Chapter 2
Fundamentals of Evolutionary Computation

Abstract The key evolutionary approaches used in the next chapters, including genetic algorithms, quantum-inspired evolutionary algorithms, ant colony optimization, particle swarm optimization and differential evolution are presented.

2.1 Introduction

Evolutionary algorithms (EAs) refer to a generic metaheuristic optimization algorithms characterized by implementations looking at a guided random search of an iterative process [49, 59, 111, 122]. EAs include a family of heuristic algorithms, called metaheuristics [10–12]. As a branch of soft computing referring to less exact calculations [115], EAs has become a well-known research area in computer science [82, 111].

Inspired by natural selection [26] and molecular genetics [18], EAs started with three research topics in the 1950s and 1960s: genetic algorithms (GAs) developed by Holland [58], evolution strategies (ES) invented by Rechenberg [102] and Schwefel [106, 107] and evolutionary programming (EP) introduced by Fogel et al. [40], and has a tremendous growth in the past three decades as witnessed by the increasing number of international conferences, workshops, papers, books and dissertations as well as more and more journals dedicated to the field. Historically, one can divide the EAs research into two groups: classic EAs and recently developed EAs. The former consists of GAs, ES, EP, and genetic programming (GP) [71–73], which were developed in the 1990s. The latter is still in a stage of rapid development and includes quantum-inspired evolutionary algorithms (QIEAs) [54, 119], simulated annealing (SA) [20, 70], ant colony optimization (ACO) [30], particle swarm optimization (PSO) [66, 90, 109], differential evolution (DE) [27, 98], estimation of distribution algorithms (EDAs) [74, 94], biogeography-based optimization (BBO) [110, 111], cultural algorithms (CA) [103], tabu search (TS) [48], artificial fish swarm algorithm (AFSA) [76, 87], artificial bee colony algorithm (ABC) [61], firefly algorithm (FA) [118], bacterial foraging optimization algorithm (BFOA) [93], teaching learning based optimization algorithm (TLBO) [100], shuffled frog leaping algorithm (SFL) [37].

G. Zhang et al., *Real-life Applications with Membrane Computing*,
Emergence, Complexity and Computation 25, DOI 10.1007/978-3-319-55989-6_2

Underlying the different variants of EAs, there are several common features: a fundamental algorithm structure show in Algorithm 1 [2, 119], a single solution or a population of tentative solutions, guided random search by an evaluation function called fitness function, iterative progress toward better solutions to the problem [36, 111]. In Algorithm 1, an EA starts from a single or a population of candidate solution(s) $P(t)$ (also called individual(s)), where t represents the number of evolutionary generations, to a problem, and then goes to an iterative search process, and finally stops when a single or a set of satisfactory solution(s) is found. The search process consists of the evaluation of candidate solution(s), the variation of individual(s) from $P(t)$ to $Q(t)$ by using various evolutionary mechanisms and the generation of the offspring individual(s) $P(t+1)$. Defining practical and robust optimization methodologies, EAs have shown outstanding characteristics, such as global search capabilities, flexibility, robust performance and adaptability, in the process of solving complex problems with combinations, discontinuities, constrains, multiple or many objectives, uncertainties or dynamics [1, 2, 36, 111]. So EAs have been increasingly widely applied to problems ranging from practical applications in industry and commerce to leading-edge scientific research [36, 60].

Algorithm 1 EA Fundamental algorithm structure

Require: A single or a population of initial solution(s) $P(t), t = 0$
1: Evaluate $P(t)$
2: **while** (not termination condition) **do**
3: $Q(t) \leftarrow$ Vary $P(t)$
4: Evaluate $Q(t)$
5: Produce $P(t+1)$ from $Q(t)$
6: $t \leftarrow t+1$
7: **end while**

This chapter is devoted to five EA variants, GAs, QIEAs, ACO, PSO and DE, which are the foundation of membrane algorithms and their engineering applications described in the next two chapters. Thus, the following sections focus on the presentation of fundamental concepts and principles, rather than on demonstrating experiments and results. One can note that the EAs discussed in this chapter are used to solve optimization problems.

2.2 Genetic Algorithms

As genetic algorithms (GAs) are the earliest, most well-known, and most widely used EAs, there are numerous publications describing them [2, 36, 68, 111]. The aim of this section is to briefly highlight the fundamental concepts and evolution principle so as to make it easy for readers to understand the following four EA variants, QIEAs, ACO, PSO and DE, and membrane algorithms in the next chapter

because the notions, operations and procedure underlying GAs are the basics of other types of EAs.

Inspired by natural selection [26] and molecular genetics [18], Holland introduced GAs in the mid-1970s [58], on the basis of the influential works of Fraser [41], Box [14], Friedberg [42], Friedberg et al. [43] and Bremermann [16] in the late 1950s. In a GA, a potential or candidate solution to an optimization problem is called an *individual*; the encoding (binary, numeric, or others) of an individual is known as its *genome* or *chromosome*. A *population* is a set consisting of a certain number of individuals. A chromosome is composed of a sequence of *genes*; specific genes are known as *genotypes*, and the problem-specific parameter representing by a genotype is termed a *phenotype*; the value of a gene is called an *allele*. The individuals at the current generation are named *parents* and correspondingly the new individuals produced by them are called *children* or *offspring*. The function used to evaluate an individual is called *fitness function* and correspondingly the function value with respect to the individual is called its *fitness*, which indicates the quality of the solution in the context of a given problem. The whole process of searching for an optimal solution to a problem is called evolution [68].

