradient-descent combined with stochastic directional search in a sophisticated scheme. Swarming behavior emerges as a consequence of the bacteria interaction through the attraction–repulsion forces that are indirectly imposed by penalizing the objective values of the bacteria. Stochasticity ensures the diverse response of the swarm.

A theoretical analysis of a simplified system of two bacteria can be found in (54). Although setting its large number of parameters can be a laborious task, BFO has gained increasing popularity with applications appearing in various fields such as filtering, learning in neural networks, pattern recognition, and scheduling (54).

## 4.5. Gravitational Search Algorithm

Gravitational search algorithm (GSA) was introduced by Rashedi et al. (56) as an agent-based optimization algorithm inspired by the behavior of body masses under gravitational forces. Specifically, the search agents are considered as objects with masses determined according to their quality. The agents move in the search space subject to gravitational forces that attract them toward heavier masses, that is, agents of better quality, thereby imitating the corresponding Newtonian laws of gravity and motion.

These laws dictate that each mass attracts other masses with a gravitational force that is proportional to the product of their masses and inversely proportional to their distance. Also, the velocity of a mass is equal to the sum of (a fraction of) its previous velocity and its acceleration (56).

Putting it formally, let $N$ be the number of search agents, forming a swarm of masses,

$$P^{(t)} = \left\{ x_1^{(t)}, \ldots, x_N^{(t)} \right\}$$

at the current iteration $t$ of the algorithm. The position of each agent is randomly initialized according to equation 6 in the $n$-dimensional search space, and $I$ and $J$ are the sets of indices defined in equation 5. Besides its position, each agent also assumes an inertial mass MI, as well as an active and a passive gravitational mass, denoted as MA and MP, respectively. These masses are determined according to a user-defined fitness function, which takes into consideration the quality of the agent, that is, its objective value, such that higher masses correspond to agents with smaller objective values in minimization case (further details are given below).

The aggregate force imposed on the $i$th search agent is defined as an $n$-dimensional vector (56),

$$F_i^{(t)} = \left( F_{i1}^{(t)}, \ldots, F_{in}^{(t)} \right)$$

with components,

$$F_{ij}^{(t)} = \sum_{k\in K} \mathcal{R}_j \, F_{[k\leftrightarrow i],j}^{(t)}, \quad j \in J, \quad K \subseteq I \setminus \{i\} \tag{18}$$

where $\mathcal{R}_j$ is a random number uniformly distributed in the range $[0, 1]$, and,

$$F_{[k\leftrightarrow i],j}^{(t)} = G^{(t)} \frac{\mathrm{MP}_i^{(t)} \, \mathrm{MA}_k^{(t)}}{R_{ik} + \varepsilon} \left( x_{kj}^{(t)} - x_{ij}^{(t)} \right), \quad j \in J, \quad k \in I \setminus \{i\}$$

with $\mathrm{MP}_i^{(t)}$ and $\mathrm{MA}_k^{(t)}$ denoting the passive and active masses of agents $i$ and $k$, respectively, at iteration $t$. The parameter $G^{(t)}$ is a gravitational "constant", that it is time-decreasing; $\varepsilon > 0$ is a small fixed value that prevents division by zero; and $R_{ik}$ is the Euclidean distance between the two agents.

The set $K \subseteq I$ includes the agents that affect the move of the $i$th agent. In the simplest case, $K$ can be the whole set $I$, that is, every mass affects all other masses. This is an exploration-oriented version that equips the algorithm with the necessary search diversity in its early iterations. However, as time passes, more exploitation-oriented steps are desirable. This can be achieved by gradually modifying $K$ to include only best-performing agents as proposed in (56).

The acceleration of the $i$th agent is then defined as the $n$-dimensional vector (56),

$$\alpha_i^{(t)} = \left(\alpha_{i1}^{(t)}, \ldots, \alpha_{in}^{(t)}\right)$$

with components,

$$\alpha_{ij}^{(t)} = \frac{F_{ij}^{(t)}}{\mathrm{MI}_i^{(t)}}, \quad j \in J$$

where $\mathrm{MI}_i$ is the inertial mass of the $i$th agent. According to the law of motion, the new positions of the search agents in the next iteration of the algorithm are given by

$$v_{ij}^{(t+1)} = \mathcal{R}_i\, v_{ij}^{(t)} + \alpha_{ij}^{(t)} \tag{19}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)} \tag{20}$$

where $i \in I$, $j \in J$, and $\mathcal{R}_i$ is a random number uniformly distributed in $[0, 1]$. The procedure is iteratively repeated until a user-defined stopping condition is satisfied.

