# Chapter 17
# Emerging Biology-based CI Algorithms

**Abstract** In this chapter, a group of (more specifically 56 in total) emerging biology-based computational intelligence (CI) algorithms are introduced. We first, in Sect. 17.1, describe the organizational structure of this chapter. Then, from Sects. 17.2 to 17.57, each section is dedicated to a specific algorithm which falls within this category, respectively. The fundamentals of each algorithm and their corresponding performances compared with other CI algorithms can be found in each associated section. Finally, the conclusions drawn in Sect. 17.58 closes this chapter.

## 17.1 Introduction

Several novel biology-based algorithms were detailed in previous chapters. In particular, Chap. 2 detailed the bacteria inspired algorithms, Chap. 3 was dedicated to the bat inspired algorithms, Chap. 4 discussed the bee inspired algorithms, Chap. 5 introduced the biogeography-based optimization algorithm, Chap. 6 was devoted to the cat swarm optimization algorithm, Chap. 7 explained the cuckoo inspired algorithms, Chap. 8 focused on the luminous insect inspired algorithms, Chap. 9 concentrated on the fish inspired algorithms, Chap. 10 targeted on the frog inspired algorithms, Chap. 11 studied the fruit fly optimization algorithm, Chap. 12 addressed the group search optimizer algorithm, Chap. 13 worked on the invasive weed optimization algorithm, Chap. 14 covered the music inspired algorithms, Chap. 15 talked about the imperialist competition algorithm, and Chap. 16 described the teaching-learning-based optimization algorithm. Apart from those quasi-mature biology principles inspired CI methods, there are some emerging algorithms also fall within this category. This chapter collects 56 of them that are currently scattered in the literature and organizes them as follows:

- Section 17.2: Amoeboid Organism Algorithm.
- Section 17.3: Artificial Searching Swarm Algorithm.

B. Xing and W.-J. Gao, *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms*, Intelligent Systems Reference Library 62,
DOI: 10.1007/978-3-319-03404-1_17, © Springer International Publishing Switzerland 2014

- Section 17.49: Society and Civilization Algorithm.
- Section 17.50: Stem Cells Optimization Algorithm.
- Section 17.51: Stochastic Focusing Search Algorithm.
- Section 17.52: Swallow Swarm Optimization.
- Section 17.53: Termite-hill Algorithm.
- Section 17.54: Unconscious Search.
- Section 17.55: Wisdom of Artificial Crowds.
- Section 17.56: Wolf Colony Algorithm.
- Section 17.57: Wolf Pack Search.

The effectiveness of theses newly developed algorithms are validated through the testing on a wide range of benchmark functions and engineering design problems, and also a detailed comparison with various traditional performance leading CI algorithms such as particle swarm optimization (PSO), genetic algorithm (GA), differential evolution (DE), evolutionary algorithm (EA), fuzzy system (FS), ant colony optimization (ACO), and simulated annealing (SA).

## 17.2 Amoeboid Organism Algorithm

In this section, we will introduce an emerging CI algorithm that is derived from the amoeboid related studies (Reece et al. 2011).

### *17.2.1 Fundamentals of Amoeboid Organism Algorithm*

Amoeboid organism algorithm (AOA) was recently proposed in Zhang et al. (2007, 2013a) and Nakagaki et al. (2000). To implement AOA for find the shortest path problem, the following steps need to be performed (Zhang et al. 2007, 2013a; Nakagaki et al. 2000):

- Step 1: Removing the edges with conductivity equals to zero.
- Step 2: Calculating the pressure of each node based on each node's current conductivity and length which can be obtained through Eq. 17.1 (Zhang et al. 2007, 2013a; Nakagaki et al. 2000):

$$\sum_i \frac{D_{ij}}{L_{ij}}\left(p_i - p_j\right) = \begin{cases} -1 & j = 1 \\ 1 & j = 2 \\ 0 & \text{otherwise} \end{cases}. \tag{17.1}$$

- Step 3: Using the pressure of each node acquired via Step 2 to compute each node's conductivity based on Eq. 17.2 (Zhang et al. 2007, 2013a; Nakagaki et al. 2000:

$$Q_{ij} = \frac{D_{ij}}{L_{ij}}\left(p_i - p_j\right), \tag{17.2}$$

where $p_i$ represents the pressure at the node $N_i$, $D_{ij}$ denotes the conductivity of the edge $M_{ij}$, and $Q_{ij}$ is used to express the flux through tube $M_{ij}$ from $N_i$ to $N_j$.

- Step 4: Evaluating the value of each edge's conductivity. If it equals to 1, moving to Step 5; otherwise, jumping to Step 7.
- Step 5: Calculating the next time flux and conductivity based on the current flux and conductivity value via Eq. 17.3 (Zhang et al. 2007, 2013a; Nakagaki et al. 2000):

$$\begin{aligned} &\sum_i Q_{i1} + I_0 = 0 \\ &\sum_i Q_{i1} - I_0 = 0 \\ &\tfrac{d}{dt} D_{ij} = f\left(\left|Q_{ij}\right|\right) - rD_{ij} \end{aligned} \quad . \tag{17.3}$$

- Step 6: Returning to Step 1.
- Step 7: Outputting the solution and terminating the algorithm.

### *17.2.2 Performance of AOA*

Six benchmark test problems with various dimensions were employed in Zhang et al. (2013a) to test the performance of the proposed AOA. From the simulation results it can be observed that AOA was able to find the optimal solutions for all cases, in particular, AOA offers better results that are reported so far in the literature.

## 17.3 Artificial Searching Swarm Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the simulation of the natural biology system.

### *17.3.1 Fundamentals of Artificial Searching Swarm Algorithm*

Artificial searching swarm algorithm (ASSA) was recently proposed in Chen (2009), Chen et al. (2009a, b, c, 2010a). The procedures of implementing ASSA are outlined as below (Chen 2009):

- Step 1: Setting up the parameters, generating the initial population, and evaluating the fitness value.
- Step 2: Dealing with the individual swarm member in turn as follows: Moving toward the calling peer by one step if a signal is received from such peer; otherwise implementing the reconnaissance mechanism. Sending a signal to other peers if a better is found; otherwise moving one step randomly.
- Step 3: Calculating the fitness value and comparing it with the best value found so far.
- Step 4: Checking whether the terminating criterion is met. If yes, stopping the algorithm; otherwise, going back to Step 2.

### *17.3.2 Performance of ASSA*

Chen (2009) tested the ASSA on a typical optimal design optimization problem for the purpose of verifying its effectiveness. The preliminary experimental results showed that ASSA outperforms GA and offers better solution quality. Chen (2009) claimed at the end of the study that the small swarm size will help ASSA to achieve a good searching capability.

## 17.4 Artificial Tribe Algorithm

In this section, we will introduce an emerging CI algorithm that is inspired by the natural tribe's survival mechanism (Magstadt 2013).

### *17.4.1 Fundamentals of Artificial Tribe Algorithm*

Artificial tribe algorithm (ATA) was recently proposed in Chen et al. (2012). The basic inspiration of ATA is renewing the tribe through the strategies of propagation and migration, and relocating the tribe by moving to a better living environment if the current one is getting worse. The two unique characteristics of ATA make it different to other popular swarm intelligence techniques: First, if the present living condition is good, the tribe will tend to propagate, through propagation strategy, the nest generation which is similar to the feature found in genetic algorithm; Second, on the contrary, if the current living situation is bad, the tribe will intend to relocate, by using migration strategy, to another place. Once they are settled, the tribe will continue to propagate. This feature of ATA and the position changing policy used in PSO are alike. Built upon the aforementioned concepts, the running flow of ATA can be described as follows (Onwubolu 2006; Chen et al. 2006, 2012; Coelho and Bernert 2009):

- Step 1: Setting parameters, initializing the tribe, and computing the fitness value.
- Step 2: Adding one to iteration counter, evaluating the current living condition of the tribe, and making decisions according to a simple rule (i.e., if living condition is good, then propagation; otherwise, migration).
- Step 3: Calculating the fitness value.
- Step 4: Determining whether the terminating criteria is met (if so, then stopping the iteration; otherwise, returning to Step 2).

### *17.4.2 Performance of ATA*

Seven benchmark test functions were employed in Chen et al. (2012) to test the performance of the proposed ATA. From the simulation results it can be observed that the tribe size is an important factor for a successful implementation of ATA. In general, the larger size we set for a tribe, the better performance we can obtain but with the cost of a reduced ATA's efficiency. On the other hand, the ATA is able to run fast with a small tribe size but which unfortunately results in low population diversity.

## 17.5 Backtracking Search Algorithm

In this section, we will introduce an emerging CI algorithm that simulates the movement exhibited by an migrating organism, namely, Brownian-like random-walk (Bolstad 2012; Durrett 1984; Shlesinger et al. 1999).

### *17.5.1 Fundamentals of Backtracking Search Algorithm*

Backtracking search algorithm (BSA) was originally proposed in Civicioglu (2013). The motivation of developing BSA is to design simpler and more effective search algorithms. Therefore, unlike many other optimization algorithms, BSA has only one controlling variable and its initial value also does affect the BSA's overall problem-solving ability. To implement BSA, the following five processes need to be performed (Civicioglu 2013):

- Process 1: Initialization. In BSA, the initial population $P$ can be defined through Eq. 17.4 (Civicioglu 2013):

$$P_{i,j} \sim U\left(low_j, up_j\right), \quad i = 1, 2, \ldots, N \text{ and } j = 1, 2, \ldots, D, \tag{17.4}$$

where the population size and problem dimension are denoted by $N$ and $D$, respectively, $U$ represents a uniform distribution, and $P_i$ stands for a target individual in the population $P$.

- Process 2: Selection-I. In BSA, the historical population *oldP* is determined at this stage for computing the search direction. The initial historical population is computed through Eq. 17.5 (Civicioglu 2013):

$$oldP_{i,j} \sim U\left(low_j, up_j\right), \quad i = 1, 2, \ldots, N \text{ and } j = 1, 2, \ldots, D. \tag{17.5}$$

  At the start of each iteration, an *oldP* redefining mechanism is introduced in BSA through the if-then rule defined by Eq. 17.6 (Civicioglu 2013):

$$\text{if } a < b \text{ then } oldp := P | a, b \sim U(0, 1), \tag{17.6}$$

  where := denotes the updating operation.
- Process 3: Mutation. At this stage, the initial form of the trial population *Mutant* is created by Eq. 17.7 (Civicioglu 2013):

$$Mutant = P + F \cdot (oldP - P). \tag{17.7}$$

- Process 4: Crossover. The final form of the trial population $T$ is generated at this stage.
- Process 5: Selection-II. At this step, a set of $T_{iS}$ which have better fitness values than the corresponding $P_{iS}$ are utilized to renew the $P_{iS}$ according to a greedy selection mechanism.

### *17.5.2 Performance of BSA*

To verify the proposed BSA, Civicioglu (2013) employed 3 test function sets in which the Set-1 involves 50 widely recognized benchmark functions, the Set-2 contains 25 benchmark problems that used in CEC 2005, and the Set-3 consists of three real-world cases used in CEC 2011. Through a detailed comparison and analysis, the results showed that BSA can solve a greater number of benchmark problems and can offer statistically better outcomes than its competitors.

## 17.6 Bar Systems Algorithm

In this section, we will introduce an emerging CI algorithm that is based on a common phenomenon observed from human social life (Ramachandran 2012a, b, c; Carlson 2013).

### 17.6.1 Fundamentals of Bar Systems Algorithm

Bar systems (BSs) algorithm was recently proposed in Acebo and Rosa (2008). The BSs algorithm was inspired by the social behaviour of the staffs or bartenders, and can be enclosed in the broader class of swarm intelligence. In the bar, bartenders have to act in a highly dynamic, asynchronous and time-critical environment, and no obvious greedy strategy (such as serving first the best customer, serving first the nearest customer or serving first the customer who has arrived first) gives good results (Acebo and Rosa 2008). Thus, the multi-agent system provides a good framework to rise to the challenge of developing a new class of adaptive and robustness systems.

In general, the crucial step in BSs algorithm is the choice of the task which the agent has to execute for the next time step. In BSs, acting as bartenders, agents operate concurrency into the environment in a synchronous manner; execute the task where they should pour the drinks. After an initial phase, the "bartenders" make their decisions according to the different problem-dependent properties (e.g., weight, speed, location, response time, maximum load, etc.), instead of making decisions randomly. Over time, if an agent is unable to adapt the environment to the preconditions of the task (such as the cost for agent to execute the task in the current state of the environment) or if it is unable to carry the task out by itself then it will be eliminated. Briefly, the BSs algorithm can be defined as a quadruple $(E, T, A, F)$ where (Acebo and Rosa 2008):

- $E$ is a (physical or virtual) environment. The state of the environment at each moment is determined by a set of state variables $(V_E)$. One of those variables is usually the time, due to the major objective of bartenders is to keep the customers waiting for a shorter time. The set of all possible states of the environment is defined as $S$ which is the set of all the possible simultaneous instantiations of the set of state variables $(V_E)$.
- $T = \{t_1, t_2, \ldots, t_M\}$ is a set of tasks to be accomplished by the agents within the environment. Each task $(t_i)$ has associated: $pre(t_i)$ denotes a set of preconditions over $V_E$ which determine whether the task $(t_i)$ can be done; $imp(t_i)$ stands for a non-negative real value which reflects the importance of the task $(t_i)$; and $urg(t_i)$ denotes a function of $V_E$ which indicates the urgency of task $(t_i)$ in the current state of the environment. It will usually be a non-decreasing function of time.
- $A = \{a_1, a_2, \ldots, a_N\}$ is a set of agents situated into the environment. Each agent $(a_i)$ can have different objective (e.g., weight, speed, location, response time, maximum load, etc.). A cost, $\text{cost}(a_i, t_i)$, is associated with each agent. If an agent is unable to adapt the environment to the preconditions of the task or if it is unable to carry the task out by itself, then the $\text{cost}(a_i, t_i)$ will be defined as infinite. In general, this cost can be divided in two parts: the cost for $a_i$ to make the environment fulfil the preconditions of task $(t_i)$, usually this can include the cost of stop doing his current tasks; and the cost for $a_i$ to actually execute $t_j$.
- $F : S \times A \times T \rightarrow \mathbf{R}$ is the function which reflects the degree to which agents are "attracted" by tasks. Overall, given a state of the environment, an agent and a

task, $F(s, a_i, t_i)$, must be defined in a way such that it increases with $imp(t_i)$ and $urg(t_i)$ and it decreases with cost$(a_i, t_i)$.

### 17.6.2 Performance of BSs

At the end of their work, Acebo and Rosa (2008) tested the applicability and efficiency of the proposed BSs algorithm on a NP-hard problem in which a group of loading robots in a commercial harbour has to be well scheduled so that all required containers are transported to the targeted ship while keeping the transportation cost as low as possible. The experiments results indicated that BSs can provide much better results than other greedy algorithms.

## 17.7 Bean Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on some studies of bean (Marcus 2013; Sizer and Whitney 2014; Maathuis 2013; Reece et al. 2011).

### 17.7.1 Fundamentals of Bean Optimization Algorithm

Bean optimization algorithm (BeOA) was recently proposed in Zhang et al. (2008b, 2013b, c). It has shown good performance in solving some difficult optimization problems such as travelling salesman problem (Zhang et al. 2012a; Li 2010) and scheduling problem (Zhang et al. 2010; Wang and Cheng 2010).

Just like other CI algorithms, a potential solution of problem space is firstly encoded into BeOA representation of search space. Situation of each individual bean can thus be expressed as vector like $X = \{x_1, x_2, x_3, \ldots, x_n\}$ indicating the current state of each bean, where $n$ is determined by the scale of problem to be resolved. The environment in which the beans are sown is mainly the solution space and the states of other beans. The basic equation of implementing BeOA is shown in Eq. 17.8 (Zhang et al. 2012a):

$$X[i] = \begin{cases} X[i] & \text{if } X[i] \text{ is a father bean} \\ X_{mb} + \text{Distribution}(X_{mb}) \cdot A & \text{if X[i] is not a father bean} \end{cases}, \tag{17.8}$$

where $X[i]$ is the position of bean $i$, $X_{mb}$ is the position of the father bean. Distribution$(X_{mb})$ is the random variable with a certain distribution of father bean in order to get the positions of its descendants. Parameter $A$ can be set according to the range of the problem to be resolved.

In addition, when the descendant beans finished locating, their fitness values are to be evaluated. The beans with most optimal fitness value will be selected as the

candidates of father beans in the next generation. The candidates of father beans should also satisfy the condition that the distance between every two father beans should be larger than the distance threshold. This condition assures that the father beans can have a fine distribution to avoid premature convergence and enhance the performance of the BeOA for global optimization. If all the conditions can be satisfied, the candidate can be set as the father bean for next generation.

### *17.7.2 Performance of BeOA*

In general, the BeOA shares many common points inspired from models of the natural evolution of species. For example, they are population-based algorithms that use operators inspired by population genetics to explore the search space (the most typical genetic operators are reproduction, mutation, and crossover). In addition, they update the population and search for the optimum with random techniques. Differences among the different biology-based CI algorithms concern the particular representations chosen for the individuals and the way genetic operators are implemented. For example, unlike GA, BeOA does not use genetic operators like mutation, they update themselves with distance threshold.

## 17.8 Bionic Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on studies related to the bionic research (Levin 2013a, b, c, d, e, f).

### *17.8.1 Fundamentals of Bionic Optimization Algorithm*

Bionic optimization (BO) algorithm was recently proposed in Song et al. (2013) for dealing with turbine layout optimization problem in a wind farm. The core concept of BO is to treat each turbine as an individual bion, attempting to be repositioned where its own power outcomes can be increased. There are several BO related studies available in the literature (Zang et al. 2010; Steinbuch 2011; Wei 2011). In Song et al. (2013), the authors defined the BO as a two-stage optimization process in which the Steps 1–6 are included in the Stage 1 and the Stage 2 contains the Steps 7–11. The detailed descriptions about each corresponding step are provided as below (Song et al. 2013):

- Step 1: When a turbine is being added to an existing wind farm, an evaluation function will be employed to assess each discretized points for the newly introduced turbine. In Song et al. (2013), the evaluation function is defined by Eq. 17.9:

$$E(\mathbf{x}) = -\frac{P(\mathbf{x})}{P_{\max}} + \sum_{i=1}^{N} D(\|\mathbf{x} - \mathbf{x}_i\|). \tag{17.9}$$

Calculating $u(\mathbf{x})$ through Eq. 17.10 to obtain the flow field for empty layout (Song et al. 2013).

$$P(\mathbf{x}) = F(u'(\mathbf{x})) = F(u(\mathbf{x})[1 - \beta c(\mathbf{x})]). \tag{17.10}$$

- Step 2: Computing the evaluation values for all the discretized points through Eq. 17.11 (Song et al. 2013):

$$E(\mathbf{x}) = -\frac{P(\mathbf{x})}{P_{\max}} + \sum_{i=1}^{N} D(\|\mathbf{x} - \mathbf{x}_i\|). \tag{17.11}$$

- Step 3: Adding a turbine at the point where the evaluation value is the least.
- Step 4: Terminating the Stage 1 if the turbine numbers pass a specified boundary.
- Step 5: Through the particle model mechanism, simulating the wake flow for all turbines and computing $c(\mathbf{x})$ through Eq. 17.12 (Song et al. 2013):

$$P(\mathbf{x}) = F(u'(\mathbf{x})) = F(u(\mathbf{x})[1 - \beta c(\mathbf{x})]). \tag{17.12}$$

- Step 6: Going back to Step 2.
- Step 7: Since the wake flow created by the later added turbines could still influence the former existing turbines, there is a necessity to further optimize the layout. At this step, one turbine with the same order as in the adding process will be removed.
- Step 8: Calculating the wake flow through the particle model mechanism.
- Step 9: Computing the evaluation values for all points.
- Step 10: Re-adding a turbine into the layout at the point with the least evaluation value.
- Step 11: Going back to Step 7.

### *17.8.2 Performance of BO*

In BO, the layout adjustment strategy within each step is controlled by the evaluation function without any randomness which make BO require much less computational time in comparison with other CI algorithm, e.g., GA and PSO. Through several case studies such as flat terrain scenario, complex terrain scenario, and grid dependency of time cost context, Song et al. (2013) claimed at the end of their study that, for the considered cases, the BO produced better solution quality, in particular for complex terrain case.

## 17.9 Blind, Naked Mole-Rats Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the blind naked mole-rats' social behaviour in looking for food resources and preventing the whole colony from the potential invasions (Mills et al. 2010).

### *17.9.1 Fundamentals of Blind, Naked Mole-Rats Algorithm*

Blind, naked mole-rats (BNMR) algorithm was recently proposed by Taherdangkoo et al. (2012a). For the purpose of simplification, the BNMR algorithm does not distinguish the soldier moles from the employed moles, i.e., these two types of moles are simply placed in one single group which is called employed moles in BNMR. To implement BNMR algorithm, the following steps need to be performed (Taherdangkoo et al. 2012a):

- First, randomly generating the initial population of the blind naked mole-rats colony across the whole problem space. In BNMR, the number of the population is designed twice as much as the food resources where each of the food resources denotes a response for target problem space. According to Taherdangkoo et al. (2012a), some parameters can be defined by Eq. 17.13:

$$x_i = x_i^{\min} + \beta\left(x_i^{\max} - x_i^{\min}\right), \quad i = 1, \ldots, S, \tag{17.13}$$

  where $x_i$ denotes the $i$th food source, $\beta$ represents a random variable which falls within $[0, 1]$, and $S$ is the total number of food sources.
- In addition, the underground temperature is also taken into account as defined by Eq. 17.14 (Taherdangkoo et al. 2012a):

$$\begin{aligned} &H(x) = \rho(x)C(x)\tfrac{\Delta T(x,t)}{\Delta t} \\ &\rho C = f_s(\rho C)_s + f_a(\rho C)_a + f_w(\rho C)_w, \\ &f_s + f_a + f_w = 1 \end{aligned} \tag{17.14}$$

  where $H(x)$ stands for the soil temperature which changes with the depth $x$, $\rho(x)$ and $C(x)$ denotes the soil's thermal properties (e.g., the density and the specific heat capacity). Although $\rho(x)$ and $C(x)$ are variables vary with the changing of environment, in BNMR, they are treated as constant which falls within $[2, 4]$, $\Delta T(x, t)/\Delta t$ is the rate of the soil temperature varying with the time, $f$ stands for the volumetric contribution of each element in the compound, and the three subscripts (i.e., $s$, $a$, and $w$) indicate the soil components (e.g., sand, air, and water).
- During the search of neighbours for food sources, the attenuation coefficient $A$ has to be updated in each iteration. The Eq. 17.15 is used to express such fact (Taherdangkoo et al. 2012a):

$$A_i^t = A_i^{t-1}\left[1 - \exp\left(\frac{-\alpha t}{T}\right)\right], \tag{17.15}$$

where $\alpha$ denotes a random number which falls within $[0, 1]$ (in BNMR, a fixed value of $\alpha = 0.95$ is employed for simplicity), and $t$ represents the iteration step.

- Then, for each food source, two employed moles will be dispatched. The acquired food sources are grouped by queen mole according to the probability of $P$ which is calculated via Eq. 17.16 (Taherdangkoo et al. 2012a):

$$P_i = \frac{Fitness_i = FS_i \times R_i}{\sum_{j=1}^{N} Fitness_j}, \tag{17.16}$$

where $Fitness_i$ is assessed by its employed moles, $FS_i$ is relative to the best food sources, $R_i$ represents the route to the food source, and $N$ stands for the food sources number.

- Finally, BNMR algorithm also takes the colony defence into account which is calculated through Eq. 17.17 (Taherdangkoo et al. 2012a):

$$B_i^t = \zeta \times B_i^{t-1}, \tag{17.17}$$

where $\zeta$ is a user defined coefficient $(\zeta \geq 1)$, and $B_i^t$ denotes the number of eliminated points for the $i$th food source during the $t$th iteration.

### *17.9.2 Performance of BNMR*

In order to show how the BNMR algorithm performs, Taherdangkoo et al. (2012a) used 24 benchmark test functions such as Shifted Sphere function, Shifted Rotated High Conditioned Elliptic Function, Shifted Rosenbrock's Function, Shifted Rotated Griewank's Function without Bounds, and Shifted Rastrigin's Function. Compared with other CI techniques (e.g., GA, PSO, SA, etc.), the BNMR algorithm has better convergence than its competitive algorithms which demonstrates that BNMR is capable of getting out of local minimum in the problem space and reaching the global minimum.

## 17.10 Brain Storm Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the outputs of human brain related research (Gross 2014; Wilson 2013; Taylor 2012).

### 17.10.1 Fundamentals of Brain Storm Optimization Algorithm

Brain storm optimization algorithm (BSOA) was recently proposed by Shi (2011a). Since the human beings are among on of the most intelligent social animals on earth, the BSO was engineered to have the ability of both convergence and divergence. The process of brainstorming (or brainwaving) is often utilized in dealing with a set of complicated problems which are not always solvable for an individual person. A detailed description about the natural human being brain storm process can be found in Shi (2011b). A typical brain storm process generally follows the eight steps (Shi 2011b; Xue et al. 2012; Zhan et al. 2012; Zhou et al. 2012; Krishnanand et al. 2013):

- Step 1: Getting together a brainstorming group of people with as diverse background as possible.
- Step 2: Generating many ideas according to the four principles (i.e., suspend judgment, anything goes, cross-fertilize, and go for quantity) of idea generation guidance.
- Step 3: Having several customers act as the owners of the problem to pick up a couple of ideas as better ideas for solving the targeted problem.
- Step 4: Using the fact that the ideas (selected in Step 3) enjoy a higher chosen probability than their competitor ideas as an evidence to generate more ideas based again on the four principles.
- Step 5: Having the customers to select several better ideas again as they did in Step 3.
- Step 6: Picking up an object randomly and using the intrinsic characteristics of the object as the indication to create more ideas (still based on the four principles of idea generation guidance).
- Step 7: Letting the customers choose several better ideas as they did in Step 3.
- Step 8: Obtaining a fairly good enough problem solution at the end of the brain storm process.

Although the three-round brain storm process, participated by a group of real human beings, can not last for too long, in a computer simulation environment, we can set the round of idea generation to a very large number as we desire.

### 17.10.2 Performance of BSOA

To test the performance of BSOA, Shi (2011b) chose ten benchmark functions (among them, five are unimodal functions, while the other five are multimodal functions). The simulation results indicated that BSOA algorithm performed reasonably well.

## 17.11 Clonal Selection Algorithm

In this section, we will introduce an emerging CI algorithm that is based on Darwin's evolutionary theory and clone related studies (Gamlin 2009; Mayfield 2013; Woodward 2008; Steinitz 2014).

### *17.11.1 Fundamentals of Clonal Selection Algorithm*

Clonal selection algorithm (CSA) was recently proposed in Castro and Zuben (2000). There are several CSA related variants and applications can be found in the literature (Castro and Zuben 2002; Campelo et al. 2005; Gao et al. 2013; Wang et al. 2009; Batista et al. 2009; Ding and Li 2009; Riff et al. 2013). Interested readers are referred to two excellent reviews (Brownlee 2007; Ulutas and Kulturel-Konak 2011) for updated information. To implement CSA, the following steps need to be performed (Castro and Zuben 2000):

- Step 1: Creating a set of candidate solutions (denoted by $P$), composing of the subset of memory cells (represented by $M$), and adding to the remaining population $(P_r)$, i.e., $P = P_r + M$.
- Step 2: According to an affinity measure, choosing the $n$ best individuals of the population, named $P_n$.
- Step 3: Cloning the population of these $n$ best individuals and giving rise to a intermediate population clones, called $C$. The clone size is regarded as an increasing function of the affinity with the antigen.
- Step 4: Submitting the population of clones to a hypermutation mechanism. A maturated antibody population is then generated and denoted by $C^*$.
- Step 5: Reselecting the improved individuals from $C^*$ to compose the memory set, i.e., $M$.
- Step 6: Replacing $d$ andibodies by novel ones (introduced through diversity strategy). In CSA, the replacement probability of lower affinity cells is in general high.

### *17.11.2 Performance of CSA*

To verify the CSA, three problem sets are considered in Castro and Zuben (2000), namely, binary character recognition task, multimodal optimization problem, and the classic travelling salesman problem. In comparison with GA, the simulation results demonstrated that CSA is a very promising CI algorithm which has showed a fine tractability regarding the computational cost.

## 17.12 Cockroach Swarm Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the behaviours observed through cockroach studies (Bell et al. 2007; Lihoreau et al. 2010; Lihoreau et al. 2012; Chapman 2013; Bater 2007; Reece et al. 2011).

### *17.12.1 Fundamentals of Cockroach Swarm Optimization Algorithm*

Cockroach swarm optimization algorithm (CSOA) was recently proposed in Chen and Tang (2010, 2011; Cheng et al. 2010). The basic concept of CSOA is that located in the $D$-dimensional search space $R^D$, there is a swarm of cockroaches which contains $N$ cockroach individuals. The $i$th individual denotes a $D$-dimensional vector $\mathbf{X}(i) = (x_{i1}, x_{i2}, \ldots, x_{iD})$ for $(i = 1, 2, \ldots, N)$, the location of each individual is a potential solution to the targeted problem. The model of CSOA consists of three behaviours, namely, chase-swarming, dispersing, and ruthless which are explained as below (Chen and Tang 2010, 2011; Cheng et al. 2010).

- Chase-swarm behaviour: Each individual cockroach $\mathbf{X}(i)$ will run after (within its visual range) a cockroach $\mathbf{P}(i)$ which carries the local optimum. This behaviour is modelled as Eq. 17.18 (Chen and Tang 2010, 2011; Cheng et al. 2010):

$$\mathbf{X}'(i) = \begin{cases} \mathbf{X}(i) + step \cdot rand \cdot [\mathbf{P}(i) - \mathbf{X}(i)] & \text{if } \mathbf{X}(i) \neq \mathbf{P}(i) \\ \mathbf{X}(i) + step \cdot rand \cdot [\mathbf{P}_g - \mathbf{X}(i)] & \text{if } \mathbf{X}(i) = \mathbf{P}(i) \end{cases}, \tag{17.18}$$

  where $\mathbf{P}_g = Opt_i\{\mathbf{X}(i),\ i = 1,\ \ldots,\ N\}$ denotes the global optimum individual cockroach, $\mathbf{P}(i) = Opt_j\{\mathbf{X}(j) \| \mathbf{X}(i) - \mathbf{X}(j) \| \leq visual,\ i = 1, \ldots, N$ and $j = 1, \ldots, N\}$, *step* represents a fixed value, and *rand* stands for a random number within the interval of $[0, 1]$.
- Dispersing behaviour: During a certain time interval, each individual cockroach will be randomly dispersed for the purpose of keeping the diversity of the current swarm. This behaviour is modelled through Eq. 17.19 (Chen and Tang 2010, 2011; Cheng et al. 2010):

$$\mathbf{X}'(i) = \mathbf{X}(i) + rand(1,\ D), \quad i = 1, \ldots, N, \tag{17.19}$$

  where $rand(1,\ D)$ is a $D$-dimensional random vector which falls within a certain interval.
- Ruthless behaviour: At a certain time interval, the cockroach which carries the current best value substitute another cockroach in a randomly selection manner.

