# Chapter 2
# Bio-inspired Computation Algorithms


**Pei Li and Haibin Duan**

**Abstract** Bio-inspired computation is the use of computers to model the living phenomena and simultaneously the study of life to improve the usage of computers. Swarm behaviors in animal groups such as bird flocks, bees, ants, fish schools, and sheep herds, as well as insects like mosquitoes, ants, and bees, often exhibit incredible abilities to solve complex problems that seem far beyond their capabilities. This chapter mainly focuses on the biological inspiration, principle, and implementation procedures of four popular bio-inspired computation algorithms including ant colony optimization (ACO), particle swarm optimization (PSO), artificial bee colony (ABC), and differential evolution (DE). Special emphasis has been laid on how the biological behavior can be transferred into a technical algorithm. Moreover, description of algorithms in more general terms and the most successful variants of these algorithms are provided. Finally, a brief introduction to other bio-inspired computation algorithms such as glowworm swarm optimization (GSM), bacteria foraging optimization (BFO), bat-inspired algorithm (BA) is presented.

## 2.1 Introduction

Bio-inspired computation, short for biologically inspired computation, is a way of developing computer systems by taking ideas from the biological world. It is a field of study that loosely knits together subfields related to the topics of connectionism, social behavior, and emergence. Briefly put, it is the use of computers to model the living phenomena and simultaneously the study of life to improve the usage of computers. Bio-inspired computation in an interdiscipline composed of many different fields, such as biology, computer science, physics, mathematics, and genetics. As we have known, biological systems have many advantages over computer systems, as they are able to solve much more complex problems beyond

---

The original version of this chapter was revised. A correction to this chapter is available at https://doi.org/10.1007/978-3-642-41196-0__9


DOI 10.1007/978-3-642-41196-0_2, © Springer-Verlag Berlin Heidelberg 2014

the capability of the contemporary computer with far less energy and much higher robustness. Many of the ideas taken from natural processes have been applied to machine learning, leading to new developments in artificial intelligence.

Bio-inspired computation is a major subset of natural computation, which is different from traditional artificial intelligence (AI) in that it often takes a more evolutionary approach to learning. In traditional AI, as the creator, programmers give their program some degree of intelligence in the process of programming. However, bio-inspired techniques often involve the method of specifying a set of simple rules, a set of simple organisms which adhere to those rules, and a method of iteratively applying those rules. In other words, bio-inspired computation takes a more bottom-up, decentralized approach. It is a new approach to enable the intelligence, as the constructed simple system is able to involve into a more complex one. Complexity gets built upon complexity until the end result is something markedly complex and quite often completely counterintuitive from what the original rules would be expected to produce.

By simulating the collective behavior of decentralized, self-organized system from nature, many optimization algorithms have been developed, which could be called as either swarm intelligence or bio-inspired computation from different perspectives. The term swarm intelligence refers to a kind of problem-solving ability that emerges in the interactions among the individuals that follow a simple rule. The concept of a swarm means multiplicity, stochasticity, randomness, and messiness, and the concept of intelligence suggests that the problem-solving method is somehow successful. Swarm behavior can be seen in bird flocks, fish schools, as well as in insects like mosquitoes and midges. Many animal groups such as fish schools and bird flocks clearly display structural order, with the behavior of the organisms so integrated that even though they may change shape and direction, they appear to move as a single coherent entity. Each individual in the group attempts to maintain a minimum distance with other members at all times. This rule has the highest priority and corresponds to a frequently observed behavior of animals in nature. If individuals are not performing an avoidance maneuver, they tend to avoid being isolated from others and to align themselves with their neighbors. A swarm can be viewed as a group of agents cooperating to achieve some purposeful behavior and achieve some goal. This collective intelligence seems to emerge from large groups composed of relatively simple agents. The agents use simple local rules to govern their actions and via the interactions of the entire group, then the swarm achieves its objectives eventually.

There is no supervisor in the colony, which means that there is no central control in the swarm and each individual has a stochastic behavior by taking advantage of her perception in the environment and her neighborhood. The agents use simple local rules to govern their actions, and via the interactions of the entire group, the swarm achieves its objectives. Note that the local rules have no relation to the global pattern. Interactions among the members through the network lead to the emergence behavior, which make the colony be able to cope with complicated situations and to find solutions to complex problems, which is called self-organization by researchers. Self-organization is a crucial feature of a swarm system which results to

global-level (macroscopic level) response by means of low-level (microscopic level) interactions. Bonabeau et al. (1999) interpreted the self-organization in swarms through four characteristics, which are respectively positive feedback, negative feedback, fluctuations, and multiple interactions. Positive feedback is a simple behavioral "rules of thumb" that promotes the creation of convenient structures. Recruitment and reinforcement such as trail laying and following in some ant species or dances in bees can be shown as examples of positive feedback. Then we have a negative feedback that counterbalances positive feedback and helps to stabilize the collective pattern. In order to avoid the saturation which might occur in terms of available foragers, food source exhaustion, crowding, or competition at the food sources, a negative feedback mechanism is needed. Fluctuations such as random walks, errors, and random task switching among swarm individuals are vital for creativity and innovation. Randomness is often crucial for emergent structures since it enables the discovery of new solutions. Multiple interactions occur since agents in the swarm use the information coming from the other agents so that the information and data spread to all network.

Millonas (1994) also defined five principles to be satisfied by a swarm to have an intelligent behavior:

1. *The proximity principle*: The swarm should be able to do simple space and time computations.
2. *The quality principle*: The swarm should be able to respond to quality factors in the environment such as the quality of foodstuffs or safety of location.
3. *The principle of diverse response*: The swarm should not allocate all of its resources along excessively narrow channels, and it should distribute resources into many nodes.
4. *The principle of stability*: The swarm should not change its mode of behavior upon every fluctuation of the environment.
5. *The principle of adaptability*: The swarm must be able to change behavior mode when the investment in energy is worth the computational price.

Ethologists have modeled the behavior of a swarm with the features described above in both low level and global level (Crina and Ajith 2006). Recently researchers have been inspired by those models, and they have provided novel problem-solving techniques based on swarm intelligence for solving difficult real-world problems such as network routing, clustering, data mining, job scheduling, and bioinformatics, to name just a few. In the last two decades, especially two approaches based on ant colony described by Colorni et al. (1991) and on fish schooling and bird flocking introduced by Kennedy and Eberhart (1995) have attracted the interest of researchers all over the world. Both approaches have been studied by many researchers, and their variants have been introduced and applied for solving several problems in different areas. In this chapter, we focus on four popular bio-inspired optimization algorithms, which are, respectively, ant colony optimization (ACO), particle swarm optimization (PSO), artificial bee colony (ABC), and differential evolution (DE).

## 2.2 Ant Colony Optimization

ACO is a metaheuristic for solving hard combinatorial optimization problems (Duan 2005, 2010; Duan et al. 2011). The inspiring source of ACO is the pheromone trail laying and following behavior of real ants, which use pheromones as a communication medium (Fig. 2.1). In analogy to the biological example, ACO is based on indirect communication within a colony of simple agents, called (artificial) ants, mediated by (artificial) pheromone trails. The pheromone trails in ACO serve as a distributed, numerical information, which the ants use to probabilistically construct solutions to the problem being solved and which the ants adapt during the algorithm's execution to reflect their search experience.

The first example of such an algorithm is ant system (AS), which was proposed using as example application the well-known traveling salesman problem (TSP). Despite encouraging initial results, AS could not compete with state-of-the-art algorithms for the TSP. Nevertheless, it had the important role of stimulating further research both on algorithmic variants, which obtain much better computational performance, and on applications to a large variety of different problems. In fact, there exist now a considerable number of applications of such algorithms where world-class performance is obtained. Examples are applications of ACO algorithms to problems such as sequential ordering, scheduling, assembly line balancing, probabilistic TSP, 2D-HP protein folding, DNA sequencing, protein–ligand docking, and packet-switched routing in Internet-like networks. The ACO metaheuristic provides a common framework for the existing applications and algorithmic variants. Algorithms which follow the ACO metaheuristic are called ACO algorithms.

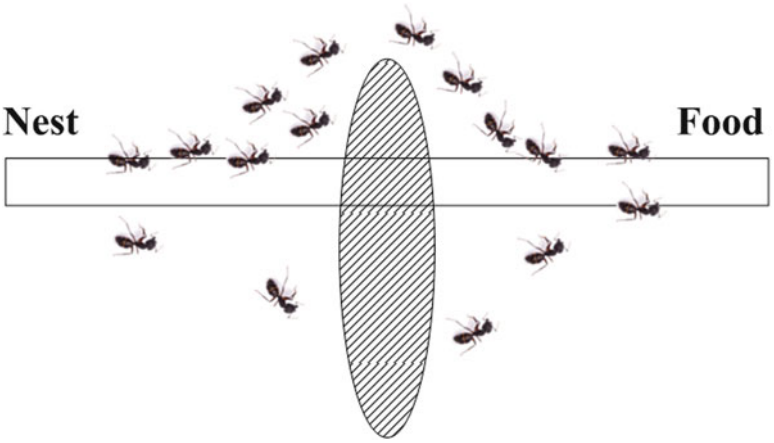


**Fig. 2.1** Schematic diagram of ACO shows that ant colony has succeeded in finding the shortest route

The (artificial) ants in ACO implement a randomized construction heuristic which makes probabilistic decisions as a function of artificial pheromone trails and possibly available heuristic information based on the input data of the problem to be solved. As such, ACO can be interpreted as an extension of traditional construction heuristics, which are readily available for many combinatorial optimization problems. Yet, an important difference with construction heuristics is the adaptation of the pheromone trails during algorithm execution to take into account the cumulated search experience.

### *2.2.1 Biological Inspiration*

Marco Dorigo and colleagues introduced the first ACO algorithm in the early 1990s (Dorigo 1992; Dorigo et al. 1996). The development of these algorithms was inspired by the observation of ant colonies. Ants are social insects. They live in colonies, and their behavior is governed by the goal of colony survival rather than being focused on the survival of individuals. The behavior that provided the inspiration for ACO is the ants' foraging behavior and, in particular, how ants can find shortest paths between food sources and their nest. When searching for food, ants initially explore the area surrounding their nest in a random manner. While moving, ants leave a chemical pheromone trail on the ground. Ants can smell pheromone. When choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromone that an ant leaves on the ground may depend on the quantity and quality of the food. The pheromone trails will guide other ants to the food source. It has been shown by Deneubourg et al. (1990) that the indirect communication between the ants via pheromone trails—known as stigmergy—enables them to find shortest paths between their nest and food sources. This is explained in an idealized setting in Fig. 2.2.