According to (56), the inertial, active, and passive gravitational masses are assumed to be equal and determined as follows:

$$\mathrm{MI}_i^{(t)} = \mathrm{MA}_i^{(t)} = \mathrm{MP}_i^{(t)} = M_i^{(t)} = \frac{m_i^{(t)}}{\sum_{j=1}^{N} m_j^{(t)}}, \quad i \in I$$

where,

$$m_i^{(t)} = \frac{f_i^{(t)} - f_{\text{worst}}^{(t)}}{f_{\text{best}}^{(t)} - f_{\text{worst}}^{(t)}}$$

with $f_i^{(t)}$ denoting the objective value of $x_i^{(t)}$, and,

$$f_{\text{best}}^{(t)} = \min_{k \in I} \left\{ f_k^{(t)} \right\}, \qquad f_{\text{worst}}^{(t)} = \max_{k \in I} \left\{ f_k^{(t)} \right\}$$

are the best and worst objective value of the swarm at iteration $t$, respectively.

GSA integrates the typical elements of swarm-based optimization algorithms. Its emergent swarming behavior emanates from the direct cooperation among the agents through their attraction forces. The use of weighted differences as well as its general context brings GSA closer to PSO than other swarm-based approaches. It has common inspiration origins also with other force-based algorithms such as the central force optimization algorithm (57).

GSA has gained increasing popularity since its development. A number of relevant applications are reported in (58). However, it has also received criticism regarding its actual relevance to the physical analogue (59). Nevertheless, it appears to be a relatively simple algorithm with promising performance.

## 5. CONCLUSIONS

The basic properties of swarm-based optimization algorithms were exposed and a number of popular approaches were presented. The main purpose of the article was to highlight the main qualities that produce emergent collective intelligence and to identify such virtues in a number of widely used algorithms for discrete and continuous optimization problems.

Naturally, the ample literature available prohibits the thorough presentation of the whole research field known as swarm intelligence, where the studied algorithms belong to. Over and above, this was not the main goal of the authors. Instead, our intention was to offer a starting point for the study of the fascinating field of swarm-based optimization by presenting some selected algorithmic approaches that prevailed in the field during the past two decades.

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ABC | artificial bee colony |
| ACO | ant colony optimization |
| AS | ant system |
| BFO | bacteria foraging optimization |
| CS | cuckoo search |
| EA | evolutionary algorithm |
| GSA | gravitational search algorithm |
| MMAS | max–min ant system |
| PSO | particle swarm optimization |
| SI | swarm intelligence |

## BIBLIOGRAPHY

1. S. Haykin. *Neural Networks and Learning Machines*. Pearson International Edition, 2009.
2. J. Schmidhuber. *Neural Networks*, **2015**, *61*, pp 85–117.
3. E. Bonabeau, G. Theraulaz, and M. Dorigo. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.
4. J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Academic Press, 2001.
5. M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer, 2009.
6. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2015.
7. C. A. Floudas and P. M. Pardalos (Eds). *Encyclopedia of Optimization*. Springer, 2009.
8. E. Sanchez, G. Squillero, and A. Tonda. *Industrial Applications of Evolutionary Algorithms*. Springer, 2012.
9. G. Luque and E. Alba. *Parallel Genetic Algorithms: Theory and Real World Applications*. Springer, 2011.
10. E. Bonabeau. *Artif. Life*, **1999**, *5*(2), pp 95–96.
11. A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2005.
12. B. Hölldobler and E. O. Wilson. *The Superorganism: The Beauty, Elegance, and Strangeness of Insect Societies*. W.W. Norton & Company, 2008.
13. C. W. Reynolds. *ACM SIGGRAPH Comput. Graph.*, **1987**, *21*(4), pp 25–34.
14. N. C. Darnton, L. Turner, S. Rojevsky, and H. C. Berg. *Biophys. J.*, **2010**, *98*, pp 2082–2090.
15. G. Majno and I. Joris. *Cells, Tissues, and Disease: Principles of General Pathology*. Wiley-Blackwell, 1996.
16. D. Helbing and P. Molnar. *Phys. Rev. E*, **1995**, *51*, pp 4282–4286.
17. L. B. Rosenberg. In *Proc. Swarm/Human Blended Intelligence Workshop (SHBI) 2015*. IEEE, 2015.