This behaviour is modelled through Eq. 17.20 (Chen and Tang 2010, 2011; Cheng et al. 2010):

$$\mathbf{X}(k) = \mathbf{P}_g, \tag{17.20}$$

where $k$ is a random integer within the interval of $[1, N]$.

Built on these three behaviours, the working procedure of the CSOA algorithm can be classified into the following steps (Chen and Tang 2010, 2011; Cheng et al. 2010):

- Step 1: Setting parameters and initializing population;
- Step 2: Search $\mathbf{P}(i)$ and $\mathbf{P}_g$;
- Step 3: Performing chase-swarming and updating $\mathbf{P}_g$;
- Step 4: Executing dispersing behaviour and updating $\mathbf{P}_g$;
- Step 5: Running ruthless behaviour;
- Step 6: Checking stopping criterion. If yes, generate output; otherwise, go back to step 2.

### *17.12.2 Performance of CSOA*

In Chen and Tang (2011), the authors made an attempt to employ CSOA to solve vehicle routing problem (VRP), more specifically, the VRP with time windows (VRPTW for short). In general the VRPTW can be stated as follows: Products are to be delivered to a group of customers by a fleet of vehicles from a central depot. The locations of the depot and the customers are known. The object is to find a suitable route which minimizes the total travel distance or cost subject to the constraints listed below.

- Each customer is visited only once by exactly one vehicle;
- Each vehicle has the fixed starting and ending point (i.e., the depot);
- The vehicles are capacitated which means the total demand of any route should not exceed the maximum capacity of an assigned vehicle;
- The visit to a customer is time restrict, i.e., each customer can only be served during a certain time period.

To test the effectiveness of the CSOA for focal problem, Chen and Tang (2011) conducted a study on VRP and VRPTW separately. The experimental results were compared with PSO and the improved PSO. Through the comparison, the authors claimed that CSOA is able to explore the optimum with higher optimal rate and shorter time.

## 17.13 Collective Animal Behaviour Algorithm

In this section, we will introduce a new CI algorithm which inspired by the collective decision-making mechanisms among the animal groups (Sulis 1997; Tollefsen 2006; Nicolis et al. 2003; Schutter et al. 2001; You et al. 2009; Couzin 2009; Aleksiev et al. 2008; Stradner et al. 2013; Zhang et al. 2012b, Niizato and Gunji 2011; Oca et al. 2011; Eckstein et al. 2012; Petit and Bon 2010).

### *17.13.1 Fundamentals of Collective Animal Behaviour Algorithm*

Collective animal behaviour (CAB) algorithm was originally proposed by Cuevas et al. (2013). In CAB, each animal position is viewed as a solution within the search space. Also, a set of rules that model the collective animal behaviours will be employed in the proposed algorithm. The main steps of CAB are outlined below (Cuevas et al. 2013):

- Initializing the population. Generate a set $\mathbf{A}$ of $N_p$ animal positions ($\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{N_p}\}$) randomly in the $D$-dimensional search space as defined by Eq. 17.21 (Cuevas et al. 2013):

$$\begin{aligned} &a_{j,i} = a_j^{low} + rand(0,1) \cdot \left(a_j^{high} - a_j^{low}\right), \\ &j = 1, 2, \ldots, D;\ i = 1, 2, \ldots, N_p. \end{aligned} \tag{17.21}$$

  where $a_j^{low}$ and $a_j^{high}$ represent the lower bound and upper bound, respectively, and $a_{j,i}$ is the $j$th parameter of the $i$th individual.
- Calculating and sorting the fitness value for each position. According to the fitness function, the best position $(B)$ which is chosen from the new individual set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_p}\}$ will be stored in a memory that includes two different elements as expressed in Eq. 17.22 (Cuevas et al. 2013):

$$\begin{cases} \mathbf{M}_g: & \text{for maintaining the best found positions} \\ & \text{in each generation} \\ \mathbf{M}_h: & \text{for storing the best history positions} \\ & \text{during the complete evolutionary process} \end{cases}. \tag{17.22}$$

- Keep the position of the best individuals. In this operation, the first $B$ elements of the new animal position set $\mathbf{A}(\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_B\})$ are generated. This behaviour rule is modelled via Eq. 17.23 (Cuevas et al. 2013):

$$\mathbf{a}_l = \mathbf{m}_h^l + \mathbf{v}, \tag{17.23}$$

where $l \in \{1, 2, \ldots, B\}$ while $\mathbf{m}_h^l$ represents the historic memory $\mathbf{M}_h$, and $\mathbf{v}$ is a random vector holding an appropriate small length.

- Move from or to nearby neighbours. This operation can be defined by Eq. 17.24 (Cuevas et al. 2013):

$$\mathbf{a}_i = \begin{cases} \mathbf{x}_i \pm r \cdot \left(\mathbf{m}_h^{\text{nearnest}} - \mathbf{x}_i\right) & \text{with probability } H \\ \mathbf{x}_i \pm r \cdot \left(\mathbf{m}_g^{\text{nearnest}} - \mathbf{x}_i\right) & \text{with probability } (1 - H) \end{cases}, \tag{17.24}$$

where $i \in \{B+1, B+2, \ldots, N_p\}$, $\mathbf{m}_h^{\text{nearnest}}$ and $\mathbf{m}_g^{\text{nearnest}}$ represent the nearest elements of $\mathbf{M}_h$ and $\mathbf{M}_g$ to $\mathbf{x}_i$, respectively, and $r$ is a random number between $[-1, 1]$.

- Move randomly. This rule is defined by Eq. 17.25 (Cuevas et al. 2013):

$$\mathbf{a}_i = \begin{cases} \mathbf{r} & \text{with probability } P \\ \mathbf{x}_i & \text{with probability } (1 - P) \end{cases}, \tag{17.25}$$

where $i \in \{B+1, B+2, \ldots, N_p\}$, and $\mathbf{r}$ is a random vector defined within the search space.

- Updating the memory. The updating procedure is as follows (Cuevas et al. 2013):
  Two memory elements are merged together as shown in Eq. 17.26 (Cuevas et al. 2013):

$$\mathbf{M}_U \left(\mathbf{M}_U = \mathbf{M}_h \cup \mathbf{M}_g\right). \tag{17.26}$$

  Based on the parameter $(\rho)$, the elements of the memory $\mathbf{M}_U$ is calculated. The $\rho$ value is computed via Eq. 17.27 (Cuevas et al. 2013):

$$\rho = \frac{\prod_{j=1}^{D} \left(a_j^{high} - a_j^{low}\right)}{10 \cdot D}, \tag{17.27}$$

  where $a_j^{low}$ and $a_j^{high}$ represent the pre-specified lower bound and the upper bound, respectively, within a $D$-dimensional space.

- Optimal determination. It is defined by Eq. 17.28 (Cuevas et al. 2013):

$$\text{Th} = \frac{Max_{fitness}(\mathbf{M}_h)}{6}, \tag{17.28}$$

  where Th represents a threshold value that decide which elements will be considered as a significant local minimum, and $Max_{fitness}(\mathbf{M}_h)$ represent the best fitness value among $\mathbf{M}_h$ elements.

### *17.13.2 Performance of CAB*

In order to evaluate the performance of CAB, a set of multimodal benchmark functions were adopted in Cuevas et al. (2013), namely, Deb's function, Deb's decreasing function, Roots function, two dimensional multimodal function, Rastringin's function, Shubert function, Griewank function, and modified Griewank function. Compared with other CI algorithms, computational results showed that CAB outperforms the other algorithms in terms of the solution quality.

## 17.14 Cultural Algorithm

In this section, we will introduce an CI algorithm that is based on the human social evolution (Mayfield 2013).

### *17.14.1 Fundamentals of Cultural Algorithm*

Cultural algorithm (CA) was originally proposed in Reynolds (1994, 1999). There are several variants and application can be found in the literature (Digalakis and Margaritis 2002; Alexiou and Vlamos 2012; Ochoa-Zezzatti et al. 2012; Srinivasan and Ramakrishnan 2012; Silva et al. 2012). In CA, the evolution process can be viewed as a dual-inheritance system in which two search spaces (i.e., the population space and the belief space) are included.

In general, the population space is used to represent a set of behavioural traits associated with each individual. On the other hand, the belief space is used to describe different domains of knowledge that the population has of the search space and it can be delivered into distinct categories, such as normative knowledge, domain specific knowledge, situational knowledge, temporal knowledge, and spatial knowledge. In other words, the belief space is used to store the information on the solution of the problem.

Furthermore, at each iteration, two functions (i.e., acceptance function and influence function) and two operators (i.e., crossover and mutation) are employed to maintain the CA algorithm. The acceptance function is used to decide which knowledge sources influence individuals. On the other hand, the influence function is used to determine which individuals and their behaviours can impact the belief space knowledge. Also, the crossover and mutation operators are used to support the population space that control the beliefs' changes in individuals.

The main steps of CA can be outlined as follows (Reynolds 1994):

- Step 1: Generate the initial population.
- Step 2: Initialize the belief space. In CA, if only two knowledge components, i.e., situational knowledge component and normative knowledge component are employed, the belief space can be defined by Eq. 17.29 (Reynolds 1994, 1999):

$$B(t) = (S(t), N(t)), \tag{17.29}$$

  where the situational knowledge component is represented by $S(t)$, and $N(t)$ denotes the normative knowledge component.

- Step 3: Evaluate the initial population.
- Step 4: Iterative procedure. First, update the belief space (with the individuals accepted). Second, apply the variation operators (under the influence of the belief space). Third, evaluate each child. Fourth, perform selection.
- Step 5: Check termination criteria.

### *17.14.2 Performance of CA*

To verify CA, a set of studies are conducted in Reynolds (1994). The experiments results demonstrated that CA is indeed a very promising solver for dealing with optimization problems.

## 17.15 Differential Search Algorithm

In this section, we will introduce an emerging CI algorithm that simulates the movement exhibited by an migrating organism, namely, Brownian-like random-walk (Bolstad 2012; Durrett 1984; Shlesinger et al. 1999).

### *17.15.1 Fundamentals of Differential Search Algorithm*

Differential search (DS) algorithm was originally proposed in Civicioglu (2012). To implement DS, the following features need to be considered (Civicioglu 2012; Sulaiman 2013):

- Feature 1: In DS, a set of artificial organisms making up a super-organism, namely, $Superorganism_g$, $g = \{1, 2, \ldots, \text{maxgeneration}\}$ in which the number of organisms is equivalent to the size of the problem (i.e., $x_{i,j}$, $j = \{1, 2, \ldots, D\}$).
- Feature 2: In DS, a member of a super-organism (i.e., an artificial organism) in its initial position can be defined through Eq. 17.30 (Civicioglu 2012):

$$x_{i,j} = rand \cdot \left(up_j - low_j\right) + low_j, \tag{17.30}$$

where $X_i = \left[x_{i,j}\right]$ represents a group of artificial organism, and the artificial super-organism can thus be expressed by $Superorganism_g = [X_i]$.

- Feature 3: In DS, the movement style for an artificial super-organism finding a stopover site is modelled by Brownian-like random walk. Several randomly chosen individuals within an artificial super-organism move forward to the targets of donor which equals to $[X_{random\_shuffling(i)}]$ for the purpose of discovering stopover sites which is generated through Eq. 17.31 (Civicioglu 2012):

$$StopoverSite = Superorganism + Scale \cdot (donor - Superorganism). \tag{17.31}$$

- Feature 4: In DS, in order to generate the scale value, a gamma-random number creator (i.e., *randg*) controlled by an uniform-random number creator (i.e., *rand*) and both falling within the range of $[0, 1]$ are employed.
- Feature 5: In DS, the numbers of individual artificial organism to join the stopover site search process are decided in an random manner.
- Feature 6: In DS, if a more fertile stopover site is discovered, a group of artificial organisms will move to the newly founded place, while the artificial super-organism will keep searching.
- Feature 7: There are only two controlling variables (i.e., $p_1$ and $p_2$) are used in DS. Through conducting a set of detailed tests, Civicioglu (2012) suggested the following values (see Eq. 17.32) can provide the best solutions for the respective problems.

$$p_1 = p_2 = 0.3 \cdot rand. \tag{17.32}$$

### 17.15.2 Performance of DS

To verify the proposed DS, Civicioglu (2012) employed two test function sets in which the Test Set-1 consists of 40 benchmark functions (e.g., Shubert function, Stepint function, Trid function, etc.) and the Test Set-2 is composed of 12 benchmark test functions which include such as Shifted Sphere function, Shifted Schwefel's function, Shifted Rastrigin Function, and Shifted Rosenbrock function. In comparison with other 8 widely used optimization algorithms through the use of statistical approaches, the experimental results demonstrated that DS is a very attractive solver for numerical optimization problems. At the end of the study, Civicioglu (2012) further applied DS to the problem of transforming the geocentric cartesian coordinates into geodetic coordinates. Compared with the other 9 classical methodologies and 8 CI algorithms which have been previously reported in dealing with the same problem, the results also confirmed the practicability and high level of accuracy of DS.

## 17.16 Dove Swarm Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on foraging behaviours observed from a dove swarm (Mills et al. 2010).

### *17.16.1 Fundamentals of Dove Swarm Optimization Algorithm*

Dove swarm optimization (DSO) algorithm was recently proposed in Su et al. (2009). The basic working principles of DSO are listed as follows (Su et al. 2009):

- Step 1: Initializing the number of doves and deploying the doves on the 2-dimensional artificial ground.
- Step 2: Setting the number of epochs $(e = 0)$, and the degree of satiety, $f_j^e = 0$ for $j = 1, \ldots, M \times N$. Initializing the multi-dimensional sense organ vector, $\vec{w}_j$ for $j = 1, \ldots, M \times N$.
- Step 3: Computing the total amount of the satiety degrees in the flock, $T(e) = \sum_{j=1}^{M \times N} f_j^e$.
- Step 4: Presenting an input pattern (i.e., piece of artificial crumb) $\vec{x}_k$ to the $M \times N$ doves.
- Step 5: Locating the dove $b_f$ closest to the crumb $\vec{x}_k$ according to the minimum-distance criterion shown in Eq. 17.33 (Su et al. 2009):

$$b_f = \arg\min_j \left\| \vec{x}_k - \vec{w}_j(k) \right\|, \quad \text{for } j = 1, \ldots, M \times N, \tag{17.33}$$

  The dove with the artificial sense organ vector which is the most similar to the artificial crumb, $\vec{x}_k$, is claimed to be the winner.

- Step 6: Updating each dove's satiety degree through Eq. 17.34 (Su et al. 2009):

$$f_j^e(new) = \frac{\left\| \vec{x}_k - \vec{w}_{b_f}(k) \right\|}{\left\| \vec{x}_k - \vec{w}_j(k) \right\|} + \lambda f_j^e(old), \quad \text{for } j = 1, \ldots, M \times N. \tag{17.34}$$

- Step 7: Selecting the dove, $b_f$, with the highest satiety degree based on the following criterion expressed as Eq. 17.35 (Su et al. 2009):

$$b_s = \arg \max_{1 \le j \le M \times N} f_j^e. \tag{17.35}$$

- Step 8: Updating the sense organ vectors and the position vectors via Eqs. 17.36 and 17.37, respectively (Su et al. 2009):

$$\vec{w}_j(k+1) = \begin{cases} \vec{w}_{b_f}(k) + \eta_w\left(\vec{x}_k - \vec{w}_{b_f}(k)\right) & \text{for } j = b_f \\ \vec{w}_j(k) & \text{for } j \neq b_f \end{cases}, \tag{17.36}$$

$$\vec{p}_j(k+1) = \vec{p}_j(k) + \eta_p \beta\left(\vec{p}_{b_s}(k) - \vec{p}_j(k)\right), \quad \text{for } j = 1, \ldots, M \times N. \tag{17.37}$$

- Step 9: Returning to Step 4 until all patterns are processes.
- Step 10: Stopping the whole training procedure if the following criterion (see Eq. 17.38) is met (Su et al. 2009):

$$\left| \sum_{j=1}^{M \times N} f_j^e - T(e) \right| \leq \varepsilon. \tag{17.38}$$

Otherwise, increasing the number of epochs by one ($e = e + 1$), and go back to Step 3 until the pre-defined limit for the number of epochs is met. The satisfaction of the criterion given above means that the total amount of satiety degree has converged to some extent.

### *17.16.2 Performance of DSO*

In general there are two main obstacles encountered in data clustering: the geometric shapes of the clusters are full of variability, and the cluster numbers are not often known a priori. In order to determine the optimal number of clusters, Su et al. (2009) employed DSO to perform data projection task, i.e., projecting high-dimensional data onto a low-dimensional space to facilitate visual inspection of the data. This process allows us to visualize high-dimensional data as a 2-dimensional scatter plot. The basic idea in their work can be described as follows (Su et al. 2009): In a data set, each data pattern, $\vec{x}$, is regarded as a piece of artificial crumb and these artificial crumbs (i.e., data patterns) will be sequentially tossed to a flock of doves on a two-dimensional artificial ground. The flock of doves adjusts its physical movements to seek these artificial crumbs. Individual members of the flock can profit from discoveries of all of the other members of the flock during the foraging procedure because an individual is usually influenced by the success of the best individual of the flock and thus has a desire to imitate the behaviour of the best individual. Gradually, the flock of the doves will be divided into several groups based on the distributions of the artificial crumbs. Those formed groups will naturally correspond to the hidden data structure in the data set. By viewing the distributions of the doves on the 2-dimensional artificial ground, we may quickly find out the number of clusters inherent in the data set. However, many practical data sets have high-dimensional data points. Therefore, the aforementioned idea has to be generalized so that it can process high-dimensional data. In the real world, each dove has a pair of eyes to find out where crumbs are, but in the artificial world, a virtual dove does not have the capability to perceive a

piece of multi-dimensional artificial crumb that is located around it. In order to cope with issue, Su et al. (2009) equipped each dove with functionalities, i.e., a multi-dimensional artificial sense organ represented as a sense organ vector, $\vec{w}$, which has the same dimensionality as a data pattern, $\vec{x}$, and a 2-dimensional position vector, $\vec{p}$, which represents its position on the 2-dimensional artificial ground. In addition to these two vectors, $\vec{w}$ and $\vec{p}$, a parameter called the satiety parameter is also attached to each dove. This special parameter endows a dove with the ability of expressing its present satiety status with respect to the food, that is, a dove with a low degree of satiety will have a strong desire to change its present foraging policy and be more willing to imitate the behaviour of the dove which performs the best among the flock.

To test the performance of DSO, five (two artificial and three real) data sets were selected in the study. These data sets include Two-Ellipse, Chromosomes, Iris, Breast Cancer, and 20-Dimensional Non-Overlapping. The projection capability of DSO was compared with the other popular projection algorithms, e.g., Sammon's algorithm. For DSO, the maximum number of epochs for every data set (excluding Iris and 20-Dimensional data sets) were set to be 5, while for the Iris and 20-Dimensional data sets, were set to be 10 and 20, respectively. The case studies showed that DSO can fulfil the projection task. Meanwhile, the performance of DSO is not so sensitive to the size of dove swarm.

## 17.17 Eagle Strategy

In this section, we will introduce an emerging strategy or search method that is based on the eagle search (hunting) behaviour.

### *17.17.1 Fundamentals of Eagle Strategy*

Eagle strategy (ES) algorithm was proposed in Yang and Deb (2010, 2012) and Gandomi et al. (2012). It is a two-stage method, i.e., exploring the search space globally using Lévy flight random walks and then employing an intensive local search mechanism for optimization, such as hill-climbing and the downhill simplex method. The main steps of ES can be described as follows (Yang and Deb 2010, 2012; Gandomi et al. 2012):

- Step 1: Initialize the population and parameters.
- Step 2: Iterative procedure. First, perform random search in the global search space defined by Eq. 17.39 (Yang and Deb 2010):

$$\text{Lévy} \sim u = t^{-\lambda}, \quad (1 < \lambda \leq 3), \tag{17.39}$$

where $\lambda = 3$ corresponds to Brownian motion, while $\lambda = 1$ has a characteristics of stochastic tunnelling.

Second, evaluate the objective functions. Third, make an intensive local search with a hypersphere via any optimization technique such as downhill simplex (i.e., Nelder-Mead) method. Fourth, calculate the fitness and keep the best solutions. Fifth, increase the iteration counter. Sixth, calculate means and standard deviations.

- Step 3: Post process results and visualization.

### *17.17.2 Performance of ES*

To evaluate the efficiency of ES, the Ackley function is adopted in Yang and Deb (2010). Compared with other CI algorithms (such as PSO and GA), the results showed that ES outperforms the others in finding the global optima with the success rates of 100 %. As all CI algorithms require a balance between the exploration and exploitation, this strategy can be combined into any algorithms [such as firefly algorithm (Yang and Deb 2010) and DE (Gandomi et al. 2012)] to improve the computational results.

## 17.18 Fireworks Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is derived from the explosion process of fireworks, an explosive devices invented by our clever ancestor, which can produce striking display of light and sound (Lancaster et al. 1998).

### *17.18.1 Fundamentals of Fireworks Optimization Algorithm*

Fireworks optimization algorithm (FOA) was recently proposed in Tan and Zhu (2010). The basic idea was when we need to find a point $x_j$ satisfying $f(x_i) = y$, a set of fireworks will be continuously fired in the potential search space until an agent (i.e., a spark in fireworks context) gets to or reasonably close to the candidate point $x_j$. Based on this understanding, to implement FOA algorithm, the following steps need to be performed (Janecek and Tan 2011; Pei et al. 2012; Tan and Zhu 2010):

- Step 1: Fireworks explosion process designing. Since the number of sparks and their coverage in the sky determines whether an explosion is good or not, Tan and Zhu (2010) first defined the number of sparks created by each firework $x_j$ through Eq. 17.40:

$$s_i = m \cdot \frac{y_{\max} - f(x_i) + \xi}{\sum_{i=1}^{n} [y_{\max} - f(x_i)] + \xi}, \tag{17.40}$$

where $m$ is a parameter used to control the total number of sparks created by the $n$ fireworks, $y_{\max} = \max(f(x_i))$ (for $i = 1, 2, \ldots, n$) stands for the maximum value of the objective function among the $y_{\max}$ fireworks, and $\xi$ represents a small constant which is used to avoid zero-division-error. Meanwhile, in order to get rid of the overwhelming effects of the splendid fireworks, bounds $s_i$ are also defined by Eq. 17.41 (Tan and Zhu 2010):

$$\hat{s}_i = \begin{cases} round(a \cdot m) & \text{if } s_i < am \\ round(b \cdot m) & \text{if } s_i > bm,\ a < b < 1, \\ round(s_i) & \text{otherwise} \end{cases} \tag{17.41}$$

where $a$ and $b$ are constant parameters.

Next, Tan and Zhu (2010) also designed the explosion amplitude via Eq. 17.42:

$$A_i = \hat{A} \cdot \frac{f(x_i) - y_{\min} + \xi}{\sum_{i=1}^{n} [f(x_i) - y_{\min}] + \xi}, \tag{17.42}$$

where $\hat{A}$ represents the maximum amplitude of an explosion, and $y_{\min} = \min(f(x_i))$ (for $i = 1, 2, \ldots, n$) denotes the minimum value of the objective function among the $n$ fireworks.

Finally, the directions of the generated sparks are computed using Eq. 17.43 (Tan and Zhu 2010):

$$z = round(d \cdot rand(0, 1)), \tag{17.43}$$

where $d$ denotes the dimensionality of the location $x$, and $rand(0, 1)$ represents an uniformly distributed number within $[0, 1]$.

- Step 2: In order to obtain a good implementation of FOA, the locations of where we want the fireworks to be fired need to be chosen properly. According to Tan and Zhu (2010), the general distance between a location $x$ and other locations can be expressed as Eq. 17.44:

$$R(x_i) = \sum_{j \in K} d(x_i, x_j) = \sum_{j \in K} \|x_i - x_j\|, \tag{17.44}$$

where $K$ denotes a group of current locations of all fireworks and sparks. The selection probability of a location $x_i$ is then defined via Eq. 17.45 (Tan and Zhu 2010):

$$p(x_i) = \frac{R(x_i)}{\sum_{j \in K} R(x_j)}. \tag{17.45}$$

### 17.18.2 Performance of FOA

To validate the performance of the proposed FOA, 9 benchmark test functions were chosen by Tan and Zhu (2010) and the comparisons were conducted among the FOA, the standard PSO, and the clonal PSO. The experiment results indicated that the FA clearly outperforms the other algorithms in both optimization accuracy and convergence speed.

## 17.19 FlockbyLeader Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the leadership pattern found in flocks of pigeon birds (Couzin et al. 2005; Giraldeau et al. 1994).

### 17.19.1 Fundamentals of FlockbyLeader Algorithm

The FlockbyLeader algorithm was proposed by Bellaachia and Bari (2012) in which the recently discovered leadership dynamic mechanisms in pigeon flocks are incorporated in the normal flocking model [i.e., Craig Reynolds' Model (Reynolds 1987)]. In every iteration, the algorithm starts by finding flock leaders. The main steps are illustrated as follows (Bellaachia and Bari 2012):

- Calculating fitness value of each flock leader $(L_i)$ according to the objective function (i.e., $d_{\max}^{L_i}$). It will be defined by Eq. 17.46 (Bellaachia and Bari 2012):

$$d_{\max}^{L_i} = \max_{o \in kNB_t(x_i)} \{\rho(x_i, o)\}, \tag{17.46}$$

  where $kNB_t(x_i)$ is the $k$-neighbourhood of $x_i$ at iteration $t$, $d_{\max}^{L_i}$ as radius associated with leader $L_i$ at iteration $t$, $x_i$ is a node in the feature graph, and $\rho(x_i, o)$ is the given distance function between objects $x_i$ and $o$.
- Ranking the LeaderAgent $(A_i)$. This procedure is defined by Eqs. 17.47–17.49, respectively (Bellaachia and Bari 2012):

$$Rank_t(A_i) = Log\left(\frac{|N_{i,t}|}{|N_t|} * 10\right) * ARF_t(A_i), \tag{17.47}$$

$$ARF_t(A_i) = \frac{|DR\_kNB_t(x_i)|}{|DR\_kNB_t(x_i)| + |D\_kNB_t(x_i)|}, \tag{17.48}$$

$$\begin{cases} \text{if } ARF_t(A_i) \geq 0.5, & \text{then } x_i \text{ is a flockleader} \\ \text{if } ARF_t(A_i) < 0.5, & \text{then } x_i \text{ is a follower} \end{cases}, \tag{17.49}$$

where $DR\_kNB_t(x_i)$ represents the dynamic reverse $k$-neighbourhood of $x_i$ at iteration $t$, $ARF_t(A_i)$ is the dynamic agent role factor of the agent $A_i$ at iteration $t$, $|N_{i,t}|$ is the number of the neighbours $A_i$ at iteration $t$, and $|N_t|$ is the number of unvisited nodes at iteration $t$.

- Performing the flocking behaviour.
- Updating the FindFlockLeaders $(G_f)$.

### *17.19.2 Performance of FlockbyLeader*

To test the efficiency of the proposed algorithm, two large datasets that one is consists of 100 news articles collected from cyberspace, and the other one is the iris plant dataset were adopted by Bellaachia and Bari (2012). Compared with other CI algorithms, the proposed algorithm is significant improve the results.

## 17.20 Flocking-based Algorithm

In this section, we will introduce an emerging CI algorithm that is derived from the emergent collective behaviour found in social animal or insects (Lemasson et al. 2009; Ballerini et al. 2008; Luo et al. 2010; Kwasnicka et al. 2011).

### *17.20.1 Fundamentals of Flocking-based Algorithm*

Flocking-based algorithm (FBA) was originally proposed in Cui et al. (2006), Picarougne et al. (2007) and Luo et al. (2010). The basic flocking model is composed of three simple steering rules (see below) that need to be executed at each instance over time, for each individual agent.

- Rule 1: Separation. Steering to avoid collision with other boids nearby.
- Rule 2: Alignment. Steering toward the average heading and speed of the neighboring flock mates.
- Rule 3: Cohesion. Steering to the average position of the neighboring flock mates.
- In the proposed algorithm, a fourth rule is added as below:
- Rule 4: Feature similarity and dissimilarity rule. Steering the motion of the boids with the similarity among targeted objects.

All these four rules can be formally express by the following equations (Cui et al. 2006):

- The function of separation rule is to act as an active boid trying to pull away before crashing into each other. The mathematical implementation of this rule is thus can be described by Eq. 17.50 (Cui et al. 2006):

$$d(P_x, P_b) \leq d_2 \Rightarrow \vec{v}_{sr} = \sum_{x}^{n} \frac{\overrightarrow{\vec{v}_x + \vec{v}_b}}{d(P_x, P_b)}, \tag{17.50}$$

where $v_{sr}$ is velocity driven by Rule 1, $d_2$ is the distance pre-defined, $v_b$ and $v_x$ are the velocities of boids $B$ and $X$.

- The function of alignment rule is to act as the active boid trying to align its velocity vector with the average velocity vector of the flock in its local neighbourhood. The degree of locality of this rule is determined by the sensor range of the active flock boid. This rule can be presented in a mathematical way through Eq. 17.51 (Cui et al. 2006):

$$d(P_x, P_b) \leq d_1 \cap d(P_x, P_b) \geq d_2 \Rightarrow \vec{v}_{ar} = \frac{1}{n} \sum_{x}^{n} \vec{v}_x, \tag{17.51}$$

where $v_{cr}$ is velocity driven by Rule 3, $d_1$ and $d_2$ are pre-defined distance, and $(P_x - P_b)$ calculates a directional vector point.

- The flock boid tries to stay with the other boids that share the similar features with it. The strength of the attracting force is proportional to the distance (between the boids) and the similarity (between the boids' feature values) which can be expressed as Eq. 17.52 (Cui et al. 2006):

$$v_{ds} = \sum_{x}^{n} (S(B, X) \times d(P_x, P_b)), \tag{17.52}$$

where $v_{ds}$ is the velocity driven by feature similarity, $S(B, X)$ is the similarity value between the features of boids $B$ and $X$.

- The flock boid attempts to stay away from other boids with dissimilar features. The strength of the repulsion force is inversely proportional to the distance (between the boids) and the similarity value (between the boids' features) which are defined by Eq. 17.53 (Cui et al. 2006):

$$v_{dd} = \sum_{x}^{n} \frac{1}{S(B, X) \times d(P_x, P_b)}, \tag{17.53}$$

where $v_{dd}$ is the velocity driven by feature dissimilarity. To get comprehensive flocking behavior, the actions of all the rules are weighted and summed to obtain a net velocity vector required for the active flock boid using Eq. 17.54 (Cui et al. 2006):

$$v = w_{sr} v_{sr} + w_{ar} v_{ar} + w_{cr} v_{cr} + w_{ds} v_{ds} + w_{dd} v_{dd}, \tag{17.54}$$

where $v$ is the boid's velocity in the virtual space, and $w_{sr}$, $w_{ar}$, $w_{cr}$, $w_{ds}$, $w_{dd}$ are pre-defined weight values.