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. 2.2. After presenting the model, we will outline the differences between the model and the behavior of real ants. Our model consists of a graph $G = (V,E)$, where $V$ consists of two nodes, namely, $v_s$ (representing the nest of the ants) and $v_d$ (representing the food source). Furthermore, $E$ consists of two links, namely, $e_1$ and $e_2$, between $v_s$ and $v_d$. To $e_1$ we assign a length of $l_1$ and to $e_2$ a length of $l_2$ such that $l_2 > l_1$. In other words, $e_1$ represents the short path between $v_s$ and $v_d$, and $e_2$ represents the long path. Real ants deposit pheromone on the paths on which they move. Thus, the chemical pheromone trails are modeled as follows. We introduce an artificial pheromone value $\tau_i$ 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 between $p_i = \tau_i/(\tau_1 + \tau_2)$ path $e_1$ and path $e_2$

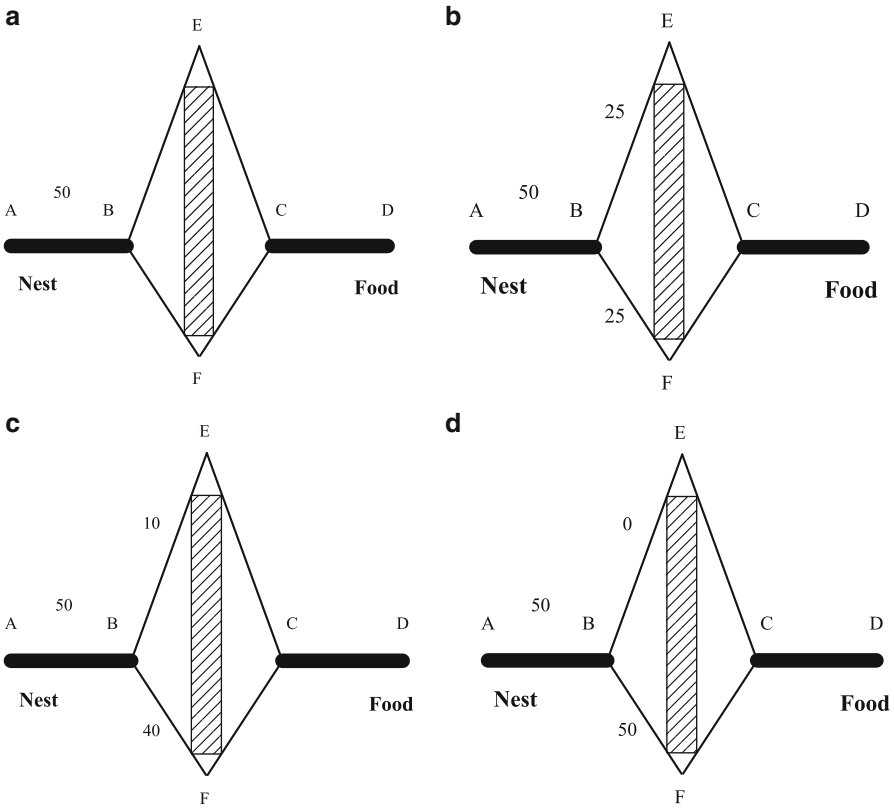


**Fig. 2.2** An experimental setting that demonstrates the shortest pathfinding capability of ant colonies

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 $v_d$, and it changes the artificial pheromone value associated to the used edge. More in detail, having chosen edge $e_i$, an ant changes the artificial pheromone value $\tau_i$ as follows:

$$\tau_i \leftarrow \tau_i + \frac{Q}{l_i} \tag{2.1}$$

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. 2.2d, in nature

the deposited pheromone is subject to an evaporation over time. We simulate this pheromone evaporation in the artificial model as follows:

$$\tau_i \leftarrow (1-\rho)\,\tau_i,\ i = 1,2 \tag{2.2}$$

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.

In the beginning, all ants are in the nest. There is no pheromone on both paths. Then the foraging starts. In probability, 50 % of ants take the short path, and the other 50 % take the long path to the food source. The ants that have taken the shorter path would arrive earlier at the food source. Therefore, when returning, the probability to take again the short path is higher. The pheromone trail on the short path receives a stronger reinforcement to take this path grows. Finally, due to the evaporation of the pheromone on the long path, the whole colony will, in probability, use the short path in probability.

## 2.2.2 *Principle of Ant Colony Optimization*

### 2.2.2.1 The First ACO Algorithm: Ant System

In AS each ant is initially put on a randomly chosen city and has a memory which stores the partial solution it has constructed so far (initially the memory contains only the start city) (Dorigo and Stützle 2003). Starting from its start city, an ant iteratively moves from city to city. When being at a city $i$, an ant $k$ chooses to go to a city $j$ with a probability given by

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum\limits_{k \in allowed_k} \left[\tau_{ik}(t)\right]^{\alpha}\left[\eta_{ik}\right]^{\beta}} & if \quad j \in allowed_k \\ 0 & otherwise \end{cases} \tag{2.3}$$

where $allowed_k = \{N - \text{tabu}_k\}$, $\alpha$ and $\beta$ are parameters that control the relative importance of trail versus visibility, $\eta_{ij}$ is the heuristic desirability, and $\eta_{ij} = 1/d_{ij}$ where $d_{ij}$ is the distance between city $i$ and city $j$ and $\tau_{ij}$ is the amount of pheromone trail on edge $(i,j)$. If $\alpha = 0$, the selection probabilities are proportional to $[\eta]^{\beta}$, and the closest cities will more likely be selected: in this case AS corresponds to a classical stochastic greedy algorithm (with multiple starting points since ants are initially randomly distributed on the cities). If $\beta = 0$, only pheromone amplification is at work: this will lead to the rapid emergence of a stagnation situation with the corresponding generation of tours which, in general, are strongly suboptimal.

The solution construction ends after each ant has completed a tour. Next, the pheromone trails are updated. In AS this is done by first lowering the pheromone trails by a constant factor (this is pheromone evaporation) and then allowing each ant to deposit pheromone on the arcs that belong to its tour:

$$\tau_{ij}\,(t+n) = (1-\rho)\cdot\tau_{ij}\,(t) + \Delta\tau_{ij} \tag{2.4}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2.5}$$

where $1-\rho\{\rho \subset (0,1)\}$ represents the evaporation of trail between time $t$ and $t+n$. The parameter $\rho$ is used to avoid unlimited accumulation of the pheromone trails and enables the algorithm to "forget" previously done bad decisions. Where $\Delta\tau_{ij}{}^{k}$ is the quantity of per unit length of pheromone trail laid on edge $(i,j)$ by the $k$th ant between time $t$ and $t+n$. In the popular ant-cycle model, it is given by

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} \; if \;\; k-th \text{ ant uses } (i,j) & \text{in its tour} \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

where $Q$ is a constant and $L_k$ is the tour length of the $k$th ant.

#### 2.2.2.2 Framework of the ACO Metaheuristic

After initialization, the metaheuristic iterates over three phases: construct ant solutions, apply local search, and update pheromones, which are described in detail as follows:

**Construct ant solutions:** A set of m artificial ants constructs solutions from elements of a finite set of available solution components $C = \{c_{ij}\}$, $i = 1, \ldots, n$, and $j = 1, \ldots, |D_i|$. A solution construction starts from an empty partial solution $s^p = \varnothing$. At each construction step, the partial solution $s^p$ is extended by adding a feasible solution component from the set $N(s^p) \subseteq \mathbb{C}$, which is defined as the set of components that can be added to the current partial solution $s^p$ without violating any of the constraints in $\Omega$. The process of constructing solutions can be regarded as a walk on the construction graph $G_C = (V,E)$.

The choice of a solution component from $N(s^p)$ is guided by a stochastic mechanism, which is biased by the pheromone associated with each of the elements of $N(s^p)$. The rule for the stochastic choice of solution components varies across different ACO algorithms, but, in all of them, it is inspired by the model of the behavior of real ants given in (2.1).

**Apply local search:** Once solutions have been constructed and before updating the pheromone, it is common to improve the solutions obtained by the ants through a

local search. This phase, which is highly problem specific, is optional although it is usually included in state-of-the-art ACO algorithms.

**Update pheromones:** The aim of the pheromone update is to increase the pheromone values associated with good or promising solutions and to decrease those that are associated with bad ones. Usually, this is achieved by decreasing all the pheromone values through pheromone evaporation and by increasing the pheromone levels associated with a chosen set of good solutions.

### *2.2.3 Ant System and Its Extensions*

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 combinatorial optimization problems. Therefore, several extensions and improvements of the original AS algorithm were introduced over the years, which show better performance than AS when applied to many optimization problems, such as Elitist AS (EAS) (Dorigo 1992), rank-based Ant System (ASrank) (Bullnheimer et al. 1999), MAX–MIN Ant System (MMAS) (Stutzle and Hoos 1997), and Ant Colony System (ACS) (Dorigo and Gambardella 1997).

A first improvement over AS was obtained by introducing the elitist strategy, which is called EAS. In this variant, the pheromone values are updated using all the solutions that were generated in the respective iteration and the best-so-far solution. It consists in giving the best tour since the start of the algorithm (called $T^{gb}$, where $gb$ stays for global best) a strong additional weight. In practice, each time the pheromone trails are updated, those belonging to the edges of the global-best tour get an additional amount of pheromone. For these edges (2.6) becomes

$$\Delta\tau_{ij}^{gb} = \begin{cases} \frac{e}{L^{gb}(t)} \; if & \text{arc}\,(i,j) \in T^{gb} \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

The arcs of $T^{gb}$ are therefore reinforced with a quantity of $e \cdot 1/L^{gb}$, where $L^{gb}$ is the length of $T^{gb}$ and $e$ is a positive integer.

Another improvement over AS is the ASrank proposed by Bullnheimer et al. (1999), which is an extension of the elitist strategy to some extent. In this approach, the best-so-far solution has the highest influence on the pheromone update at each iteration, while a selection of the best solutions constructed at that current iteration influences the update in accordance with their rankings. It sorts the ants according to the lengths of the tours they generated, and, after each tour construction phase, only the $(\omega - 1)$ best ants and the global-best ant are allowed to deposit pheromone. The $r$th best ant of the colony contributes to the pheromone update with a weight given by $\max\{0, \omega - r\}$, while the global-best tour reinforces the pheromone trails with weight $\omega$. Then (2.4) and (2.5) becomes therefore

$$\tau_{ij}\,(t+n) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{r=1}^{\omega}(\omega - r)\cdot\triangle\tau_{ij}^{r}(t) + \omega\cdot\triangle\tau_{ij}^{gb}(t) \tag{2.8}$$

where ${}_{\triangle}\tau_{ij}^{r}(t) = 1/L^{r}(t)$ and ${}_{\triangle}\tau_{ij}^{gb}(t) = 1/L^{gb}$.

As one of the most successful ACO variants, MMAS introduces upper and lower bounds to the values of the pheromone trails, as well as a different initialization of their values. In MMAS, the allowed range of the pheromone trail strength is limited to the interval $[\tau_{\min},\tau_{\max}]$, that is, $\tau_{\min} \leq \tau_{ij} \leq \tau_{\max}$, $\forall\ \tau_{ij}$, and the pheromone trails are initialized to the upper trail limit, which allows a higher exploration at the start of the algorithm. The value of this bound is updated each time a new best-so-far solution is found by the algorithm. Depending on some convergence measure, at each iteration, either the iteration-best update or the global-best update rule is used for updating the pheromone values. At the start of the algorithm, the iteration-best update rule is used more often, while during the run of the algorithm, the frequency with which the global-best update rule is used increases.