Underlying various variants of GAs, there is a common algorithm structure shown in Algorithm 2, where each step is detailed as follows:

Algorithm 2 GA Algorithm structure

Require: A group of random generated initial solutions $P(t), t = 0$
1: Evaluate $P(t)$
2: **while** (not termination condition) **do**
3: Select individuals from $P(t)$ to form $Q_1(t)$ for crossover
4: Crossover $Q_1(t)$ to form $Q_2(t)$
5: Mutate $Q_2(t)$ to form $Q_3(t)$
6: Evaluate $Q_3(t)$
7: $P(t+1) \leftarrow Q_3(t)$
8: $t \leftarrow t+1$
9: **end while**

Step 0: An initial population $P(t)$, $t = 0$, consisting of a certain number of individuals is randomly generated. Each individual is composed of a sequence of codes such as binary, numeric and permutation codes.

Step 1: Each individual in $P(t)$ is evaluated by using the fitness function associated with the optimization problem. Thus, each individual has assigned a fitness value.

Step 2: The termination criterion may be the prescribed maximal number of evolutionary generations or the preset difference between the best solution searched and the optimal/desired solution of the optimization problem.

Step 3: Determine each pair of individuals in $P(t)$ to perform crossover operation. As usual the roulette-wheel selection with respect to fitness, which is also called fitness-proportional selection or fitness-proportionate selection, is used. The selected population is represented as $Q_1(t)$.

Step 4: Swap partial genes of each pair of selected individuals in $Q_1(t)$ with each other by a probabilistic value called crossover probability. The crossovered population is denoted as $Q_2(t)$. It is well known that the crossover probability should be assigned a bigger value.
Step 5: Each gene of each individual in $Q_2(t)$ is mutated by a probabilistic value called mutation probability, which is usually set to a smaller value. The mutated population is denoted as $Q_3(t)$. The uniform mutation is a popular approach.
Step 6: This step is similar to Step 1, i.e., each individual in $Q_3(t)$ is evaluated.
Step 7: The individuals in $Q_3(t)$ are assigned to $P(t+1)$ as the offspring individuals.
Step 8: The evolutionary generation t increases by 1.

The GA research was mainly developed in the past decades with respect to encoding techniques, selection, crossover operators, mutation operators, fitness functions, hybridization with other techniques and theoretical analysis. As usual the individual representation, selection methods, crossover and mutation operators, and fitness functions depend on the optimization problem. The individuals in GAs could be represented by using various types of codes, such as binary and m-ary codes, numeric values, permutation codes and quantum-inspired bits. The analysis of various representations can be found in [13, 54, 95, 104]. Most of the researches on GAs are related to the modification of selection, crossover and mutation operators. So numerous variants of selection, crossover and mutation operators and their effect on the GA performance were reported in literature [15, 44, 55, 62, 63, 81, 89, 92]. Moreover, the influence of crossover and mutation probabilities were also investigated [3, 4, 79]. To select a suitable fitness function to a real-world application problem is also an important issue. In [60], a comprehensive survey of the research on fitness approximation in GAs was reviewed with respect to approximation levels, approximate model management schemes and model construction techniques. Recent research on GAs principally focused on the hybridization with other techniques such as tabu search, simulated annealing, quantum computing, rough set, fuzzy logic theory and other types of EAs. These investigations mentioned above are more concerned with the question of *whether* GAs work. Actually, the theoretical analysis of GAs answers satisfactorily the questions of *how* or *why* GAs work, which are important and challenging issues in the further advance of GAs, even of EAs [111]. Some methods like schema theory, Markov models and Fourier and Walsh transforms have been applied to analyze the GAs behavior [2, 59, 111].

2.3 Quantum-Inspired Evolutionary Algorithms

The past three decades have witnessed the use of various properties from quantum physics to devise a new kind of computers, quantum computers [46, 88]. In contrast with classical computers processing binary digits (*bits*), quantum computers work by handling quantum bits (*qubits*), which are the smallest information units that can be stored in a two-state quantum computer [56]. A qubit can be in a superposition

of the usual '0' and '1' states other than themselves. Thus, a quantum particle could simultaneously be in many incompatible states [88]. Each superposition, $|\psi\rangle$ can be represented as a linear sum of the basis states, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are numbers that denote the corresponding states' probability amplitudes. The values $|\alpha|^2$ and $|\beta|^2$ are the probabilities that the observation of a qubit in state $|\psi\rangle$ will render a '0' or '1' state, respectively [47], and normalization property requires that $|\alpha|^2 + |\beta|^2 = 1$. A quantum gate can be used to modify the state of a qubit [56]. A quantum system $|\psi_n\rangle$ with n qubits can represent 2^n states simultaneously [5, 50] as

$$|\psi_n\rangle = \sum_{j=1}^{2^n} C_j |S_j\rangle, \tag{2.1}$$

where C_j is the probability amplitude of the jth state S_j described by the binary string $(x_1 x_2 \cdots x_n)$, where x_i, $i = 1, 2, \cdots, n$, is either 0 or 1. Nonetheless, the system will "collapse" to a single state if a quantum state is observed.