### 17.20.2 Performance of FBA

Document clustering is an essential operation used in unsupervised document organization, automatic topic extraction, and information retrieval. It provides a structure for organizing large bodies of data (in text form) for efficient browsing and searching. Cui et al. (2006) utilized FBA for document clustering analysis. A synthetic data set and a real document collection (including 100 news articles collected from the Internet) were used in their study. In the synthesis data set, four data types were included with each containing 200 2-dimensional $(x, y)$ data objects. Parameters $x$ and $y$ are distributed according to Normal distribution $N(\mu, \sigma)$; while for the real document collection data set, 100 news articles collected from the Internet at different time stages were categorized by human experts and manually clustered into 12 categories such as Airline safety, Iran Nuclear, Storm Irene, Volcano, and Amphetamine. In order to reduce the impact of the length variations of different documents, Cui et al. (2006) further normalized each file vector to make it in unit length. Each term stands one dimension in the document vector space. The total number of terms in the 100 stripped test files is thus 4,790 (i.e., 4,790 dimensions). The experimental studies were carried out on the synthetic and the real document collection data sets, respectively, among FBA and other popular clustering algorithms such as ant clustering algorithm and K-means algorithm. The final testing results illustrated that the FBA can have better performance with fewer iterations in comparison with the K-means and ant clustering algorithm. In the meantime, the clustering results generated by FBA were easy to be visualized and recognized even by an untrained human user.

## 17.21 Flower Pollinating Algorithm

In this section, we will introduce an emerging CI algorithm that is derived from the findings related to pollination studies (Acquaah 2012; Alonso et al. 2012)

### 17.21.1 Fundamentals of Flower Pollinating Algorithm

Flower pollinating algorithm (FPA) was originally proposed in Yang (2012). To implement FPA, the following four rules need to be followed (Yang 2012; Yang et al. 2013):

- Rule 1: Treating the biotic and cross-pollination as a global pollination process, and pollen-carrying pollinators following Lévy flights. In FPA, this rule can be defined by Eq. 17.55 (Yang 2012; Yang et al. 2013):

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \gamma L(\lambda)\left(\mathbf{x}_i^t - \mathbf{g}_*\right), \tag{17.55}$$

where $\mathbf{x}_i^t$ denotes the pollen $i$ or solution vector $\mathbf{x}_i$ at the $t$th iteration, $\mathbf{g}_*$ stands for the best solution found so far among all solutions at the current generation.

- Rule 2: For local pollination, abiotic and self-pollination are employed.
- Rule 3: Insects can play the role of pollinators for developing flower constancy. In FPA, the value of flower constancy is set equivalent to a probability called reproduction which is proportional to the similarity of two flowers involved. For modelling the local pollination, both Rule 2 and Rule can be expressed as Eq. 17.56 (Yang 2012; Yang et al. 2013):

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \varepsilon\left(\mathbf{x}_j^t - \mathbf{x}_k^t\right), \tag{17.56}$$

where the pollen from different flowers of the same plant species is denoted by $\mathbf{x}_j^t$ and $\mathbf{x}_k^t$, respectively.

- Rule 4: Controlling the interaction or switching between the local and global pollination through a switch probability parameter $p$ which falls within the range of $[0, 1]$. In FPA, a slightly biased mechanism is added here for local pollination.

### *17.21.2 Performance of FPA*

The FPA was originally developed in Yang (2012) for dealing with single objective optimization problems. Ideally, it would be great that a new algorithm can be verified on all available test function. Nevertheless, this is quite a time-consuming job. Therefore, Yang (2012) selected a set of benchmark testing functions to check the effectiveness of FPA. The preliminary experimental results demonstrated that FPA is indeed a very effective optimization algorithm.

## 17.22 Goose Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the characteristics of Canada geese flight (Hagler 2013) and the PSO algorithm.

### *17.22.1 Fundamentals of Goose Optimization Algorithm*

Goose optimization algorithm (GOA) was proposed by Liu et al. (2006). Since then, this and similar ideas have attracted a steadily increasing amount of

researchers, such as Sun and Lei (2009), Cao et al. (2012) and Dai et al. 2013). The main steps of GOA are described as follows (Sun and Lei 2009):

- Step 1: Initialize the population.
- Step 2: Calculate each goose's current fitness and ascertain each goose's individual optimum ($pfbest$) and its corresponding position ($pbest$).
- Step 3: Update each goose's local optimum ($pbest_i$)
- Step 4: Sort the population according to each goose's historical individual optimum ($pfbest_i$) in every generation and receive the sorted population ($spop$).
- Step 5: Replace the $i$th goose's global optimal with the $(i-1)$th goose's individual optimum of the sorted population.
- Step 6: Improve the velocity-location as defined by Eqs. 17.57 and 17.58, respectively (Sun and Lei 2009):

$$v_{id}^{k+1} = \omega \cdot v_{id}^{k} + \alpha\left(spop_{id}^{k} - x_{id}^{k}\right) + \beta\left(pbest_{(i-1)d}^{k} - x_{id}^{k}\right), \tag{17.57}$$

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1}, \tag{17.58}$$

  where $\alpha(spop_{id}^{k} - x_{id}^{k})$ can be regarded as a crossover operation between the $i$th goose of the current population and the $i$th goose of the stored population, $\beta(pbest_{(i-1)d}^{k} - x_{id}^{k})$ can be viewed as a crossover operation between the foregoing acquired goose position and the $(i-1)$th goose position of the stored population, and $\omega \cdot v_{id}^{k}$ can be perceived as a mutation operation by which the crossed geese are disturbed randomly.
- Step 7: Rank the solutions and store the current best as optimal fitness value as defined by Eq. 17.59 (Sun and Lei 2009):

$$\begin{cases} \text{if } f\left(x_{temp}\right) - f(x_i) < 0, & \text{then } x_{temp} \\ \text{if } f\left(x_{temp}\right) - f(x_i) > 0, & \text{then } x_i \end{cases}, \tag{17.59}$$

  where $x_{temp}$ is the new goose position that is generated by mutation operator.
- Step 8: Check the termination criteria.

### *17.22.2 Performance of GOA*

To test the efficiency of GOA, a set of travelling salesman benchmark problems were adopted in Sun and Lei (2009). Compared with other CI algorithms (such as GA, SA), computational results showed that GOA outperforms the others in terms of convergence speed and the quality of the solutions.

## 17.23 Great Deluge Algorithm

In this section, we will introduce a new CI algorithm that is based flood related research (Samuels et al. 2009).

### *17.23.1 Fundamentals of Great Deluge Algorithm*

Great deluge algorithm (GDA) was originally proposed by Dueck (1993). There are several GDA related variants and applications can be found in the literature (Burke et al. 2004; Ravi 2004; Weigert et al. 2006; Sacco et al. 2006; AL-Milli 2010; Nahas et al. 2010; Ghatei et al. 2012; Abdullah et al. 2009). In order to implement GDA, the following facts need to be taken into account (Weigert et al. 2006; Dueck 1993):

- Normally, every place within in the search space can be reached at the beginning of the GDA.
- With the time advances, the landscape of the search space will be divided into several islands according to Eq. 17.60 (Weigert et al. 2006; Dueck 1993):

$$\begin{aligned} p_i &= \Theta(L_i - C_i) \\ L_i &= L_{i-1} - \Delta L \end{aligned}, \tag{17.60}$$

  where the water level is denoted by $L$, and the rain quantity is represented by $\Delta L$.
- When the GDA is used to deal with the minimization problem, it can be renamed to great drought algorithm, through not technically necessary. In such situation, $\Delta L$ will actually refer to the water evaporation quantity. The "walker" in the original GDA will have be replaced by an artificial "fish" which continuously search for a place with sufficient water.
- The water level and rain quantity are controlling variables which play a key role in GDA. While the probability of satisfaction is independent of $\Delta C$ which depends only on the absolute value of the objective function $C$.

### *17.23.2 Performance of GDA*

In order to evaluate the performance of GDA, two typical travelling salesman problems, i.e., the 442-city problem and the 532-city problem were selected in Dueck (1993). The experimental results demonstrated that GDA has the ability of finding the equally good results reported in the literature, but with much easier implementation effort. By further testing GDA on much harder problem such as chip placement case, the GDA generated better results than other known methods (including the results obtained by SA).

## 17.24 Grenade Explosion Method

In this section, we will introduce a new CI algorithm that is inspired by the mechanism of grenade explosion. In general, there are three types of grenade, i.e., explosive grenades, chemical grenades, and gas grenades (Adams 2004). Although it is a small bomb that is hurled by hand, it is particularly effective in knocking out enemy positions.

### *17.24.1 Fundamentals of Grenade Explosion Method*

Grenade explosion method (GEM) was proposed in Ahrari et al. (2009) and Ahrari and Atai (2010). The core idea behind GEM is when grenade explodes, the thrown pieces of shrapnel destruct the objects near the explosion location. The main procedures of GEM are listed as follows (Ahrari et al. 2009; Ahrari and Atai 2010):

- Initializing the population. The initial grenades $(N_g)$ are generated in random locations in an $n$-dimension search space $\vec{X}_i \in [-1,1]^n$, $\left(i = 1,\ldots,N_g\right)$.
- Generate a point $(X')$ around the $j$th grenade through Eq. 17.61 (Ahrari et al. 2009; Ahrari and Atai 2010):

$$X'_j = \{X_m + sign(r_m) \cdot |r_m|^p \cdot L_e\}, \quad j = 1,2,\ldots,N_q, \tag{17.61}$$

  where $X = \{X_m\}$, $m = 1,2,\ldots,n$ is the current location in the $n$-dimension search space, $r_m$ is a uniformly distributed random number in $[-1,1]$, $L_e$ is the length of explosion along each coordinate, and $p$ is a constant that defined as Eq. 17.62 (Ahrari et al. 2009; Ahrari and Atai 2010):

$$p = \max\left\{1, n \cdot \frac{\log(R_t/L_e)}{\log(T_w)}\right\}, \tag{17.62}$$

  where $T_w$ is the probability that a produced piece of shrapnel collides an object in $n$-dimension hyper-box which circumscribes the grenade's territory, and $R_t$ is the territory radius.

If $X'$ is outside the feasible space, transport it to a new location inside the feasible region (i.e., $[-1,1]^n$) as defined by Eq. 17.63 (Ahrari et al. 2009; Ahrari and Atai 2010):

$$\text{if} \quad X'_j \notin [-1,1]^n \quad \Rightarrow \quad \left(B'_j = \frac{X'_j}{|\text{Largest component of } X'_j \text{ in value}|}\right)$$
$$\rightarrow \quad B''_j = r'_j \cdot \left(B'_j - X\right) + X \tag{17.63}$$
$$\begin{cases} j = 1 \text{ to } N_q \text{ (Shrapnel Number)} \\ 0 < r'_j < +1 \text{ (Random Number)} \end{cases},$$

where $X'_j$ is the collision location outside the feasible space, $B''_j$ is the new location inside the feasible space, and $N_q$ is the number of shrapnel pieces.

- Evaluate the distance between each grenade based on the territory radius $(R_t)$. If $X'$ is a distance of at least $R_t$ apart from the location of grenades $(1, 2, \ldots, i-1)$, then $X'$ is accepted.
- Calculate the fitness of the new generated points around the $j$th grenade. If the fitness of the best point is better than current location of the $j$th grenade, move the grenade to the location of the best point.
- Reduce $R_t$. For increasing the ability of global investigation, the territory radius will be reduced according to Eq. 17.64 (Ahrari et al. 2009; Ahrari and Atai 2010):

$$R_t = \frac{R_{t-initial}}{(R_{rd})^{(\text{iteration No}/\text{total No of iterations})}}, \tag{17.64}$$

where $R_{rd}$ is user defined (set before the algorithm starts).

Also, the length of explosion $(L_e)$ is reduced via Eq. 17.65 (Ahrari et al. 2009; Ahrari and Atai 2010):

$$L_e = (L_{e-initial})^m (R_t)^{1-m}, \quad 0 \le m \le 1, \tag{17.65}$$

where $m$ can be constant during the algorithm, or reduced from a higher value to a lower one.

### *17.24.2 Performance of GEM*

To demonstrate the efficiency of GEM, a set of optimization benchmark functions such as De Jong's function, Goldstein and Price function, Branin function, Martin and Gaddy function, Rosenbrock function, Schwefel function, and Hyper Sphere function were employed in Ahrari and Atai (2010). Compared with other CI methods (e.g., GA, ACO), computational results showed that GEM can perform well in finding all global minima.

## 17.25 Group Leaders Optimization Algorithm

In this section, we will introduce a new CI algorithm that inspired by the influence of the leaders in social groups and cooperative co-evolutionary mechanism (Creel 1997; Theiner et al. 2010; Mosser and Packer 2009).

### *17.25.1 Fundamentals of Group Leaders Optimization Algorithm*

Group leaders optimization algorithm (GLOA) was proposed by Daskin and Kais (2011). In order to implement GLOA, the following procedure need to be followed (Daskin and Kais 2011):

- Step 1: Generate $p$ number of population for each group randomly.
- Step 2: Calculate fitness values for all members in all groups.
- Step 3: Determine the leaders for each group.
- Step 4: Mutation and recombination.
- Step 5: Parameter transfer from other groups (one way crossover).
- Step 6: Repeat Steps 3–5 until a termination criterion is satisfied.

### *17.25.2 Performance of GLOA*

To demonstrate the efficiency of GLOA, a set of single and multi-dimensional optimization functions were adopted in Daskin and Kais (2011), namely Beale function, Easom function, Goldstein-Price's function, Shubert's function, Rosenbrock's Banana function, Griewank's function, Ackley's function, Sphere function, and Rastrigin function. Computational results showed that GLOA is very flexible and rarely gets trapped in local minima.

## 17.26 Harmony Elements Algorithm

In this section, we will introduce an emerging CI algorithm that is inspired by the human life model in traditional Chinese medicine and graph theory.

### 17.26.1 Fundamentals of Harmony Elements Algorithm

Harmony elements algorithm (HEA) or five-element string algorithm was recently proposed in Cui et al. (2008, 2009) and Rao et al. (2009). The five-elements theory posits wood, fire, earth, metal, and water as the basic elements of the material world, such as people, companies, games, plants, music, art and so on. In terms of traditional Chinese medicine, this theory is used to interpret the relationship between the physiology and pathology of the human body and the natural environment. In other words, they are metaphors for describing how things interact and relate with each other. To implement HEA, the following steps need to be followed (Cui et al. 2008, 2009):

- Step 1: Random initialization: Stochastically creating $2N$ five-element strings as candidate solutions, then grouping the candidate solutions into two string vectors (two element matrices) where the first one is denoted by $Q_{\min}$ and the second one is represented by $Q_{\max}$. The searching range for the $i$th component of the system state $\underline{x}$ is $[u_{\min}, u_{\max}]$.
- Step 2: $2N$ string cycles generation. By applying $\lambda[\,]$ to $Q_{\min}$ and $Q_{\max}$, respectively, ten string vectors can be created by Eq. 17.66 (Cui et al. 2009):

$$\begin{aligned} Q_i &= \lambda^{(i-1)}[Q_{\min}], \quad i = 1, 2, 3, 4, 5 \\ Q_i &= \lambda^{(i-6)}[Q_{\max}], \quad i = 6, 7, 8, 9, 10. \end{aligned} \tag{17.66}$$

- Step 3: Ranking the strings. Fitness checking and best-worst string vectors generation.
- Step 4: Best element selection and worst element removal. Performing packed-rolling operation and worst elements excising operation.
- Step 5: Checking whether the stopping criterion is met. If yes, terminating the HEA and outputting the results; otherwise, return to Step 1.

### 17.26.2 Performance of HEA

To verify the proposed HEA, Cui et al. (2009) employed 3 benchmark test functions, namely, Rosenbrock function, Rastrigin function, and Griewank function. In comparison with other CI algorithms (e.g., GA), the experimental results demonstrated that HEA's excellent global searching ability with very attractive speed and impressive solution quality. All these make HEA a quite promising optimization algorithm.

## 17.27 Human Group Formation Algorithm

In this section, we will introduce an emerging CI algorithm that is derived from a common phenomenon of individuals classification observed from human society (Frank 1998; Magstadt 2013; Ramachandran 2012a, b, c; Mayfield 2013; Howell 2014).

### *17.27.1 Fundamentals of Human Group Formation Algorithm*

Human group formation (HGF) algorithm was recently proposed in Thammano and Moolwong (2010). The key concept of this algorithm is about the behaviour of in-group members that try to unite with their own group as much as possible, and at the same time maintain social distance from the out-group members. To implement HGF algorithm, the following steps need to be performed (Thammano and Moolwong 2010):

- Step 1: Cluster centres representation refers to the number of classes, number of available input patterns, and number, type, and scale of the features available to the clustering algorithm. At first, there are a total of $Q$ clusters, which is equal to the number of target output classes.
- Step 2: Accuracy selection is usually measured by a distance function defined on pairs of patterns as shown in Eq. 17.67 (Thammano and Moolwong 2010):

$$\begin{aligned} &\text{Accuracy} = \frac{\sum_{i=1}^{p} A_i}{P} \\ &A_i = \begin{cases} 1, & \text{if } J \in Y_i \\ 0, & \text{otherwise} \end{cases} \\ &J = \arg_j \min(d_j(X_i)), d_j(X_i) = \left\| X_i - z_j \right\| \end{aligned}, \tag{17.67}$$

  where $P$ denotes the total number of patterns in the training data set; $J$ represents the index of a cluster whose reference pattern is the closest match to the incoming input pattern $X_i$; $Y_i$ stands for the target output of the $i$th input pattern; $z_j$ refers to the centre of the $j$th cluster; and $d_j(X_i)$ states the Euclidean distance between the input pattern $X_i$ and the centre of the $j$th cluster.

- Step 3: The grouping/formation step can be performed in a way that in-group member try to unite with their own group and maintain social distance from the non-members as much as possible, update the centre value of each cluster $(Z_j)$ by using Eq. 17.68 (Thammano and Moolwong 2010):

$$\begin{aligned} Z_{jk}^{new} &= Z_{jk}^{old} + \Delta Z_{jk} \\ \Delta Z_{jk} &= \sum_{m \in q} \eta_{jm} \beta_j \delta_{jm} (Z_{mk} - Z_{jk}) - \sum_{n \notin q} \eta_{jn} \beta_j \delta_{jn} (Z_{nk} - Z_{jk}), \end{aligned} \tag{17.68}$$

where $k$ $(k = 1, 2, 3, \ldots, k)$ is the number of features in the input pattern; $q$ is the class to which the $j$th cluster belongs; $\eta_{jm} = e^{-\left[\left(Z_{jk} - Z_{mk}\right)/\sigma\right]^2}$ and $\eta_{jn} = e^{-\left[\left(Z_{jk} - Z_{nk}\right)/\sigma\right]^2}$ have values between 0 and 1which determine the influence of $m$th and $n$th clusters on the $j$th cluster. In general, the further apart $m$th and $n$th clusters are from the $j$th cluster, the lower the values of $\eta_{jm}$ and $\eta_{jn}$; $\beta_j$ is the velocity of the $j$th cluster with respect to its own ability to move in the search space; and $\delta_{jm}$ is the parameter to prevent clusters of the same class from being too close to one another and normally with respect to two factors: (1) the distance between the $j$th cluster and the $m$th cluster, and (2) the territorial boundary of the clusters $(T)$. If the distance between the $j$th cluster and the $m$th cluster is less than $T$, the value of $\delta_{jm}$ will be decreased by a predefined amount. After each centre is updated, if the accuracy is higher, save this new center value and then continue updating the next cluster centre; if it is lower, discard the new center value and return to the previous centre; and if it does not change, save the new center value and decrease the value of $\beta_j$ by a predefined amount.

- Step 4: Cluster validity analysis is the assessment of clustering procedure's output. The cluster which satisfies the Eq. 17.69 will be deleted (Thammano and Moolwong 2010):

$$-\frac{1}{2\log_2\left(\frac{n_j}{p}\right)}\left(\frac{n_j^q}{n_j}\right)\left(\frac{\sum_{\forall X_i^j \in q}\left\|X_i^j - z_j\right\|}{n_j}\right) < \rho, \tag{17.69}$$

where $n_j$ is the number of input patterns in the $j$th cluster; $n_j^q$ is the number of input patterns in the $j$th cluster whose target outputs $(Y)$ are $q$; $X_i^j$ is the $i$th input pattern in the $j$th cluster; and $\rho$ is the vigilance parameter.

- Step 5: Recalculating the accuracy of the model according to Eq. 17.67 (Thammano and Moolwong 2010):
- Step 6: For each remaining cluster, if the distance between the new centre updated in step 3 and the previous centre is less than 0.0001 $(\|Z_{jk}^{new} - Z_{jk}^{old}\| < 0.0001)$, randomly pick $k$ small numbers between $-0.1$ and 0.1, and then add them to the centre value of the cluster. The purpose of this step is to prevent the premature convergence of the proposed algorithm to sub-optimal solutions.
- Step 7: Terminating process is to check the end condition, if it is satisfied, stop the loop; if not, examine the following conditions: (1) if the accuracy of the model improves over the previous iteration, randomly select one input pattern from the training data set of each target output class that still has error. Then go to step 2; and (2) if the accuracy does not improve, randomly select the input patterns, a number equal to the number of clusters deleted in step 4, from the training data set of each target output class. Then go to step 2.

### 17.27.2 Performance of HGF

To test the performance of HGF, Thammano and Moolwong (2010) employed 16 data sets (4 artificial and 12 real-world). The experimental results were compared with the fuzzy neural network, the radial basis function network, and the learning vector quantization network. The performance comparisons demonstrated that the validity of the proposed HGF algorithm.

## 17.28 Hunting Search Algorithm

In this section, we will introduce an emerging CI algorithm that is inspired by the group hunting of animals, such as African wild dogs (Gusset and Macdonald 2010), rodents (Ebensperger 2001), and wolves (Muro et al. 2011). Although these hunters have difference behavioural patterns during the hutting process, they are share a natural phenomenon in which all of them look for a prey in a group.

### 17.28.1 Fundamentals of Hunting Search Algorithm

Hunting search (HuS) algorithm was recently proposed in Oftadeh et al. (2010). To implement HuS algorithm, the following steps need to be performed (Oftadeh et al. 2010):

- Step 1: Initialize the optimization problem and algorithm parameters [such as hunting group size ($HGS$), maximum movement toward the leader ($MML$), and hunting group consideration rate ($HGCR$)].
- Step 2: Initialize the hunting group (HG) based on the number of hunters ($HGS$).
- Step 3: Moving toward the leader. The new hunters' positions are generated via Eq. 17.70 (Oftadeh et al. 2010):

$$x_i' = x_i + rand \cdot MML \cdot \left(x_i^L - x_i\right), \tag{17.70}$$

  where $x' = \left(x_1', x_2', \ldots, x_N'\right)$ represents the new hunters' positions, $MML$ is the maximum movement toward the leader, $rand$ is a uniform random number which varies between 0 and 1, and $x_i^L$ is the position value of the leader for the $i$th variable.

- Step 4: Position correction-cooperation between members. The updating rule of the real value correction and digital value correction are given by Eqs. 17.71 and 17.72 respectively (Oftadeh et al. 2010):

$$x_i^{j'} \leftarrow \begin{cases} x_i^{j'} \in \{x_i^1, x_i^2, \ldots, x_i^{HGS}\} & \text{with probability } HGCR \\ x_i^{j'} = x_i^j \pm R_a & \text{with probability } (1 - HGCR) \end{cases}, \quad (17.71)$$
$$i = 1, \ldots, N;$$
$$i = 1, \ldots, N;$$

$$xd_{ik}^{j'} \leftarrow \begin{cases} d_{ik}^{j'} \in \{d_{ik}^1, d_{ik}^2, \ldots, d_{ik}^{HGS}\} & \text{with probability } HGCR \\ d_{ik}^{j'} = d_{ik}^j \pm a & \text{with probability } (1 - HGCR) \end{cases}, \quad (17.72)$$
$$i = 1, \ldots, N;$$
$$j = 1, \ldots, HGS$$
$$k = 1, \ldots, M \text{ (number of digits in each variable)}$$

where $HGCR$ is the probability of choosing one value from the hunting group stored in the HG, $(1 - HGCR)$ is the probability of doing a position correction, $a$ can be any number between 1 and 9, and $R_a$ is an arbitrary distance radius for the continuous design variable as defined by Eq. 17.73 (Oftadeh et al. 2010):

$$R_a(it) = R_{a_{\min}}(\max(x_i) - \min(x_i)) \exp\left(\frac{Ln\left(\frac{R_{a_{\min}}}{R_{a_{\max}}}\right) \cdot it}{itm}\right), \quad (17.73)$$

where $it$ is the iteration number, $\max(x_i)$ and $\min(x_i)$ are the maximum or minimum possible value of variable $x_i$, respectively, $R_{a_{\max}}$ and $R_{a_{\min}}$ are the maximum and minimum of relative search radius of the hunter, respectively, and $itm$ is the maximum number of iterations in the optimization process.

- Step 5: Reorganizing the hunting group. The rule for members' recognition is defined by Eq. 17.74 (Oftadeh et al. 2010):

$$x_i' = x_i^L \pm rand \cdot (\max(x_i) - \min(x_i)) \cdot \alpha \exp(-\beta \cdot EN), \quad (17.74)$$

where $x_i^L$ is the position value of the leader for the $i$th variable, $rand$ is a uniform random number which varies between 0 and 1, $\max(x_i)$ and $\min(x_i)$ are the maximum and minimum possible values of variable $x_i$, respectively, $EN$ counts the number of times that the group has been trapped until this step, and $\alpha$ and $\beta$ are positive real values.

- Step 6: Termination. Repeat Steps 3–5 until the termination criterion is satisfied.

### 17.28.2 Performance of HuS

In order to show how the HuS algorithm performs, different unconstrained and constrained standard benchmark test functions were adopted in Oftadeh et al. (2010). Compared with other CI techniques (e.g., EA, GA, PSO, ACO, etc.), the performance of HuS algorithm is very competitive.

## 17.29 Krill Herd Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the krill swarm related studies (Verdy and Flierl 2008; Brierley and Cox 2010; Goffredo and Dubinsky 2014).

### *17.29.1 Fundamentals of Krill Herd Algorithm*

Krill herd (KH) algorithm was recently proposed in Gandomi and Alavi (2012). In order to implement the KH algorithm, the following steps need to be followed (Gandomi and Alavi 2012; Wang et al. 2013):

- Step 1: Defining the simple boundaries, algorithm parameters, and so on.
- Step 2: Initialization. Stochastically creating the initial population within the search space.
- Step 3: Fitness evaluation. Evaluating each individual krill based on its position.
- Step 4: Calculating motion conditions. In KH algorithm, the motion caused by the presence of other individual krill is computed via Eq. 17.75 (Gandomi and Alavi 2012):

$$N_i^{new} = N^{\max}\alpha_i + \omega_n N_i^{old}, \tag{17.75}$$

where $\alpha_i$ equals to $\alpha_i^{local} + \alpha_i^{\text{target}}$, and $N^{\max}$ represents the maximum induced speed, $N_i^{old}$ denotes the last induced motion. Meanwhile, the foraging motion for the $i$th krill individual is defined by Eq. 17.76 (Gandomi and Alavi 2012):

$$F_i = V_f\beta_i + \omega_f F_i^{old}, \tag{17.76}$$

where $\beta_i$ is equivalent to $\beta_i^{food} + \beta_i^{best}$, and the foraging speed is denoted by $V_f$. Finally, the physical diffusion motion of the krill is treated as a random process. This motion can be expressed in Eq. 17.77 (Gandomi and Alavi 2012):

$$D_i = D^{\max}\delta, \tag{17.77}$$

where $D^{\max}$ denotes the maximum diffusion speed, and $\delta$ represents a random direction vector. All three motions can be defined by using the following Lagrangian model (see Eq. 17.78) (Gandomi and Alavi 2012):

$$\frac{dX_i}{dt} = N_i + F_i + D_i, \tag{17.78}$$

where $N_i$ denotes the motion induced by other individual krills.

- Step 5: Implementing the genetic operators.
- Step 6: Updating the position of each individual krill within the search space.

- Step 7: Checking whether the stopping condition is met. If not, returning to Step 3; otherwise, terminating the algorithm.

### *17.29.2 Performance of KH*

In order to show how the KH algorithm performs, 20 benchmark test functions such as Sphere function, Goldstein and Price function, Griewank function, and Ackley function are employed in Gandomi and Alavi (2012). Compared with other CI techniques, the performance of KH algorithm is very competitive.

## 17.30 League Championship Algorithm

In this section, we will introduce an emerging CI algorithm that is based on some interesting findings relative to sports science (Smolin and Grosvenor 2010; Abernethy et al. 2013).

### *17.30.1 Fundamentals of League Championship Algorithm*

League championship algorithm (LCA) was recently proposed in Kashan (2009). In order to model an artificial championship environment, there are the following 6 idealization rules employed in LCA (Kashan 2009, 2011; Kashan and Karimi 2010):

- Rule 1: In LCA, playing strength is defined as the capability of one team defeating the other team.
- Rule 2: The game results is not predictable even if the team's playing strength is know perfectly.
- Rule 3: The winning probability of a team $i$ over the other team $j$ is assumed to be the same, no matter from which team's viewpoint.
- Rule 4: In the basic version of LCA, tie is not taken into account which means win or loss will be the only game result option.
- Rule 5: Teams only concentrate on the forthcoming math and with no interest of the other distant future game.
- Rule 6: When team $i$ defeats the other team $j$, any strength of assisting team $i$ in winning will have a dual weakness in causing team $j$ to lose.

In order to implement LCA algorithm, the following modules need to be well designed (Kashan 2009, 2011; Kashan and Karimi 2010):

- Module 1: Creating the league timetable. In LCA, an important step is to simulate a real championship environment by establishing a schedule which forms a "virtual season". For instance, a single round-robin schedule mechanism can be employed for ensuring that each team plays against every other team once in each virtual season.
- Module 2: Confirming the winner or loser. Using the playing strength criterion, the winner or loser in LCA is identified in a random manner. Based on the abovementioned Rule 1, the expected chance of winning for team $i$ (or $j$) can be defined as Eq. 17.79 (Kashan 2011):

$$p_i^t = \frac{f\left(X_j^t\right) - \hat{f}}{f\left(X_j^t\right) + f(X_i^t) - 2\hat{f}}. \tag{17.79}$$

- Module 3: Deploying a suitable mixture of team members. Since the strengths and weaknesses of the each individual team member are not the same, it is often important for coach to generate a good team members mixture by taking various constraint into account. In LCA, a similar process is also performed through an artificial analysis mechanism, more specifically, an artificial SWOT (denoting strengths, weaknesses, opportunities, and threats) analysis is utilized for generating a suitable focus strategy. Based on a thorough analysis, in order to get a new formation of team, the random number of changes made in $B_i^t$ (i.e., best team formation for team $i$ at week $t$) can be computed through Eq. 17.80 (Kashan 2011):

$$q_i^t = \left\lceil \frac{\ln\left(1 - \left(1 - (1 - p_c)^{n-q_0+1}\right)r\right)}{\ln(1 - p_c)} \right\rceil + q_0 - 1, \quad q_i^t \in \{q_0, q_0 + 1, \ldots, n\}, \tag{17.80}$$

  where $r$ denotes a random number which falls within the range of $[0, 1]$, and $p_c < 1$, $p_c \neq 0$ represents a controlling variable.