ACS, which was introduced by Dorigo and Gambardella (1997), differs from the original AS algorithm in more aspects than just in the pheromone update. In this approach, the importance of exploitation of information collected by previous ants with respect to exploration of the search space is increased, which is achieved via two mechanisms. ACS improves over AS by increasing the importance of exploitation of information collected by previous ants with respect to exploration of the search space. This is achieved via two mechanisms. First, a strong elitist strategy is used to update pheromone trails, which means only the ant that has produced the best solution is allowed to update pheromone trails, according to a pheromone trail update rule similar to that used in AS:

$$\tau_{ij}\,(t+n) = (1-\rho)\cdot\tau_{ij}(t) + \rho\cdot\tau_{ij}^{best}(t) \tag{2.9}$$

The best ant can be the iteration-best ant, that is, the best in the current iteration, or the global-best ant, that is, the ant that made the best tour from the start of the trial.

Second, ants choose the next city to move to using a so-called pseudorandom proportional rule: with probability $q_0$, they move to the city $j$ for which the product between pheromone trail and heuristic information is maximum, that is, $j = \arg\max_{j\in N_i^k}\left\{\tau_{ij}(t)\cdot\eta_{ij}^{\beta}\right\}$, while with probability $1-q_0$, they operate a biased exploration in which the probability ${p_{ij}}^{k}(t)$ is the same as in AS. When the parameter $q_0$ is set to a value close to 1, as it is the case in most ACS applications, exploitation is favored over exploration. It is obvious that, when $q_0 = 0$, the probabilistic decision rule becomes the same as in AS.

Besides, ACS differs from previous ACO algorithms also because ants update the pheromone trails while building solutions as that in ant quantity and in ant density. In practice ACS ants "eat" some of the pheromone trail on the edges they visit. This operation favors exploitation, counterbalancing this way the other

two abovementioned modifications that strongly favor exploitation of the collected knowledge about the problem. In this way, the probability that a same path is used by all the ants is decreased. In other words, it helps to avoid to be trapped into local optimum. ACS has been made more performing over other variants also by the addition of local search routines that take the solution generated by ants to their local optimum just before the pheromone update.

Although retaining some of the original biological inspiration, they are less and less biologically inspired and more and more motivated by the need of making ACO algorithms competitive with state-of-the-art algorithms or improve their performance. Nevertheless, many aspects of the original AS remain, such as the need for a colony, the role of autocatalysis, the cooperative behavior mediated by artificial pheromone trails, the probabilistic construction of solutions biased by artificial pheromone trails and local heuristic information, the pheromone updating guided by solution quality, and the evaporation of pheromone trail, which is the same in all ACO algorithms. For more information about the variants of ACO such as hypercube framework (HCF), reader can refer to Dorigo and Blum (2005). Ant algorithms are receiving increasing attention in the scientific community as a promising novel approach to distributed control and optimization.

## 2.3 Particle Swarm Optimization

The initial ideas on particle swarms of Kennedy (a social psychologist) and Eberhart (an electrical engineer) were essentially aimed at producing computational intelligence by exploiting simple analogues of social interaction (Kennedy and Eberhart 1995) rather than purely individual cognitive abilities. The first simulations were influenced by Heppner and Grenander's work (Heppner and Grenander 1990) and involved analogues of bird flocks searching for corn. These soon developed into a powerful optimization method-PSO.

### *2.3.1 Biological Inspiration*

A number of scientists have created computer simulations of various interpretations of the movement of organisms in a bird flock or fish school. Notably, Reynolds (1987) and Heppner and Grenander (1990) presented simulations of bird flocking. Reynolds was intrigued by the aesthetics of bird-flocking choreography, and Heppner, a zoologist, was interested in discovering the underlying rules that enabled large numbers of birds to flock synchronously, often changing direction suddenly, scattering and regrouping, etc. Both of these scientists had the insight that local processes, such as those modeled by cellular automata, might underlie the unpredictable group dynamics of bird social behavior. Both models relied heavily on

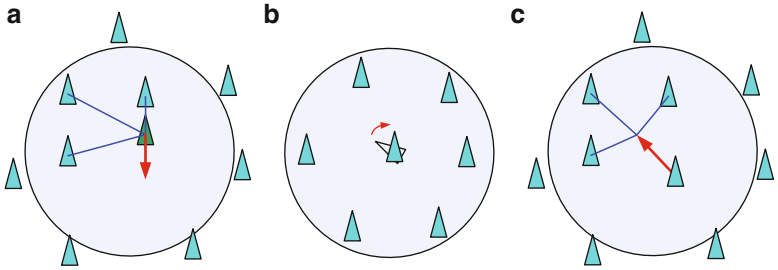


**Fig. 2.3** Boid model. (**a**) Separation. Each agent tries to move away from its neighbors if they are too close. (**b**) Alignment. Each agent steers towards the average heading of its neighbors. (**c**) Cohesion. Each agent tries to go towards the average position of its neighbors

manipulation of interindividual distances; that is, the synchrony of flocking behavior was thought to be a function of birds' efforts to maintain an optimum distance between themselves and their neighbors (Fig. 2.3).

It has been believed that social sharing of information among conspeciates offers an evolutionary advantage: this hypothesis was fundamental to the development of PSO. One motive for developing the simulation was to model human social behavior, which is of course not identical to fish schooling or bird flocking. The important difference is its abstractness. Birds and fish adjust their physical movement to avoid predators, seek food and mates, optimize environmental parameters such as temperature, etc. Humans adjust not only physical movement but cognitive or experiential variables as well. We do not usually walk in step and tum in unison (though some fascinating research in human conformity shows that we are capable of it); rather, we tend to adjust our beliefs and attitudes to conform with those of our social peers.

This is a major distinction in terms of contriving a computer simulation, for at least one obvious reason: collision. Two individuals can hold identical attitudes and beliefs without banging together, but two birds cannot occupy the same position in space without colliding. It seems reasonable, in discussing human social behavior, to map the concept of change into the bird/fish analogue of movement. This is consistent with the classic Aristotelian view of qualitative and quantitative change as types of movement. Thus, besides moving through three-dimensional physical space and avoiding collisions, humans change in abstract multidimensional space, collision-free. Physical space of course affects informational inputs, but it is arguably a trivial component of psychological experience. Humans learn to avoid physical collision by an early age, but navigation of $n$-dimensional psychosocial space requires decades of practice, and many of us never seem to acquire quite all the skills we need.

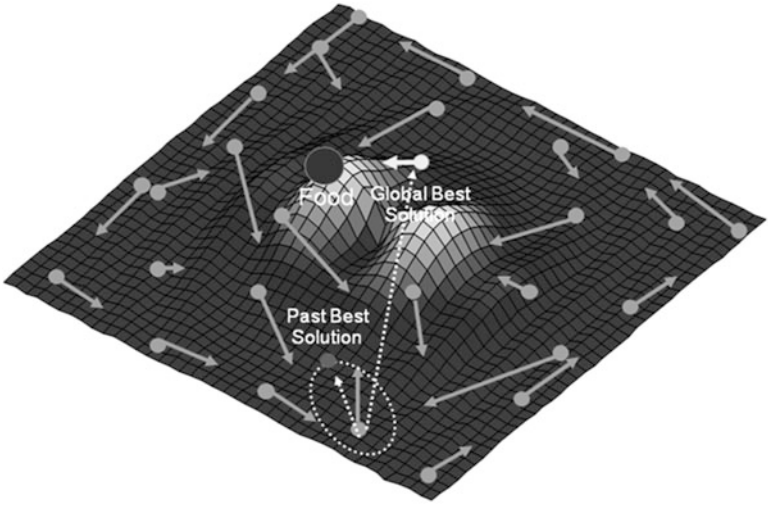


**Fig. 2.4** Schematic diagram of PSO © [2002] IEEE (Reprinted, with permission, from Duan and Liu (2010))

## *2.3.2 Principle of Particle Swarm Optimization*

### 2.3.2.1 The Framework of PSO

In PSO, a number of simple entities, the particles, are placed in the search space of some problem or function, and each evaluates the objective function at its current location. Each particle then determines its movement through the search space by combining some aspect of the history of its own current and best (best fitness) locations with those of one or more members of the swarm, with some random perturbations (Duan and Xing 2009; Duan and Liu 2010; Duan et al. 2011). The next iteration takes place after all particles have been moved. Eventually the swarm as a whole, like a flock of birds collectively foraging for food, is likely to move close to an optimum of the fitness function. Each individual in the particle swarm is composed of three $D$-dimensional vectors, where $D$ is the dimensionality of the search space. These are the current position $\boldsymbol{x}_i$, the previous best position $\boldsymbol{p}_i$, and the velocity $\boldsymbol{v}_i$. Shi and Eberhart (1998) firstly introduced the inertia weights $w$ into the basic PSO model, by adjusting $w$ to improve the performances of the PSO algorithm. Figure 2.4 describes the schematic diagram of PSO.

### 2.3.2.2 Original Version

The current position $\boldsymbol{x}_i$ can be considered as a set of coordinates describing a point in space. At each iteration of the algorithm, the current position is evaluated as a

problem solution. If that position is better than any that has been found so far, then the coordinates are stored in the second vector, $\boldsymbol{p}_i$. The value of the best function result so far is stored in a variable that can be called $pbest_i$ (for "previous best"), for comparison on later iterations. The objective, of course, is to keep finding better positions and updating $\boldsymbol{p}_i$ and $pbest_i$. New points are chosen by adding $\boldsymbol{v}_i$ coordinates to $\boldsymbol{x}_i$, and the algorithm operates by adjusting $\boldsymbol{v}_i$, which can effectively be seen as a step size (Poli et al. 2007).

The particle swarm is more than just a collection of particles. A particle by itself has almost no power to solve any problem; progress occurs only when the particles interact. Problem solving is a population-wide phenomenon, emerging from the individual behaviors of the particles through their interactions. In any case, populations are organized according to some sort of communication structure or topology, often thought of as a social network. The topology typically consists of bidirectional edges connecting pairs of particles, so that if $j$ is in $i$'s neighborhood, $i$ is also in $j$'s. Each particle communicates with some other particles and is affected by the best point found by any member of its topological neighborhood. This is just the vector $\boldsymbol{p}_i$ for that best neighbor, which we will denote with $\boldsymbol{p}_g$. The potential kinds of population "social networks" are hugely varied, but in practice certain types have been used more frequently:

$$\begin{aligned} \boldsymbol{v}_i\,(t+1) &= \boldsymbol{v}_i(t) + c_1 \cdot rand_1\,(\boldsymbol{x}_i(t) - \boldsymbol{p}_i(t)) + c_2 \cdot rand_2\,(\boldsymbol{x}_i(t) - \boldsymbol{p}_g(t)) \\ \boldsymbol{x}_i\,(t+1) &= \boldsymbol{x}_i(t) + \boldsymbol{v}_i\,(t+1) \end{aligned} \tag{2.10}$$

where each individual particle $i$ has the following properties: a position vector in search space $\boldsymbol{x}_i(t)$ at time $t$, a velocity vector $\boldsymbol{v}_i(t)$ at time $t$, and a personal best position in search space $\boldsymbol{p}_i(t)$. The personal best position $\boldsymbol{p}_i(t)$ corresponds to the position in search space where particle $i$ had the minimum fitness value $pbest_i$ determined by the objective function (in a minimization problem). The global-best position denoted by $p_g(t)$ represents the position yielding the lowest error among all the $\boldsymbol{p}_i(t)$, which has the best fitness value $gbest$ among all the particles. Two pseudorandom sequences, $rand_1 \sim (0, 1)$ and $rand_2 \sim (0, 1)$ are used to effect the stochastic algorithm nature.