Inspired by quantum computing, a computational method called quantum-inspired computation is designed to solve various problems in the context of a classical computing paradigm [83]. Amongst the quantum-inspired computation topics, a quantum-inspired evolutionary algorithms (QIEA) is receiving renewed attention. A QIEA is a novel EA *for a classical computer* rather than for a quantum machine (or computer). Generally speaking, a QIEA is designed by integrating the EA framework with quantum-inspired bits (Q-bits), quantum-inspired gates (Q-gates) and probabilistic observation.

Conventional EAs use several different representations to encode solutions onto chromosomes, such as symbolic, binary, and numeric representations [57]. While in a QIEA, a novel probabilistic description, Q-bit representation, of Q-bit individuals is used. A Q-bit individual is represented as a string of Q-bits. The basic computing unit in a QIEA, Q-bit, is defined as a column vector

$$[\alpha \;\; \beta]^T, \tag{2.2}$$

where α and β are real numbers satisfying the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. Equation (2.2) is usually written as $\alpha|0\rangle + \beta|1\rangle$ in quantum mechanics ket-notation. The values $|\alpha|^2$ and $|\beta|^2$ are the probabilities that the Q-bit will be found in the '0' or '1' state, respectively, in quantum theory [54]. By using a probabilistic observation, each Q-bit can be rendered into one binary bit. Algorithm 3 shows the observation process, where x is the observed value of the Q-bit shown in (2.2). Differing from the binary representation that uses 0 or 1 to deterministically represent a bit, the Q-bit representation uses a Q-bit to describe a probabilistic linear superposition of 0 and 1. The Q-bit representation can be easily extended to multi-Q-bit systems.

Algorithm 3 Observation process in the QIEA [54]

Require: A Q-bit $[\alpha \ \ \beta]$
1: **if** $random[0, 1) < |\alpha|^2$ **then**
2: $x \leftarrow 0$
3: **else**
4: $x \leftarrow 1$
5: **end if**

In what follows, the QIEA in [54] is taken as an example to detail the QIEA algorithm. Algorithm 4 shows the pseudocode QIEA algorithm, where each step is described below.

Algorithm 4 Pseudocode algorithm of the QIEA in [54]

Require: An initial population $Q(t), t = 0$
1: Make $P(t)$ by observing the states of $Q(t)$
2: Evaluate $P(t)$
3: Store all solutions in $P(t)$, into $B(t)$ and the best solution $\mathbf{b}$ in $B(t)$
4: **while** (not termination condition) **do**
5: $t \leftarrow t + 1$
6: Make $P(t)$ by observing the states of $Q(t - 1)$
7: Evaluate $P(t)$
8: Update $Q(t)$ using Q-gates
9: Store all solutions in $P(t)$, into $B(t - 1)$ and the best solution $\mathbf{b}$ in $B(t)$
10: **if** (migration condition) **then**
11: Migrate $\mathbf{b}$ or $\mathbf{b}_j^t$ to $B(t)$ globally or locally, respectively
12: **end if**
13: **end while**

Step 0: In the step of "initialize $Q(t)$", a population $Q(0)$ with n multi-Q-bit individuals is produced, $Q(t)$=$\{\boldsymbol{q}_1^t, \boldsymbol{q}_2^t, \cdots, \boldsymbol{q}_n^t\}$, at the generation moment $t = 0$, where $\boldsymbol{q}_i^t$ $(i = 1, 2, \cdots, n)$ is an arbitrary individual in $Q(t)$, denoted as

$$\boldsymbol{q}_i^t = \begin{bmatrix} \alpha_{i1}^t | \alpha_{i2}^t | \cdots | \alpha_{im}^t \\ \beta_{i1}^t | \beta_{i2}^t | \cdots | \beta_{im}^t \end{bmatrix}, \tag{2.3}$$

where m is the string length of the Q-bit individual, that is, the number of Q-bits used in each individual's representation. The values α_{ij}^t and β_{ij}^t, $j = 1, 2, \cdots, m, t = 0$, are initialized by the same probability amplitude $1/\sqrt{2}$, which guarantees that all possible states are superposed with the same probability at the beginning.

Step 1: By independently observing each Q-bit of $Q(t)$ (where at this stage $t = 0$), using the process described in Algorithm 3, binary solutions in $P(t)$, $P(t) = \{\boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \cdots, \boldsymbol{x}_n^t\}$, are obtained, where each $\boldsymbol{x}_i^t$ $(i = 1, 2, \cdots, n)$ is a binary solution with m bits. Each bit '0' or '1' is the observed value of a Q-bit $[\alpha_{ij}^t \ \ \beta_{ij}^t]^T$ in $\boldsymbol{q}_i^t$, respectively, $j = 1, 2, \cdots, m$.

Step 2: The binary solution $\boldsymbol{x}_i^t$ $(i = 1, 2, \cdots, n)$ in $P(t)$ is evaluated thus obtaining its fitness.

Step 3: In this step, all solutions in $P(t)$ are stored into $B(t)$, where $B(t) = \{\boldsymbol{b}_1^t, \boldsymbol{b}_2^t, \cdots, \boldsymbol{b}_n^t\}$ and $\boldsymbol{b}_i^t = \boldsymbol{x}_i^t$ $(i = 1, 2, \cdots, n)$ (again, at this stage, $t = 0$). Furthermore, the best binary solution $\boldsymbol{b}$ in $B(t)$ is also stored.