### *17.30.2 Performance of LCA*

To verify the capability of LCA, Kashan (2009) employed 5 benchmark test functions which include such as Sphere function, Rosenbrock function, Rastrigin function, Ackley function, and Schwefel function. In comparison with other CI techniques (e.g., PSO), the simulation results proved that LCA is a dependable method which can converge very fast to the global optimal.

## 17.31 Membrane Algorithm

In this section, we will introduce an emerging CI algorithm that is based on some studies relative to biological membrane (Reece et al. 2011; Yeagle 2005) and some of its basic features inspired membrane computing (Păun 2000, 2002; Gheorghe et al. 2012; Xiao et al. 2013; Maroosi and Muniyandi 2013; Muniyandi and Zin 2013; Kim 2012; Gofman 2012; Nabil et al. 2012; Zhang et al. 2011; Murphy 2010; Aman 2009; Sedwards 2009; Păun 2007; Nguyen et al. 2008; Woodworth 2007; Ishdorj 2006; Zaharie and Ciobanu 2006; Ciobanu et al. 2003).

### *17.31.1 Fundamentals of Membrane Algorithm*

Membrane algorithm (MA), an approach built on membrane system or P-system diagram (Păun 2000, 2002), was initially proposed in Nishida (2005):

- Component 1: A set of regions which are normally divided by nested membranes.
- Component 2: Each individual region contains a sub-algorithm and several tentative solutions of the targeted optimization problem.
- Component 3: Solution transferring strategy between adjacent regions.

Once the initial settings are done, the following steps need to be performed for implementing MA algorithm (Nishida 2005):

- Step 1: Simultaneously updating the solutions by using the sub-algorithm existing in each individual region.
- Step 2: Sending the best and worst solutions to all the adjacent inner and outer regions, respectively. This mechanism is performed for each region.
- Step 3: Repeating the solutions updating and transferring procedure until a stopping criterion is met.
- Step 4: Outputting the best solution found in the innermost region.

### *17.31.2 Performance of MA*

Nishida (2005) employed the classic travelling salesman problem as a benchmark for verifying the performance of MA. The simulation results demonstrated that the performance of MA is very attractive. As Nishida (2005) commented in the work: On one hand, since other CI algorithms such as GA and SA can be used to play the role of sub-algorithm, an MA is likely to be able to avoid the local optimal. On the other hand, since different sub-algorithms are separated by membranes and the communications happen only among adjacent regions, MA can be easily implemented in other types of computing systems such as parallel, distributed, and grid

computing. All these merits make MA a promising candidate in defeating "No Free Lunch Theorem (Wolpert and Macready 1997)".

## 17.32 Migrating Birds Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is inspired by the v-flight formation of the migrating birds. It gets this name because of the similarity of the shape the birds that, through one bird leading the flock and two lines of other birds following it, make to the letter "V" (Shettleworth 2010). In addition, the v-formation is one example of the fluid dynamics at work (Hagler 2013). Also, it is believed as a very efficient way for long distance flying due to it is possible to save energy and it can help birds avoid collisions (Badgerow and Hainsworth 1981; Cutts and Speakman 1994; Lissaman and Shollenberger 1970).

### *17.32.1 Fundamentals of Migrating Birds Optimization Algorithm*

Migrating birds optimization (MBO) algorithm was proposed by Duman et al. (2012). In MBO, it is assumed that after flying fro some time, when the leader birds gets tired, it goes to the end of the line and one of the birds following it takes the leader position. As a result, MBO is capable of finding more areas of the feasible solution space by looking at the neighbour solutions. The main steps of MBO are outlined below (Duman et al. 2012):

- Step 1: Initializing the population and parameters.
- Step 2: Repeating the following procedure till stopping criteria met. First, randomly select a leading bird $(i)$. Second, calculate its fitness function. Third, randomly select a neighbour bird among $k$ available neighbour birds (e.g., $j$). Fourth, if $(F_i > F_j)$, then replace the $j$ by the new solution. Fifth, improve each solution $(s_r)$ in the flock (except leader) by evaluating neighbours' wing-tip spacing (WTS) through Eq. 17.81 (Duman et al. 2012):

$$WTS_{opt} = -0.05b, \tag{17.81}$$

  where $WTS_{opt}$ represents the optimum WTS, and $b$ is the wing span. Sixth, calculate fitness and keep the best solutions. Seventh, rank the solutions and store the current best as optimal fitness value.
- Step 3: Posting process and visualizing results.

### 17.32.2 Performance of MBO

To test the performance of the MBO algorithm, a series of quadratic assignment problems were taken as the benchmarks (Duman et al. 2012). Compared with other CI algorithm (e.g., TS, SA, and GA), MBO obtained very successful results.

## 17.33 Mine Blast Algorithm

In this section, we will introduce a new CI algorithm that is based on the observation of the mine bombs (a notorious invention by human) explosion in real world. Just like the volcano or earthquake (Rose 2008), with such force, the mine bombs will be blasted into billions of tiny pieces. In addition, the thrown pieces of shrapnel remain bore with other mine bombs near the explosion area resulting in their explosion.

### 17.33.1 Fundamentals of Mine Blast Algorithm

Mine blast algorithm (MBA) was proposed by Sadollah et al. (2012, 2013). The main steps of MBA are listed as follows (Sadollah et al. 2012, 2013):

- Step 0: Initializing the population. The initial population is generated by a first shot $(X_0^f)$ explosion producing a number of individuals. The first shot point value is updated via Eq. 17.82 (Sadollah et al. 2012, 2013):

$$X_0^{new} = LB + rand \cdot (UB - LB), \tag{17.82}$$

  where $LB$ and $UB$ are the lower and upper bonds of the problem, respectively, and $X_0^{new}$ is the new generated first shot point.
- Step 1: Initializing the parameters.
- Step 2: Check the condition of exploration constant $(\mu)$.
- Step 3: If condition of exploration constant is satisfied, calculate the distance of shrapnel pieces and their location, otherwise, go to Step 10. The calculating equations are given by Eq. 17.83 (Sadollah et al. 2012, 2013):

$$\begin{aligned} d_{n+1}^f &= d_n^f \cdot (|randn|)^2, \quad n = 0, 1, 2, \ldots \\ X_{e(n+1)}^f &= d_{n+1}^f \cdot \cos(\theta), \quad n = 0, 1, 2, \ldots \end{aligned}, \tag{17.83}$$

  where $X_{e(n+1)}^f$ is the location of exploding mine bomb, $d_{n+1}^f$ is the distance of the thrown shrapnel pieces in each iteration, and *randn* is normally distributed pseudorandom number (obtained using *randn* function in MATLAB).

- Step 4: Calculate the direction of shrapnel pieces through Eq. 17.84 (Sadollah et al. 2012, 2013):

$$m_{n+1}^{f} = \frac{F_{n+1}^{f} - F_{n}^{f}}{X_{n+1}^{f} - X_{n}^{f}}, \quad n = 0, 1, 2, 3, \ldots, \tag{17.84}$$

where $F$ is the function value of the $X$, and $m_{n+1}^{f}$ is the direction of shrapnel pieces.

- Step 5: Generate the shrapnel pieces and compute their improved locations via Eq. 17.85 (Sadollah et al. 2012, 2013):

$$X_{n+1}^{f} = X_{e(n+1)}^{f} + \exp\left(-\sqrt{\frac{m_{n+1}^{f}}{d_{n+1}^{f}}}\right) \cdot X_{n}^{f}, \quad n = 0, 1, 2, 3, \ldots, \tag{17.85}$$

where $X_{e(n+1)}^{f}$, $d_{n+1}^{f}$, and $m_{n+1}^{f}$ are the location of exploding mine bomb collided by shrapnel, the distance of shrapnel and the direction (slope) of the thrown shrapnel in each iteration, respectively.

- Step 6: Check the constraints for generated shrapnel pieces.
- Step 7: Save the best shrapnel piece as the best temporal solution.
- Step 8: Does the shrapnel piece have the lower function value than the best temporal solution?
- Step 9: If true, exchange the position of the shrapnel with the best temporal solution. Otherwise, go to Step 10.
- Step 10: Calculate the distance of shrapnel pieces and their locations, then return to Step 4. The calculating equations are given by Eq. 17.86 (Sadollah et al. 2012, 2013):

$$\begin{aligned} d_{n+1}^{f} &= \sqrt{\left(X_{n+1}^{f} - X_{n}^{f}\right)^{2} + \left(F_{n+1}^{f} - F_{n}^{f}\right)^{2}}, \quad n = 0, 1, 2, \ldots \\ X_{e(n+1)}^{f} &= d_{n}^{f} \cdot rand \cdot \cos(\theta), \qquad\qquad\quad n = 0, 1, 2, \ldots \end{aligned}, \tag{17.86}$$

where $X_{e(n+1)}^{f}$ is the location of exploding mine bomb, *rand* is a uniformly distributed random number, and $\theta$ is the angle of the shrapnel which is calculated through Eq. 17.87 (Sadollah et al. 2012, 2013):

$$\theta = 360/N_{s}, \tag{17.87}$$

where $N_s$ is the number of shrapnel pieces which are produced by the mine bomb explosion.

- Step 11: Reduce the distance of the shrapnel pieces according to Eq. 17.88 (Sadollah et al. 2012, 2013):

$$d_n^f = \frac{d_{n+1}^f}{\exp(k/\alpha)}, \quad n = 1, 2, 3, \ldots, \tag{17.88}$$

where $\alpha$ and $k$ are the reduction constant which is user parameter and depends on the complexity of the problem and iteration number, respectively.

- Step 12: Check the convergence criteria. If the stopping criterion is satisfied, the algorithm will be stopped. Otherwise, return to Step 2.

### *17.33.2 Performance of MBA*

To test the efficiency of MBA, five well-known truss structures problems were adopted in Sadollah et al. (2012), namely, 10-bar truss, 15-bar truss, 52-bar truss, 25-bar truss, and 72-bar truss. Compared with other CI algorithms (e.g., PSO), computational results showed that MBA clearly outperforms the others in terms of convergence speed and computational cost.

## 17.34 Monkey Search Algorithm

In this section, we will introduce an emerging CI algorithm that is inspired by the monkey foraging behaviour (King et al. 2011; Mills et al. 2010; Sueur et al. 2010; Lee and Quessy 2003; Taffe and Taffe 2011).

### *17.34.1 Fundamentals of Monkey Search Algorithm*

Monkey search algorithm (MSA) was proposed by Mucherino and Seref (2007). In MSA, the food is viewed as the desirable solutions and the branches of the trees are illustrated as perturbations between two neighbouring feasible solutions. In addition, at each iteration, the starting solution is viewed as the root of a branch and the new neighbour solution is given at the tip of the same branch. The height of the trees (i.e., the functional distance between the two solutions, $h_t$) is determined by the random perturbation. Also, it is assumed that when the monkeys look for food, they will also learn which branches lead to better food resources. The main steps of MSA are described as follows (Mucherino and Seref 2007):

- Step 1: Initialize populations and parameters.
- Step 2: Repeat till stopping criteria met. First, randomly select a branch of a tree ($n_w$) as root. Second, calculate its fitness function. Third, perform the perturbations process to generate a new solution at the tip of the same branch as follows (Mucherino and Seref 2007): (1) Random changes to $X_{cur}$, as in the SA methods; (2) Crossover operator applied for generating a child solution from the

parents $X_{cur}$ and $X_{best}$, as in GA; (3) The mean solution built from $X_{cur}$ and $X_{best}$, inspired by ACO; (4) Directions that lead $X_{cur}$ to $X_{best}$, as in directional evolution; (5) Creating solutions from $X_{cur}$ and $X_{best}$ and introducing random notes, as in harmony search.

- Step 3: Check the termination criteria.

In addition, for avoiding local optima, the predetermined number of $n_m$ best solutions (i.e., the memory bank) are updated by each successive tree.

### *17.34.2 Performance of MSA*

To test the performance of MSA, two global optimization problem of finding stable conformations of clusters of atoms' energy functions (i.e., Lennard Jones potential energy and Morse potential energy) were adopted in Mucherino and Seref (2007). In addition, a protein folding problem (i.e., the tube model) is also considered as test function. Compared with other CI algorithms (such as SA), computational results showed that the proposed algorithm outperforms others in terms of the quality of solutions.

## 17.35 Mosquito Host-Seeking Algorithm

In this section, we will introduce an emerging CI algorithm that is inspired by the host-seeking behaviour of mosquitoes (Levin 2013d).

### *17.35.1 Fundamentals of Mosquito Host-Seeking Algorithm*

Mosquito host-seeking algorithm (MHSA) was proposed by Feng et al. (2009). According to the observation of mosquito host-seeking behaviour, there is only female mosquitoes search the host to attract (Woodward 2008). Recently, based on mosquito host-seeking behaviour, Cummins et al. (2012) proposed a new mathematical model to describe the effect of spatial heterogeneity. Furthermore, it is possible to design artificial female mosquitoes that, by seeking towards the hosts which emit an odor (e.g., $CO_2$), find the shortest path between the two nodes corresponding to the nest and to the food source. The following description, which is developed using the travelling salesman problem as a running example, is completed for implementing the proposed algorithm.

- Step 1: Initializing the population and parameters.

- Step 2: Calculate the distance between $(u_{ij}(t))$ between an artificial mosquito and the host at time $t$ in parallel as defined by Eq. 17.89 (Feng et al. 2009):

$$u_{ij}(t) = \exp\left(-c_{ij}(t) r_{ij}(t) x_{ij}(t)\right), \tag{17.89}$$

where $x_{ij}$ is the sex value of each artificial mosquito $(m_{ij})$ as defined by Eq. 17.90 (Feng et al. 2009):

$$\begin{cases} x_{ij} = 1, & m_{ij} \text{ is female} \\ x_{ij} = 0, & m_{ij} \text{ is male} \end{cases}, \tag{17.90}$$

and $c_{ij}$ represents the relative strength of the artificial mosquito. It can be defined as Eq. 17.91 (Feng et al. 2009):

$$\begin{cases} t = 0, & c_{ij} = \max\limits_{i,j} d_{ij} - d_{ij} \\ t > 0, & c_{ij} \in [0, 1] \end{cases}, \tag{17.91}$$

where $c_{ij}$ represents the distance between city pair $\left(C_i, C_j\right)$, and $d_{ij}$ is defined by Eq. 17.92 (Feng et al. 2009):

$$d_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}. \tag{17.92}$$

The utility of all artificial mosquitoes will be summarized as Eq. 17.93 (Feng et al. 2009):

$$J(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} u_{ij}(t). \tag{17.93}$$

- Step 3: Calculating the motion of artificial mosquitoes via Eqs. 17.94–17.96, respectively (Feng et al. 2009):

$$\frac{du_{ij}(t)}{dt} = \psi_1(t) + \psi_2(t), \tag{17.94}$$

$$\psi_1(t) = -u_{ij}(t) + \gamma v_{ij}(t), \tag{17.95}$$

$$\begin{aligned} \psi_2(t) = & \left[-\lambda_1 - \lambda_2 \frac{\partial J(t)}{\partial u_{ij}(t)} - \lambda_3 \frac{\partial P(t)}{\partial u_{ij}(t)} - \lambda_4 \frac{\partial Q(t)}{\partial u_{ij}(t)}\right] \\ & \times \left\{ \left[\frac{\partial u_{ij}(t)}{\partial r_{ij}(t)}\right]^2 + \left[\frac{\partial u_{ij}(t)}{\partial c_{ij}(t)}\right]^2 \right\} \end{aligned}, \tag{17.96}$$

where $\gamma > 1$, and $v_{ij}(t)$ is a piecewise linear function of $u_{ij}(t)$ defined by Eq. 17.97 (Feng et al. 2009):

$$v_{ij}(t) = \begin{cases} 0 & \text{if } u_{ij}(t) < 0 \\ u_{ij}(t) & \text{if } 0 \le u_{ij}(t) < 1 \\ 1 & \text{if } u_{ij}(t) > 1 \end{cases}, \tag{17.97}$$

and $P(t)$ and $Q(t)$ represent the attraction functions as defined by Eqs. 17.98 and 17.99, respectively (Feng et al. 2009):

$$P(t) = \varepsilon^2 \ln \sum_{i=1}^{n} \sum_{j=1}^{n} \exp\left[-u_{ij}^2(t) \Big/ 2\varepsilon^2\right] - \varepsilon^2 \ln nn, \tag{17.98}$$

$$Q(t) = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} r_{ij}(t) x_{ij}(t) - 2 \right|^2 - \sum_{i,j} \int_0^{u_{ij}} \left\{ [1 + \exp(-10x)]^{-1} - 0.5 \right\} dx, \tag{17.99}$$

where $0 < \varepsilon < 1$, and $r_{ij}(t)$ is defined by Eq. 17.100 (Feng et al. 2009):

$$\begin{cases} r_{ij} = 1, & Z \text{ pass } p_{ij} \\ r_{ij} = 0, & Z \text{ not pass } p_{ij} \end{cases}, \tag{17.100}$$

where $Z$ is the shortest path through $n$ cities, and $p_{ij}$ is the path between $C_i$ and $C_j$.

The general hybrid attraction function for artificial mosquito can be defined by Eq. 17.101 (Feng et al. 2009):

$$E_{ij}(t) = -\lambda_1 u_{ij}(t) - \lambda_2 J(t) - \lambda_3 P(t) - \lambda_4 Q(t), \tag{17.101}$$

where $E_{ij}(t) =$ is the hybrid attrition function, and $0 < \lambda_1, \lambda_2, \lambda_3, \lambda_4 < 1$.

- Step 4: For each artificial mosquito, calculate the value of $dr_{ij}(t)$ and $dc_{ij}(t)$ in order to increase the problem's dynamically according to Eqs. 17.102 and 17.103, respectively (Feng et al. 2009):

$$\frac{dr_{ij}(t)}{dt} = -\lambda_1 \frac{\partial u_{ij}(t)}{\partial r_{ij}(t)} - \lambda_2 \frac{\partial J(t)}{\partial r_{ij}(t)} - \lambda_3 \frac{\partial P(t)}{\partial r_{ij}(t)} - \lambda_4 \frac{\partial Q(t)}{\partial r_{ij}(t)}, \tag{17.102}$$

$$\frac{dc_{ij}(t)}{dt} = -\lambda_1 \frac{\partial u_{ij}(t)}{\partial c_{ij}(t)} - \lambda_2 \frac{\partial J(t)}{\partial c_{ij}(t)} - \lambda_3 \frac{\partial P(t)}{\partial c_{ij}(t)} - \lambda_4 \frac{\partial Q(t)}{\partial c_{ij}(t)}, \tag{17.103}$$

where $\dfrac{\partial Q(t)}{\partial u_{ij}(t)} = -\left\{ \left[1 + \exp\left(-10(t)u_{ij}(t)\right)\right]^{-1} - 0.5 \right\}$.

- Step 5: Updating the both value of $(r_{ij}(t))$ and $(c_{ij}(t))$ in parallel through Eqs. 17.104 and 17.105 (Feng et al. 2009):

$$r_{ij}(t+1) = r_{ij}(t) + \frac{dr_{ij}(t)}{dt}, \tag{17.104}$$

$$c_{ij}(t+1) = c_{ij}(t) + \frac{dc_{ij}(t)}{dt}. \tag{17.105}$$

- Step 6: If all $\frac{du_{ij}(t)}{dt} = 0$, then finish successfully; otherwise, go to Step 2.

### *17.35.2 Performance of MHSA*

The travelling salesman problem has attracted many researchers from different fields. Recently, Feng et al. (2009) used MHSA for finding near-optimum solutions to the travelling salesman problem. Computational results showed that the proposed algorithm performs well and easy to jump into local optimal. Also, it is easy to adapt to a wide range of the travelling salesman problem.

## 17.36 Oriented Search Algorithm

In this section, we will introduce an emerging algorithm that is inspired by the human helping behaviour. Helping behaviours are activities where people intend to assist other person to solve the problems, such as to relieve distress (Stukas and Clary 2012).

### *17.36.1 Fundamentals of Oriented Search Algorithm*

Oriented search algorithm (OSA) was proposed by Zhang et al. (2008a) that mimics the helping behaviour of a little girl when she lost her way in deep forest. In OSA, the optimal solution of the objective function is the lost girl who can transmit information for help in order to be found immediately. The main steps of OSA are illustrated as follows (Zhang et al. 2008a):

- Step 1: Initialization the population and parameters. For example, the objective function for each search individuals $(f(x_{0ji}))$ and the position of search individuals $(x_{0ji})$ are defined by Eqs. 17.106–17.108, respectively (Zhang et al. 2008a):

$$x_{0ji} = X_{\min i} + (X_{\max i} - X_{\min i}) * random_{ji}(0,1), \tag{17.106}$$

$$x_{0ji} = c_i, \quad \text{where } c_i \in [X_{\min i}, X_{\max i}], \tag{17.107}$$

$$x_{0ji} = c_i, \quad \text{where } c_i \in [X_{\min i}, X_{\max i}], \tag{17.108}$$

  where $a_{ji}$ and $c_i$ are constants.

- Step 2: Exploration walks procedure. First, generate the strategy of updating $\Delta x_{tji}$ via Eqs. 17.109 and 17.110, respectively (Zhang et al. 2008a):

$$x_{tji} = x_{0ji} + \Delta x_{tji}, \tag{17.109}$$

$$\Delta x_{tji} = \left(x_{tji\_global} \cdot \left(1 + w \cdot randn_{tji}(0,1) - x_{tji}\right) \cdot random(0,1)\right), \tag{17.110}$$

  where $x_{tji\_global}$ denotes the current optimal position of the objective function, and $w$ is a variable which can adjust the variable trend of oriented-neighbour-space.

Second, explore new position of the current search individual $(x_{tji})$. Third, evaluate the quality of the objective function $(f_{tj} = f(x_{tji}))$. Fourth, if $f_{tj} \leq f_{(t-1)j}$, then $x_{0ji} = x_{tji}$. Fifth, update the current position of the objective function optimal solution $(x_{tji\_global})$. Sixth, check the termination criteria.

- Step 3: Posting process and visualizing results.

### *17.36.2 Performance of OSA*

To test the performance of OSA, a reactive power optimization problem was adopted in Zhang et al. (2008a). Compared with other algorithms, computational results showed that OSA has better convergence property and precision. Also, it is capable of escaping from the local optima.

## 17.37 Paddy Field Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the concept of sowing is carried out in accordance with individual fitness value and neighbour numbers of seed so that they will grow towards the best environment (optimal solution) (Maathuis 2013; Acquaah 2012).

### *17.37.1 Fundamentals of Paddy Field Algorithm*

Paddy field algorithm (PFA) was recently proposed in Premaratne et al. (2009). In general, situation of each individual seed or plant can be illustrated as vector $X = (x_1, x_2, \ldots, x_k)$ and the fitness or objective function of $X$ is denoted by $Y = f(X)$. Depending on the nature of the parameter space each dimension of the seed can be bonded such that Eq. 17.111 holds (Premaratne et al. 2009; Wang et al. 2011):

$$x_j \in \left[\left(x_j\right)_{\min}, \left(x_j\right)_{\max}\right]. \tag{17.111}$$

To implement PFA algorithm, the following steps need to be performed (Premaratne et al. 2009):

- Sowing behaviour: The algorithm operates by initially scattering seeds (initial population $p_0$) at random in an uneven field. The values of seed dimensions are uniformly distributed depending on the bounds of the parameter space.
- Selection behaviour: When the seeds produce plants, the best plants are selected depending on a threshold $(y_t)$, which can be used to determine the number of seeds of a plant. The reason for having a threshold is to control the population of the system. That means, a plant will be selected to the next iteration only if its fitness value $(y)$ is greater than the threshold as defined by Eq. 17.112 (Premaratne et al. 2009).

$$y \geq y_t. \tag{17.112}$$

- Seeding behaviour: In this stage, each plant develops a number of seeds proportional to its health. The total quantity of seeds $(s)$ produced by any plant would be a function of the plant fitness function and the maximum number of seeds $(q_{\max})$ as defined by Eq. 17.113 (Premaratne et al. 2009):

$$s = \phi[f(x), q_{\max}], \tag{17.113}$$

  In general, the fitness function is depending on its fitness in proportion to the fittest plant of the population $(y_{\max})$ as shown in Eq. 17.114 (Premaratne et al. 2009):

$$s = q_{\max}\left[\frac{y - y_t}{y_{\max} - y_t}\right]. \tag{17.114}$$

- Pollination behaviour: In any paddy field, the strong ones (best solution) have greater opportunity to pass their seeds to future generations via pollination behaviour. This behaviour is a major factor either via animals or through wind. High population density would increase the chance of pollination for pollen carried by the wind. That means, the plant with more neighbours (i.e. neighbourhood function $N$) will be better pollinated. The number of viable seeds $(s_v)$ produced by a plant can be expressed as Eq. 17.115 (Premaratne et al. 2009):

$$s_v = N\phi[f(x), q_{\max}], \quad 0 \leq N \leq 1. \tag{17.115}$$

  In order to satisfy this condition, a sphere of radius $(a)$ is used. For two plants $x_j$ and $x_k$, the perimeter formula (see Eq. 17.116) (Premaratne et al. 2009)

$$n\left(x_j, x_k\right) = \left\|x_j - x_k\right\| - a, \tag{17.116}$$

  is used. If the two are within the sphere, then $n < 0$. From this for a particular plant, the number of neighbours $(v_j)$ can be determined. Once this is done, the

pollination factor for that plant can be obtained from Eq. 17.117 (Premaratne et al. 2009),

$$N_j = e^{\left[\frac{v_j}{v_{\max}} - 1\right]}, \tag{17.117}$$

where $v_{\max}$ is maximum neighbour number of the plant.

- Dispersion behaviour: In order to prevent getting stuck in local minima, the seeds of each plant are dispersed and then the cycle stars again from the selection stage. In PFA, when dispersing, the dimension values take a Gaussian distribution which could provide a faster convergence in local search. The new seed will land on a location in the parameter space given by Eq. 17.118 (Premaratne et al. 2009):

$$X_{seed}^{i+1} = F\left(x^i, \sigma\right), \tag{17.118}$$

where $\sigma$ is the coefficient of dispersion, which can determine the dispersion degree of produced seeds.

### *17.37.2 Performance of PFA*

The difference between PFA and other nature-inspired algorithms (such as evolutionary algorithms) is PFA uses pollination and dispersal between individuals as mainly operators. In addition, unlike the basic version of PSO, in PFA, the random numbers are not generated by applying the uniform distribution function. Instead, the Gaussian probability distribution function is applied. This offers the advantage of enhanced search capability while maintaining adequate exploitation capability (Premaratne et al. 2009).

## 17.38 Photosynthetic Algorithm

The process of which the carbon atoms in $CO_2$ are incorporated into glucose, $C_6H_{12}O_6$, in green plants is normally referred to as photosynthesis (Whitten et al. 2014). It is often regarded as one of the key biological process in the biosphere (Dubinsky 2013; Carpentier 2011). Normally, oxygenic photosynthesis can be found occurring in cyanobacteria, algae and land plants (Dubinsky 2013). Although the actual process is quite complex, the following net equation (see Eq. 17.119) can be used to simply describe the phenomenon of photosynthesis (Whitten et al. 2014; Reece et al. 2011; Jelinek 2013; Hobbs et al. 2013):

$$6CO_2 + 6H_2O \xrightarrow[\text{chlorophyll}]{\text{sunlight}} C_6H_{12}O_6 + 6O_2, \tag{17.119}$$

where chlorophyll contains magnesium ions which are bond to porphyrin rings and it is critical substance for photosynthesis.

In nature, ribulose-1, 5-bisphosphate carboxylase/oxygenase (Rubisco for short) is the most abundant protein on Eearth, comprising almost the half of the protein in leaves. Basically Rubisco catalyses the carboxylation of ribulose-1, 5-bishosphate (RuBP), generating two molecules of 3-phosphoglycerate (3-PGA). Rubisco is a vary useful bifunctional enzyme that fixes the liberated $CO_2$ in the chloroplasts of photosynthetic organism through its carboxylase activity (Dubinsky 2013). This irreversible first step of photosynthesis is therefore the entering point for carbon into the biosphere (Carpentier 2011).

On our planet, most of the energy required to develop and sustain life is supplied by the capture of sunlight by photosynthetic organisms (Carpentier 2011). Photosynthesis is thus often treated as the source of global food, feed, fibre, and timber production as well as biomass-based bio-energy. The renewability is the main characteristic of each of these products of photosynthesis. For instance the main products photosynthesis are starch and sucrose where the latter is also the main form of carbon translocated from leaves to other organs in plants (Dubinsky 2013). Since photosynthesis is, in itself, a multidisciplinary research area which involves such as agriculture, environmental sciences, forestry, plant genetics, photobiology, photophysics, plant physiology, and biochemistry, the detailed explanation of many of its general and fundamental research methods and recent advances is out of the scope of the present book, interested readers are referred to the corresponding studies, e.g., (Acquaah 2012; Carpentier 2011; Dubinsky 2013; Maathuis 2013), for more relative information.

For the rest of this section, we will introduce an emerging CI algorithm which is based on the findings extracted from photosynthesis research.

### *17.38.1 Fundamentals of Photosynthetic Algorithm*

Motivated by the principle of Benson-Calvin cycle Phase-1 and the reaction that happens in the chloroplast subcellular compartment for photorespiration, photosynthetic algorithm (PA) was originally proposed in Murase (2000). To perform the PA, the following calculation processes need to be followed (Murase 2000):

- First, randomly generating the intensity of light.
- Second, evaluating the fixation rate of $CO_2$ via the following equation (also refer to as the stimulation function in the PA algorithm) based on the light intensity (Murase 2000). This is a unique characteristic of the PA algorithm. Such stimulation often happens as a result of randomly changed light intensity which in turn adjusts the influential degree on the elements of RuBP [i.e., ribulose-1, 5-bishosphate (Carpentier 2011)] by photorespiration as shown in Eq. 17.120.

$$C = \frac{V_{\max}}{1 + A/L}, \tag{17.120}$$

where the $CO_2$ fixation rate is denoted by $C$, $V_{\max}$ represents the maximum $CO_2$ fixation rate, $A$ stands for the affinity of $CO_2$, and $L$ is used to express the light intensity.

- Third, based on the fixation rate obtained from the stage above, one of two cycles, either Benson-Calvin or photorespiration will be selected at this stage. For both cycles, Murase (2000) utilized 16-bit strings which shuffles based on carbon molecules recombination rule in photosynthetic pathways.
- Then after certain rounds of iterations, an amount of GAPs, i.e., glyceraldehyde-3-phosphate (Dubinsky 2013), are generated for representing intermediate knowledge strings in the PA algorithm. Each GAP is composed of 16 bits. The fitness of these GAPs will be evaluated at this stage. The best fit GAP will remain as a DHAP [i.e., di-hydroxyacetone phosphate (Carpentier 2011)] which is referred to as the current estimated value.