The PSO algorithm consists of repeated application of (2.10). In theory, particles of a swarm may benefit from the prior discoveries and experiences of all the members of a swarm when foraging. The key point of PSO is that particles in the swarm share information with each other, which offers some sort of evolutionary advantage. Therefore, due to the simple concept, easy implementation, and quick convergence, PSO has gained much attention and wide applications in solving continuous nonlinear optimization problems. The process for implementing the original PSO is described as follows:

*Step 1* Initialize a population array of particles with random positions and velocities on $D$ dimensions in the search space.

*Step 2* For each particle, evaluate the desired optimization fitness function in $D$ variables.

*Step 3* Compare particle's fitness evaluation with its $pbest_i$. If current value is better than $pbest_i$, then set $pbest_i$ equal to the current value and $\boldsymbol{p}_i(t)$ equal to the current location $\boldsymbol{x}_i(t)$ in the $D$-dimensional space.

*Step 4* Identify the particle in the neighborhood with the best success so far, and assign its index to the variable $g$.

*Step 5* Change the velocity and position of the particle according to the (2.10).

*Step 6* If a criterion is met (usually a sufficiently good fitness or a maximum number of iterations), stop. Otherwise, go to Step 2.

#### 2.3.2.3 Other Variants of PSO

Motivated by the desire to better control the scope of the search, reduce the importance of $V_{\max}$, and perhaps eliminate it altogether, the following modification of the PSO's update equations was proposed (Shi and Eberhart 1998):

$$\begin{aligned} \boldsymbol{v}_i\,(t+1) &= w \cdot \boldsymbol{v}_i(t) + c_1 \cdot rand_1\,(\boldsymbol{x}_i(t) - \boldsymbol{p}_i(t)) + c_2 \cdot rand_2\,(\boldsymbol{x}_i(t) - \boldsymbol{p}_g(t)) \\ \boldsymbol{x}_i\,(t+1) &= \boldsymbol{x}_i(t) + \boldsymbol{v}_i\,(t+1) \end{aligned} \tag{2.11}$$

where $w$ was termed the "inertia weight." If we interpret $c_1$ and $c_2$ as the external force, $\boldsymbol{f_i}$, acting on a particle, then the change in a particle's velocity (i.e., the particle's acceleration) can be written as ${}_{\Delta}v_i = \boldsymbol{f}_i + (1-w)\boldsymbol{v}_i$. That is, the constant $1-w$ acts effectively as a friction coefficient, and so $w$ can be interpreted as the fluidity of the medium in which a particle moves. This perhaps explains why researchers have found that the best performance could be obtained by initially setting $w$ to some relatively high value (e.g., 0.9), which corresponds to a system where particles move in a low viscosity medium and perform extensive exploration, and gradually reducing $w$ to a much lower value (e.g., 0.4), where the system would be more dissipative and exploitative and would be better at homing into local optima. It is even possible to start from values of $w > 1$, which would make the swarm unstable, provided that the value is reduced sufficiently to bring the swarm in a stable region.

With (2.11) and an appropriate choice of $w$ and of the acceleration coefficients, $c_1$ and $c_2$, the PSO can be made much more stable so much so that one can either do without $V_{\max}$ or set $V_{\max}$ to a much higher value, such as the value of the dynamic range of each variable. In this case, $V_{\max}$ may improve performance, though with use of inertia or constriction techniques, it is no longer necessary for damping the swarm's dynamics.

Though the earliest researchers recognized that some form of damping of the dynamics of a particles (e.g., $V_{\max}$) was necessary, the reason for this was not understood. But when the particle swarm algorithm is run without restraining velocities in some way, these rapidly increase to unacceptable levels within a few iterations. Kennedy (1998) noted that the trajectories of nonstochastic one-dimensional particles contained interesting regularities when $c_1 + c_2$ was between

0.0 and 4.0. Clerc's analysis of the iterative system led him to propose a strategy for the placement of "constriction coefficients" on the terms of the formulas; these coefficients controlled the convergence of the particle and allowed an elegant and well-explained method for preventing explosion, ensuring convergence, and eliminating the arbitrary $V_{\max}$ parameter. The analysis also takes the guesswork out of setting the values of $c_1$ and $c_2$. Clerc and Kennedy (2002) noted that there can be many ways to implement the constriction coefficient. One of the simplest methods of incorporating it is the following:

$$\boldsymbol{v}_i\,(t+1) = \chi\,(\boldsymbol{v}_i(t)+c_1\cdot rand_1(\boldsymbol{x}_i(t)-\boldsymbol{p}_i(t))+c_2\cdot rand_2(\boldsymbol{x}_i(t)-\boldsymbol{p}_g(t)))$$

$$\boldsymbol{x}_i\,(t+1) = \boldsymbol{x}_i(t)+\boldsymbol{v}_i\,(t+1) \tag{2.12}$$

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \tag{2.13}$$

When Clerc's constriction method is used, $\phi$ is commonly set to 4.1, and the constant multiplier $\chi$ is approximately 0.7298. The results in the previous velocity being multiplied by 0.7298 and each of the two $(\boldsymbol{p}-\boldsymbol{x})$ terms being multiplied by a random number limited by $0.7298 \times 2.05 \approx 1.49618$. The constricted particles will converge without using any $V_{\max}$ at all. However, subsequent experiments and applications concluded that a better approach to use as a prudent rule of thumb is to limit $V_{\max}$ to $X_{\max}$, the dynamic range of each variable on each dimension, in conjunction with (2.12) and (2.13). The result is a PSO algorithm with no problem-specific parameters. And this is the canonical particle swarm algorithm of today. Note that a PSO with constriction is algebraically equivalent to a PSO with inertia.

Indeed, (2.11) and (2.12) can be transformed into one another via the mapping $w \leftrightarrow \chi$ and $c_i \leftrightarrow \chi c_i$. So, the optimal settings suggested by Clerc correspond to $w = 0.7298$ and $c_1 = c_2 = 1.49618$ for a PSO with inertia.

### 2.3.3 Parameters and Population Topology

The role of inertia weight $w$ in (2.11) is considered critical for the convergence behavior of PSO. The inertia weight is employed to control the impact of the previous history of velocities on the current one. Accordingly, the parameter $w$ regulates the trade-off between the global (wide-ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e., fine-tuning the current search area. A suitable value for the inertia weight $w$ usually provides balance between global and local exploration abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. Initially, the inertia weight is set as a constant. However, some experiment results indicate that it is better to initially set the inertia to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined

solutions. Thus, an initial value around 1.2 and gradually reducing towards 0 can be considered as a good choice for $w$. A better method is to use some adaptive approaches (e.g., fuzzy controller), in which the parameters can be adaptively fine-tuned according to the problems under consideration (Grosan and Abraham 2011).

The parameters $c_1$ and $c_2$ in (2.11) are not critical for the convergence of PSO. However, proper fine-tuning may result in faster convergence and alleviation of local minima. As default values, usually, $c_1 = c_2 = 2$ are used, but some experiment results indicate that $c_1 = c_2 = 1.49$ might provide even better results. Recent work reports that it might be even better to choose a larger cognitive parameter, $c_1$, than a social parameter, $c_2$, but with $c_1 + c_2 \leq 4$.

The first particle swarms evolved out of bird-flocking simulations of a type described by Reynolds (1987) and Heppner and Grenander (1990). In these models, the trajectory of each bird's flight is modified by application of several rules, including some that take into account the birds that are nearby in physical space. So, early PSO topologies were based on proximity in the search space. Kennedy and Mendes studied the various population topologies on the PSO performance. Different concepts for neighborhoods could be envisaged. It can be observed as a spatial neighborhood when it is determined by the Euclidean distance between the positions of two particles or as a sociometric neighborhood (e.g., the index position in the storing array).

The next topology to be introduced, the *gbest* topology (for "global best"), was one where the best neighbor in the entire population influenced the target particle. While this may be conceptualized as a fully connected graph, in practice it only meant that the program needed to keep track of the best function result that had been found and the index of the particle that found it.

The *gbest* is an example of static topology, i.e., one where neighbors and neighborhoods do not change during a run. The *lbest* topology (for "local best") is another static topology, which was introduced in Eberhart and Kennedy (1995). It is a simple ring lattice where each individual was connected to $K = 2$ adjacent members in the population array, with toroidal wrapping (naturally, this can be generalized to $K > 2$). This topology had the advantage of allowing parallel search, as subpopulations could converge in diverse regions of the search space. Where equally good optima were found, it was possible for the population to stabilize with particles in the good regions, but if one region was better than another, it was likely to attract particles to itself. Thus this parallel search resulted in a more thorough search strategy; though it converged more slowly than the *gbest* topology, *lbest* was less vulnerable to the attraction of local optima.

Several classical communications structures from social psychology (Bavelas 1950), including some with small-world modifications, were experimented with in Kennedy (1999). Circles, wheels, stars, and randomly assigned edges were tested on a standard suite of functions. The most important finding was that there were important differences in performance depending on the topology implemented; these differences depended on the function tested, with nothing conclusively suggesting that any one was generally better than any other.