Step 4: The termination criterion may be the prescribed maximal number of evolutionary generations or the preset difference between the best solution searched and the optimal/desired solution of the optimization problem.

Step 5: The evolutionary generation t increases by 1.

Step 6: This step is similar to Step 1. Observation of the states of $Q(t-1)$ produces the binary solutions in $P(t)$.

Step 7: This step is similar to Step 2.

Step 8: In this step, all the individuals in $Q(t)$ are modified by applying Q-gates. The QIEA uses a *quantum rotation gate* as a Q-gate. To be specific, the jth Q-bit in the ith Q-bit individual $\boldsymbol{q}_i^t$, $j = 1, 2, \cdots, m$, $i = 1, 2, \cdots, n$, is updated by applying the current Q-gate $G_{ij}^t(\theta)$

$$G_{ij}^t(\theta) = \begin{bmatrix} \cos\theta_{ij}^t & -\sin\theta_{ij}^t \\ \sin\theta_{ij}^t & \cos\theta_{ij}^t \end{bmatrix}, \tag{2.4}$$

where θ_{ij}^t is an adjustable Q-gate rotation angle. Thus, the update procedure for the Q-bit $[\alpha_{ij}^t \;\; \beta_{ij}^t]^T$ can be described as

$$\begin{bmatrix} \alpha_{ij}^{t+1} \\ \beta_{ij}^{t+1} \end{bmatrix} = G_{ij}^t(\theta) \begin{bmatrix} \alpha_{ij}^t \\ \beta_{ij}^t \end{bmatrix}, \tag{2.5}$$

where θ_{ij}^t is defined as

$$\theta_{ij}^t = s(\alpha_{ij}^t, \beta_{ij}^t)\Delta\theta_{ij}^t, \tag{2.6}$$

and $s(\alpha_{ij}^t, \beta_{ij}^t)$ and $\Delta\theta_{ij}^t$ are the sign and the value of θ_{ij}^t, respectively. The particular values used in the QIEA in [54] are illustrated in Table 2.1, in which $f(\cdot)$ is the fitness function, $s(\alpha_{ij}^t, \beta_{ij}^t)$ depends on the sign of $\alpha_{ij}^t\beta_{ij}^t$, and b and x are certain bits of the searched best solution $\boldsymbol{b}$ and the current solution $\boldsymbol{x}$, respectively. It is worth pointing out that Table 2.1 was derived from a maximization problem and hence the condition $f(\boldsymbol{x}) \geq f(\boldsymbol{b})$ should be replaced by $f(\boldsymbol{x}) \leq f(\boldsymbol{b})$ if a minimization problem is to be considered.

Step 9: This step is similar to Step 3. The better candidate between $\boldsymbol{x}_i^t$ in $P(t)$ and $\boldsymbol{b}_i^{t-1}$ in $B(t-1)$, $i = 1, 2, \cdots, n$, is selected and stored into $B(t)$. Simultaneously, the best candidate $\boldsymbol{b}$ in $B(t)$ is also stored.

Steps 10–11: This step includes local and global migrations, where a *migration* in this algorithm is defined as the process of copying $\boldsymbol{b}_j^t$ in $B(t)$ or $\boldsymbol{b}$ to $B(t)$. A global migration is realized by substituting $\boldsymbol{b}$ for all the solutions in $B(t)$, and a local migration is realized between each pair of neighboring solutions in $B(t)$, i.e., by

Table 2.1 Lookup table of θ_{ij}^t, where $f(\cdot)$ is the fitness, $s(\alpha_{ij}^t, \beta_{ij}^t)$ is the sign of θ_{ij}^t, and b and x are certain bits of the searched best solution $\boldsymbol{b}$ and the current solution $\boldsymbol{x}$, respectively [54]

x	b	$f(\boldsymbol{x}) \geq f(\boldsymbol{b})$	$\Delta\theta_{ij}^t$	$s(\alpha_{ij}^t, \beta_{ij}^t)$	
				$\alpha_{ij}^t \beta_{ij}^t \geq 0$	$\alpha_{ij}^t \beta_{ij}^t < 0$
0	0	false	0	± 1	± 1
0	0	true	0	± 1	± 1
0	1	false	0.01π	$+1$	-1
0	1	true	0	± 1	± 1
1	0	false	0.01π	-1	$+1$
1	0	true	0	± 1	± 1
1	1	false	0	± 1	± 1
1	1	true	0	± 1	± 1

substituting the better one of two neighboring solutions for the other solution. For more information about the migrations, see [54].

In summary, in QIEA, Q-bits are applied to represent genotype individuals; Q-gates are employed to operate on Q-bits to generate offspring; and the genotypes and phenotypes are linked by a probabilistic observation process. QIEAs were firstly introduced by Narayanan and Moore in the 1990s to solve the traveling salesman problem [84], in which the crossover operation was performed based on the concept of interference. The contribution of Narayanan and Moore signaled the potential advantage of introducing quantum computational parallelism into the evolutionary algorithm framework. No further attention was paid to QIEAs until a practical algorithm was proposed by [53, 54], but they are now viewed as an emergent theme in evolutionary computation. In the last sixteen years have been considered various variants of QIEAs to solve a large number of problems (for a comprehensive survey see [119]). The main characteristics of QIEAs can be summarized as follows:

- A QIEA uses a novel representation, *Q-bit representation*, to describe individuals of a population. Q-bit representation provides probabilistically a linear superposition of multiple states.
- A QIEA employs a *Q-gate* guiding the individuals toward better solutions [54] to produce the individuals at the next generation.
- A QIEA can exploit the search space for a global solution with a small number of individuals, even with one element [54].