18. C. Voglis, K. E. Parsopoulos, and I. E. Lagaris. *Soft Comput.*, **2012**, *16*(8), pp 1373–1392.
19. M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.
20. S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels. *Naturwissenschaften*, **1989**, *76*(12), pp 579–581.
21. M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.
22. M. Dorigo, V. Maniezzo, and A. Colorni. *IEEE Trans. Syst. Man Cybern. B*, **1996**, *26*(1), pp 29–41.
23. T. Stützle and H. H. Hoos. *Future Gener. Comput. Syst.*, **2000**, *16*(8), pp 889–914.
24. R. C. Eberhart and J. Kennedy. In *Proc. 6th Symposium on Micro Machine and Human Science*, IEEE Service Center, Piscataway, NJ, 1995, pp 39–43.
25. A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2006.
26. J. Kennedy. In *Encyclopedia of Machine Learning*, Sammut, C. ; Webb, G., Eds; Springer, 2010; pp 760–766.
27. T. Liao, D. Molina, M. A. Montes de Oca, and T. Stützle. *Evol. Comput.*, **2014**, *22*(2), pp 351–359.
28. M. Clerc and J. Kennedy. *IEEE Trans. Evol. Comput.*, **2002**, *6*(1), pp 58–73.
29. K. E. Parsopoulos and M. N. Vrahatis. *Particle Swarm Optimization and Intelligence: Advances and Applications*. Information Science Publishing (IGI Global), Hershey, PA, USA, 2010.
30. C. Voglis, P. E. Hadjidoukas, K. E. Parsopoulos, D. G. Papageorgiou, I. E. Lagaris, and M. N. Vrahatis. *Comput. Phys. Commun.*, **2015**, *197*, pp 190–211.
31. M. Clerc. *Particle Swarm Optimization*. ISTE Ltd, 2006.
32. S. Kiranyaz, T. Ince, and M. Gabbouj. *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Springer-Verlag, Berlin, 2014.
33. A. Kishk. *Particle Swarm Optimizaton: A Physics-Based Approach*. Morgan and Claypool Publishers, 2008.
34. A. E. Olsson, Ed., *Particle Swarm Optimization: Theory, Techniques and Applications*. Nova Science Pub Inc., 2011.
35. D. Parasuraman. *Handbook of Particle Swarm Optimization: Concepts, Principles & Applications*. Auris Reference, 2012.
36. J. Sun, C.-H. Lai, and X.-J. Wu. *Particle Swarm Optimisation: Classical and Quantum Perspectives*. CRC Press, 2011.
37. K. E. Parsopoulos. In *Handbook of Heuristics*; Resende, M., Marti, R., and Pardalos, P., Eds.; Springer, 2017.
38. A. Banks, J. Vincent, and C. Anyakoha. *Nat. Comput.*, **2007**, *6*(4), pp 467–484.
39. A. Banks, J. Vincent, and C. Anyakoha. *Nat. Comput.*, **2008**, *7*(1), pp 109–124.
40. R. Poli, J. Kennedy, and T. Blackwell. *Swarm Intell.*, **2007**, *1*(1), pp 33–57.
41. A. Rezaee Jordehi and J. Jasni. *J. Exp. Theor. Artif. Intell.*, **2013**, *25*(4), pp 527–542.
42. R. Thangaraj, M. Pant, A. Abraham, and P. Bouvry. *Appl. Math. Comput.*, **2011**, *217*(12), pp 5208–5226.
43. X.-S. Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2010.
44. K. N. Krishnanand and D. Ghose. *Swarm Intell.*, **2009**, *3*(2), pp 87–124,
45. D. Karaboga. An Idea Based on Honey Bee Swarm for Numerical Optimization. Technical Report, Erciyes University, Department of Computer Engineering, Technical Report TR06, 2005.
46. D. Karaboga and B. Basturk. *J. Glob. Optim*, **2007**, *39*(3), pp 459–471.
47. D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga. *Artif. Intell. Rev.*, **2014**, *42*(1), pp 21–57.
48. D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi. The Bees Algorithm. Technical report, Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.
49. D. T. Pham and M. Castellani. *Soft Comput.*, **2014**, *18*(5), pp 871–903.
50. X.-S. Yang and S. Deb. In *Proc. of the World Congress on Nature and Biologically Inspired Computing (NaBIC 2009)*, IEEE Publications, pp 210–214.
51. X.-S. Yang and S. Deb. *Neural Comput. Appl.*, **2014**, *24*(1), pp, 169–174.
52. X.-S. Yang, Ed. *Cuckoo Search and Firefly Algorithm: Theory and Applications*. Springer, 2014.
53. K. M. Passino. *IEEE Control Syst. Mag.*, **2002**, *22*(3), pp 52–67.
54. S. Das, A. Biswas, S. Dasgupta, and A. Abraham. In *Foundations of Computational Intelligence Volume 3*; Abraham, A.; Hassanien, A.-E.; Siarry, P.; Engelbrecht, A., Eds.; Springer, 2009.
55. Y. Liu and K. M. Passino. *J. Optim. Theory Appl.*, **2002**, *115*(3), pp 603–628.
56. E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi. *Inform. Sci.*, **2009**, *179*(13), pp 2232–2248.
57. R. A. Formato. *Prog. Electromagn. Res.*, **2007**, *77*, pp 425–491.
58. N. Mohd Sabri, M. Puteh, and M. Rusop Mahmood. *Int. J. Adv. Soft Comput. Appl.*, **2013**, *5*(3), pp 1–39.
59. M. Gauci, T. J. Dodd, and R. Groß. *Nat. Comput.*, **2012**, *11*(4), pp 719–720.

Konstantinos E. Parsopoulos

University of Ioannina,
Ioannina, Greece

Marco A. Montes de Oca

Clypd Inc.,
Somerville, MA, USA