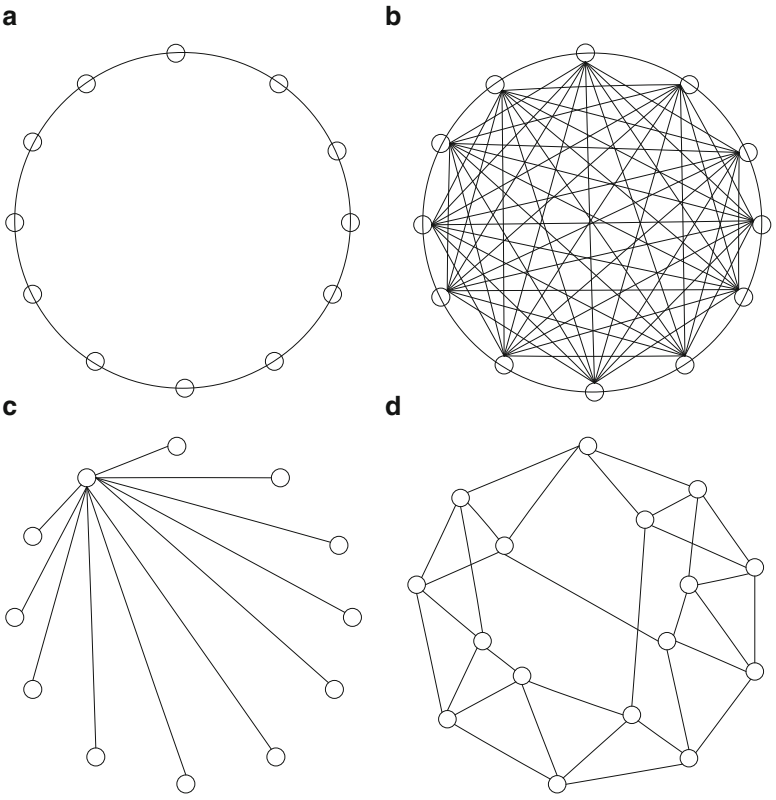


**Fig. 2.5** Some neighborhood topologies of PSO. (**a**) Local-best topology. (**b**) Global-best topology. (**c**) Star topology. (**d**) Von Neumann topology

Numerous aspects of the social-network topology were tested in Kennedy and Mendes (2002) (Fig. 2.5). For instance, the effect of including the target particle in its neighborhood (as opposed to allowing only "external" influences) was evaluated, finding, surprisingly, that whether or not the particle belonged to its neighborhood had little impact on behavior. 1343 random graphs were generated and then modified to meet certain criteria, including mean degree, clustering, and the standard deviations of those two measures to introduce homogeneous and heterogeneous structures. Several regular topologies were included, as well; these were the *gbest* and *lbest* versions mentioned above, as well as a von Neumann topology which defined neighborhoods on a grid, a "pyramid" topology, and a handmade graph with clusters of nodes linked sparsely.

One finding that emerged was the relative superiority of the von Neumann structure. This topology possesses some of the parallelism of *lbest*, yet nodes have degree $K = 4$; thus the graph is more densely connected than *lbest* but less densely than *gbest*.

## 2.4 Artificial Bee Colony

ABC algorithm was originally presented by Karaboga and Basturk (2007), under the inspiration of collective behavior on honeybees, and it has been proved to possess a better performance in function optimization problem, compared with genetic algorithm, DE and PSO. As we know, usual optimization algorithms conduct only one search operation in one iteration, for example, the PSO algorithm carries out global search at the beginning and local search in the later stage. Compared with the usual algorithms, the major advantage of ABC algorithm lies in that it conducts both global search and local search in each iteration, and as a result the probability of finding the optimal parameters is significantly increased, which efficiently avoid local optimum to a large extent.

### *2.4.1 Biological Inspiration*

A very interesting swarm in nature is honeybee swarm that allocates the tasks dynamically and adapts itself in response to changes in the environment in a collective intelligent manner. The honeybees have photographic memories; space-age sensory and navigation systems, possibly even insight skills; and group decision-making process during selection of their new nest sites, and they perform tasks such as queen and brood tending, storing, retrieving and distributing honey and pollen, communicating, and foraging. These characteristics are incentive for researchers to model the intelligent behaviors of bees. Before presenting the algorithms described to use intelligent behaviors and their applications, behavior of the colony is explained.

Bees are social insects living as colonies. There are three kinds of bees in a colony: drones, queen, and workers. Foraging is the most important task in the hive. Many studies (Seeley 1985) have investigated the foraging behavior of each individual bee and what types of external information (such as odor, location information in the waggle dance, the presence of other bees at the source or between the hive and the source) and internal information (such as remembered source location or source odor) affect this foraging behavior. Foraging process starts with leaving the hive of a forager in order to search food source to gather nectar. After finding a flower for herself, the bee stores the nectar in her honey stomach. Based on the conditions such as richness of the flower and the distance of the flower to the hive, the bee fills her stomach in about 30–120 min, and honey-making process begins with the secretion of an enzyme on the nectar in her stomach. After coming back to the hive, the bee unloads the nectar to empty honeycomb cells, and some extra substances are added in order to avoid the fermentation and the bacterial attacks. Filled cells with the honey and enzymes are covered by wax.

After unloading the nectar, the forager bee which has found a rich source performs special movements called “dance” on the area of the comb in order to share her information about the food source such as how plentiful it is and its direction and distance and recruits the other bees for exploiting that rich source. While dancing, other bees touch her with their antenna and learn the scent and the taste of the source she is exploiting. She dances on different areas of the comb in order to recruit more bees and goes on to collect nectar from her source. There are different dances performed by bees depending on the distance information of the source: round dance, waggle dance, and tremble dance. If the distance of the source to the hive is less than 100 m, round dance is performed, while if the source is far away, waggle dance is performed. Round dance does not give direction information. In case of waggle dance, direction of the source according to the sun is transferred to other bees. Longer distances cause quicker dances. The tremble dance is performed when the foraging bee perceives a long delay in unloading its nectar.

Forager bees use a maplike organization of spatial memory for homing and food source search flights. This organization is based on the computations of two experienced vectors or on viewpoints and landmarks. There are two perspectives of which one certainly true is not known. First one is that bees use stimuli obtained during their flights. The second one is that they encode the spatial information in their dances into their maplike spatial memory (Menzel et al. 2006).

A honeybee colony needs to divide its workforce so that the appropriate number of individuals is allocated for each of the many tasks. Bees are specialized in order to carry out every task in the hive. However, there is a controversy about which factors have roles on the specialization of bees, such as their age, hormones, and individual predisposition coming from their genetic determination (Dornhaus et al. 1998), and also the allocation of tasks can dynamically change. For example, when food is drought, younger nurse bees will also join to foraging process. Depending on the swarm intelligent behaviors of a bee swarm noted above, several approaches have been introduced and applied to solve problems.

Karlvon Frisch, a famous Nobel Prize winner, found that in nature, although each bee only performs one single task, yet through a variety of information communication ways between bees such as waggle dance and special odor, the entire colony can always easily find food resources that produce relative high amount of nectar, hence realize its self-organizing behavior.

### *2.4.2 Principle of Artificial Bee Colony*

In order to introduce the self-organization model of forage selection that leads to the emergence of collective intelligence of honeybee swarms, first, we need to define three essential components: food sources, unemployed foragers, and employed foragers (Duan et al. 2010, 2011; Xu et al. 2010; Yu and Duan 2012):

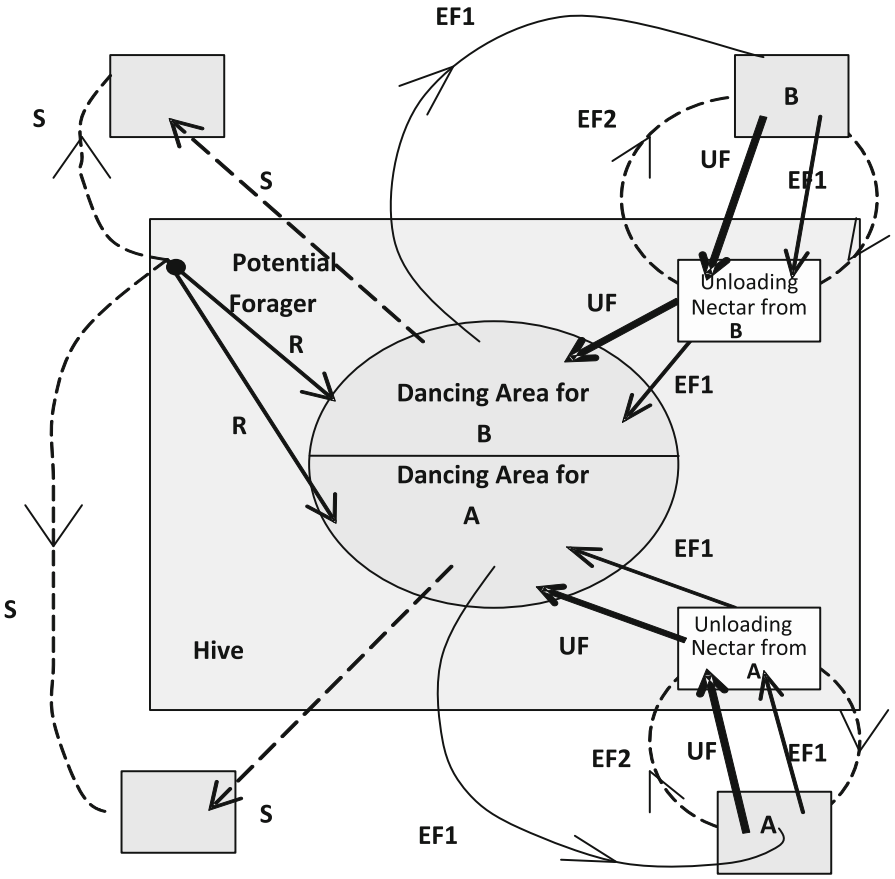


**Fig. 2.6** The behavior of honeybee foraging for nectar (Reprinted from Xu and Duan (2010), with kind permission from Elsevier)

**1. Food Sources**

For the sake of simplicity, the "profitability" of a food source (A and B in Fig. 2.6) can be represented with a single quantity. The position of a food source represents a possible parameter solution to the optimization problem, and the nectar amount of a food source corresponds to the similarity value of the associated solution.

**2. Unemployed Foragers**

If it is assumed that a bee has no knowledge about the food sources in the search field, the bee initializes its search as an unemployed forager. Unemployed foragers are continually at look out for a food source to exploit. There are two types of unemployed foragers: scouts and onlookers.

**Scouts (S in Fig. 2.6):** If the bee starts searching spontaneously for new food sources without any knowledge, it will be a scout bee.

**Onlookers (R in Fig. 2.6):** The onlookers wait in the nest and search the food source through sharing information of the employed foragers, and there is a greater probability of onlookers choosing more profitable sources.

**3. Employed Foragers**

They are associated with a particular food source which they are currently exploiting. They carry with them information about this particular source and the profitability of the source and share this information with a certain probability. After the employed foraging bee loads a portion of nectar from the food source, it returns to the hive and unloads the nectar to the food area in the hive. There are three possible options related to residual amount of nectar for the foraging bee.

If the nectar amount decreases to a low level or is exhausted, the foraging bee abandons the food source and becomes an unemployed bee (UF in Fig. 2.6).

If there are still sufficient amount of nectar in the food source, it can continue to forage without sharing the food source information with the nest mates (EF2 in Fig. 2.6).

Or it can go to the dance area to perform waggle dance for informing the nest mates about the food source (EF1 in Fig. 2.6).

In this way, the bees finally can construct a relative good solution of the multimodal optimization problems.

At the initial moment, all the bees without any prior knowledge play the role of detecting bees. After a random search for bee sources, the detecting bees can convert into any kind of bees above in accordance with the profit of the searched food sources. The changing rules are described as follows:

When the profit of the food source the bee searched is higher than the threshold, it becomes a leading bee, goes on exploring nectar, and also recruits more bees (EF1) to explore together. When the profit of related food source is relative low, it gives up the food source and again becomes a detecting bee to search for new food source (UF). When the profit is less than certain threshold, it follows leading bees to explore nectar. When searching times around hive exceed a certain limit but still the bees could not find a good resource, it abandons the source and finds a new one.