Currently there is intensive research in this area, but there are some aspects that need to be addressed from the perspectives of theoretical research, engineering applications, comparative experiments, extensions of QIEAs and hybrid algorithms. These issues were presented in detail in [119].

2.4 Ant Colony Optimization

Instead of simulating the process of natural selection, some researchers introduced novel algorithms by simulating the collective behavior of decentralized, self-organized colonies. Ant colony optimization (ACO), originally proposed by Dorigo and co-workers in 1991 [33] and later explicitly defined in [32], is such a metaheuristic approach for combinatorial optimization problem inspired by the foraging behavior of ants. In nature, to find the shortest path from the nest to a food source, ant colonies exploit a positive feedback mechanism by laying and detecting the chemical trail (pheromone) on the ground during their trips. More pheromone is left when more ants go through the trip, which improves the probability of other ants choosing this trip. Furthermore, the pheromone has a decreasing action over time because of evaporation of trail. In the ACO metaphor, a generic combinatorial optimization problem is transformed into a shortest path problem which is encoded as a graph; a number of paths are constructed by artificial ants walking on the graph based on a probabilistic model using pheromone; the cost of the generated path is utilized to modify the pheromone, and hence to bias the generation of further paths.

ACO was initially applied to solve traveling salesman problem (TSP) [32], one of the well-known NP-complete problems and most intensively studied combinatorial optimization problems in the areas of optimization, operational research, theoretical computer science, and computational mathematics. The TSP can be described as follows [121]. Given a set C of N cities, i.e., $C = \{c_1, c_2, \cdots, c_N\}$, and a set D of the pairwise travel costs, $D = \{d_{ij} | i, j \in \{1, 2, \cdots, N\}, i \neq j\}$, it is requested to find the minimal cost of the path taken by a salesman visiting each of the cities just once and returning to the starting point. More generally, the task is to find a Hamiltonian tour with a minimal length in a connected, directed graph with a positive weight associated to each edge. If $d_{ij} = d_{ji}$, the TSP is symmetric in the sense that traveling from city c_i to city c_j costs just as much as traveling in the opposite direction, otherwise, it is asymmetric. This section uses symmetric TSP as an example to describe ACO.

ACO is an iterative metaheuristic. At each iteration, a number of paths are constructed based on stochastic decisions which are biased by pheromone and heuristic information. These paths are used for updating the pheromone in order to bias further solutions towards promising regions of the search space. Algorithm 5 gives the pseudocode of a generic ACO algorithm. In the pseudocode, a local search procedure may be applied for further improving the solutions constructed by ants. The use of such a procedure is optional; however, it has been observed that its use improves the algorithms's overall performance. The most used and well-known tour improvement local searches are 2-opt and 3-opt [69], in which two and three edges of a tour are exchanged, respectively.

Algorithm 5 Pseudocode of a generic ACO

Require: $t = 0$
1: Pheromone trail initialization
2: **while** (not termination condition) **do**
3: Construct tours
4: Apply local search (*optional*)
5: Update pheromone
6: $t \leftarrow t + 1$
7: **end while**

The most well-known ACO algorithms in literature include the earliest ant system (AS) [33, 34], MAX-MIN ant system [114], hyper-cube ant system [9], and ant colony system (ACS) [29], and they differ in the way to construct tours and/or update pheromone. According to the studies in [8, 28, 31], the ACS is one of the most powerful ACO algorithms. Therefore, we take it as an example to describe the ACO algorithm. Algorithm 6 shows the pseudocode of an ACS algorithm, where each step is described below.

Algorithm 6 Pseudocode algorithm of the ACS in [29]

Require: $t = 0$
1: Pheromone trail initialization
2: **while** (not termination condition) **do**
3: Randomly place M ants in the N nodes
4: **for** $k = 1, 2, \ldots, M$ **do**
5: **for** $n = 1, 2, \ldots, N$ **do**
6: Ants moving
7: **end for**
8: Evaluate the length of the path construct by ant k
9: Local pheromone updating
10: **end for**
11: Global pheromone updating
12: $t \leftarrow t + 1$
13: **end while**

Step 1: At the beginning of a run, the initial pheromone value τ_0 is set to be $1/ND_a$, where N is the number of cities in a TSP and D_a is the length of a feasible tour generated randomly or by the nearest-neighbor heuristic.

Step 2: The termination criterion may be the prescribed maximal number of generations or the preset difference between the best path searched and the optimal/desired path of the problem.

Step 3: The M ants are randomly positioned on the N nodes of the TSP graph as the initial state of tour construction.

Step 4: The ants construct paths one by one.