Taking into account the fundamental process described above, the steps of implementing PA can be summarized as follows (Murase 2000; Alatas 2011; Yang 2005):

- Step 1: Initializing the following problem parameters such as $f(x)$ (the object function), $x_i$ (the decision variable), $N$ (the number of decision variables), and the boundary of constraints.
- Step 2: Initializing the following problem parameters such as DHAPs, and $CO_2$ fixation parameters (e.g., affinity $A$, maximum fixation rate $V_{\max}$, and light intensity $L$).
- Step 3: Calculating $CO_2$ concentration, determining $O_2/CO_2$ concentration ration, and setting Benson-Calvin/photorespiration frequency ratio.
- Step 4: Evaluating if the stopping criteria are met. If yes, the algorithm stops; otherwise, go to the next step.
- Step 5: Depending the fixation rate of $CO_2$, the 16-bit strings are shuffled in either Benson-Calvin or photorespiration cycle.
- Step 6: Comparing the fitness value where the poor results will be removed and the desired DHAP strings and results will be remained.
- Step 7: Updating the light intensity and the next round of iteration of the PA algorithm starts.

### *17.38.2 Performance of PA*

In order to verify the proposed PA, the finite element inverse analysis problem was employed in Murase (2000). The prediction of the elastic moduli of the finite element model via PA was quite satisfied. The overall performance demonstrated by this preliminary application make PA a very attractive optimization algorithm.

## 17.39 Population Migration Algorithm

In this section, we will introduce an emerging CI algorithm that is inspired by the population migrating mechanism (Ramachandran 2012a, b, c).

### *17.39.1 Fundamentals of Population Migration Algorithm*

Population migration algorithm (PMA) was originally proposed in Zhou and Mao (2003). There are several variants and application can be found in the literature (Zhang et al. 2009; Zhang and Zhou 2009; Wang et al. 2010; Lu and Liu 2011; Zhao and Liu 2009, 2011). To implement PMA, the following components need to be considered (Zhang and Zhou 2009; Zhou and Mao 2003):

- Component 1: In PMA, the social-cooperation strategy of PMA can be defined by Eq. 17.121 (Zhang and Zhou 2009):

$$\alpha^t = [x_{best}, 1, \text{population migration}, x_{best}]. \tag{17.121}$$

- Component 2: In PMA, the self-adaptation strategy can be divided into two parts, namely, population flow and population proliferation. Mathematically, the self-adaptation mechanism can be defined as Eq. 17.122 (Zhang and Zhou 2009):

$$\beta^t = \left[\left(\text{pop}_{flow},\ \text{pop}_{proliferation}\right)\text{shrinkage the beneficial region}\right]. \tag{17.122}$$

- Component 3: Competition, i.e., population updating strategy can be described as Eq. 17.123 (Zhang and Zhou 2009):

$$\gamma^t = [\mu = \lambda, (\mu, \lambda),\ \text{record and update } x_{best} \text{ and } f(x_{best})]. \tag{17.123}$$

### *17.39.2 Performance of PMA*

To verify the proposed PMA, a set of experimental studies were conducted in Zhou and Mao (2003). The simulation results demonstrated that PMA is a very attractive optimization problem solver.

## 17.40 Roach Infestation Optimization

In this section, we will introduce an emerging CI algorithm that is based on the collective behaviour of some insects, e.g., roach (Bater 2007; Chapman 2013).

### 17.40.1 Fundamentals of Roach Infestation Optimization Algorithm

Roach infestation optimization (RIO) algorithm was proposed by Havens et al. (2008) that inspired by the recent observation of cockroaches' social (both collective and individual) behaviours. Typically, there are three types of behaviour are employed in RIO, i.e., search behaviour, social behaviour, and hungry (foraging) behaviour. Each one is outlined as follows (Havens et al. 2008):

- Search behaviour (*Find_Darkness*): As RIO is a cockroach-inspired PSO, this behaviour is defined as Eq. 17.124 (Havens et al. 2008):

$$\vec{v}_i = C_0\vec{v}_i + C_{\max}\vec{R}_{1\cdot} * \left(\vec{p}_i - \vec{x}_i\right), \tag{17.124}$$

  where $\vec{v}_i$ is the velocity of the $i$th agent (cockroach), $\vec{x}_i$ is the current location, $\vec{p}_i$ is the best location found by the $i$th agent, $\{C_0, C_{\max}\}$ are parameters, $\vec{R}_1$ is a vector of uniform random numbers and $.*$ is element-by-element vector multiplication.
- Social behaviour (*Find_Friends*): The agents will socialize and share their information by setting the darkest local location $(\vec{l})$ as shown in Eq. 17.125 (Havens et al. 2008):

$$\vec{l}_i = \vec{l}_j = \arg\min_k \left\{F\left(\vec{p}_k\right)\right\}, \quad k = \{i,j\}, \tag{17.125}$$

  where $\vec{l}_i$ is the group best solution, $\{i,j\}$ are the indices of the two socializing cockroaches, and $\vec{p}_k$ is darkest known location for the individual cockroach agent (i.e., the best personal location).
- Hunger behaviour (*Find_Food*): To model this behaviour, a parameter called hunger counter $(hunger_i)$ is employed. The main procedure deals with the hungry degree checking of the agents which based on a threshed $(t_{hunger})$. If not (i.e., $hunger_i < t_{hunger}$), update the agent's velocity as shown in Eq. 17.125 (Havens et al. 2008):

$$\begin{aligned}\vec{v}_i &= C_0\vec{v}_i + C_{\max}\vec{R}_{1\cdot} * \left(\vec{p}_i - \vec{x}_i\right) + C_{\max}\vec{R}_{2\cdot} * \left(\vec{l}_i - \vec{x}_i\right)\\ \vec{x}_i &= \vec{x}_i + \vec{v}_i,\end{aligned} \tag{17.126}$$

  otherwise, the agent will be transported to other food location $(\vec{b})$ randomly. Also, this piece of food is randomly relocated.

### 17.40.2 Performance of RIO

To check the efficiency of RIO, a number of numerical examples were adopted in Havens et al. (2008), such as Sphere function, Rastrigin function, Rosenbrock function, Ackley function, Griewank function, Michalewicz function, Easom function, and Hump function. Compared with other CI algorithms (e.g., PSO), the proposed algorithm is more effective for optimizing highly-modal function.

## 17.41 Saplings Growing Up Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the behaviours observed from saplings (Schnell and Priyadarshan 2012; Tidball and Krasny 2014; Reece et al. 2011).

### 17.41.1 Fundamentals of Saplings Growing Up Algorithm

Saplings growing up algorithm (SGuA) was originally proposed in Karci (2007a, b, c) and Karci and Alatas (2006). To implement SGuA, the following steps need to be performed (Karci 2007a, b, c; Karci and Alatas 2006):

- Sowing phase: In sowing phase, a uniform population method and divide-and-generate paradigm was proposed for initial population generating. Initially, two saplings are set where $S_0 = \{u_1, u_2, \ldots, u_n\}$ and $S_1 = \{l_1, l_2, \ldots, l_n\}$, $n$ is the length of sampling and this case the dividing factor $(k)$ is considered as $k = 1$, $u_i$and $l_i$ are upper and lower bounds for corresponding variables. Then the factor $k$ is determined. For $k = 2$ and two extra $S_2$ and $S_3$ are divided from $S_0$ and $S_1$ as shown in Eqs. 17.127 and 17.128, respectively (Karci 2007a, b, c; Karci and Alatas 2006):

$$S_2 = \left\{r_1, r_2, \ldots, r_{n/2}, r_{n/2+1}, r_{n/2+2}, \ldots, r_n\right\}, \tag{17.127}$$

$$S_3 = \left\{r \cdot l_1, r \cdot l_2, \ldots, r \cdot l_{n/2}, r \cdot u_{n/2+1}, r \cdot u_{n/2+2}, \ldots, r \cdot u_n\right\}, \tag{17.128}$$

  where $r$ is a random number such as $0 \leq r \leq 1$. Let us consider the population $P$ size as $|P|$ and the number of elements in the set of generated saplings $S$ as $|S|$. So if $|S| < |P|$, then the value of $k$ is increased by 1, and $2^3 - 2 = 8 - 2 = 6$ saplings can be derived from $S_0$ and $S_1$, which are not in $S$, since $S_0$ and $S_1$ are divided into three parts. The remaining saplings in the garden will be obtained by applying same method with increasing the value of $k$. This process goes on until $|S| \geq |P|$. Hereafter, the first $|P|$ elements of the set $S$ are taken as saplings' population.

- Growing up phase: This phase contains three operators: mating, branching, and vaccinating operators.
  The aim of mating to generate a new sapling from currently existing saplings (global search) by interchanging current exist information between temporary solutions. In general, the distance between two saplings affects the mating process' taking place or not, and it depends on the distance between current pair (i.e., it has greater probability for near saplings and has small probability for saplings far away to each other). Let $P_m(S_1, S_2)$ can be computed in the following two ways as shown in Eqs. 17.129 and 17.130, respectively (Karci 2007a, b, c; Karci and Alatas 2006):

$$P_m(S_1, S_2) = 1 - \frac{\left(\sum_{i=1}^{n} \left(s_{1,i} - s_{2,i}\right)^2\right)^{\frac{1}{2}}}{R}, \tag{17.129}$$

$$P_m(S_1, S_2) = \frac{\left(\sum_{i=1}^{n} \left(s_{1,i} - s_{2,i}\right)^2\right)^{\frac{1}{2}}}{R}\left(1 - \frac{\left(\sum_{i=1}^{n} \left(s_{1,i} - s_{2,i}\right)^2\right)^{\frac{1}{2}}}{R}\right), \tag{17.130}$$

  where $R = \left(\sum_{i=1}^{n} \left(u_i - l_i\right)^2\right)^{\frac{1}{2}}$, $u_i$ and $l_i$ are upper and lower bounds for corresponding variables.
  Branching: each sapling consists of branches, and initially each sapling contains no branches $(P(s_{1,j}|s_{1,i}) = 1)$ and it is a body. In order to grow up a branch on any point (i.e., a new sapling) from currently exist saplings, the author used probabilistic method for determination of branch position depending on the currently exist branches position. It aims at embedding/removing new knowledge into/from the current solutions set. Let $S_1 = s_{1,1}s_{1,2}\ldots s_{1,i}\ldots s_{1,n}$ be a sapling. If a branch occurs in point $s_{1,i}$, then the probability of this pint could be calculated in two ways listed below (see Eqs. 17.131 and 17.132, respectively) (Karci 2007a, b, c; Karci and Alatas 2006): linear and non-linear. The distance between $s_{1,i}$ and $s_{1,j}$ (where $i \neq j$) can be considered as $|j - i|$ or $|i - j|$.

$$\text{linear case: } P(s_{1,j}|s_{1,i}) = 1 - \frac{1}{(|j - i|)^2}, \quad i \neq j, \tag{17.131}$$

$$\text{non-linear case: } P(s_{1,j}|s_{1,i}) = 1 - \frac{1}{e^{(|j-i|)^2}}, \quad i \neq j. \tag{17.132}$$

Vaccinating: aims to generate new saplings from currently exist saplings which are similar; since the dissimilarity of saplings affects the success of vaccinating process (i.e., vaccinating success is proportional to the dissimilarity of both saplings). In SGuA, if $\text{Dis}(S_1, S_2) \geq$ threshold, the dissimilarity of saplings is computed in the following two ways shown in Eqs. 17.133 and 17.134, respectively (Karci 2007a, b, c; Karci and Alatas 2006):

$$S_1 = \begin{cases} s_{1,i} & \text{if } s_{1,i} = s_{2,i} \\ random(1) & \text{if } s_{1,i} \neq s_{2,i} \end{cases}, \tag{17.133}$$

$$S_2 = \begin{cases} s_{2,i} & \text{if } s_{2,i} = s_{1,i} \\ random(1) & \text{if } s_{2,i} \neq s_{1,i} \end{cases}, \tag{17.134}$$

where $S_1$ and $S_2$ are obtained as consequence of applying vaccinating process to $S_1$ and $S_2$; $random(1)$ generates a random number which is 0 or 1.
The initial value of threshold depends on the problem solvers. For example, $G$ and $H$ are saplings and the similarity of $S_1$ and $S_2$ is $\text{Dis}\,(S_1, S_2) = \sum_{i=1}^{n} \left|s_{1,i} - s_{2,i}\right| / u_i - l_i$. If $\text{Dis}\,(S_1, S_2) \geq n \cdot \varepsilon$, where $\varepsilon$ is a user-defined constant $(0 < \varepsilon < 1)$, then $S_1$ and $S_2$ are vaccinated through Eqs. 17.135 and 17.136, respectively (Karci 2007a, b, c; Karci and Alatas 2006):

$$S_1 = \begin{cases} s_{1,i} & \text{if } \frac{\left|s_{1,i} - s_{2,i}\right|}{u_i - l_i} \leq \varepsilon \\ s_{2,i} & \text{if } \frac{\left|s_{1,i} - s_{2,i}\right|}{u_i - l_i} > \varepsilon \end{cases}, \tag{17.135}$$

$$S_2 = \begin{cases} s_{2,i} & \text{if } \frac{\left|s_{1,i} - s_{2,i}\right|}{u_i - l_i} \leq \varepsilon \\ s_{1,i} & \text{if } \frac{\left|s_{1,i} - s_{2,i}\right|}{u_i - l_i} > \varepsilon \end{cases}. \tag{17.136}$$

In fact, the vaccinating process is opposite to mating process, since vaccinating operator uses dissimilarity in the garden. Thus, the vaccinating operator can also compute the distance between saplings and then compute the probability of saplings as defined by Eq. 17.137 (Karci 2007a, b, c; Karci and Alatas 2006):

$$P_v(S_1, S_2) = \frac{\left(\sum_{i=1}^{n} \left(s_{1,i} - s_{2,i}\right)^2\right)^{1/2}}{R}, \tag{17.137}$$

where $P_v(S_1, S_2)$ is the probability of $S_1$ and $S_2$ to be vaccinated.

### *17.41.2 Performance of SGuA*

Compare with GA, the SGuA has some unique characteristics (Karci 2007b): (1) it uses objective function for determination of quality of saplings in contrast to GA due to the difficulty of defining fitness function; (2) the SGuA uses less parameter determined by the user (only one for vaccinating operator) and obtained better results with less time steps with respect to GA; (3) it uses similarity and dissimilarity properties in to current solutions set with property of new information not adding in neighbor points which GA did; (4) GA is a global search method but SGuA contains both local and global search steps. Furthermore, one of the unique

features of the PA is that the operator within SGuA can be applied in two different ways: sequentially and separately. Those processes allow the SGuA more flexible and faster than others.

## 17.42 Seeker Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is inspired by the act of human searching behaviour.

### *17.42.1 Fundamentals of Seeker Optimization Algorithm*

Seeker optimization algorithm (SeOA) was originally proposed in Dai et al. (2006, 2007). There are several variants and applications can be found in the literature (Dai et al. 2009a, b, 2010a, b; Shaw et al. 2011)

- Step 1: Initialization. Creating $S$ positions which are described as Eq. 17.138 (Dai et al. 2007):

$$\{x_i(t)|x_i(t) = (x_{i1}, x_{i2}, \ldots, x_{iD}); \quad i = 1, 2, \ldots, S; \quad t = 0\}. \tag{17.138}$$

  The positions are randomly and uniformly distributed in the parametric space.

- Step 2: Computing and evaluating each seeker's fitness value.
- Step 3: Performing searching strategy. Giving search variables which includes centre position vector, searching direction, searching radius, and trust degree.
- Step 4: Updating position. The new position of each seeker can be computed through Eq. 17.139 (Dai et al. 2007):

$$\vec{x}(t+1) = \vec{c} + \vec{d} \cdot \vec{r} \cdot \sqrt{-\log(\vec{\mu})}. \tag{17.139}$$

- Step 5: Checking whether the stopping criterion is met. If yes, terminating the algorithm; otherwise, return to Step 3.

### *17.42.2 Performance of SeOA*

To verify the proposed SeOA, 6 typical testing functions with varying complexities and number of variables were employed in Dai et al. (2007). These functions included such as Goldstein and Price function, De Jong's function 2, and Griewangk's function. In comparison with other CI algorithms (e.g., GA, PSO), SeOA outperformed its competitors in all cases.

## 17.43 Self-organising Migrating Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the competitive-cooperative behaviour of intelligent creatures.

### *17.43.1 Fundamentals of Self-organising Migrating Algorithm*

Self-organising migrating algorithm (SOMA) was first proposed in Zelinka and Lampinen (2000). There are several variants and applications can be found in the literature (Nolle et al. 2005; Senkerik et al. 2010; Zelinka et al. 2009; Davendra and Zelinka 2009; Davendra et al. 2013). Two evolutionary operators (i.e., mutation and crossover) are employed in SOMA to maintain the perturbation of individuals and movement of an element. The main steps of SOMA are outlined as follows (Davendra et al. 2013):

- Step 1: Initializing the parameters.
- Step 2: Creation the populations. The population is generated via Eqs. 17.140 and 17.141, respectively (Davendra et al. 2013):

$$P = \left\{X_1^t, X_2^t, \ldots, X_\beta^t\right\}, \tag{17.140}$$

$$X_i^t = \left(x_{i,1}^t, x_{i,2}^t, \ldots, x_{i,n}^t\right), \quad \text{where } x_{i,j}^t : \begin{cases} i = 1, \ldots, \beta \\ j = 1, \ldots, N \end{cases}, \tag{17.141}$$

  where $\beta$ is the number of individuals, $x_{i,j}^t$ represents the element in each individual, and $N$ is the dimension of the problem.

- Step 3: Iterative procedure. First, Evaluate the cost function of each individual as follows (Davendra et al. 2013):

$$C_i^t = f\left(X_i^t\right),\ i = 1, \ldots, \beta. \tag{17.142}$$

Second, create the perturbation matrix $(\mathbf{A}_{i,j}(PRT))$ for each element $(x_{i,j}^t)$ in an individual $(X_i^t)$ as defined by Eq. 17.143 (Davendra et al. 2013):

$$\mathbf{A}_{i,j} = \begin{cases} 1 & \text{if } rand(\,) < PRT \\ 0 & \text{otherwise} \end{cases}, \begin{cases} i = 1, 2, \ldots, \beta \\ j = 1, 2, \ldots, N \end{cases}, \tag{17.143}$$

where $PRT \in [0, 1]$ is a parameter that to achieve perturbation.

Third, all individuals perform their run towards the selected individual (leader), which has the best fitness for the migration as follows (Davendra et al. 2013):

$$x_{i,j}^{t} = x_{i,j}^{t-1} + \left(x_{L,j}^{t-1} - x_{i,j}^{t-1}\right) s\mathbf{A}_{i,j}, \tag{17.144}$$

where $x_{i,j}^{t}$ is new candidate individual, $x_{i,j}^{t-1}$ is the original individual, $x_{L,j}^{t-1}$ is the leader individual, $s \in [0, \text{path length}]$, and $\mathbf{A}_{i,j}$ is perturbation matrix.

Fourth, Calculate the cost function and keep the best solutions.

- Step 4: Post process and visualize results.

### *17.43.2 Performance of SOMA*

To test the performance of SOMA, a set of experimental studies are conducted in Zelinka and Lampinen (2000). Computational results showed that SOMA is very competitive.

## 17.44 Sheep Flock Heredity Model Algorithm

In this section, we will introduce an emerging CI algorithm that is based on some observations from sheep herd (Mills et al. 2010).

### *17.44.1 Fundamentals of Sheep Flock Heredity Model Algorithm*

Sheep flock heredity model (SFHM) algorithm was originally proposed in Nara et al. (1999) and Kim and Ahn (2001). There are several applications can be found in the literature (Chandrasekaran et al. 2006; Subbaiah et al. 2009; Anandaraman 2011; Venkumar and Sekar 2012; Anandaraman et al. 2012; Mukherjee et al. 2012). The natural evolution phenomenon of sheep flocks can be associated to the genetic operations of string which we can define the following two kinds of genetic operation: (1) Traditional genetic operations between two strings; and (2) Genetic operations between sub-strings within one string. This kind of genetic operation is referred to as the "multi-stage genetic operation". In summary, to implement SFHM algorithm, the following steps need to be performed (Chandrasekaran et al. 2006; Nara et al. 1999; Mukherjee et al. 2012; Kim and Ahn 2001):

- Step 0: Initializing the population of artificial sheep herd.
- Step 1: Selecting the parent, setting the probability of sub-chromosome level crossover, performing the sub-chromosome level of crossover.
- Step 2: Selecting two sequences from population, setting crossover probability, performing chromosome level of crossover.
- Step 3: Checking the termination condition.

### 17.44.2 Performance of SFHM

To verify the proposed SFHM, Kim and Ahn (2001) tested it on a 23 generators' maintenance scheduling problem. It was assumed in the study that the system load will increase by 2 % per annual. The simulation results showed that the proposed SFHM outperform its competitor algorithm with a better solution quality and almost twice of the calculation times' reduction.

## 17.45 Simple Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on two very simple mechanisms, namely, exploration and exploitation.

### 17.45.1 Fundamentals of Simple Optimization Algorithm

Simple optimization (SPOT) algorithm, a population-based approach, was recently proposed in Hasançebi and Azad (2012). Briefly, the SPOT algorithm can be summarized as follows (Hasançebi and Azad 2012):

- Step 1: Generating a population of stochastically proposed candidate solutions and initiating the exploration procedure.
- Step 2: Evaluating the members of population according to an objective function.
- Step 3: Determining the best candidate solution among the whole population group.
- Step 4: Calculating the standard deviation of each column of population.
- Step 5: Starting the exploitation process and creating a set of new candidate solutions through Eq. 17.145 (Hasançebi and Azad 2012):

$$x_{new(i)} = x_{best(i)} + \lambda_2 \cdot R_{(i)}, \tag{17.145}$$

  where $\lambda_2 = 0.5\lambda_1$, in comparison with the exploration stage.
- Step 6: Evaluating the newly created candidate solutions.
- Step 7: Replacing the worst population members with the better new ones.
- Step 8: Determining the best candidate solution within the population.
- Step 9: Calculating the standard deviation again for each column of population.
- Step 10: Activating the exploration procedure and creating a set of new candidate solutions through Eq. 17.146 (Hasançebi and Azad 2012):

$$x_{new(i)} = x_{best(i)} + \lambda_1 \cdot R_{(i)}, \tag{17.146}$$

where $\lambda_1$ stands for a positive constant, and $R_{(i)}$ represents a normal distributed random number with a mean zero and a stand deviation of $\sigma_{R(i)}$.

- Step 11: Evaluating the newly created candidate solutions.
- Step 12: Checking whether the stopping criterion is met. If not, repeating the algorithm from Step 3; otherwise, terminates the algorithm.

### *17.45.2 Performance of SPOT*

Hasançebi and Azad (2012) employed two well-known benchmark engineering optimization problems, namely, welded beam and pressure vessel design optimization, to verify the proposed SPOT algorithm. Compared with the best available methods found in the literature, the results obtained by SPOT is very attractive.

## 17.46 Slime Mold Algorithm

In this section, we will introduce an emerging CI algorithm that is based on slime mold related studies (Newell 1978).

### *17.46.1 Fundamentals of Slime Mold Algorithm*

Slime mold algorithm (SMA) was recently proposed in Li et al. (2011). There are several slime mold related applications can be found in the literature (Umedachi et al. 2010; Tero et al. 2010; Adamatzky and Oliveira 2011; Li et al. 2011; Shann 2008) To implement SSOA, the following steps need to be performed (Li et al. 2011):

- Step 1: Initializing slime mold around the one or more food sources.
- Step 2: The slime mold will stream towards a newly introduced food source based on a gradient descent rule. The total field $\varphi$ is computed via Eq. 17.147 (Li et al. 2011):

$$\varphi = \sum_{i=1}^{N} (f_i - \bar{f}) \cdot \ln\left(|\mathbf{x} - \mathbf{x}_i|^2\right), \tag{17.147}$$

where the number of discovered food sources is denoted by $N$, $f_i$ is the remaining nutrient amount found at the $i$th food source, and $\bar{f}$ is the mean of all $f_i$.

- Step 3: Once slime mold arrives at a newly discovered food resource, such food resource will be linked to the network. Nutrient values of the new connected food resources will gradually decay.

- Step 4: Checking whether the stopping criterion is met. If yes, the algorithm terminates; otherwise, i.e., unconnected food sources still existing, return to Step 2.

### *17.46.2 Performance of SMA*

Based on the proposed SMA, Li et al. (2011) presented two self-organizing routing protocols for wireless sensor networks by considering both efficiency and robustness. The simulation conducted in their study proved that the proposed protocol is effective in building network connectivities, with a trade-off between efficiency and robustness.

## 17.47 Social Emotional Optimization Algorithm

In this section, we will introduce an emerging CI algorithm which is inspired from the human society. As we know, group decisions are very important for us and they have been studied for millennia. They are range from small-scale decisions, e.g., some advices taken by groups of relatives, friends or colleagues, to large-scale decisions, e.g., nation-wide democratic electrons and international agreements (Conradt and List 2009).

### *17.47.1 Fundamentals of Social Emotional Optimization Algorithm*

Social emotional optimization algorithm (SEOA) was originally proposed in Xu et al. (2010), Wu et al. (2011), Wei et al. (2010), Cui and Cai (2010), Chen et al. (2010b) and Cui et al. (2010, 2011). Each person is viewed as a solution. Through cooperation and competition mechanisms, the personal social status will be increased and the best one will win and output as the final solution. The main steps of SEOA are outlined as follows:

- Step 1: Initializing all individuals randomly in the search space. In the fist step, all individuals' emotion indexes are set to 1 as shown in Eq. 17.148 (Cui et al. 2012):

$$\vec{x}_j(1) = \vec{x}_j(0) \oplus \text{Manner}_1, \tag{17.148}$$

where $\vec{x}_j(1)$ represents the social position of the $j$th individual in the initialization period, $\oplus$ means the operation. The movement phase of $\text{Manner}_1$ is defined by Eq. 17.149 (Cui et al. 2012):

$$\text{Manner}_1 = -k_1 \cdot rand_1 \cdot \sum_{w=1}^{L} \left( \vec{x}_w(0) - \vec{x}_j(0) \right), \tag{17.149}$$

where $k_1$ is a parameter used to control the emotion changing size, *rand* is a random number with uniform distribution, $L$ represents the worst individuals that are selected to provide a remainder for the $j$th individual to avoid the wrong behaviour.

- Step 2: Computing the fitness value of each individual according to the objective function.
- Step 3: For the $j$th individual, determining the value $\vec{X}_{j,best}(0)$.
- Step 4: For all population, determining the value $\vec{Status}_{best}(0)$.
- Step 5: Determining three emotional index via Eq. 17.150 (Cui et al. 2012):

$$\begin{cases} \vec{x}_j(t+1) = \vec{x}_j(t) \oplus \text{Manner}_2 & \text{If } BI_j(t+1) < TH_1 \\ \vec{x}_j(t+1) = \vec{x}_j(t) \oplus \text{Manner}_3 & \text{If } TH_1 \le BI_j(t+1) < TH_2 \\ \vec{x}_j(t+1) = \vec{x}_j(t) \oplus \text{Manner}_4 & \text{otherwise} \end{cases}, \tag{17.150}$$

where $TH_1$ and $TH_2$ are two thresholds aiming to restrict the different behaviour manner.

- Step 6: Determining different decisions according to Eqs. 17.151–17.153, respectively (Cui et al. 2012):

$$\text{Manner}_2 = k_3 \cdot rand_3 \cdot \left( \vec{X}_{j,best}(t) - \vec{x}_j(t) \right) + k_2 \cdot rand_2 \cdot \left( \vec{Status}_{best}(t) - \vec{x}_j(t) \right), \tag{17.151}$$

$$\begin{aligned} \text{Manner}_3 = {} & k_3 \cdot rand_3 \cdot \left( \vec{X}_{j,best}(t) - \vec{x}_j(t) \right) \\ & + k_2 \cdot rand_2 \cdot \left( \vec{Status}_{best}(t) - \vec{x}_j(t) \right) \\ & - k_1 \cdot rand_1 \cdot \sum_{w=1}^{L} \left( \vec{x}_w(0) - \vec{x}_j(0) \right), \end{aligned} \tag{17.152}$$

$$\text{Manner}_4 = k_3 \cdot rand_3 \cdot \left( \vec{X}_{j,best}(t) - \vec{x}_j(t) \right) - k_1 \cdot rand_1 \cdot \sum_{w=1}^{L} \left( \vec{x}_w(0) - \vec{x}_j(0) \right), \tag{17.153}$$

where $\vec{Status}_{best}(t)$ represents the best society status position obtained from all people previously, and $\vec{X}_{j,best}(t)$ denotes the best status value obtained by the $j$th

individual previously. Both can be defined by Eqs. 17.154 and 17.155, respectively (Cui et al. 2012):

$$\vec{Status}_{best}(t) = \arg\min\left\{f\left(\vec{x}_w(h)|1 \le h < t\right)\right\}, \tag{17.154}$$

$$\vec{X}_{j,best}(t) = \arg\min\left\{f\left(\vec{x}_j(h)|1 \le h < t\right)\right\}. \tag{17.155}$$

- Step 7: Making mutation operation.
- Step 8: Checking termination criteria, if it is satisfied, output the best solution; otherwise, go to Step 3.

### *17.47.2 Performance of SEOA*

To test the performance of SEOA, a set of benchmark functions were adopted in Cui et al. (2012). Compared with other CI algorithms, SEOA has a remarkable superior performance in terms of accuracy and convergence speed.

## 17.48 Social Spider Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on some findings regarding the spider colony (Bater 2007; Chapman 2013; Levin 2013a, b, c, d, e, f).

### *17.48.1 Fundamentals of Social Spider Optimization Algorithm*

Spider algorithm has been around for a while for dealing with like search engine optimization (Whitehouse and Lubin 1999; Jonassen 2006; Du et al. 2005). Social spider optimization algorithm (SSOA) was recently proposed in Cuevas et al. (2013). To implement SSOA, the following steps need to be performed (Cuevas et al. 2013):

- Step 1: Setting the total number of $n$-dimensional colony members as $N$ and defining the number of male $N_{male}$ and female $N_{female}$ spiders in the entire colony $\mathbf{S}$ based on Eq. 17.156 (Cuevas et al. 2013):

$$\begin{aligned} N_{male} &= N - N_{female} \\ N_{female} &= floor[(0.9 - rand \cdot 0.25) \cdot N], \end{aligned} \tag{17.156}$$

where *rand* stands for a random number which falls within the range of $[0, 1]$, and $floor(\cdot)$ indicates the mapping between a real and an integer numbers.

- Step 2: Initializing stochastically the female and male members and computing the mating radius according to Eq. 17.157 (Cuevas et al. 2013):

$$r = \frac{\sum_{j=1}^{n} \left( p_j^{high} - p_j^{low} \right)}{2 \cdot n}. \tag{17.157}$$

- Step 3: Calculating the weight of each spider in colony $\mathbf{S}$ through Eq. 17.158 (Cuevas et al. 2013):

$$w_i = \frac{J(\mathbf{s}_i) - worst_{\mathbf{s}}}{best_{\mathbf{s}} - worst_{\mathbf{s}}}, \tag{17.158}$$

where $J(\mathbf{s}_i)$ denotes the fitness value acquired through the evaluation of the spider position $\mathbf{s}_i$ with regard to the objective function $J(\cdot)$.