In ABC algorithm, the position of a food source represents a possible solution to the optimization problem, and the nectar amount of a food source corresponds to the quality (fitness) of the associated solution. The number of the employed bees or the onlooker bees is equal to the number of solutions in the population. At the first step, the ABC generates a randomly distributed initial population, which is corresponding to the food source positions. After initialization, the population of the positions (solutions) is subject to repeated cycles, $T = 1, 2, \ldots, T_{\max}$, of the search processes of the employed bees, the onlooker bees, and the scout bees. An employed bee produces a modification on the position (solution) in her memory depending on the local information (visual information) and tests the nectar amount (fitness value) of the new source (new solution). If the nectar amount of the new one is higher than

that of the previous one, the bee memorizes the new position and forgets the old one. Otherwise she keeps the position of the previous one in her memory. After all employed bees complete the search process, they share the nectar information of the food sources and their position information with the onlooker bees. An onlooker bee evaluates the nectar information taken from all employed bees and chooses a food source with a probability related to its nectar amount. As in the case of the employed bee, she produces a modification on the position in her memory and checks the nectar amount of the candidate source. If the nectar is higher than that of the previous one, the bee memorizes the new position and forgets the old one.

### *2.4.3 Algorithmic Structure of Artificial Bee Colony*

In ABC algorithm, each cycle of the search consists of three steps: sending the employed bees onto their food sources and evaluating their nectar amounts; after sharing the nectar information of food sources, the selection of food source regions by the onlookers and evaluating the nectar amount of the food sources; and determining the scout bees and then sending them randomly onto possible new food sources. At the initialization stage, a set of food sources is randomly selected by the bees, and their nectar amounts are determined. At the first step of the cycle, these bees come into the hive and share the nectar information of the sources with the bees waiting on the dance area. A bee waiting on the dance area for making decision to choose a food source is called onlooker, and the bee going to the food source visited by herself just before is named as employed bee. After sharing their information with onlookers, every employed bee goes to the food source area visited by herself at the previous cycle since that food source exists in her memory and then chooses a new food source by means of visual information in the neighborhood of the one in her memory and evaluates its nectar amount. At the second step, an onlooker prefers a food source area depending on the nectar information distributed by the employed bees on the dance area. As the nectar amount of a food source increases, the probability of that food source chosen also increases. After arriving at the selected area, she chooses a new food source in the neighborhood of the one in the memory depending on visual information as in the case of employed bees. The determination of the new food source is carried out by the bees based on the comparison process of food source positions visually. At the third step of the cycle, when the nectar of a food source is abandoned by the bees, a new food source is randomly determined by a scout bee and replaced with the abandoned one. At each cycle at most, one scout goes outside for searching a new food source, and the number of employed and onlooker bees is selected to be equal to each other. These three steps are repeated through a predetermined number of cycles called maximum cycle number $T_{\max}$ or until a termination criterion is satisfied.

Define $N_s$ as the total number of bees, $N_e$ as the colony size of the employed bees, and $N_u$ as the size of unemployed bees, which satisfy the equation $N_s = N_e + N_u$. We usually set $N_e$ equal to $N_u$. $D$ is the dimension of individual solution vector,

$S = \mathbb{R}^D$ represents individual search space, and $S^{N_e}$ denotes the colony space of employed bees. An employed bee colony can be expressed by $N_e$ dimension vector $\overrightarrow{X} = (X_1, \ldots, X_{N_e})$, where $X_i \in S$ and $i \leq N_e$. $\overrightarrow{X}(0)$ means the initial employed bee colony, while $\overrightarrow{X}(n)$ represents employed bee colony in the $n$th iteration. Denote $f: S \to R^+$ as the fitness function, and the standard ABC algorithm can be expressed as follows:

*Step 1* Randomly initialize a set of feasible solutions $(X_1, \ldots, X_{N_s})$, and the specific solution $X_i$ can be generated by

$$X_i^j = X_{\min}^j + rand\,(0,1)\left(X_{\max}^j - X_{\min}^j\right) \tag{2.14}$$

where $j \in \{1,2, \ldots,D\}$ is the $j$th dimension of the solution vector. Calculate the fitness value of each solution vector respectively, and set the top $N_e$ best solutions as the initial population of the employed bees $\overrightarrow{X}(0)$.

*Step 2* For an employed bee in the $n$th iteration $X_i(n)$, search new solutions in the neighborhood of the current position vector according to the following equation:

$$V_i^j = X_i^j + \varphi_i^j\left(X_i^j - X_k^j\right) \tag{2.15}$$

where $V \in S$, $j \in \{1,2, \ldots,D\}$, $k \in \{1,2, \ldots,N_e\}, k \neq i$, $k$, and $j$ are randomly generated. $\varphi_i^j$ is a random number between -1 and 1. It controls the production of neighbor food sources around $X_i^j$ and represents the comparison of two food positions visually by a bee. As can be seen from the above equation, as the difference between the parameters of $X_i^j$ and $X_k^j$ decreases, the perturbation on the position $X_i^j$ gets decreased, too. Thus, as the search approaches the optimum solution in the search space, the step length is adaptively reduced.

Generally, this searching process is actually a random mapping from individual space to individual space, and this process can be denoted with $T_m: S \to S$, and its probability distribution is clearly only related to current position vector $X_i(n)$, and has no relation with past location vectors as well as the iteration number $n$.

*Step 3* Apply the greedy selection operator $T_s: S^2 \to S$ to choose the better solution between searched new vector $V_i$ and the original vector $X_i$ into the next generation. Its probability distribution can be described as follows:

$$P\left\{T_s\left(X_i, V_i\right) = V_i\right\} = \begin{cases} 1, f\left(V_i\right) \geq f\left(X_i\right) \\ 0, f\left(V_i\right) < f\left(X_i\right) \end{cases} \tag{2.16}$$

The greedy selection operator ensures that the population is able to retain the elite individual, and accordingly the evolution will not retreat. Obviously, the distribution of $T_s$ has no relation with the iteration $n$.

*Step 4* Each unemployed bee selects an employed bee from the colony according to their fitness values. The probability distribution of the selection operator $T_{s1}$ : $S^{N_e} \rightarrow S$ can described as follows:

$$P\left\{T_{s1}\left(\overrightarrow{X}\right) = X_i\right\} = \frac{f\left(X_i\right)}{\sum_{m=1}^{N_e} f\left(X_m\right)} \tag{2.17}$$

*Step 5* The unemployed bee searches in the neighborhood of the selected employed bee's position to find new solutions (see (2.15)). The updated best fitness value can be denoted with $f\_best$, and the best solution parameters can be expressed with ($x_1$,$x_2$, ...,$x_D$).

*Step 6* If a position cannot be improved further through a predetermined number of cycles, then that food source is assumed to be abandoned. The value of predetermined number of cycles is an important control parameter of the ABC algorithm, which is called "limit" for abandonment. If the searching times surrounding an employed bee *Bas* exceed a certain threshold *Limit*, but still could not find better solutions, then the location vector can be reinitialized randomly according to the following equation:

$$X_i\left(n+1\right) = \begin{cases} X_{\min} + rand\left(0,1\right)\left(X_{\max} - X_{\min}\right), & Bas_i \geq Limit \\ X_i(n), & Bas_i < Limit \end{cases} \tag{2.18}$$

*Step 7* If not, go to Step (2). If the iteration value is larger than the maximum number of the iteration (i.e., $T > T_{\max}$), output the optimal fitness value $f\_best$ and correlative parameters ($x_1$,$x_2$, ...,$x_D$). If not, go to Step (2).

Step (6) is a most prominent aspect making ABC algorithm different from other algorithms, which is designed to enhance the diversity of the population to prevent the population from trapping into the local optimum. Obviously, this step can improve the probability of finding the best solution efficiently and make the ABC algorithm perform much better.

Totally, ABC algorithm employs four different selection processes (Karaboga and Akay 2009):

1. A global probabilistic selection process, in which the probability value is calculated by (2.17) used by the onlooker bees for discovering promising regions.
2. A local probabilistic selection process carried out in a region by the employed bees and the onlookers depending on the visual information such as the color, shape, and fragrance of the flowers (sources) (bees will not be able to identify the type of nectar source until they arrive at the right location and discriminate among sources growing there based on their scent) for determining a food source around the source in the memory in a way described by (2.15).

3. A local selection called greedy selection process carried out by onlooker and employed bees in that if the nectar amount of the candidate source is better than that of the present one, the bee forgets the present one and memorizes the candidate source produced by (2.15). Otherwise, the bee keeps the present one in the memory.
4. A random selection process carried out by scouts as defined in (2.18).

## 2.5 Differential Evolution

### *2.5.1 Biological Inspiration*

Researchers have been looking into nature for years for inspiration with the purpose of tackling complex computational problems. Optimization is ubiquitous in natural processes. For example, every species had to adapt their physical structures to fit to the environments they were in and to strengthen their survival ability all the time. The underlying relation between optimization and biological evolution led to the development of an important paradigm of computational intelligence, the evolutionary computing techniques for performing very complex search and optimization.

Evolutionary computation uses iterative progress, such as growth or development in a population. This population is then selected in a guided random search using parallel processing to achieve the desired end. The paradigm of evolutionary computing techniques dates back to early 1950s, when the idea to use Darwinian principles for automated problem solving originated. It was not until the 1960s that three distinct interpretations of this idea started to be developed in three different places. Evolutionary programming (EP) was introduced by Lawrence J. Fogel in the United States, while almost simultaneously, I. Rechenberg and H.-P. Schwefel introduced evolution strategies (ESs) in Germany. Almost a decade later, John Henry Holland from the University of Michigan at Ann Arbor devised an independent method of simulating the Darwinian evolution to solve practical optimization problems and called it the genetic algorithm (GA). These areas developed separately for about 15 years. From the early 1990s on, they are unified as different representatives of one technology, called evolutionary computing. Also since the early 1990s, a fourth stream following the same general ideas started to emerge, which is genetic programming (GP). Nowadays, the field of nature-inspired metaheuristics is mostly constituted by the evolutionary algorithms (EA) as well as the swarm intelligence algorithms. Also the field extends in a broader sense to include self-organizing systems, artificial life (digital organism), memetic and cultural algorithms, harmony search, artificial immune systems, and learnable evolution model.