Steps 5–7: Each ant constructs a whole path step by step using a pseudorandom proportional rule. Specifically, the kth ant in the ith city chooses the next city j by using the following formula

$$j = \begin{cases} \arg\max_{l \in \mathcal{N}_i^k} \{[\tau_{il}]^\alpha [\eta_{il}]^\beta\}, \text{ if } q \leq q_0 \\ J, \qquad\qquad \text{otherwise} \end{cases} \tag{2.7}$$

where arg max{·} stands for the argument of the maximum, that is to say, the set of points of the given argument for which the value of the given expression attains its maximum value; τ_{il} is the pheromone value of the edge connecting the ith node and the lth node; η_{il} is a heuristic information value, equal to the inverse of the distance between the ith and lth cities; the parameters α and β ($\alpha > 0$ and $\beta > 0$) determine the relative importance of the pheromone value τ_{il} and the heuristic information η_{il}; $\mathcal{N}_i^k$ ($\mathcal{N}_i^k \subseteq \mathcal{N}$) is the set of all nodes of the TSP graph that the kth ant in the ith city can visit; q_0 ($0 \leq q_0 \leq 1$) is a user-defined parameter specifying the distribution ratio of the two choices; q is a random number generated by using a uniform distribution function in the interval [0, 1]; J means that the next city j is chosen by using a random proportional rule, i.e., the kth ant in the ith city visits the city j at the next step according to the probability

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \ j \in \mathcal{N}_i^k \\ 0, \qquad\qquad \text{otherwise} \end{cases} \tag{2.8}$$

Step 8: At each time an ant construct a whole path, the length of this path is evaluated and compared with the best path stored. If the new path is better than the stored best path, the best path is updated.

Step 9: Ant releases a mount of pheromone on edges at its every traveling when it completes a path construction procedure. In ACS, an ant updates the pheromone value τ_{ij} of the tour by applying a local pheromone update rule, defined as follows

$$\tau_{ij} = (1 - \upsilon)\tau_{ij} + \upsilon\tau_0 \tag{2.9}$$

where υ ($0 < \upsilon < 1$) is a local pheromone decay coefficient. The local pheromone update is used to encourage subsequent ants to choose other edges and, hence, to produce different solutions, by decreasing the pheromone value on the traversed edges.

Step 11: In this step, the globally best ant, i.e., the ant which constructs the shortest tour form the beginning of the trial, is allowed to deposit additional pheromone via a global pheromone update rule. To be specific, the pheromone value τ_{ij} of the edge connecting the ith node and the jth node is modified by

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij} \tag{2.10}$$

where ρ ($0 < \rho \leq 1$) is a global pheromone decay coefficient which is also called pheromone evaporation rate, and $\Delta\tau_{ij}$ is

$$\Delta\tau_{ij} = \begin{cases} 1/D_b, & \text{if } (i, j) \in T_b \\ 0, & \text{otherwise} \end{cases} \tag{2.11}$$

where D_b is the length of the shorted path searched so far, and T_b is the path corresponding to D_b.

Step 12: The iteration counter t increases by 1.

At present, the research of ACO focuses on three main aspects, i.e., improvement of different ACO algorithms, applications, and theoretic analysis. Regarding the performance improvement, researchers proposed a large variety of ACO variants by designing new path construct schemes, pheromone update schemes, mixing with various local search operators, or even incorporating novel mechanism like chaos [75]. As for applications, although ACO was originally introduced in connection to TSP, it is now recognized as one of the state-of-the-art methods for solving other kinds of discrete optimization problems, such as assignment problems, scheduling problems, graph coloring, vehicle routing problems, design of communication networks. Furthermore, in recent years, some researchers have extended its use for continuous optimization problems, multi-objective discrete problems and dynamic problems. Since experimental results show better performance of ACO over other meta-heuristics, researchers have paid much attention to the ACO theory to explain why and how it works. The first convergence proof of ACO was given in [51]. Since then various convergence proofs for various ACO variants have been published, e.g. [19, 30, 52]. For more details of the progress of ACO, the readers can refer to the comprehensive survey papers [6, 28].

2.5 Particle Swarm Optimization

Particle swarm optimization (PSO) is another well-known population-based meta-heuristic approach proposed by Kennedy and Eberhart in 1995 for continuous optimization problems [65]. This technique was motivated by social behavior of bird flock. In PSO, each individual is called a "particle" with properties being described by the current position vector, its velocity vector and its personal best position vector, which represents a potential solution to a problem. Instead of using genetic operators (e.g., crossover, mutation) to evolve individuals, the trajectory of each particle is adjusted by dynamically altering its velocity according to its own flying experience and its companion's experience.

Suppose there are N particles in a PSO, and each particle is treated as a point in a D-dimensional space, representing a candidate solution to the problem. Each particle is characterized by the current position vector $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D})$, velocity vector $\mathbf{v}_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,D})$ and its personal best position vector $\mathbf{p}_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,D})$, $i = 1, 2, \ldots, N$. The particle with its personal best position which returns the best fitness value among the population is called the global best particle and its position is recorded as $\mathbf{p}_g = (p_{g,1}, p_{g,2}, \ldots, p_{g,D})$, where g is the

index of the global best particle. Algorithm 7 shows the pseudocode PSO algorithm, where each step is described below [25, 77, 90, 116, 123].