- Step 4: Moving female spiders according to the female cooperative operator modelled as shown in Eq. 17.159 (Cuevas et al. 2013):

$$\mathbf{f}_i^{k+1} = \begin{cases} \mathbf{f}_i^k + \alpha \cdot Vibc_i \cdot \left( \mathbf{s}_c - \mathbf{f}_i^k \right) + \beta \cdot Vibb_i \cdot \left( \mathbf{s}_b - \mathbf{f}_i^k \right) \\ \quad + \delta \cdot \left( rand - \frac{1}{2} \right) \text{ with probability } PF \\ \mathbf{f}_i^k - \alpha \cdot Vibc_i \cdot \left( \mathbf{s}_c - \mathbf{f}_i^k \right) - \beta \cdot Vibb_i \cdot \left( \mathbf{s}_b - \mathbf{f}_i^k \right) \\ \quad + \delta \cdot \left( rand - \frac{1}{2} \right) \text{ with probability } 1 - PF \end{cases}, \tag{17.159}$$

where $\alpha$, $\beta$, $\delta$, and *rand* are random numbers which fall within the range of $[0, 1]$.

- Step 5: Similarly moving male spiders according to the male cooperative operator expressed as Eq. 17.160 (Cuevas et al. 2013):

$$\mathbf{m}_i^{k+1} = \begin{cases} \mathbf{m}_i^k + \alpha \cdot Vibf_i \cdot \left( \mathbf{s}_f - \mathbf{m}_i^k \right) + \delta \cdot \left( rand - \frac{1}{2} \right) & \text{if } w_{N_{female}+i} > w_{N_{female}+m} \\ \mathbf{m}_i^k + \alpha \cdot \left( \frac{\sum_{h=1}^{N_{male}} \mathbf{m}_h^k \cdot w_{N_{female}+h}}{\sum_{h=1}^{N_{male}} w_{N_{female}+h}} - \mathbf{m}_i^k \right) & \text{if } w_{N_{female}+i} \leq w_{N_{female}+m} \end{cases}, \tag{17.160}$$

where $\mathbf{s}_f$ indicates the nearest female spider to the male individual.

- Step 6: Performing the mating operation.
- Step 7: Checking whether the stopping criterion is satisfied. If yes, the algorithm terminates; otherwise, return to Step 3.

### 17.48.2 Performance of SSOA

Different to other evolutionary algorithms, in SSOA, each individual spider is modelled by taking its gender into account. This design allows incorporating computational mechanisms to avoid critical flaws and incorrect exploration-exploitation trade-off. In order to show how the SSOA performs, Cuevas et al. (2013) collected a comprehensive set of 19 benchmark test function from the literature. In comparison with other CI algorithms (e.g., PSO), the experimental results confirmed an acceptable performance of the SSOA in terms of the solution quality.

## 17.49 Society and Civilization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on some findings regarding the human social behaviour (Irvine 2013; Müller 2013; Gross 2014; Bhugra et al. 2013; Adair 2007; Chen and Lee 2008; Savage 2012; Magstadt 2013; Chalmers 2010).

### 17.49.1 Fundamentals of Society and Civilization Algorithm

Society and civilization algorithm (SCA) was recently proposed in Ray and Liew (2003). To implement US algorithm, the following steps need to be performed (Ray and Liew 2003):

- In SCA, for each individual, $c$ indicates the constraint satisfaction vector denoted by $c = [c_1, c_2, \ldots, c_s]$ as shown in Eq. 17.161 (Ray and Liew 2003):

$$c_i = \begin{cases} 0 & \text{if } i\text{th constraint satisfied} \\ & i = 1, 2, \ldots, s \\ -g_i(\mathbf{x}) & \text{if } i\text{th constraint violated} \\ & i = 1, 2, \ldots, q \\ h_i(\mathbf{x}) - \delta & \text{if } i\text{th constraint violated} \\ & i = q+1, q+2, \ldots, q+r \\ \delta - h_i(\mathbf{x}) & \text{if } i\text{th constraint violated} \\ & i = q+r+1, q+r+2, \ldots, s \end{cases}. \tag{17.161}$$

- Creating $N$ random individuals standing for a civilization.
- Evaluating each individual according to the computed objective function value and constraints.

- Building societies so that $\bigcup_{i=1}^{K(t)} Soc_i(t) = Civ(t)$ and $Soc_i(t) \cap Soc_j(t) = \phi$, for $i \neq j$.
- Identifying society leaders so that $Soc\_L_i(t) \subset Soc_i(t)$, $i = 1, 2, \ldots, K(t)$.
- Performing move function $Move(\ )$.

### *17.49.2 Performance of SCA*

In order to show how the SCA algorithm performs, Ray and Liew (2003) employed 4 well-studied benchmark engineering design optimization problems such as welded beam design, spring design, speed reducer design, and the three-bar truss design. Compared with the best results obtained from the literature, the proposed SCA performed consistently well on all cases. Meanwhile, the algorithm exhibits a robust convergence for all selected problems which make it a very attractive optimization algorithm.

## 17.50 Stem Cells Optimization Algorithm

During the past three decades, our understanding of embryonic cells has increased dramatically. The humbling beginning started thirty years ago when embryonic stem cells were first cultured from mouse embryos. It was only 15 years later, we were able to derive human embryonic stem cells from human embryos that were donated from early blastocysts which are no more need for in vitro fertilization (Sell 2013). Throughout all creature's life, the balance between cell birth and death is largely regulated by complex genetic systems in response to various growth and death signals. During this dynamic procedure, stem cells are present within most if not all multicellular organisms and are crucial for developing, tissue repairing, as well as aging and cancer (Resende and Ulrich 2013; Sell 2013). Briefly, stems cells can be defined as biological cells which have the ability of self-renewal and the capability in differentiating into various cell types (Sell 2013). They are thus regarded as one of the most important biological components which is essential to the proper growth and development in the process of embryogenesis. Since the detailed information regarding the stem cells is out of the scope of the present book, interested readers are referred to the corresponding studies, e.g., (Resende and Ulrich 2013; Sell 2013), for more recent advancement in this field.

For the rest of this section, we will introduced an emerging CI algorithm which is based on the findings of some important characteristics of stem cells.

### 17.50.1 Fundamentals of Stem Cells Optimization Algorithm

Stem cells optimization algorithm (SCOA) was originally proposed in Taherdangkoo et al. (2011, 2012b). To perform the SCOA algorithm, the following procedure needs to be followed (Taherdangkoo et al. 2012b):

- First, dividing the problem space into sections. The process can be accomplished totally in a random manner;
- Second, generating the initial population randomly and uniformly distributed in the whole search space of the target problem. At this stage, similar to most optimization algorithms, a variable matrix needs to be established for the purpose of obtaining a feedback with respect to problem variables. In SCOA, the key stem cells features are used to form the initial variable matrix. Such features may include liver cells, intestinal cells, blood cells, neurons, heart muscle cells, pancreatic islets cells, etc. Basically, the initial matrix can be express as Eq. 17.162 (Taherdangkoo et al. 2012b):

$$\text{Population} = \begin{bmatrix} X_1 \\ X_2 \\ \cdots \\ X_N \end{bmatrix}, \tag{17.162}$$

  where $X_i = \text{Stem Cells} = [SC_1, SC_2, \ldots, SC_N]$; $i = 1, 2, \ldots, N$.
  In SCOA, some initialized parameters are defined as follows: $M$ represents the maximum of stem cells; $P$ stands for population size $(10 < P \leq M)$; $C_{Optimum}$ indicates the best of stem cell in each iteration; $\chi$ denotes the penalty parameter which is used to stop the growth of stem cell; and $sc^i$ is the $i$th stem cell in the population.
- Third, the cost of each stem cell is obtained a criterion function which is determined based on the nature of the target problem. In SCOA, two types of memory, namely, local- and global-memory, are defined for each cell in which the local-memory is used to store the cost of each stem cell, and the global-memory stores the best cost among all locally stored cost values;
- Then, a self-renewal process will be performed which involves only the best cells of each area. At this stage, the information of each area's best cells will be shared and the cell that possesses the best cost will thus be chosen. In SCOA, such cell is designed to play a more important role than other cells. Briefly, the stem cells' self-renewal operation is computed through Eq. 17.163 (Taherdangkoo et al. 2012b):

$$SC_{Optimum}(t+1) = \zeta SC_{Optimum}(t), \tag{17.163}$$

  where the iteration number is denoted by $t$, $SC_{Optimum}$ represents the best stem cell found in each iteration, and $\zeta$ is a random number which falls within $[0, 1]$.

- Next, the above mentioned procedure will continue until the SCOA arrives at the goal of getting the best cell while keeping the value of cost function as low as possible. This is acquired via Eq. 17.164 (Taherdangkoo et al. 2012b):

$$x_{ij}(t+1) = \mu_{ij} + \varphi\big(\mu_{ij}(t) - \mu_{kj}(t)\big), \tag{17.164}$$

  where the $i$th stem cell position for the solution dimension $j$ is represented by $x_{ij}$, the iteration number is denoted by $t$, two randomly selected stem cells for the solution dimension $j$ are denoted by $\mu_{ij}$ and $\mu_{kj}$, respectively, and $\varphi(\tau)$ (if $\mu_{ij}(t) - \mu_{kj}(t) = \tau$) generates a random variable falls within $[-1, 1]$.
- Finally, the best stem cell is selected when it has the most power relative to other cells. The comparative power can be computed via Eq. 17.165 (Taherdangkoo et al. 2012b):

$$\varsigma = \frac{SC_{Optimum}}{\sum_{i=1}^{N} SC_i}, \tag{17.165}$$

  where $\varsigma$ stands for stem cells' comparative power, $SC_{Optimum}$ denotes the stem cells selected in terms of cost, and $N$ represents the final population size, i.e., when the best solution is obtained and the SCOA algorithm terminates.

### *17.50.2 Performance of SCOA*

Similar to other optimization algorithms, SCOA is also a population-based algorithm. But the difference between SCOA and other CI algorithms lies in that it employs minimal constraints and thus has a simpler implementation. The converging speed of SCOA is faster than other algorithms in virtue of its simplicity and its capability of escaping from local minima (Taherdangkoo et al. 2012b).

## 17.51 Stochastic Focusing Search Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the behaviours observed from human randomized search.

### *17.51.1 Fundamentals of Stochastic Focusing Searching Algorithm*

Stochastic focusing searching (SFS) algorithm was recently proposed in Zheng et al. (2009). In order to implement SFS, the following steps need to be considered (Zheng et al. 2009; Wang et al. 2008):

- In SFS, the targeted optimization problems are regarded as minimization problems, and the particles are controlled according to Eqs. 17.166–17.168, respectively (Zheng et al. 2009).

$$\vec{v}_i(t) = \begin{cases} Rand(\,) \cdot [R_{ti} - \vec{x}_i(t-1)] & \text{if } fun[\vec{x}_i(t-1)] \geq fun[\vec{x}_i(t-2)] \\ \vec{v}_i(t-1) & \text{if } fun[\vec{x}_i(t-1)] < fun[\vec{x}_i(t-2)] \end{cases}, \tag{17.166}$$

$$\vec{x}_i(t) = \vec{v}_i(t) + \vec{x}_i(t-1), \tag{17.167}$$

$$\vec{x}_i(t) = \vec{x}_i(t-1) \quad \text{if } fun[\vec{x}_i(t)] \geq fun[\vec{x}_i(t-1)], \tag{17.168}$$

  where $fun[\vec{x}_i(t)]$ denotes the objective function value of $\vec{x}_i(t)$, and $R_{ti}$ represents a random chosen position in the neighbour space $R_t$ of $\vec{g}_{best}$.
- It can be observed that each individual particle search in a decreasing $R_t$. Therefore, an appropriate selection of $w$is crucial not only to the convergence of particles, but also to the avoidance of local optimal. Accordingly $w$ can be defined by Eq. 17.169 (Zheng et al. 2009):

$$w = \left(\frac{G-t}{G}\right)^{\delta}, \tag{17.169}$$

  where the maximum generation is denoted by $G$, and $\delta$ denotes a positive number.

### *17.51.2 Performance of SFS*

To evaluate the proposed SFS, Zheng et al. (2009) employed a test suite of benchmark functions collected from the literature. The testing function group consists of a diverse set of problems such as functions 1–5 are unimodal, function 8–13 are multimodal function, and function 14–23 are low-dimensional function. In comparison with other CI algorithms (e.g., DE, PSO), the experimental results demonstrated that SFS posses a better global searching capability and faster converging speed for most of the testing cases.

## 17.52 Swallow Swarm Optimization Algorithm

In this section, we will introduce an emerging CI algorithm that is based on the social behaviour of swallow swarm.

### 17.52.1 Fundamentals of Swallow Swarm Optimization Algorithm

Swallow swarm optimization (SSO) algorithm was originally proposed in Neshat et al. (2013). The algorithm shares some similarities with other population-based CI algorithms, e.g., PSO, but with several key differences. Basically, there are three types of swallows in the proposed SSO algorithm (Neshat et al. 2013):

- Explorer swallow ($e_i$): The major population of swallow colony is composed of explorer swallows. Their main task is to search the target problem space. After arriving at a potential solution point, an explorer swallow will use a special to guide the group moving toward such place. If the place is indeed the best one found in the problem space, this swallow will play a role of head leader ($HL_i$). However, if the place is a good but not the best position in comparison with other locations, this swallow will be selected as a local leader ($LL_i$). Otherwise, each explorer swallow $e_i$ will make a random movement according to Eqs. 17.170–17.181, respectively (Neshat et al. 2013):

$$V_{HL_{i+1}} = V_{HL_i} + \alpha_{HL} rand(\,)(e_{best} - e_i) + \beta_{HL} rand(\,)(HL_i - e_i), \tag{17.170}$$

$$\alpha_{HL} = \{\text{if } (e_i = 0 \| e_{best} = 0) \rightarrow 1.5\}, \tag{17.171}$$

$$\alpha_{HL} = \begin{cases} \text{if } (e_i < e_{best}) \&\& (e_i < HL_i) & \rightarrow \frac{rand(\,) \cdot e_i}{e_i \cdot e_{best}} \quad (\text{where } e_i, e_{best} \neq 0) \\ \text{if } (e_i < e_{best}) \&\& (e_i > HL_i) & \rightarrow \frac{2 \cdot rand(\,) \cdot e_{best}}{1/(2 \cdot e_i)} \quad (\text{where } e_i \neq 0) \\ \text{if } (e_i > e_{best}) & \rightarrow \frac{e_{best}}{1/(2 \cdot rand(\,))} \end{cases}, \tag{17.172}$$

$$\beta_{HL} = \{\text{if } (e_i = 0 \| e_{best} = 0) \rightarrow 1.5\}, \tag{17.173}$$

$$\beta_{HL} = \begin{cases} \text{if } (e_i < e_{best}) \&\& (e_i < HL_i) & \rightarrow \frac{rand(\,) \cdot e_i}{e_i \cdot HL_i} \quad (\text{where } e_i, HL_i \neq 0) \\ \text{if } (e_i < e_{best}) \&\& (e_i > HL_i) & \rightarrow \frac{2 \cdot rand(\,) \cdot HL_i}{1/(2 \cdot e_i)} \quad (\text{where } e_i \neq 0) \\ \text{if } (e_i > e_{best}) & \rightarrow \frac{HL_i}{1/(2 \cdot rand(\,))} \end{cases}, \tag{17.174}$$

$$V_{LL_{i+1}} = V_{LL_i} + \alpha_{LL} rand(\,)(e_{best} - e_i) + \beta_{LL} rand(\,)(LL_i - e_i), \tag{17.175}$$

$$\alpha_{LL} = \{\text{if } (e_i = 0 \| e_{best} = 0) \rightarrow 2\}, \tag{17.176}$$

$$\alpha_{LL} = \begin{cases} \text{if } (e_i < e_{best}) \&\& (e_i < LL_i) & \rightarrow \frac{rand(\,) \cdot e_i}{e_i \cdot e_{best}} & (\text{where } e_i, e_{best} \neq 0) \\ \text{if } (e_i < e_{best}) \&\& (e_i > LL_i) & \rightarrow \frac{2 \cdot rand(\,) \cdot e_{best}}{1/(2 \cdot e_i)} & (\text{where } e_i \neq 0) \\ \text{if } (e_i > e_{best}) & \rightarrow \frac{e_{best}}{1/(2 \cdot rand(\,))} & \end{cases}, \tag{17.177}$$

$$\beta_{LL} = \{\text{if } (e_i = 0 \| e_{best} = 0) \rightarrow 2\}, \tag{17.178}$$

$$\beta_{LL} = \begin{cases} \text{if } (e_i < e_{best}) \&\& (e_i < LL_i) & \rightarrow \frac{rand(\,) \cdot e_i}{e_i \cdot LL_i} & (\text{where } e_i, LL_i \neq 0) \\ \text{if } (e_i < e_{best}) \&\& (e_i > LL_i) & \rightarrow \frac{2 \cdot rand(\,) \cdot LL_i}{1/(2 \cdot e_i)} & (\text{where } e_i \neq 0) \\ \text{if } (e_i > e_{best}) & \rightarrow \frac{LL_i}{1/(2 \cdot rand(\,))} & \end{cases}, \tag{17.179}$$

$$V_{i+1} = V_{HL_{i+1}} + V_{LL_{i+1}}, \tag{17.180}$$

$$e_{i+1} = e_i + V_{i+1}. \tag{17.181}$$

Vector $V_{HL_i}$ has an important impact on explorer swallow's behaviour. Each explorer swallow $e_i$ uses the nearest swallow $LL_i$ for the purpose of calculating the vector of $V_{LL_i}$

.

- Aimless swallow $(o_i)$: When the search begins, these swallows doe not occupy a good position. Their task is thus doing random search which means their positions are not affected by $HL_i$ and $LL_i$. The role of an aimless swallow is more like a scout. If an aimless swallow $o_i$ gets a better solution during its searching, it will replace its position with the nearest explorer swallow's position and then continue with search. This process is defined by Eq. 17.182 (Neshat et al. 2013):

$$o_{i+1} = o_i + \left[ rand((-1, 1)) \cdot \frac{rand(\min_s, \max_s)}{1 + rand(\,)} \right]. \tag{17.182}$$

- Leader swallow $(l_i)$: Unlike the PSO which has only one leader particle, in SSO, there may have $n_l$ leader swallows that are distributed or gathered in the problem space. The best leader is called leader head, while the others are called local leader. They are candidate desirable solutions that we are looking for.

### 17.52.2 Performance of SSO

In order to show how the SSO algorithm performs, Neshat et al. (2013) employed 16 benchmark test functions such as Sphere function, Rosenbrock function, Quadric function, Rastrigin function, Rotated Rastrigin function, and Rotated Ackley function. In comparison with other CI techniques (e.g., variant PSO, etc.), the SSO algorithm is one of the best optimization approaches in the category of swarm intelligence. It thus has the ability in solving optimization problems encountered in different scenarios.

## 17.53 Termite-Hill Algorithm

In this section, we will introduce an emerging CI algorithm that is based on hill building behaviour observed from real termites (Keller and Gordon 2009).

### 17.53.1 Fundamentals of Termite-Hill Algorithm

Termite-hill algorithm (ThA) was originally proposed (Zungeru et al. 2012) for dealing with wireless sensor networks routing problem. The key components of ThA are detailed as below (Zungeru et al. 2012):

- Component 1: Pheromone table. Pheromone plays an important role in leveraging an effective and efficient communication in real world (Touhara 2013). Similarly in ThA, the pheromone is updated through Eq. 17.183 (Zungeru et al. 2012):

$$T'_{r,s} = T_{r,s} + \gamma, \tag{17.183}$$

  where $\gamma$ can be expressed as Eq. 17.184 (Zungeru et al. 2012):

$$\gamma = \frac{N}{E - \left(\frac{E_{\min} - N_j}{E_{av} - N_j}\right)}, \tag{17.184}$$

  where $E$ denotes the initial energy of the nodes.
- Component 2: Route selection. In ThA, each of the routing tables carried by the nodes is initialized with a uniform probability distribution defined as Eq. 17.185 (Zungeru et al. 2012):

$$P_{s,d} = \frac{1}{N_k}, \tag{17.185}$$

  where $N_k$ denotes the number of nodes in the network.

### 17.53.2 Performance of ThA

To verify the proposed ThA, Zungeru et al. (2012) conducted a series of experimental studies. The simulation results demonstrated that ThA is a very competitive routing algorithm in terms of energy consumption, throughput, and network lifetime.

## 17.54 Unconscious Search Algorithm

In this section, we will introduce an emerging CI algorithm that is based on some findings from human brain and psychology studies (Irvine 2013; Chalmers 2010; Müller 2013; Gross 2014; Bhugra et al. 2013; Şen 2014).

### 17.54.1 Fundamentals of Unconscious Search Algorithm

Unconscious search (US) algorithm was recently proposed in Ardjmand and Amin-Naseri (2012). The key concept of US is to use some common features found between optimization and psychoanalysis to design search algorithm. To implement US algorithm, the following steps need to be performed (Ardjmand and Amin-Naseri 2012):

- Initially, a group of suitable solutions $P = \left(P_1, P_2, \ldots, P_{|MM|}\right)$ is created. $|MM|$ is the size of the measurement matrix $(MM)$ where the assorted group of best feasible solutions. In US, $MM$ can be expressed as Eq. 17.186 (Ardjmand and Amin-Naseri 2012):

$$MM = \left\{P_q \middle| C\left(P_q\right) < C\left(P_{q+1}\right), \quad q = 1, 2, \ldots, |MM|\right\}, \tag{17.186}$$

  where a translation function is employed in US to map the value of the objective function of any solution $P_q$ into a range of $(\alpha, 1-\alpha)$ for $\alpha$ falls within $(0, 1)$ and $P_q \in MM$. In US, the translation function $f_{t_l}$ is defined according to Eq. 17.187 (Ardjmand and Amin-Naseri 2012):

$$f_{t_l}\left(C\left(P_q\right)\right) = \frac{1}{1 + e^{a\left(C\left(P_q\right)\right)+b}}, \quad P_q \in MM, \tag{17.187}$$

  where $f_{t_l}$ is a sigmoid function, and $a$ and $b$ are variables of $f_{t_l}$ which will be updated in each iteration.
- In US, a displacement memory, $\Pi$, is employed to remember the displace pattern in the solutions which can be calculated through Eqs. 17.188 and 17.189, respectively (Ardjmand and Amin-Naseri 2012):

$$\Pi = \left\{ \left( \Pi_{jI}, \Pi_{jE} \right) \middle| j = 1, 2, \ldots, |X| \right\}, \tag{17.188}$$

$$\begin{aligned} \Pi_{jI} &= \left\{ \pi_{jIi} \middle| i = 1, 2, \ldots, n; j = 1, 2, \ldots, |X| \right\} \\ \Pi_{jE} &= \left\{ \pi_{jEi} \middle| i = 1, 2, \ldots, n; j = 1, 2, \ldots, |X| \right\} \end{aligned}, \tag{17.189}$$

where the number of decision parameters is denoted by $n$, and $\pi_{jIi}$ and $\pi_{jEi}$ are computed via Eqs. 17.190 and 17.191, respectively (Ardjmand and Amin-Naseri 2012):

$$\begin{aligned} &\pi_{jIi} = \left( \sum_{MS} f_{t_I}\left( C\left( P_q \right) \right) \right) \\ &P_q \in MM;\ P_q(i) \in X_j;\ j = 1, 2, \ldots, |X|. \\ &q = 1, 2, \ldots, |MM|;\ i = 1, 2, \ldots, n \end{aligned} \tag{17.190}$$

$$\pi_{jEi} = \left( \sum_{MS} h \right); \quad j = 1, 2, \ldots, |X|; \quad i = 1, 2, \ldots, n, \tag{17.191}$$

where $MS \in \mathbb{Z}^+ - \{0\}$ represents the memory size.

- By using the displacement memory, $\Pi$, a new solution, $S_1$, will be created. In US, the $i$th solution component $S_1(i)$ will be allocated with a possible range $X_j$ in solution space with a probability defined as Eq. 17.192 (Ardjmand and Amin-Naseri 2012):

$$\text{Prob}\left\{ S_1(i) \in X_j \right\} = \frac{\frac{\pi_{jIi}}{1+\left(\pi_{jEi}\right)^{\beta}}}{\sum_{j=1}^{|X|} \frac{\pi_{jIi}}{1+\left(\pi_{jEi}\right)^{\beta}}}, \tag{17.192}$$

where the probability function is denoted by Prob, and $\beta$ represents a predetermined constant.

- Once a displacement-free solution is reached, in order to remove the condensational resistance pattern, a condensational memory, $\Pi'$, is introduced in US and it is defined as Eq. 17.193 (Ardjmand and Amin-Naseri 2012):

$$\begin{aligned} \Pi' &= \left\{ \left( \Pi_i^+, \Pi_i^- \right) \middle| i = 1, 2, \ldots, n \right\} \\ \Pi_i^+ &= \left\{ \left( \pi_{iI}^+, \pi_{iE}^+ \right)^T \middle| i = 1, 2, \ldots, n \right\} \\ \Pi_i^- &= \left\{ \left( \pi_{iI}^-, \pi_{iE}^- \right)^T \middle| i = 1, 2, \ldots, n \right\} \end{aligned}, \tag{17.193}$$

where $\pi_{iI}^+ = \sum_{MS} f_{t_I}\left( C\left( P_q \right) \right)$, $\pi_{iE}^+ = \sum_{MS} h$, $\pi_{iI}^- = \sum_{MS} f_{t_I}\left( C\left( P_q \right) \right)$, and $\pi_{iE}^- = \sum_{MS} h$

### 17.54.2 Performance of US

Overall, US is a multi-start CI algorithm which contains three phases, namely, construction, construction review, and local search. To test the performance of US algorithm, Ardjmand and Amin-Naseri (2012) employed one bounded and six unbounded benchmark test functions and engineering design optimization problems which include such as Rosenbrock function, Goldsten and Price function, Wood function, and pressure vessel design problem. Compared with other CI algorithms, the results obtained by US is very competitive. The parameter analysis carried out in (Ardjmand and Amin-Naseri 2012) demonstrated that US is a robust and easy-to-use approach in dealing with hard optimization problems.

## 17.55 Wisdom of Artificial Crowds Algorithm

In this section, we will introduce an emerging CI algorithm that is based on some findings regarding the human collective intelligence (Irvine 2013; Chalmers 2010; Müller 2013; Gross 2014; Bhugra et al. 2013; Adair 2007; Chen and Lee 2008; Savage 2012).

### 17.55.1 Fundamentals of Wisdom of Artificial Crowds Algorithm

Wisdom of artificial crowds (WoAC) algorithm was recently proposed in Ashby and Yampolskiy (2011), Port and Yampolskiy (2012) and Yampolskiy et al. (2012). The WoAC is a post-processing algorithm in which independently decision-making artificial agents aggregate their personal solutions to reach an agreement about which answer is superior to all other solutions presented in the population (Yampolskiy et al. 2012). To implement WoAC algorithm, the following steps need to be performed (Yampolskiy et al. 2012):

- Setting up an automatic aggregation mechanism which collecting the individual solutions and producing a common solution that reflects frequent local structures of individual solutions.
- After establishing an agreement matrix, in order to transform agreements between solutions and costs, a nonlinear monotonic transformation function is applied as shown in Eq. 17.193 (Yampolskiy et al. 2012):

$$c_{ij} = 1 - I^{-1}_{a_{ij}}(b_1, b_2), \tag{17.194}$$

where $I^{-1}_{a_{ij}}(b_1, b_2)$ represents the inverse regularized beta function.

### 17.55.2 Performance of WoAC

To test the performance of WoAC algorithm, Yampolskiy et al. (2012) employed the classic travelling salesman problem as a benchmark. Compared with other CI algorithms (e.g., GA), the results obtained by WoAC is very competitive, in particular with an average of 3–10 % solutions quality improvement.

## 17.56 Wolf Colony Algorithm

As one of the largest species of the genus Canis, Northern gray wolves exhibit some of the most complex intra-specific social behaviour within the carnivores (Macdonald et al. 2004). Such behaviours include such as living in social units (i.e., packs), hunting socially, participating in group care of young offspring, and group defences of food and territory (Muro et al. 2011). According to Fuller et al. (2003) and Vucetich et al. (2004) in real world environment, living and foraging in form of packs is commonly observed when the prey base is composed of large ungulates, when the risk of losing food to scavengers is high, and when territorial defence is critical. Wolves hunt large ungulates, e.g., moose (Sand et al. 2006), in pack of two or more animals (Fuller et al. 2003) for the purpose of, e.g., reducing foraging cost (Packer and Caro 1997). In Muro et al. (2011), the authors employed two simple decentralized rules [via conventional CI approach, i.e., multi-agent system (MAS)] to regenerate the main features of the wolf pack hunting behaviour. The rules developed in their study are (1) moving towards the prey until a minimum safe distance to the prey is acquired, and (2) moving away from the other wolves (under the situation of close enough to the prey) that are adjacent to the safe distance to the prey. The detailed information regarding the wolf pack hunting behaviour is out of the scope of present book. For the rest of this section, we will introduce an emerging CI algorithm that is based on the hunting behaviours observed from a wolf colony.

### 17.56.1 Fundamentals of Wolf Colony Algorithm

Wolf colony algorithm (WCA) was originally proposed in Liu et al. (2011). Let $D$ represents the dimension of the search space, $n$ denotes the individual number, $X_i$ stands for the position of the $i$th artificial wolf, then we have Eq. 17.197 (Liu et al. 2011):

$$X_i = (X_{i1}, \ldots, X_{id}, \ldots, X_{iD}), \tag{17.195}$$

where $1 \leq i \leq n$, and $1 \leq d \leq D$.

The WCA algorithm mimics several behaviours that are commonly found in a wolf colony (Liu et al. 2011).

- Searching behaviour: $q$ artificial wolves are initialized to detect the possible quarry activities for the purpose of increasing the probability of discovering the quarry. Suppose that $q$ scout wolves are the wolves that are closest to the quarry, max$dh$ denotes the maximum searching number, $XX_i$ is the location of the $i$th scout wolf (totally $h$ locations are created around the candidate wolf), and the $j$th searching position is denoted by $Y_j$, then we have Eq. 17.195 (Liu et al. 2011):

$$Y_j = XX_i + randn \cdot stepa, \tag{17.196}$$

  where a uniformly distributed random number (falling within $[-1, 1]$) is denoted by *randn*, *stepa* represents the searching step. The searching behaviour will be terminated under the following situations, i.e., the searching number is greater than max$dh$ or the current location is better than the optimal searching location.
- Besieging behaviour: Once a quarry is discovered by scout wolves, howl is normally used to notify other wolves about the position of the quarry. Let the location of a quarry in the $d$th searching space after the $k$thiteration is denoted by $G_d^k$, and $X_{id}^k$ stands for the position of the $i$th artificial wolf, then we have Eq. 17.197 (Liu et al. 2011):

$$X_{id}^{k+1} = X_{id}^k + rand \cdot stepb \cdot \left(G_d^k - X_{id}^k\right), \tag{17.197}$$

  where *rand* represents a uniformly distributed random number (falling within $[0, 1]$), *stepb* denotes the searching step, the iteration number is represented by $k$, and the range of $[XMIN_d, XMAX_d]$ is used to stand for the $d$th position.
- Food assignment behaviour: Assigning food to the strongest wolves first, and then to other wolves is often observed in a colony of wolves. Based on this observation, in WCA, the wolves (denoted by$m$) with the worst performances will be replaced by newly generated $m$ artificial wolves which are randomly distributed within the wolf colony. This mechanism can assist the WCA algorithm in avoiding the local optimum.