The DE algorithm emerged as a very competitive form of evolutionary computing more than a decade ago. The first written article on DE appeared as a technical report

by Storn and Price (1995). One year later, the success of DE was demonstrated at the First International Contest on Evolutionary Optimization in May 1996, which was held in conjunction with the 1996 IEEE International Conference on Evolutionary Computation (CEC). DE finished third at the First International Contest on Evolutionary Optimization (1st ICEO), which was held in Nagoya, Japan. DE turned out to be the best evolutionary algorithm for solving the real-valued test function suite of the 1st ICEO (the first two places were given to non-evolutionary algorithms, which are not universally applicable but solved the test problems faster than DE). Price presented DE at the Second International Contest on Evolutionary Optimization in 1997, and it turned out as one of the best among the competing algorithms. Two journal articles describing the algorithm in sufficient details followed immediately in quick succession. In 2005 CEC competition on real parameter optimization, on 10-*D* problems classical DE secured 2*nd* rank and a self-adaptive DE variant called SaDE secured third rank although they performed poorly over 30-*D* problems.

DE, like most popular EAs, is a population-based tool. DE, unlike other EAs, generates offspring by perturbing the solutions with a scaled difference of two randomly selected population vectors, instead of recombining the solutions under conditions imposed by a probabilistic scheme. In addition, DE employs a one-to-one spawning logic which allows replacement of an individual only if the offspring outperforms its corresponding parent (Duan et al., 2011). DE has been seen as an attractive optimization tool for continuous optimization for the following four reasons: (1) Compared to most other EAs, DE is much more simple and straightforward to implement. (2) As indicated by the recent studies on DE despite its simplicity, DE exhibits much better performance in comparison with several algorithms in solving a wide variety of problems including unimodal, multimodal, separable, non-separable, and so on. (3) The number of control parameters in DE is very few (*Cr*, *F*, and *NP* in classical DE). (4) The space complexity of DE is low as compared to some of the most competitive real parameter optimizers.

## *2.5.2 Principle of Differential Evolution*

### 2.5.2.1 Initialization of the Parameter Vectors

DE algorithm mainly has three evolutionary operations, namely, mutation, recombination, and selection. The positions of individuals are represented as real-coded vectors which are randomly initialized inside the limits of the given search space in the beginning of an optimization process. The individuals are evolved during the optimization process by applying mutation, recombination, and selection to each individual in every generation. A stopping criterion determines after the building of every new generation if the optimization process should be terminated.

Like other evolutionary algorithms, DE also deals with a population of solutions. Suppose that the initial solution population has *NP* individuals and the search

space is $D$ dimensional, the solution vector in continuous space can be represented by $x_i = [x_{i1}, x_{i2}, \cdots, x_{iD}], (i = 1, \cdots, NP)$. Let there be some criteria of optimization, usually named fitness or cost function. Then the optimization goal of DE algorithm is to find the values of the variables that minimize the fitness, that is, to find

$$x^* : f\left(x^*\right) = \min_x f(x)$$

#### 2.5.2.2 Mutation with Difference Vectors

Biologically, "mutation" means a sudden change in the gene characteristics of a chromosome. In the context of the evolutionary computing paradigm, mutation is also seen as a change or perturbation with a random element. In DE literature, a parent vector from the current generation is called target vector, a mutant vector obtained through the differential mutation operation is known as donor vector, and finally an offspring formed by recombining the donor with the target vector is called trial vector. In one of the simplest forms of DE mutation, to create the donor vector for each $i$th target vector from the current population, three other distinct parameter vectors, say $\boldsymbol{x}_{r1}$, $\boldsymbol{x}_{r2}$, and $\boldsymbol{x}_{r3}$, are sampled randomly from the current population. The indices $r_1$, $r_2$, and $r_3$ are mutually exclusive integers randomly chosen from the range [1, $NP$], which are also different from the base vector index $i$. These indices are randomly generated once for each mutant vector. Now the difference of any two of these three vectors is scaled by a scalar number $F$ (that typically lies in the interval [0.4, 1]), and the scaled difference is added to the third one whence we obtain the donor vector $\boldsymbol{v}_i$. We can express the process as

$$\boldsymbol{v}_i = \boldsymbol{x}_{r_1} + F \times (\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3}) \tag{2.19}$$

The process is illustrated on a 2-$D$ parameter space in Fig. 2.7.

#### 2.5.2.3 Crossover

To enhance the potential diversity of the population, a crossover operation comes into play after generating the donor vector through mutation. In this way, the individuals of the population are updated by means of the recombination operation. By coping components from the mutation vector $\boldsymbol{v}_i$ and the target vector $\boldsymbol{x}_i$ in dependence, the trial vector $\boldsymbol{u}_i$ was generated. This process can be written as the following equation:

$$u_{ji} = \begin{cases} v_{ji}, \ if \ randb \leq CR \ or \ j = randr, & j = 1, \cdots, D \\ x_{ji}, \ if \ randb > CR \ or \ j \neq randr, & j = 1, \cdots, D \end{cases} \tag{2.20}$$

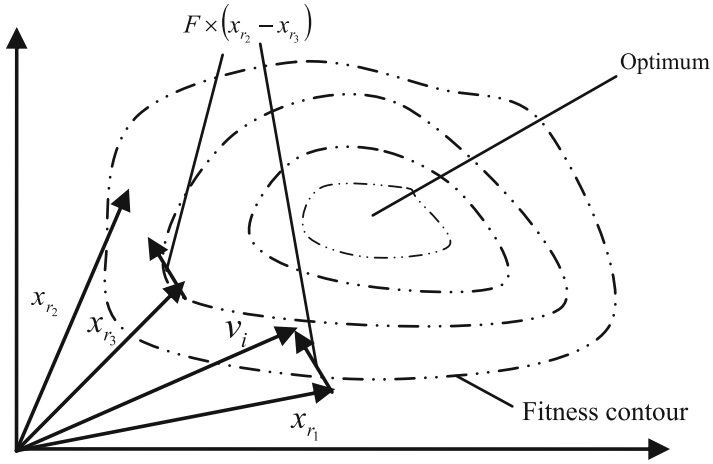


**Fig. 2.7** Mutation process of DE

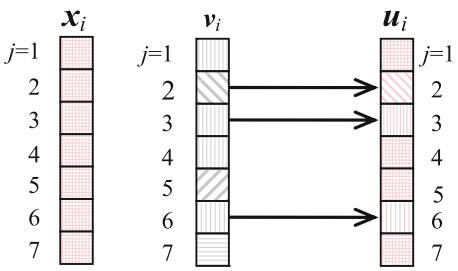


**Fig. 2.8** Recombination process of DE

where the random number $randb \in [0,1]$, the recombination control parameter $CR$ is a constant in the interval [0,1]. $randr$ is an integer randomly chosen from [1,$D$]. The recombination process is described in Fig. 2.8.

#### 2.5.2.4 Selection

To keep the population size constant over subsequent generations, the next step of the algorithm calls for selection to determine whether the target or the trial vector survives to the next generation. Then, selection operation, as a deterministic process in DE algorithm, is implemented to choose the better individuals with lower fitness function value between the target vector and the trial vector, which is inherited by the next generation, expressed as

$$x_i(t+1) = \begin{cases} u_i, & if \ f(u_i) \le f(x_i) \\ x_i, & else \end{cases} \tag{2.21}$$

This selection scheme allows only improvement but not deterioration of the fitness function value; it is called greedy. Selection operator ensures that the best fitness function value cannot get lost when moving from one generation to the next, which usually results in the fast convergence behavior. Therefore, if the new trial vector yields an equal or lower value of the objective function, it replaces the corresponding target vector in the next generation; otherwise the target is retained in the population. Hence, the population either gets better (with respect to the minimization of the objective function) or remains the same in fitness status but never deteriorates.

### 2.5.3 Control Parameters of Differential Evolution

There are three main control parameters of the DE algorithm: the mutation scale factor $F$, the crossover constant $CR$, and the population size $NP$.

The population size is related to the amount of possible moving vectors. Over all the possible moves given by a population, some moves are beneficial in the search for the optimum, while some others are ineffective and result in a waste of computational effort. Therefore, too small a population size can contain too limited an amount of moves, while too large a population size may contain a high number of ineffective moves which can likely mislead the search. To some extent the population sizing of a DE is analogous to the other EAs, if too small it could cause premature convergence, and if too large it could cause stagnation. A good value can be found by considering the dimensionality of the problem similar to what is commonly performed for the other EAs. A guideline is given in Price and Storn (1997) where a setting of Spop equal to ten times the dimensionality of the problem is proposed. However, this indication is not confirmed by a recent study in Neri and Tirronen (2008) where it is shown that a population size lower than the dimensionality of the problem can be optimal in many cases.

Regarding the scale factor $F$ and the crossover rate $CR$, these settings may be a difficult task. The setting of these two parameters is neither an intuitive nor a straightforward task but is unfortunately crucial for guaranteeing the algorithmic functioning. Several studies have thus been proposed in literature. The study reported in Lampinen and Zelinka (2000) arrives at the conclusion, after an empirical analysis, that usage of $F = 1$ is not recommended, since according to a conjecture of the authors, it leads to a significant decrease in explorative power. Analogously, the setting $CR = 1$ is also discouraged since it would dramatically decrease the amount of possible offspring solutions. In Price and Storn (1997), the settings $F \in [0.5,1]$ and $CR \in [0.8,1]$ are recommended. The empirical analysis reported in Zielinski et al. (2006) shows that in many cases the setting of $F \geq 0.6$ and $CR \geq 0.6$ leads to results having better performance.

Several studies highlight that an efficient parameter setting is very dependent on problems (e.g., $F = 0.2$ could be a very efficient setting for a certain fitness landscape and completely inadequate for another problem). This result can be seen

as a confirmation of the validity of the No Free Lunch Theorem (Wolpert and Macready 1997) with reference to the DE schemes.

The problem of the parameter setting is emphasized when DE is employed for handling difficulties of real-world applications such as high dimensionality and noisy optimization problems. Clearly, the risk of the DE stagnation is higher for larger decision spaces and worsens as the number of dimensions of the problem increases. A large decision space (in terms of dimensions) requires a wide range of possible moves to enhance its capability of detecting new promising solutions. Since, as mentioned before, an enlargement in population size causes an increase in the set of potential ineffective moves, a proper choice of $F$ and $CR$ becomes a crucial aspect in the success of DE.

## 2.6 Other Algorithms

### *2.6.1 Glowworm Swarm Optimization*

Glowworm swarm optimization (GSO) is a novel algorithm designed by Krishnanand and Ghose (2009) for the simultaneous computation of multiple optima of multimodal functions. The algorithm shares a few features with some better known swarm intelligence-based optimization algorithms, such as ACO and PSO, but with several significant differences. The agents in GSO are thought of as glowworms that carry a luminescence quantity called luciferin along with them. The glowworms encode the fitness of their current locations, evaluated using the objective function, into a luciferin value that they broadcast to their neighbors. The glowworm identifies its neighbors and computes its movements by exploiting an adaptive neighborhood, which is bounded above by its sensor range. Each glowworm selects, using a probabilistic mechanism, a neighbor that has a luciferin value higher than its own and moves towards it. These movements—based only on local information and selective neighbor interactions—enable the swarm of glowworms to partition into disjoint subgroups that converge on multiple optima of a given multimodal function.