Algorithm 7 Pseudocode algorithm of the PSO

Require: An initial population of N particles with positions $P(t)$ and velocities $V(t)$, $t = 0$
Evaluate the particles
Initialize personal best and global best
while (not termination condition) **do**
  for $i = 1, 2, \ldots, N$ **do**
    Change the velocity and position
    Evaluate the particle
    Update personal and global best positions
  end for
  $t \leftarrow t + 1$
end while

Step 0: In this step, uniform distribution on $[x_j^{min}, x_j^{max}]$ $(j = 1, 2, \ldots, D)$ in the jth dimension is used to generate the initial current position vector $\mathbf{x}_i$ for the ith particle, where x_j^{min} and x_j^{max} are lower limit and upper limit of particle positions in the jth dimension. Similarly, the initial velocity vector $\mathbf{v}_i$ is initialized by choosing its jth component randomly in $[-v_j^{max}, v_j^{max}]$ $(j = 1, 2, \ldots, D)$, where v_j^{max} is the upper limit of velocities in the jth dimension. v_j^{max} is an important parameter that determines the search behavior of the algorithm. If v_j^{max} is too small, particles may become trapped in local optima, unable to move far away to a better position. On the other hand, if v_j^{max} is too large, particles might fly past good solutions.

Step 1: The performance of each particle is measured according to a pre-defined fitness function.

Step 2: For each particle, set its personal best position as the current position, i.e., $\mathbf{p}_i = \mathbf{x}_i$, $i = 1, 2, \ldots, N$. Also, identify the global best position $\mathbf{p}_g$ based on the fitness value of the particles.

Step 3: The termination criterion may be the prescribed maximal number of generations or the preset difference between the best solution searched and the optimal/desired solution of the problem.

Step 4: The particle flies one by one.

Step 5: The velocity and position of the ith particle are updated according to the following equation,

$$\mathbf{v}_i = \mathbf{v}_i + c_1 r_1 (\mathbf{p}_i - \mathbf{x}_i) + c_2 r_2 (\mathbf{p}_g - \mathbf{x}_i) \tag{2.12}$$

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i \tag{2.13}$$

where c_1 and c_2 are acceleration coefficients, r_1 and r_2 are two different sequences of random numbers uniformly distributed over $(0, 1)$. In (2.12), the first part represents the previous velocity, which provides the necessary momentum for particles to roam across the search space; the second part is the "cognition" part,

which represents the private thinking of the particle itself; the third part is the "social" part, which represents the collaboration among the particles in finding the global optimal solution. Equation (2.12) is used to calculate the particle's new velocity and the particle flies toward a new position according to (2.13). In this step, if the particle's velocity on jth dimension exceeds the maximum value v_j^{max}, then it is clamped to v_j^{max}.

Step 6: The performance of the particle is measured according to a pre-defined fitness function.

Step 7: Comparing particle's fitness with its personal best performance. If current value is better than its personal best fitness, then update its personal best fitness as the current fitness and set $\mathbf{p}_i = \mathbf{x}_i$. Also, comparing particle's fitness with the population's overall previous best. If current value is better that the previous best value, then update the global best fitness as the current value and set $\mathbf{p}_g = \mathbf{x}_i$.

Step 9: The iteration counter t increases by 1.

The original PSO has been found performing well in solving some simple problems, however, its performance is not satisfactory when solving complex problems. Therefore, a considerable amount of work has been done in developing the original PSO. For example, in [108], to reduce the importance of v_j^{max}, Shi and Eberhart introduced the concept of inertia weight in the calculation of velocities to balance the local and global search, and later they further improved the algorithm performance with a linearly varying inertia weigh over the iterations. In [101], time-varying acceleration coefficients are introduced to control the local search and the convergence to the global optimum solution. In fully informed particle swarm algorithm [80], the particle is affected by all its neighbors, sometimes with no influence from its own previous success. In [77], a novel learning strategy whereby all other particles' historical best information is proposed to update a particle's velocity, which enables the diversity of the swarm to be preserved to discourage premature convergence. Instead of moving toward a kind of stochastic average of personal best position and global best position, particles moving toward points defined by personal best position and local best position is also widely investigated, where the best position is the location of the particle's neighborhood defined by a certain topology. Currently, various topologies have been studied, such as simple ring lattice, small-world modifications [64, 117], or von Neumann structure [67]. Some theoretical analysis for PSO approaches has been developed. For example, in [25], Clerc and Kennedy analyzed a particles trajectory as it moves in discrete time from the algebraic view and in continuous time from the analytical view. In [24], Clerc analyzed the distribution of velocities of a particle in order to observe algorithm behavior in stagnation phases. As for applications, PSO has been applied across various areas, such as classification, pattern recognition, planning, signal processing, power system, controller design. For more information of the important work in PSO, the readers can refer to the survey paper [96].

2.6 Differential Evolution

Differential Evolution (DE) is a meta-heuristic approach originally proposed by Storn and Price in 1996 for handling continuous optimization problems [112, 113]. Rather than using natural selection or colony collective behavior, DE relies on engineering aspects. In 1994, Price published a genetic annealing algorithm [97], which is a population-based, combinatorial optimization algorithm that implements an annealing criterion via thresholds. Later, Genetic Annealing has been used to solve Chebyshev polynomial fitting problem. As the performance of genetic annealing was not very satisfactory because of its slow convergence and difficulties to set effective control parameters, Price introduced floating-point encoding, arithmetic operations and differential mutation operator in genetic annealing algorithm. As a result, these alterations transformed the combinatorial algorithm genetic annealing into a numerical optimizer, which becomes the first generation of DE. Due to its distinguished characteristics, such as few control parameters, simple and straightforward implementation, remarkable performance and low complexity, DE [27, 86] has been recognized as a competitive continuous optimization technique.