Taking into account the fundamental behaviours described above, the steps of implementing WCA can be summarized as follows (Liu et al. 2011):

- Step 1: Initializing the following parameters such as $n$ (the individual number), *maxk* (the maximum iteration number), $q$ (the number of the searching artificial wolf), $h$ (the searching direction), *maxdh* (the maximum searching number), *stepa* (the searching step), *stepb* (the besieging step), $m$ (the number of the worst artificial wolves), and the $i$th $(1 \le i \le n)$ artificial wolf's $(X_i)$ position.
- Step 2: Forming the group of searching wolves ($q$ optimal artificial wolves) and each member of searching wolves moves according to $Y_j = XX_i + randn \cdot stepa$.
- Step 3: Choosing the best location of the searching artificial wolves as the quarry's position. Updating each artificial wolf's position based on $X_{id}^{k+1} =$

$X_{id}^k + rand \cdot stepb \cdot \left(G_d^k - X_{id}^k\right)$. If $X_{id} < XMIN_d$, then set $X_{id} = XMIN_d$; if $X_{id} \geq XMAX_d$, set $X_{id} = XMAX_d$.

- Step 4: Updating the wolf colony following the food assignment behaviour, i.e., replacing the worst $m$ artificial wolves with $m$ newly generated artificial wolves.
- Step 5: Evaluating the stopping criteria. If the circulation steps of WCA equals the predetermined maximum iteration number, the algorithm stops and outputs the current best position of artificial wolves; otherwise, WCA continues to run (i.e., returning to Step 2).

### *17.56.2 Performance of WCA*

In order to test the performance of WCA, Liu et al. (2011) employed 5 benchmark test functions such as Sphere function, Rosenbrock function, Schwefel function, and Rastrigin function. In comparison with other CI techniques (e.g., PSO and GA), WCA showed a good convergence and a strong global searching capability.

## 17.57 Wolf Pack Search Algorithm

In this section, we will introduce another wolf (in particular, wolf pack search behaviour) inspired CI algorithm (Cordoni 2009, Heylighen 1992).

### *17.57.1 Fundamentals of Wolf Pack Search Algorithm*

Wolf pack search (WPS) algorithm was originally proposed by Yang et al. (2007). Briefly, WPS works as follows (Yang et al. 2007):

- Initializing step.
- Initializing a pack of wolves in a random manner.
- Comparing and determining the best wolf *GBest* and its fitness *GBFit*.
- Circulating and updating the Eq. 17.198 (Yang et al. 2007):

$$wolf_{new} = wolf + step \cdot (GBest - wolf)/|GBest - wolf|. \tag{17.198}$$

- If the fitness of $wolf_{new}$ is better than *GBFit*, replacing *GBest* and *GBFit* with $wolf_{new}$ and its corresponding fitness value, respectively.

### *17.57.2 Performance of WPS*

In Yang et al. (2007), the WPS algorithm was hybridized with honeybee mating optimization algorithm to form WPS-MBO. By testing it on classical travelling salesman problem and a set of benchmark functions (e.g., Rosenbrock function, Schwefel function, and generalized Rastrigin function), the WPS-MBO showed a very attractive performance.

## 17.58 Conclusions

In this chapter, 56 emerging biology-based CI methodologies are discussed. Although most of them are still in their infancy, their usefulness has been demonstrated throughout the preliminary corresponding studies. Interested readers are referred to them as a starting point for a further exploration and exploitation of these innovative CI algorithms.

## References

Abdullah, S., Turabieh, H., & Mccollum, B. (2009). A hybridization of electromagnetic-like mechanism and great deluge for examination timetabling problems. *Hybrid Metaheuristics, LNCS* (Vol. 5818, pp. 60–72). Berlin: Springer.

Abernethy, B., Kippers, V., Hanrahan, S. J., Pandy, M. G., Mcmanus, A. M., & Mackinnon, L. (2013). *Biophysical foundations of human movement,*, Champaign: Human Kinetics. ISBN 978-1-4504-3165-1.

Acebo, E. D., & Rosa, J. L. D. L. (2008, April 1–4). Introducing bar systems: A class of swarm intelligence optimization algorithms. In *AISB 2008 Symposium on Swarm Intelligence Algorithms and Applications*, University of Aberdeen (pp. 18–23). The Society for the Study of Artificial Intelligence and Simulation of Behaviour.

Acquaah, G. (2012). *Principles of plant genetics and breeding.* River Street: Wiley. ISBN 978-0-470-66476-6.

Adair, J. (2007). *Develop your leadership skills*. London: Kogan Page Limited. ISBN 0-7494-4919-5.

Adamatzky, A., & Oliveira, P. P. B. D. (2011). Brazilian highways from slime mold's point of view. *Kybernetes, 40*, 1373–1394.

Adams, S. (2004). *World War I*. London: Dorling Kindersley Limited. ISBN 1-4053-0298-4.

Ahrari, A., & Atai, A. A. (2010). Grenade explosion method: A novel tool for optimization of multimodal functions. *Applied Soft Computing, 10*, 1132–1140.

Ahrari, A., Shariat-Panahi, M., & Atai, A. A. (2009). GEM: a novel evolutionary optimization method with improved neighborhood search. *Applied Mathematics and Computation, 210*, 379–386.

Al-Milli, N. R. (2010). Hybrid genetic algorithms with great deluge for course timetabling. *International Journal of Computer Science and Network Security, 10*, 283–288.

Alatas, B. (2011). Photosynthetic algorithm approaches for bioinformatics. *Expert Systems with Applications, 38*, 10541–10546.

Aleksiev, A. S., Longdon, B., Christmas, M. J., Sendova-Franks, A. B., & Franks, N. R. (2008). Individual and collective choice: Parallel prospecting and mining in ants. *Naturwissenschaften, 95*, 301–305.

Alexiou, A., & Vlamos, P. (2012). A cultural algorithm for the representation of mitochondrial population. *Advances in Artificial Intelligence, 2012*, 1–7.

Alonso, C., Herrera, C. M., & Ashman, T.-L. (2012). A piece of the puzzle: A method for comparing pollination quality and quantity across multiple species and reproductive events. *New Phytologist, 193*, 532–542.

Aman, B. (2009). *Spatial dynamic structures and mobility in computation.* Unpublished Doctoral Thesis, Romania Academy.

Anandaraman, C. (2011). An improved sheep flock heredity algorithm for job shop scheduling and flow shop scheduling. *International Journal of Industrial Engineering Computations, 2*, 749–764.

Anandaraman, C., Sankar, A. M., & Natarajan, R. (2012). Evolutionary approaches for scheduling a flexible manufacturing system with automated guided vehicles and robots. *International Journal of Industrial Engineering Computations, 3*, 627–648.

Ardjmand, E., & Amin-naseri, M. R. (2012). Unconscious search: A new structured search algorithm for solving continuous engineering optimization problems based on the theory of psychoanalysis. In Y. Tan, Y. Shi, & Z. Ji (Eds.), *ICSI 2012, Part I, LNCS* (Vol. 7331, pp. 233–242). Berlin: Springer.

Ashby, L. H. & Yampolskiy, R. V. (2011). Genetic algorithm and wisdom of artificial crowds algorithm applied to light up. In *16th International Conference on Computer Games (GAMES 2011)*, (pp. 27–32). IEEE.

Badgerow, J. P., & Hainsworth, F. R. (1981). Energy savings through formation flight? A re-examination of the vee formation. *Journal of Theoretical Biology, 93*, 41–52.

Ballerini, M., Cabibbo, N., Candelier, R., Cavagna, A., Cisbani, E., Giardina, I., et al. (2008). Empirical investigation of starling flock: A benchmark study in collective animal behaviour. *Animal Behaviour, 76*, 201–215.

Bater, L. (2007). *Incredible insects: Answers to questions about miniature marvels*. Vero Beach: Rourke Publishing LLC. Post Office Box 3328. ISBN 978-1-60044-348-0.

Batista, L. D. S., Guimarães, F. G., & Ramírez, J. A. (2009). A distributed clonal selection algorithm for optimization in electromagnetics. *IEEE Transactions on Magnetics, 45*, 1598–1601.

Bell, W. J., Roth, L. M., & Nalepa, C. A. (2007). *Cockroaches: Ecology, behavior, and natural history.* Maryland: The Johns Hopkins University Press. ISBN 978-0-8018-8616-4.

Bellaachia, A., & Bari, A. (2012). Flock by leader: A novel machine learning biologically inspired clustering algorithm. In Y. Tan, Y. Shi, & Z. Ji (Eds.), *ICSI 2012, Part I, LNCS* (Vol. 7332, pp. 117–126). Berlin: Springer.

Bhugra, D., Ruiz, P., & Gupta, S. (2013). *Leadership in psychiatry*. Hoboken: Wiley. ISBN 978-1-119-95291-6.

Bolstad, T. M. (2012). *Brownian motion*. Department of Physics and Technology, University of Bergen.

Brierley, A. S., & Cox, M. J. (2010). Shapes of krill swarms and fish schools emerge as aggregation members avoid predators and access oxygen. *Current Biology, 20*, 1758–1762.

Brownlee, J. (2007). Clonal selection algorithms. CIS Technical Report, 070209A, 1–13.

Burke, E., Bykov, Y., Newall, J., & Petrovic, S. (2004). A time-predefined local search approach to exam timetabling problems. *IIE Transactions, 3*, 509–528.

Campelo, F., Guimarães, F. G., Igarashi, H., & Ramírez, J. A. (2005). A clonal selection algorithm for optimization in electromagnetics. *IEEE Transactions on Magnetics, 41*, 1736–1739.

Cao, C.-H., Wang, L.-M., Han, C.-Y., Zhao, D.-Z., & Zhang, B. (2012). Geese PSO optimization in geometric constraint solving. *Information Technology Journal, 11*, 504.

Carlson, N. R. (2013). *Physiology of behavior.* New Jersey: Pearson Education, Inc. ISBN 978-0-205-23948-1.

Carpentier, R. (2011). *Photosynthesis research protocols*. New York: Springer. ISBN 978-1-60671-924-6.

Castro, L. N. D., & Zuben, F. J. V. (2000, July). The clonal selecton algorithm with engineering applications. In *Workshop on Artificial Immune Systems and Their Applications*, Las Vegas, USA, pp. 1–7.

Castro, L. N. D., & Zuben, F. J. V. (2002). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation, 6*, 239–251.

Chalmers, D. J. (2010). *The character of consciousness*. USA: Oxford University Press. ISBN 978-0-195-31111-2.

Chandrasekaran, M., Asokan, P., Kumanan, S., & Balamurugan, T. (2006). Sheep flocks heredity model algorithm for solving job shop scheduling problems. *International Journal of Applied Management and Technology, 4*, 79–100.

Chapman, R. F. (2013). *The insects: structure and function*. In S. J. Simpson & A. E. Douglas (Eds.). New York: Cambridge University Press. ISBN 978-0-521-11389-2.

Chen, C.-C., & Lee, Y.-T. (Eds.). (2008). *Leadership and management in China: Philosophies, theories, and practices*. Cambridge: Cambridge University Press. ISBN 978-0-511-40909-7.

Chen, K., Li, T., & Cao, T. (2006). Tribe-PSO: A novel global optimization algorithm and its application in molecular docking. *Chemometrics and Intelligent Laboratory Systems, 82*, 248–259.

Chen, T. (2009). A simulative bionic intelligent optimization algorithm: Artificial searching swarm algorithm and its performance analysis. In *IEEE International Joint Conference on Computational Sciences and Optimization* (*CSO*) (pp. 864–866).

Chen, T., Liu, Z., Shu, Q., & Zhang, L. (2009a). On the analysis of performance of the improved artificial searching swarm algorithm. In *IEEE 2nd International Conference on Intelligent Networks and Intelligent Systems* (*ICINIS*) (pp. 502–506).

Chen, T., Pang, L., Du, J., Liu, Z., & Zhang, L. (2009b). Artificial searching swarm algorithm for solving constrained optimization problems. In *IEEE International Conference on Intelligent Computing and Intelligent Systems* (*ICIS*) (pp. 562–565) .

Chen, T., Wang, Y., & Li, J. (2012). Artificial tribe algorithm and its performance analysis. *Journal of Software, 7*, 651–656.

Chen, T., Wang, Y., Pang, L., Liu, Z., & Zhang, L. (2010a). An improved artificial searching swarm algortihm and its performance analysis. In *IEEE 2nd International Conference on Computer Modeling and Simulation* (*ICCMS*) (pp. 260–263).

Chen, T., Zhang, L., Liu, Z., Pang, L., & Shu, Q. (2009c). On the analysis of performance of the artificial searching swarm algorithm. In *IEEE 5h International Conference on Natural Computation* (*ICNC*) (pp. 365–368).

Chen, Y., Cui, Z., & Zeng, J. (2010b, July 7–9). Structural optimization of lennard-jones clusters by hybrid social cognitive optimization algorithm. In F. Sun, Y. Wang, J. Lu, B. Zhang, W. Kinsner, & L. A. Zadeh (Eds.), In *9th International Conference on Cognitive Informatics (ICCI)* (pp. 204–208). IEEE. Beijing, China.

Chen, Z., & Tang, H. (2010). Cockroach swarm optimization. In *IEEE 2nd International Conference on Computer Engineering and Technology* (*ICCET*) (pp. 652–655).

Chen, Z., & Tang, H. (2011). Cockroach swarm optimization for vehicle routing problems. *Energy Procedia, 13*, 30–35.

Cheng, L., Xu, Y.-H., Zhang, H.-B., Qian, Z.-L., & Feng, G. (2010). New bionics optimization algorithm: food truck-cockroach swarm optimization algorithm (in Chinese). *Computer Engineering, 36*, 208–209.

Ciobanu, G., Desai, R., & Kumar, A. (2003). Membrane systems and distributed computing. In G. Păun (Ed.), *WMC-CdeA 2002, LNCS* (Vol. 2597, pp. 187–202). Berlin: Springer.

Civicioglu, P. (2012). Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm. *Computers and Geosciences, 46*, 229–247.

Civicioglu, P. (2013). Backtracking search optimization algorithm for numerical optimization problems. *Applied Mathematics and Computation, 219*, 8121–8144.

Coelho, L. D. S., & Bernert, D. L. D. A. (2009). PID control design for chaotic synchronization using a tribes optimization approach. *Chaos, Solitons and Fractals, 42*, 634–640.

Conradt, L., & List, C. (2009). Group decisions in humans and animals: A survey. *Philosophical Transaction of the Royal Society B, 364*, 719–742.

Cordoni, G. (2009). Social play in captive wolves (*Canis lupus*): Not only an immature affair. *Behaviour, 146*, 1363–1385.

Couzin, I. D. (2009). Collective cognition in animal groups. *Trends in Cognitive Sciences, 13*, 36–43.

Couzin, I. D., Krause, J., Franks, N. R., & Levin, S. A. (2005). Effective leadership and decision-making making in animal groups on the move. *Nature, 434*, 513–516.

Creel, S. (1997). Cooperative hunting and group size: Assumptions and currencies. *Animal Behaviour, 54*, 1319–1324.

Cuevas, E., Cienfuegos, M., Zaldívar, D., & Pérez-Cisneros, M. (2013). A swarm optimization algorithm inspired in the behavior of the social-spider. *Expert Systems with Applications*. doi: http://dx.doi.org/10.1016/j.eswa.2013.05.041.

Cuevas, E., Zaldívar, D., & Pérez-Cisneros, M. (2013b). A swarm optimization algorithm for multimodal functions and its application in multicircle detection. *Mathematical Problems in Engineering, 2013*, 1–22.

Cui, X., Gao, J., & Potok, T. E. (2006). A flocking based algorithm for document clustering analysis. *Journal of Systems Architecture, 52*, 505–515.

Cui, Y., Guo, R., & Guo, D. (2009). A naïve five-element string algorithm. *Journal of Software, 4*, 925–934.

Cui, Y. H., Guo, R., Rao, R. V., & Savsani, V. J. (2008, December 15–17) Harmony element algorithm: A naive initial searching range. In *International Conference on Advances in Mechanical Engineering*, S.V. (pp. 1–6). National Institute of Technology, Gujarat, India.

Cui, Z., & Cai, X. (2010, July 7–9). Using social cognitive optimization algorithm to solve nonlinear equations. In F. Sun, Y. Wang, J. Lu, B. Zhang, W. Kinsner & L. A. Zadeh (Eds.), In *9th International Conference on Cognitive Informatics* (*ICCI*) (pp. 199–203). Beijing, China. IEEE.

Cui, Z., Cai, X., & Shi, Z. (2011). Social emotional optimization algorithm with group decision. *Scientific Research and Essays, 6*, 4848–4855.

Cui, Z., Shi, Z., & Zeng, J. (2010). Using social emotional optimization algorithm to direct orbits of chaotic systems. In B. K. Panigrahi, S. Das, P. N. Suganthan & S. S. Dash (Eds.), *Swarm, Evolutionary, and Memetic Computing, LNCS* (Vol. 6466, pp. 389–395). Berlin: Springer.

Cui, Z., Xu, Y., & Zeng, J. (2012). Social emotional optimization algorithm with random emotional selection strategy. In R. Parpinelli (Ed.), *Theory and New Applications of Swarm Intelligence, Chap. 3* (pp. 33–50). Croatia: InTech. ISBN 978-953-51-0364-6.

Cummins, B., Cortez, R., Foppa, I. M., Walbeck, J., & Hyman, J. M. (2012). A spatial model of mosquito host-seeking behavior. *PLoS Computational Biology, 8*, 1–13.

Cutts, C. J., & Speakman, J. R. (1994). Energy savings in formation flight of pink-footed geese. *Journal of Experimental Biology, 189*, 251–261.

Dai, C., Chen, W., & Zhu, Y. (2006, November). Seeker optimization algorithm. In *IEEE International Conference on Computational Intelligence and Security* (pp. 225–229). Guangzhou, China.

Dai, C., Chen, W., Song, Y., & Zhu, Y. (2010a). Seeker optimization algorithm: A novel stochastic search algorithm for global numerical optimization. *Journal of Systems Engineering and Electronics, 21*, 300–311.

Dai, C., Chen, W., & Zhu, Y. (2010b). Seeker optimization algorithm for digital IIR filter design. *IEEE Transactions on Industrial Electronics, 57*, 1710–1718.

Dai, C., Chen, W., Zhu, Y., & Zhang, X. (2009a). Reactive power dispatch considering voltage stability with seeker optimization algorithm. *Electric Power Systems Research, 79*, 1462–1471.

Dai, C., Chen, W., Zhu, Y., & Zhang, X. (2009b). Seeker optimization algorithm for optimal reactive power dispatch. *IEEE Transactions on Power Systems, 24*, 1218–1231.

Dai, C., Zhu, Y., & Chen, W. (2007). Seeker optimization algorithm. In Y. Wang, Y. Cheung & H. Liu (Eds.), *CIS 2006, LNAI* (Vol. 4456. pp. 167–176). Berlin: Springer.

Dai, S., Zhuang, P., & Xiang, W. (2013). GSO: An improved PSO based on geese flight theory. In Y. Tan, Y. Shi, & H. Mo (Eds.), *Advances in Swarm Intelligence, ICSI 2013, Part I, LNCS* (Vol. 7928, pp. 87–95). Berlin: Springer.

Daskin, A., & Kais, S. (2011). Group leaders optimization algorithm. *Molecular Physics, 109*, 761–772.

Davendra, D., & Zelinka, I. (2009). Optimization of quadratic assignment problem using self-organinsing migrating algorithm. *Computing and Informatics, 28*, 169–180.

Davendra, D., Zelinka, I., Bialic-Davendra, M., Senkerik, R., & Jasek, R. (2013). Discrete self-organising migrating algorithm for flow-shop scheduling with no-wait makespan. *Mathematical and Computer Modelling, 57*, 100–110.

Digalakis, J. G., & Margaritis, K. G. (2002). A multipopulation cultural algorithm for the electrical generator scheduling problem. *Mathematics and Computers in Simulation, 60*, 293–301.

Ding, S., & Li, S. (2009). Clonal selection algorithm for feature selection and parameters optimization for support vector machines. In *IEEE 2nd International Symposium on Knowledge Acquisition and Modeling* (pp. 17–20).

Du, Y., Li, H., Pei, Z., & Peng, H. (2005). Intelligent spider's algorithm of search engine based on keyword. *ECTI Transactions on Computer and Information Theory, 1*, 40–49.

Dubinsky, Z. (Ed.). (2013). *Photosynthesis*. InTech: Croatia. ISBN 978-953-51-1161-0.

Dueck, G. (1993). New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics, 104*, 86–92.

Duman, E., Uysal, M., & Alkaya, A. F. (2012). Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem. *Information Sciences, 217*, 65–77.

Durrett, R. (1984). *Brownian motion and martingales in analysis.* Belmont: Wadsworth Advanced Books and Software, A Division of Wadsworth, Inc. ISBN 0-534-03065-3.

Ebensperger, L. A. (2001). A review of the evolutionary causes of rodent group-living. *Acta Theriologica, 46*, 115–144.

Eckstein, M. P., Das, K., Pham, B. T., Peterson, M. F., Abbey, C. K., Sy, J. L., et al. (2012). Neural decoding of collective wisdom with multi-brain computing. *NeuroImage, 59*, 94–108.

Feng, X., Lau, F. C. M., & Gao, D. (2009). A new bio-inspired approach to the traveling salesman problem. In J. Zhou (Ed.), *Complex 2009, Part II, LNICST*, (Vol. 5, pp. 1310–1321). Institute for Computer Sciences, Social Informatics and Telecommunications Engineering.

Frank, S. A. (1998). *Foundations of social evolution*. New Jersey: Princeton University Press. ISBN 0-691-05933-0.

Fuller, T. K., Mech, L. D., & Cockrane, J. F. (2003). Wolf population dynamics. In L. D. Mech & L. Boitani (Eds.), *Wolves: Behavior, Ecology and Conservation* (pp. 161–191). Chicago: University of Chicago Press.

Gamlin, L. (2009). *Evolution*. New York: Dorling Kindersley Limited. ISBN 978-0-7566-5028-5.

Gandomi, A. H., & Alavi, A. H. (2012). Krill herd: A new bio-inspired optimization algorithm. *Communications in Nonlinear Science and Numerical Simulation, 17*, 4831–1845.

Gandomi, A. H., Yang, X.-S., Talatahari, S., & Deb, S. (2012). Coupled eagle strategy and differential evolution for unconstrained and constrained global optimization. *Computers and Mathematics with Applications, 63*, 191–200.

Gao, S., Chai, H., Chen, B., & Yang, G. (2013). Hybrid gravitational search and clonal selection algorithm for global optimization. In Y. Tan, Y. Shi & H. Mo (Eds.), *Advances in Swarm Intelligence, LNCS* (Vol. 7929, pp. 1–10). Hybrid gravitational search and clonal selection algorithm for global optimization. Berlin: Springer.

Ghatei, S., Khajei, R. P., Maman, M. S., & Meybodi, M. R. (2012). A modified PSO using great deluge algorithm for optimization. *Journal of Basic and Applied Scientific Research, 2*, 1362–1367.

Gheorghe, M., Păun, G., Rozenberg, G., Salomaa, A., & Verlan, S. (Eds.). (2012). *Membrane computing*. Berlin: Springer. ISBN 978-3-642-28023-8.

Giraldeau, L.-A., Soos, C., & Beauchamp, G. (1994). A test of the producer-scrounger foraging game in captive flocks of spice finches, *Lonchura punctulata*. *Behavioral Ecology and Sociobiology, 34*, 251–256.

Goffredo, S., & Dubinsky, Z. (2014). *The Mediterranean Sea: Its history and present challenges*. New York: Springer. ISBN 978-94-007-6703-4.

Gofman, Y. (2012). *Computational studies of the interactions of biologically active peptides with membrane.* Unpublished Doctoral Thesis, Universität Hamburg.

Gross, R. (2014). *Psychology: The science of mind and behaviour*. London: Hodder Education. ISBN 978-1-4441-0831-6.

Gusset, M., & Macdonald, D. W. (2010). Group size effects in cooperatively breeding African wild dogs. *Animal Behaviour, 79*, 425–428.

Hagler, G. (2013). *Modelinig ships and space craft*. Berlin: Springer. ISBN 978-1-4614-4595-1.

Hasançebi, O., & Azad, S. K. (2012). An efficient metaheuristic algorithm for engineering optimization: SPOT. *International Journal of Optimization in Civil Engineering, 2*, 479–487.

Havens, T. C., Spain, C. J., Salmon, N. G., & Keller, J. M. (2008, September 21–23). Roach infestation optimization. In *IEEE Swarm Intelligence Symposium* (pp. 1–7). St. Louis MO, USA.

Heylighen, F. (1992). Evolution, selfishness and cooperation. *Journal of Ideas, 2*, 70–76.

Hobbs, R. J., Higgs, E. S., & Hall, C. M. (2013). *Novel ecosystems: Intervening in the new ecological world order*. Hoboken: Wiley. ISBN 978-1-118-35422-3.

Howell, D. C. (2014). *Fundamental statistics for the behavioral sciences*. Belmont: Cengage Learning. ISBN 978-1-285-07691-1.

Irvine, E. (2013). *Consciousness as a scientific concept: A philosophy of science perspective*. Dordrecht: Springer. ISBN 978-94-007-5172-9.

Ishdorj, T.-O. (2006). *Membrane computing, neural inspirations, gene assembly in Ciliates.* Unpublished Doctoral Thesis, University of Seville.

Janecek, A., & Tan, Y. (2011). Swarm intelligence for non-negative matrix factorization. *International Journal of Swarm Intelligence Research, 2*, 12–34.

Jelinek, R. (2013). *Biomimetics: A molecular perspective*. Berlin/Boston: Walter de Gruyter. ISBN 978-3-11-028117-0.

Jonassen, T. M. (2006, June). Symbolic dynamics, the spider algorithm and finding certain real zeros of polynomials of high degree. In *8th International Mathematica Symposium* (pp. 1–16). Avignon.

Karci, A. (2007a). Human being properties of saplings growing up algorithm. In *International Conference on Computational Cybernetics* (*ICCC*) (pp. 227–232). IEEE.

Karci, A. (2007b). Natural inspired computational intelligence method: saplings growing up algorithm. In *International Conference on Computational Cybernetics (ICCC)*, pp. 221–226. IEEE.

Karci, A. (2007c). Theory of saplings growing up algorithm. In *Adaptive and Natural Computing Algorithms, LNCS* (Vol. 4431, pp. 450–460). Berlin: Springer.

Karci, A., & Alatas, B. (2006). Thinking capability of saplings growing up algorithm. In *Intelligent Data Engineering and Automated Learning* (*IDEAL 2006*)*, LNCS* (Vol. 4224, pp. 386–393). Berlin: Springer.

Kashan, A. H. (2009). League championship algorithm: a new algorithm for numerical function optimization. In *IEEE International Conference of Soft Computing and Pattern Recognition* (*SoCPAR*) (pp. 43–48).

Kashan, A. H. (2011). An efficient algorithm for constrained global optimization and application to mechanical engineering design: League championship algorithm (LCA). *Computer-Aided Design, 43*, 1769–1792.

Kashan, A. H., & Karimi, B. (2010, 18–23 July). A new algorithm for constrained optimization inspired by the sport league championships. In *World Congress on Computational Intelligence* (*WCCI*) (pp. 487–494). CCIB, Barcelona, Spain.

Keller, L., & Gordon, É. (2009). *The lives of ants (translated by James Grieve).* Oxford: Oxford University Press Inc. ISBN 978–0–19–954186–7.

Kim, H., & Ahn, B. (2001). A new evolutionary algorithm based on sheep flocks heredity model. In *Conference on Communications, Computers and Signal Processing* (pp. 514–517). IEEE.

Kim, Y.-B. (2012). *Distributed algorithms in membrane systems.* Unpublished Doctoral Thesis, University of Auckland.

King, A. J., Sueur, C., Huchard, E., & Cowlishaw, G. (2011). A rule-of-thumb based on social affiliation explains collective movements in desert baboons. *Animal Behaviour, 82*, 1337–1345.

Krishnanand, K. R., Hasani, S. M. F., & Panigrahi, B. K. (2013). Optimal power flow solution using self-evolving brain–storming inclusive teaching-learning-based algorithm. In Y. Tan, Y. Shi, & H. Mo (Eds.), *ICSI 2013, Part I, LNCS* (Vol. 7928, pp. 338–345). Berlin: Springer.

Kwasnicka, H., Markowska-Kaczmar, U., & Mikosik, M. (2011). Flocking behaviour in simple ecosystems as a result of artificial evolution. *Applied Soft Computing, 11*, 982–990.

Lancaster, R., Butler, R. E. A., Lancaster, J. M., & Shimizu, T. (1998). *Fireworks: Principles and practice.* New York: Chemical Publishing Co., Inc. ISBN 0-8206-0354-6.

Lee, D., & Quessy, S. (2003). Visual search is facilitated by scene and sequence familiarity in rhesus monkeys. *Vision Research, 43*, 1455–1463.

Lemasson, B. H., Anderson, J. J., & Goodwin, R. A. (2009). Collective motion in animal groups from a neurobiological perspective: The adaptive benefits of dynamic sensory loads and selective attention. *Journal of Theoretical Biology, 261*, 501–510.

Levin, S. A. (2013a). *Encyclopedia of biodiversity*. Oxford: Academic Press, Elsevier Inc. ISBN 978-0-12-384719-5.

Levin, S. A. (2013b). *Encyclopedia of biodiversity*. Oxford: Academic Press, Elsevier Inc. ISBN 978-0-12-384719-5.

Levin, S. A. (2013c). *Encyclopedia of biodiversity*. Oxford: Academic Press, Elsevier Inc. ISBN 978-0-12-384719-5.

Levin, S. A. (2013d). *Encyclopedia of biodiversity*. Oxford: Academic Press, Elsevier Inc. ISBN 978-0-12-384719-5.

Levin, S. A. (2013e). *Encyclopedia of biodiversity*. Oxford: Academic Press, Elsevier Inc. ISBN 978-0-12-384719-5.

Levin, S. A. (2013f). *Encyclopedia of biodiversity,*. London: Academic Press, Elsevier Inc. ISBN 978-0-12-384719-5.

Li, K., Torres, C. E., Thomas, K., Rossi, L. F., & Shen, C.-C. (2011). Slime mold inspired routing protocols for wireless sensor networks. *Swarm Intelligence, 5*, 183–223.

Li, Y. (2010, October 22–24). Solving TSP by an ACO-and-BOA-based hybrid algorithm. In *IEEE International Conference on Computer Application and System Modeling* (*ICCASM*), (Vol. 12, pp. 189–192).