### *2.6.2 Bacteria Foraging Optimization*

Natural selection tends to eliminate animals with poor "foraging strategies" (methods for locating, handling, and ingesting food) and favor the propagation of genes of those animals that have successful foraging strategies since they are more likely to enjoy reproductive success (they obtain enough food to enable them to reproduce). After many generations, poor foraging strategies are either eliminated or shaped into good ones. Such evolutionary principles have led scientists to hypothesize that it is appropriate to model the activity of foraging as an

optimization process. Passino (2002) proposed a bacteria foraging optimization by simulating the chemotactic (foraging) behavior of *E. coli* bacteria. There are algorithmic analogies between the genetic algorithm and the above optimization model for foraging. There are analogies between the fitness function and the nutrient concentration function (both a type of "landscape"), selection and bacterial reproduction (bacteria in the most favorable environments gain a selective advantage for reproduction), crossover and bacterial splitting (the children are at the same concentration, whereas with crossover they generally end up in a region around their parents on the fitness landscape), and mutation and elimination and dispersal. However, the algorithms are not equivalent, and neither is a special case of the other. Each has its own distinguishing features. The fitness function and nutrient concentration functions are not the same (one represents likelihood of survival for given phenotypic characteristics, whereas the other represents nutrient/noxious substance concentrations or for other foragers predator/prey characteristics). Crossover represents mating and resulting differences in offspring, something we ignore in the bacterial foraging algorithm (we could, however, have made less than perfect copies of the bacteria to represent their splitting). Moreover, mutation represents gene mutation and the resulting phenotypical changes, not physical dispersal in an environment.

### *2.6.3 Bat-Inspired Algorithm*

Most microbats are insectivores. Microbats use a type of sonar, called echolocation, to detect prey, avoid obstacles, and locate their roosting crevices in the dark. These bats emit a very loud sound pulse and listen for the echo that bounces back from the surrounding objects. Their pulses vary in properties and can be correlated with their hunting strategies, depending on the species. Most bats use short, frequency-modulated signals to sweep through about an octave, while others more often use constant-frequency signals for echolocation. Their signal bandwidth varies and depends on the species and often increased by using more harmonics. Yang (2010) formulated a new Bat Algorithm for continuous constrained optimization problems. Though the implementation is more complicated than many other metaheuristic algorithms, however, it does show that it utilizes a balanced combination of the advantages of existing successful algorithms with innovative feature based on the echolocation behavior of bats. New solutions are generated by adjusting frequencies, loudness, and pulse emission rates, while the proposed solution is accepted or does not depend on the quality of the solutions controlled or characterized by loudness and pulse rate which are in turn related to the closeness or the fitness of the locations/solution to the global optimal solution. Moreover, the author argues that PSO and harmony search are the special cases of the Bat Algorithm under appropriate simplifications.

## 2.7 Conclusions

In this chapter, we focus on four popular bio-inspired computation algorithms, which are, respectively, ACO, PSO, ABC, and DE. PSO and ACO are currently the most popular algorithms in the swarm intelligence domain. Dorigo and his colleagues introduced the first ACO algorithms in the early 1990s, which is a metaheuristic suitable for solving hard combinatorial optimization problems. The inspiring source of ACO is the pheromone trail laying and following behavior of real ants, which use pheromones as a communication medium. Ants are social insects, being interested mainly in the colony survival rather than individual survival. Of interests is ants' ability to find the shortest path from their nest to food. This idea was the source of the algorithms inspired from ants' behavior. The initial ideas on particle swarms of Kennedy and Eberhart were essentially aimed at producing computational intelligence by exploiting simple analogues of social interaction, which soon developed into a powerful optimization method-PSO. Unlike in the other evolutionary computation techniques, each particle in PSO is also associated with a velocity. Particles fly through the search space with velocities, which are dynamically adjusted according to their historical behaviors. Therefore, the particles have the tendency to fly towards the better and better search area over the course of search process. ABC was originally presented by Karaboga and Basturk, under the inspiration of collective behavior on honeybees, and it has been proved to possess a better performance in function optimization problem. Compared with the usual algorithms, the major advantage of ABC algorithm lies in that it conducts both global search and local search in each iteration, and as a result the probability of finding the optimal parameters is significantly increased, which efficiently avoid local optimum to a large extent. The DE algorithm emerged as a very competitive form of evolutionary computing more than a decade ago. DE has been seen as an attractive optimization tool for continuous optimization for the following reasons: (1) simple and straightforward for implementation, (2) better performance (compared with several other algorithms in solving a variety of problems including unimodal, multimodal, separable, non-separable, and so on), (3) few control parameters, and (4) less space complexity. Besides, we have also given a brief introduction of other bio-inspired computation algorithms such as GSM, BFO, and BA.

## References

Bavelas A (1950) Communication patterns in task-oriented groups. J Acoust Soc Am 22(6):725–730

Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press, New York

Bullnheimer B, Hartl RF, Strauss C (1999) A new rank based version of the Ant System: a computational study. Central European J Operations Res Econom 7(1):25–38

Clerc M, Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evolut Comput 6(1):58–73

Colorni A, Dorigo M, Maniezzo V (1991) Positive feedback as a search strategy. Techn Rep, politecnico di milano

Crina G, Ajith A (2006) Stigmergic optimization: inspiration, technologies and perspectives. In: Stigmergic optimization. Springer Berlin Heidelberg, pp 1–24

Deneubourg J-L, Aron S, Goss S, Pasteels JM (1990) The self-organizing exploratory pattern of the argentine ant. J Insect Behav 3(2):159–168

Dorigo M (1992) Optimization, learning and natural algorithms. PhD Thesis, Politecnico di Milano, Italy

Dorigo M, Blum C (2005) Ant colony optimization theory: a survey. Theor Comput Sci 344(2):243–278

Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evolut Comput 1(1):53–66

Dorigo M, Stützle T (2003) The ant colony optimization metaheuristic: algorithms, applications, and advances. In: Glover F, Kochenberger GA (eds) Handbook of Metaheuristics. Springer, Boston, MA, pp 250–285

Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern B Cybern 26(1):29–41

Dornhaus A, Klügl F, Puppe F, Tautz J (1998) Task selection in honeybees-experiments using multi-agent simulation. In: Proceedings of The Third German Workshop on Artificial Life, Bochum. Verlag Harry Deutsch, pp 171–183

Duan H (2005) Ant colony algorithms: theory and applications. Science Press, Beijing, China

Duan H (2010) Ant colony optimization: principle, convergence and application. In: Bijaya Ketan Panigrahi, Yuhui Shi, Lim M-H (eds) Handbook of Swarm Intelligence. Springer Berlin Heidelberg, pp 373–388

Duan H, Liu S (2010) Non-linear dual-mode receding horizon control for multiple unmanned air vehicles formation flight based on chaotic particle swarm optimisation. IET Control Theory Appl 4(11):2565–2578

Duan H, Xing Z (2009) Improved quantum evolutionary computation based on particle swarm optimization and two-crossovers. Chin Phys Lett 26(12):120304

Duan H, Xu C, Xing Z (2010) A hybrid artificial bee colony optimization and quantum evolutionary algorithm for continuous optimization problems. Int J Neural Syst 20(01):39–50

Duan H, Zhang X, Xu C (2011) Bio-inspired computing. Science Press, Beijing, China

Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya. IEEE, pp 39–43

Grosan C, Abraham A (2011) Swarm intelligence. In: Intelligent systems: a modern approach. Springer, Berlin, Heidelberg, pp 409–422

Heppner F, Grenander U (1990) A stochastic nonlinear model for coordinated bird flocks. In: Krasner S (ed) The ubiquity of chaos. AAAS Publications, Washington, DC, pp 233–238

Karaboga D, Akay B (2009) A comparative study of artificial bee colony algorithm. Appl Math Comput 214(1):108–132

Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39(3):459–471

Kennedy J (1998) The behavior of particles. Evolutionary programming VII. In: David Hutchison, Takeo Kanade, Josef Kittler (eds) Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp 579–589

Kennedy J (1999) Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Proceedings of the 1999 Congress on Evolutionary Computation, Washington, DC. IEEE, pp 1931–1938

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ. IEEE, pp 1942–1948

Kennedy J, Mendes R (2002) Population structure and particle swarm performance. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02), Honolulu, HI. IEEE, pp 1671–1676

Krishnanand K, Ghose D (2009) Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. Swarm Intell 3(2):87–124

Lampinen J, Zelinka I (2000) On stagnation of the differential evolution algorithm. In: Proceedings of 6th International Mendel Conference Soft Computing, Brno, Czech Republic. pp 76–83

Menzel R, De Marco RJ, Greggers U (2006) Spatial memory, navigation and dance behaviour in Apis mellifera. J Comp Physiol A 192(9):889–903

Millonas MM (1994) Swarms, phase transitions, and collective intelligence. In: Artificial life III. Reading, MA. Addison-Wesley, pp 417–445

Neri F, Tirronen V (2008) On memetic differential evolution frameworks: a study of advantages and limitations in hybridization. In: Proceedings of IEEE Congress on Evolutionary Computation, 2008 (IEEE World Congress on Computational Intelligence), Hong Kong. IEEE, pp 2135–2142

Passino KM (2002) Biomimicry of bacterial foraging for distributed optimization and control. IEEE Contr Syst 22(3):52–67

Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization: an overview. Swarm Intell 1(1):33–57

Price K, Storn R (1997) Differential evolution–a simple evolution strategy for fast optimization. Dr Dobb's J 22(4):18–24

Reynolds CW (1987) Flocks, herds and schools: a distributed behavioral model. Comput Graphics 21(4):25–34

Seeley TD (1985) Honeybee ecology: a study of adaptation in social life. Princeton University Press, Princeton

Shi Y, Eberhart R. (1998) A modified particle swarm optimizer. In: Proceedings of The 1998 IEEE International Conference on Evolutionary Computation (IEEE World Congress on Computational Intelligence), Anchorage, AK. IEEE, pp 69–73

Storn R, Price K (1995) Differential Evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces. Techn Rep International Computer Science Institute, Berkeley, CA

Stutzle T, Hoos H (1997) MAX-MIN ant system and local search for the traveling salesman problem. In: Proceeding of IEEE Conference on Evolutionary Computation, Indianapolis, IN. IEEE, pp 309–314

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evolut Comput 1(1):67–82

Xu C, Duan H, Liu F (2010) Chaotic artificial bee colony approach to Uninhabited Combat Air Vehicle (UCAV) path planning. Aerosp Sci Technol 14(8):535–541

Yang X-S (2010) A new metaheuristic bat-inspired algorithm. In: Gonzalez JR et al. (eds) Nature inspired cooperative strategies for optimization (NISCO 2010). Studies in computational intelligence, vol 284. Springer, Berlin, pp 65–74

Yu J, Duan H (2012) Artificial Bee Colony approach to information granulation-based fuzzy radial basis function neural networks for image fusion. Optik 124(17):3103–3111

Zielinski K, Weitkemper P, Laur R, Kammeyer K-D (2006) Parameter study for differential evolution using a power allocation problem including interference cancellation. In: Proceedings of IEEE Congress on Evolutionary Computation, Vancouver, BC. IEEE, pp 1857–1864