Lihoreau, M., Costa, J. T., & Rivault, C. (2012). The social biology of domiciliary cockroaches: colony structure, kin recognition and collective decisions. *Insectes Sociaux.* doi: 10.1007/s00040-012-0234-x.

Lihoreau, M., Deneubourg, J.-L., & Rivault, C. (2010). Collective foraging decision in a gregarious insect. *Behavioral Ecology and Sociobiology, 64*, 1577–1587.

Lissaman, P. B. S., & Shollenberger, C. A. (1970). Formation flight of birds. *Science, 168*, 1003–1005.

Liu, C., Yan, X., Liu, C., & Wu, H. (2011). The wolf colony algorithm and its application. *Chinese Journal of Electronics, 20*, 212–216.

Liu, J. Y., Guo, M. Z., & Deng, C. (2006). Geese PSO: An efficient improvement to particle swarm optimization. *Computer Science, 33*, 166–168.

Lu, Y., & Liu, X. (2011). A new population migration algorithm based on the chaos theory. In *IEEE 2nd International Symposium on Intelligence Information Processing and Trusted Computing* (*IPTC*) (pp. 147–10).

Luo, X., Li, S., & Guan, X. (2010). Flocking algorithm with multi-target tracking for multi-agent systems. *Pattern Recognition Letters, 31*, 800–805.

Maathuis, F. J. M. (2013). *Plant mineral nutrients: Methods and protocols*. New York: Springer. ISBN 978-1-62703-151-6.
Macdonald, D. W., Creel, S., & Mills, M. G. L. (2004). Society: Canid society. In D. W. Macdonald & C. Sillero-Zubiri (Eds.), *Biology and Conservation of Wild Carnivores* (pp. 85–106). Oxford: Oxford University Press.
Magstadt, T. M. (2013). *Understanding politics: ideas, institutions, and issues*. Cengage Learning: Belmont. ISBN 978-1-111-83256-8.
Marcus, J. B. (2013). *Culinary nutrition: The science and practice of healthy cooking*. Waltham: Elsevier. ISBN 978-0-12-391882-6.
Maroosi, A., & Muniyandi, R. C. (2013). Membrane computing inspired genetic algorithm on multi-core processors. *Journal of Computer Science, 9*, 264–270.
Mayfield, J. E. (2013). *The engine of complexity: Evolution as computation*. New York: Columbia University Press. ISBN 978-0-231-16304-0.
Mills, D. S., Marchant-Forde, J. N., McGreevy, P. D., Morton, D. B., Nicol, C. J., Phillips, C. J. C., et al. (Eds.). (2010). *The encyclopedia of applied animal behaviour and welfare*. Wallingford: CAB International. ISBN 978-0-85199-724-7.
Mosser, A., & Packer, C. (2009). Group territoriality and the benefits of sociality in the African lion, *Panthera leo*. *Animal Behaviour, 78*, 359–370.
Mucherino, A., & Seref, O. (2007). Monkey search: A novel metaheuristic search for global optimization. In *AIP Conference Proceedings* (Vol. 953, pp. 162–173).
Mukherjee, R., Chakraborty, S., & Samanta, S. (2012). Selection of wire electrical discharge machining process parameters using non-traditional optimization algorithms. *Applied Soft Computing, 12*, 2506–2516.
Müller, V. C. (Ed.). (2013). *Philosophy and theory of artificial intelligence*. Berlin: Springer. ISBN 978-3-642-31673-9.
Muniyandi, R. C., & Zin, A. M. (2013). Membrane computing as the paradigm for modeling system biology. *Journal of Computer Science, 9*, 122–127.
Murase, H. (2000). Finite element inverse analysis using a photosynthetic algorithm. *Computers and Electronics in Agriculture, 29*, 115–123.
Muro, C., Escobedo, R., Spector, L., & Coppinger, R. P. (2011). Wolf-pack (Canis lupus) hunting strategies emerge from simple rules in computational simulations. *Behavioural Processes, 88*, 192–197.
Murphy, N. (2010). *Uniformity conditions for membrane system uncovering complexity below P*. Unpublished Doctoral Thesis, National University of Ireland Maynooth.
Nabil, E., Badr, A., & Farag, I. (2012). *A membrane-immune algorithm for solving the multiple 0-1 knapsack problem* (pp. 3–15). LVII: Informatica.
Nahas, N., Nourelfath, M., & Aït-Kadi, D. (2010). Iterated great deluge for the dynamic facility layout problem. Canada: Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, Report No.: CIRRELT-2010-20.
Nakagaki, T., Yamada, H., & Tóth, Á. (2000). Maze-solving by an amoeboid organism. *Nature, 407*, 470.
Nara, K., Takeyama, T., & Kim, H. (1999). A new evolutionary algorithm based on sheep flocks heredity model and its application to scheduling problem. In *IEEE International Conference on Systems, Man, and Cybernetics* (*SMC*) (pp. VI-503–VI-508).
Neshat, M., Sepidnam, G., & Sargolzaei, M. (2013). Swallow swarm optimization algorithm: A new method to optimization. *Neural Computing & Application*, *23*, 429–454. doi: 10.1007/s00521-012-0939-9.
Newell, P. C. (1978). Genetics of the cellular slime molds. *Annual Review of Genetics, 12*, 69–93.
Nguyen, V., Kearney, D., & Gioiosa, G. (2008). An implementation of membrane computing using reconfigurable hardware. *Computing and Informatics, 27*, 551–569.
Nicolis, S. C., Detrain, C., Demolin, D., & Deneubourg, J. L. (2003). Optimality of collective choices: a stochastic approach. *Bulletin of Mathematical Biology, 65*, 795–808.
Niizato, T., & Gunji, Y.-P. (2011). Metric–topological interaction model of collective behavior. *Ecological Modelling, 222*, 3041–3049.

Nishida, T. Y. (2005, July 18–21). Membrane algorithm: An approximate algorithm for NP-complete optimization problems exploiting P-systems. In R. Freund, G. Lojka, M. Oswald, & G. Păun (Eds.), In *6th International workshop on membrane computing* (*WMC*) (pp. 26–43). Vienna, Austria. Institute of Computer Languages, Faculty of Informatics, Vienna University of Technology.

Nolle, L., Zelinka, I., Hopgood, A. A., & Goodyear, A. (2005). Comparison of an self-organizing migration algorithm with simulated annealing and differential evolution for automated waveform tuning. *Advanced Engineering Software, 36*, 645–653.

Oca, M. A. M. D., Ferrante, E., Scheidler, A., Pinciroli, C., Birattari, M., & Dorigo, M. (2011). Majority-rule opinion dynamics with differential latency: A mechanism for self-organized collective decision-making. *Swarm Intelligence, 5*, 305–327.

Ochoa-Zezzatti, A., Bustillos, S., Jaramillo, R., & Ruíz, V. (2012). Improving practice sports in a largest city using a cultural algorithm. *International Journal of Combinatorial Optimization Problems and Informatics, 3*, 14–20.

Oftadeh, R., Mahjoob, M. J., & Shariatpanahi, M. (2010). A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search. *Computers and Mathematics with Applications, 60*, 2087–2098.

Onwubolu, G. C. (2006). Performance-based optimization of multi-pass face milling operations using Tribes. *International Journal of Machine Tools and Manufacture, 46*, 717–727.

Packer, C., & Caro, T. M. (1997). Foraging costs in social carnivores. *Animal Behaviour, 54*, 1317–1318.

Păun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences, 61*, 108–143.

Păun, G. (2002). A guide to membrane computing. *Theoretical Computer Science, 287*, 73–100.

Păun, G. (2007). Tracing some open problems in membrane computing. *Romanian Journal of Information Science and Technology, 10*, 303–314.

Pei, Y., Zheng, S., Tan, Y., & Takagi, H. (2012, October 14–17). An empirical study on influence of approximation approaches on enhancing fireworks algorithm. In *IEEE International Conference on Systems, Man, and Cybernetics* (*IEEE SMC 2012*) (pp. 1322–1327). Seoul, Korea.

Petit, O., & Bon, R. (2010). Decision-making processes: The case of collective movements. *Behavioural Processes, 84*, 635–647.

Picarougne, F., Azzag, H., Venturini, G., & Guinot, C. (2007). A new approach of data clustering using a flock of agents. *Evolutionary Computation, 15*, 345–367.

Port, A. C., & Yampolskiy, R. V. (2012). Using a GA and wisdom of artificial crowds to solve solitaire battleship puzzles. In *IEEE 17th International Conference on Computer Games* (*CGAMES 2012*) (pp. 25–29).

Premaratne, U., Samarabandu, J., & Sidhu, T. (2009, December 28–31). A new biologically inspired optimization algorithm. In *IEEE 4th International Conference on Industrial and Information Systems* (*ICIIS*) (pp. 279–284). Sri Lanka.

Ramachandran, V. S. (2012a). *Encyclopedia of human behavior*. London: Elsevier. ISBN 978-0-12-375000-6.

Ramachandran, V. S. (2012b). *Encyclopedia of human behavior*. London: Elsevier. ISBN 978-0-12-375000-6.

Ramachandran, V. S. (2012c). *Encyclopedia of human behavior*. London: Elsevier. ISBN 978-0-12-375000-6.

Rao, R. V., Vakharia, D. P., & Savsani, V. J. (2009). Mechanical engineering design optimisation using modified harmony elements algorithm. *International Journal of Design Engineering, 2*, 116–135.

Ravi, V. (2004). Optimization of complex system reliability by a modified great deluge algorithm. *Asia-Pacific Journal of Operational Research, 21*, 487–497.

Ray, T., & Liew, K. M. (2003). Society and civilization: an optimization algorithm based on the simulation of social behavior. *IEEE Transactions on Evolutionary Computation, 7*, 386–396.

Reece, J. B., Urry, L. A., Cain, M. L., Wasserman, S. A., Minorsky, P. V., & Jackson, R. B. (2011). *Campbell biology.* San Francisco: Pearson Education, Inc. ISBN 978-0-321-55823-7.

Resende, R. R., & Ulrich, H. (2013). *Trends in stem cell proliferation and cancer research*. Dordrecht: Springer. ISBN 978-94-007-6210-7.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics, 21*, 25–34.

Reynolds, R. G. (1994). An introduction to cultural algorithms. In A. V. Sebald & L. J. Fogel (Eds.) *The 3rd Annual Conference on Evolutionary Programming* (pp. 131–139). World Scientific Publishing.

Reynolds, R. G. (1999). Cultural algorithms: theory and application In D. Corne, M. Dorigo & Glover, F. (Eds.), *New Ideas in Optimization.* NY: McGraw-Hill.

Riff, M. C., Montero, E., & Neveu, B. (2013). Reducing calibration effort for clonal selection based algorithms. *Knowledge-Based Systems, 41*, 54–67.

Rose, S. V. (2008). *Volcano and earthquake*. New York: Dorling Kindersley Limited. ISBN 978-0-7566-3780-4.

Sacco, W. F., Oliveira, C. R. E. D., & Pereira, C. M. N. A. (2006). Two stochastic optimization algorithms applied to nuclear reactor core design. *Progress in Nuclear Energy, 48*, 525–539.

Sadollah, A., Bahreininejad, A., Eskandar, H., & Hamdi, M. (2012). Mine blast algorithm for optimization of truss structures with discrete variables. *Computers and Structures, 102–103*, 49–63.

Sadollah, A., Bahreininejad, A., Eskandar, H., & Hamdi, M. (2013). Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems. *Applied Soft Computing, 13*, 2592–2612.

Samuels, P., Huntington, S., Allsop, W., & Harrop, J. (2009). *Flood risk management: Research and practice*. London: Taylor & Francis Group. ISBN 978-0-415-48507-4.

Sand, H., Wikenros, C., Wabakken, P., & Liberg, O. (2006). Effects of hunting group size, snow depth and age on the success of wolves hunting moose. *Animal Behaviour, 72*, 781–789.

Savage, N. (2012). Gaining wisdom from crowds. *Communications of the ACM, 55*, 13–15.

Schnell, R. J., & Priyadarshan, P. M. (2012). *Genomics of tree crops*. New York: Springer. ISBN 978-1-4614-0919-9.

Schutter, G. D., Theraulaz, G., & Deneubourg, J.-L. (2001). Animal–robots collective intelligence. *Annals of Mathematics and Artificial Intelligence, 31*, 223–238.

Sedwards, S. (2009). *A natural computation approach to biology: Modelling cellular processes and populations of cells with stochastic models of P systems.* Unpublished Doctoral Thesis, University of Trento.

Sell, S. (2013). *Stem cells handbook*. New York: Springer. ISBN 978-1-4614-7695-5.

Şen, Z. (2014). *Philosophical, logical and scientific perspectives in engineering*. Heidelberg: Springer. ISBN 978-3-319-01741-9.

Senkerik, R., Zelinka, I., Davendra, D., & Oplatkova, Z. (2010). Utilization of soma and differential evolution for robust stabilization of chaotic logistic equation. *Computers and Mathematics with Applications, 60*, 1026–1037.

Shann, M. (2008). *Emergent behavior in a simulated robot inspired by the slime mold.* Unpublished Bachelor Thesis, University of Zurich.

Shaw, B., Banerjee, A., Ghoshal, S. P., & Mukherjee, V. (2011). Comparative seeker and bio-inspired fuzzy logic controllers for power system stabilizers. *Electrical Power and Energy Systems, 33*, 1728–1738.

Shettleworth, S. J. (2010). *Cognition, evolution, and behavior*. New York: Oxford University Press. ISBN 978-0-19-531984-2.

Shi, Y. (2011a). Brain storm optimization algorithm. In Y. Tan, Y. Shi & G. Wang (Eds.), *ICSI 2011, Pat I, LNCS* (Vol. 6728, pp. 303–309). Berlin: Springer.

Shi, Y. (2011b). An optimization algorithm based on brainstorming process. *International Journal of Swarm Intelligence Research, 2*, 35–62.

Shlesinger, M. F., Klafter, J., & Zumofen, G. (1999). Above, below and beyond Brownian motion. *American Journal of Physics, 67*, 1253–1259.

Silva, D. J. A. D., Teixeira, O. N., & Oliveira, R. C. L. D. (2012). Performance study of cultural algorithm based on genetic algorithm with single and multi population for the MKP. In S. Gao (Ed.), *Bio-inspired computational algorithms and their applications.* Rijeka: InTech.

Sizer, F. S., & Whitney, E. (2014). *Nutrition: Concepts and controversies*. Belmont: Cengage Learning. ISBN 978-1-133-60318-4.

Smolin, L. A., & Grosvenor, M. B. (2010). *Healthy eathing_a guide to nutrition: Nutrition for sports and exercise*. New York: Infobase Publishing. ISBN 978-1-60413-804-7.

Song, M. X., Chen, K., He, Z. Y., & Zhang, X. (2013). Bionic optimization for micro-siting of wind farm on complex terrain. *Renewable Energy, 50*, 551–557.

Srinivasan, S., & Ramakrishnan, S. (2012). Nugget discovery with a multi-objective cultural algorithm. *Computer Science and Engineering: An International Journal, 2*, 11–25.

Steinbuch, R. (2011). Bionic optimisation of the earthquake resistance of high buildings by tuned mass dampers. *Journal of Bionic Engineering, 8*, 335–344.

Steinitz, M. (2014). *Human monoclonal antibodies: Methods and protocols*. New York: Springer. ISBN 978-1-62703-585-9.

Stradner, J., Thenius, R., Zahadat, P., Hamann, H., Crailsheim, K., & Schmickl, T. (2013). *Algorithmic requirements for swarm intelligence in differently coupled collective systems*. Chaos: Solitons and Fractals. 50.

Stukas, A. A., & Clary, E. G. (2012). Altruism and helping behavior. In V. S. Ramachandran, (Ed.), *Encyclopedia of human behavior* (2nd ed.). London: Elsevier, Inc. ISBN 978-0-12-375000-6.

Su, M.-C., Su, S.-Y., & Zhao, Y.-X. (2009). A swarm-inspired projection algorithm. *Pattern Recognition, 42*, 2764–2786.

Subbaiah, K. V., Rao, M. N., & Rao, K. N. (2009). Scheduling of AGVs and machines in FMS with makespan criteria using sheep flock heredity algorithm. *International Journal of Physical Sciences, 4*, 139–148.

Sueur, C., Deneubourg, J.-L., & Petit, O. (2010). Sequence of quorums during collective decision making in macaques. *Behavioral Ecology and Sociobiology, 64*, 1875–1885.

Sulaiman, M. H. (2013, March 15–17). Differential search algorithm for economic dispatch with valve-point effects. In *2nd International Conference on Engineering and Applied Science* (*ICEAS*) (pp. 111–117). Tokyo: Toshi Center Hotel.

Sulis, W. (1997). Fundamental concepts of collective intelligence. *Nonlinear Dynamics, Psychology, and Life Sciences, 1*, 35–53.

Sun, J., & Lei, X. (2009). Geese-inspired hybrid particle swarm optimization algorithm. In *International Conference on Artificial Intelligence and Computational Intelligence* (pp. 134–138). IEEE.

Taffe, M. A., & Taffe, W. J. (2011). Rhesus monkeys employ a procedural strategy to reduce working memory load in a self-ordered spatial search task. *Brain Research, 1413*, 43–50.

Taherdangkoo, M., Shirzadi, M. H., & Bagheri, M. H. (2012a). A novel meta-heuristic algorithm for numerical function optimization_blind, naked mole-rats (BNMR) algorithm. *Scientific Research and Essays, 7*, 3566–3583.

Taherdangkoo, M., Yazdi, M., & Bagheri, M. H. (2011). Stem cells optimization algorithm. *LNBI* (Vol. 6840, pp. 394–403). Berlin: Springer.

Taherdangkoo, M., Yazdi, M., & Bagheri, M. H. (2012b). A powerful and efficient evolutionary optimization algorithm based on stem cells algorithm for data clustering. *Central European Journal of Computer Science, 2*, 1–13.

Tan, Y., & Zhu, Y. (2010). Fireworks algorithm for optimization. In Y. Tan, Y. Shi & K. C. Tan (Eds.), *ICSI 2010, Part I, LNCS* (Vol. 6145, pp. 355–364). Berlin: Springer

Taylor, K. (2012). *The brain supremacy: Notes from the frontiers of neuroscience*. Oxford: Oxford University Press. ISBN 978-0-19-960337-4.

Tero, A., Takagi, S., Saigusa, T., Ito, K., Bebber, D. P., Fricker, M. D., et al. (2010). Rules for biologically inspired adaptive network design. *Science, 237*, 439–442.

Thammano, A., & Moolwong, J. (2010). A new computational intelligence technique based on human group formation. *Expert Systems with Applications, 37*, 1628–1634.

Theiner, G., Allen, C., & Goldstone, R. L. (2010). Recognizing group cognition. *Cognitive Systems Research, 11*, 378–395.

Tidball, K. G., & Krasny, M. E. (2014). *Greening in the red zone: Disaster, resilience and community greening*. Heidelberg: Springer. ISBN 978-90-481-9946-4.

Tollefsen, D. P. (2006). From extended mind to collective mind. *Cognitive Systems Research, 7*, 140–150.

Touhara, K. (2013). *Pheromone signaling: Methods and protocols*. New York: Springer. ISBN 978-1-62703-618-4.

Ulutas, B. H., & Kulturel-Konak, S. (2011). A review of clonal selection algorithm and its applications. *Artificial Intelligence Review, 36*, 117–138.

Umedachi, T., Takeda, K., Nakagaki, T., Kobayashi, R., & Ishiguro, A. (2010). Fully decentralized control of a soft-bodied robot inspired by true slime mold. *Biological Cybernetics, 102*, 261–269.

Venkumar, P., & Sekar, K. C. (2012). Design of cellular manufacturing system using non-traditional optimization algorithms. In V. Modrák & R. S. Pandian (Eds.), *Operations management research and cellular manufacturing systems: Innovative methods and approaches, Chap. 6* (pp. 99–139). Hershey: IGI Global.

Verdy, A., & Flierl, G. (2008). Evolution and social behavior in krill. *Deep-Sea Research II, 55*, 472–484.

Vucetich, J. A., Peterson, R. O., & Waite, T. A. (2004). Raven scavenging favours group foraging in wolves. *Animal Behaviour, 67*, 1117–1126.

Wang, G., Guo, L., Gandomi, A. H., Cao, L., Alavi, A. H., Duan, H., et al. (2013). Lévy-flight krill herd algorithm. *Mathematical Problems in Engineering, 2013*, 1–14.

Wang, P., & Cheng, Y. (2010). Relief supplies scheduling based on bean optimization algorithm. *Economic Research Guide, 8*, 252–253.

Wang, S., Dai, D., Hu, H., Chen, Y.-L., & Wu, X. (2011). RBF neural network parameters optimization based on paddy field algorithm. In *International Conference on Information and Automation* (*ICIA*) (pp. 349–353). June, Shenzhen, China. IEEE.

Wang, W., Feng, Q., & Zheng, Y. (2008, November 19–21). A novel particle swarm optimization algorithm with stochastic focusing search for real-parameter optimization. In *11th Singapore International Conference on Communication Systems* (*ICCS*) (pp. 583–587). Guangzhou, China. IEEE.

Wang, X., Gao, X.-Z., & Ovaska, S. J. (2009). Fusion of clonal selection algorithm and harmony search method in optimization of fuzzy classification systems. *International Journal of Bio-Inspired Computation, 1*, 80–88.

Wang, Z.-R., Ma, F., Ju, T., & Liu, C.-M. (2010). A niche genetic algorithm with population migration strategy. In *IEEE 2nd International Conference on Information Science and Engineering* (*ICISE*) (pp. 912–915).

Wei, G. (2011). Optimization of mine ventilation system based on bionics algorithm. *Procedia Engineering, 26*, 1614–1619.

Wei, Z. H., Cui, Z. H., & Zeng, J. C. (2010, September 26–28). Social cognitive optimization algorithm with reactive power optimization of power system. In *2010 International Conference on Computational Aspects of Social Networks* (*CASoN*) (pp. 11–14). Taiyuan, China.

Weigert, G., Horn, S., & Werner, S. (2006). Optimization of manufacturing processes by distributed simulation. *International Journal of Production Research, 44*, 3677–3692.

Whitehouse, M. E. A., & Lubin, Y. (1999). Competitive foraging in the social spider *Stegodyphus dumicola*. *Animal Behaviour, 58*, 677–688.

Whitten, K. W., Davis, R. E., Peck, M. L., & Stanley, G. G. (2014). *Chemistry*. Belmont: Cengage Learning. ISBN 13: 978-1-133-61066-3.

Wilson, C. (2013). *Brainstroming and beyond: a user-centered design method.* Waltham: Morgan Kaufmann, Elsevier Inc. ISBN 978-0-12-407157-5.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*, 67–82.

Woodward, J. (2008). *Climate change*. New York: Dorling Kindersley Limited. ISBN 978-07566-3771-2.

Woodworth, S. (2007). *Computability limits in membrane computing*. Unpublished Doctoral Thesis, University of California, Santa Barbara.

Wu, J., Cui, Z., & Liu, J. (2011, August 18–20). Using hybrid social emotional optimization algorithm with metropolis rule to solve nonlinear equations. In Y. Wang, A. Celikyilmaz, W. Kinsner, W. Pedrycz, H. Leung & L. A. Zadeh (Eds.), *10th International Conference on Cognitive Informatics and Cognitive Computing* (ICCI & CC) (pp. 405–411). Banff, AB. IEEE.

Xiao, J.-H., Huang, Y.-F., & Cheng, Z. (2013). A bio-inspired algorithm based on membrane computing for engineering design problem. *International Journal of Computer Science Issues, 10*, 580–588.

Xu, Y. C., Cui, Z. H., & Zeng, J. C. (2010). Social emotional optimization algorithm for nonlinear constrained optimization problems. In *1st International Conference on Swarm, Evolutionary and Memetic Computing* (*SEMCCO*) (pp. 583–590).

Xue, J., Wu, Y., Shi, Y., & Cheng, S. (2012). Brain storm optimization algorithm for multi-objective optimization problems. In Y. Tan, Y. Shi & Z. Ji (Eds.), *ICSI 2012, Part I, LNCS* (Vol. 7331, pp. 513–519). Berlin: Springer.

Yampolskiy, R. V., Ashby, L., & Hassan, L. (2012). Wisdom of artificial crowds: A metaheuristic algorithm for optimization. *Journal of Intelligent Learning Systems and Applications, 4*, 98–107.

Yang, C., Tu, X., & Chen, J. (2007). Algorithm of marriage in honey bees optimization based on the wolf pack search. In *IEEE International Conference on Intelligent Pervasive Computing* (*IPC*) (pp. 462–467).

Yang, X.-S. (2005). Biology-derived algorithms in engineering optimization. In S. Olarius & A. Zomaya (Eds.), *Handbook of Bioinspired Algorithms and Applications, Chap. 32* (pp. 585–596). Boca Raton: CRC Press.

Yang, X.-S. (2012). Flower pollination algorithm for global optimization. *Unconventional Computation and Natural Computation, LNCS* (Vol. 7445, pp. 240–249). Berlin: Springer.

Yang, X.-S., & Deb, S. (2010). Eagle strategy using Lévy walk and firefly algorithms for stochastic optimization. In J. R. Gonzalez (Ed.), *Nature Inspired Cooperative Strategies for Optimization* (*NISCO 2010*), *SCI* (Vol. 284, pp. 101–111). Berlin: Springer.

Yang, X.-S., & Deb, S. (2012). Two-stage eagle strategy with differential evolution. *International Journal of Bio-Inspired Computation, 4*, 1–5.

Yang, X.-S., Karamanoglu, M., & He, X. (2013). Multi-objective flower algorithm for optimization. *Procedia Computer Science, 18*, 861–868.

Yeagle, P. L. (Ed.). (2005). *The structure of biological membranes*. Boca Raton: CRC Press. ISBN 0-8493-1403-8.

You, S. K., Kwon, D. H., Park, Y.-I., Kim, S. M., Chung, M.-H., & Kim, C. K. (2009). Collective behaviors of two-component swarms. *Journal of Theoretical Biology, 261*, 494–500.

Zaharie, D., & Ciobanu, G. (2006). Distributed evolutionary algorithms inspired by membranes in solving continuous optimization problems. In H. J. Hoogeboom (Ed.), *WMC 7, LNCS* (Vol. 4361, pp. 536–553). Berlin: Springer.

Zang, H., Zhang, S., & Hapeshi, K. (2010). A review of nature-inspired algorithms. *Journal of Bionic Engineering, 7*, S232–S237.

Zelinka, I., & Lampinen, J. (2000). Soma: Self-organizing migrating algorithm. In *The 6th International Conference on Soft Computing, Brno, Czech Republic*.

Zelinka, I., Senkerik, R., & Navratil, E. (2009). Investigation on evolutionary optimization of chaos control. *Chaos, Solitons and Fractals, 40*, 111–129.

Zhan, Z.-H., Zhang, J., Shi, Y.-H., & Liu, H.-L. (2012, June 10–15) A modified brain storm optimization. In *World Congress on Computational Intelligence* (*WCCI*) (pp. 1–8). Brisbane, Australia. IEEE.

Zhang, G., Cheng, J., & Gheorghe, M. (2011). A membrane-inspired approximate algorithm for traveling salesman problems. *Romanian Journal of Information Science and Technology, 14*, 3–19.

Zhang, G., Yang, H., & Liu, Z. (2007). Using watering algorithm to find the optimal paths of a maze. *Computer, 24*, 171–173.

Zhang, W., Luo, Q. & Zhou, Y. (2009). A method for training RBF neural networks based on population migration algorithm. In *International Conference on Artificial Intelligence and Computational Intelligence* (*AICI*) (pp. 165–169). IEEE.

Zhang, W., & Zhou, Y. (2009). Description population migration algorithm based on framework of swarm intelligence. In *IEEE WASE International Conference on Information Engineering* (*ICIE*) (pp. 281–284).

Zhang, X., Chen, W., & Dai, C. (2008a, April 6–9) Application of oriented search algorithm in reactive power optimization of power system. *DRPT 2008* (pp. 2856–2861). Nanjing, China. DRPT.

Zhang, X., Sun, B., Mei, T., & Wang, R. (2010, November 28–30). Post-disaster restoration based on fuzzy preference relation and bean optimization algorithm. In *Youth Conference on Information Computing and Telecommunications* (*YC-ICT*) (pp. 271–274). IEEE.

Zhang, X., Jiang, K., Wang, H., Li, W., & Sun, B. (2012a). An improved bean optimization algorithm for solving TSP. In Y. Tan, Y. Shi & Z. Ji (Eds.), *ICSI 2012, Part I, LNCS* (Vol. 7331, pp. 261–267). Berlin: Springer.

Zhang, X., Huang, S., Hu, Y., Zhang, Y., Mahadevan, S., & Deng, Y. (2013a). Solving 0-1 knapsack problems based on amoeboid organism algorithm. *Applied Mathematics and Computation, 219*, 9959–9970.

Zhang, X., Sun, B., Mei, T., & Wang, R. (2013b). A novel evolutionary algorithm inspired by beans dispersal. *International Journal of Computational Intelligence Systems, 6*, 79–86.

Zhang, X., Wang, H., Sun, B., Li, W., & Wang, R. (2013c). The Markov model of bean optimization algorithm and its convergence analysis. *International Journal of Computational Intelligence Systems, 6*, 609–615.

Zhang, X., Wang, R., & Song, L. (2008b). A novel evolutionary algorithm: Seed optimization algorithm. *Pattern Recognition and Artificial Intelligence, 21*, 677–681.

Zhang, Z.-W., Zhang, H., & Li, Y.-B. (2012b). Biologically inspired collective construction with visual landmarks. *Journal of Zhejiang University-SCIENCE C (Computers and Electronics), 13*, 315–327.

Zhao, Q., & Liu, X. (2011). An improved multi-objective population migration optimization algorithm. In *2nd International Symposium on Intelligence Information Processing and Trusted Computing* (*IPTC*) (pp. 143–146). IEEE.

Zheng, Y., Chen, W., Dai, C., & Wang, W. (2009). Stochastic focusing search: A novel optimization algorithm for real-parameter optimization. *Journal of Systems Engineering and Electronics, 20*, 869–876.

Zhou, D., Shi, Y., & Cheng, S. (2012). Brain storm optimization algorithm with modified step-size and individual generation. In Y. Tan, Y. Shi & Z. Ji (Eds.), *ICSI 2012, Part I, LNCS* (Vol. 7331, pp. 243–252). Berlin: Springer.

Zhou, Y., & Liu, B. (2009). Two novel swarm intelligence clustering analysis methods. In *IEEE Fifth International Conference on Natural Computation* (*ICNC*) (pp. 497–501).

Zhou, Y., & Mao, Z. (2003). A new search algorithm for global optimization: Population migration algorithm. *Journal of South China University of Technology, A31*, 1–5.

Zungeru, A. M., Ang, L.-M., & Seng, K. P. (2012). Termite-hill: Performance optimized swarm intelligence based routing algorithm for wireless sensor networks. *Journal of Network and Computer Applications, 35*, 1901–1917.