Similar to GA or PSO, DE also maintains a population during the evolution. Let $P(t) = \{\mathbf{x}_1^t, \mathbf{x}_2^t, \ldots, \mathbf{x}_N^t\}$ be the population at the tth iteration, and $\mathbf{x}_i^t = (x_{i,1}^t, x_{i,2}^t, \ldots, x_{i,D}^t)$ $(i = 1, 2, \ldots, N)$ be the ith individual in $P(t)$ that represents a potential solution to the problem, where N is the population size and D is the number of decision variables of the problem. Starting with an initial population $P(t)(t = 0)$, the optimization process involves three basic steps, i.e., mutation, crossover and selection. Algorithm 8 shows the pseudocode of a basic DE, where each step is described below [21–23, 86].

Algorithm 8 Pseudocode algorithm of the basic DE

Require: An initial population $P(t), t = 0$
1: Evaluate the population $P(t)$
2: **while** (not termination condition) **do**
3: Mutate to form $V(t)$
4: Crossover to form $U(t)$
5: Evaluate $U(t)$
6: Selection to form $P(t+1)$
7: $t \leftarrow t + 1$
8: **end while**

Step 0: The initial population $P(0) = \{\mathbf{x}_1^0, \mathbf{x}_2^0, \ldots, \mathbf{x}_N^0\}$ is produced, where each component of an individual is uniformly and randomly sampled in the feasible space, that is,

$$x_{i,j}^0 = x_j^{min} + rand(0, 1) \cdot (x_j^{max} - x_j^{min}), \tag{2.14}$$

where $i = 1, 2, \ldots, N$; $j = 1, 2, \ldots, D$; $rand(0, 1)$ is a uniformly distributed random variable within the interval [0,1], x_j^{min} and x_j^{max} are the lower and upper bound of the jth decision variable.

Step 1: The performance of each individual is evaluated according to a pre-defined fitness function.

Step 2: The termination criterion may be the prescribed maximal number of iterations or the preset difference between the best solution searched and the optimal/desired solution of the problem.

Step 3: The mutation operator is performed on $\mathbf{x}_i^t$ (called target vector) $(i = 1, 2, \ldots, N)$ to create a mutant vector $\mathbf{v}_i^t$ (called donor vector) by perturbing a randomly selected vector $\mathbf{x}_{r_1}^t$ with the difference of two other randomly selected vector $\mathbf{x}_{r_2}^t$ and $\mathbf{x}_{r_3}^t$. This operation is formulated as

$$\mathbf{v}_i^t = \mathbf{x}_{r_1}^t + F \cdot (\mathbf{x}_{r_2}^t - \mathbf{x}_{r_3}^t) \tag{2.15}$$

where $\mathbf{x}_{r_1}^t$, $\mathbf{x}_{r_2}^t$ and $\mathbf{x}_{r_3}^t$ are distinct vectors randomly selected from the current population $P(t)$, and they are selected a new for each mutation operation. $F \in (0, 1)$ is a constant called differential factor, which scales the differential vector $(\mathbf{x}_{r_2}^t - \mathbf{x}_{r_3}^t)$ added to the base vector $\mathbf{x}_{r_1}^t$.

Step 4: Following mutation, the donor individual $\mathbf{v}_i^t$ $(i = 1, 2, \ldots, N)$ is recombined with the target individual $\mathbf{x}_i^t$ to produce an offspring $\mathbf{u}_i^t$ (called trial vector) by using a binomial crossover operator, which is a typical case of genes' exchange, formulated as

$$u_{i,j}^t = \begin{cases} v_{i,j}^t, \text{ if } rand_j(0, 1) \leq Cr \text{ or } j = j_{\text{rand}} \\ x_{i,j}^t, \text{ otherwise} \end{cases}, \tag{2.16}$$

where $j = 1, 2, \ldots, D$; $Cr \in (0, 1)$ is a crossover rate which is used to control the diversity of the population; and $j_{\text{rand}} \in \{1, 2, \ldots, D\}$ is a random integer generated once for each individual $\mathbf{x}_i^t$. The condition $j = j_{\text{rand}}$ makes sure that at least one component of $\mathbf{u}_i^t$ inherits from $\mathbf{v}_i^t$ so that $\mathbf{u}_i^t$ will not be identical with $\mathbf{x}_i^t$.

Step 5: The trial vector $\mathbf{u}_i^t$ $(i = 1, 2, \ldots, N)$ is evaluated according to a pre-defined fitness function.

Step 6: The selection operator is performed on $P(t)$ and $U(t)$ to construct the population $P(t+1)$ by choosing vectors between the trial vectors and their corresponding target vectors following the formula

$$\mathbf{x}_i^{t+1} = \begin{cases} \mathbf{u}_i^t, \text{ if } f(\mathbf{u}_i^t) \leq f(\mathbf{x}_i^t) \\ \mathbf{x}_i^t, \text{ otherwise,} \end{cases}, \tag{2.17}$$

where $f(\cdot)$ is a fitness function.

Step 7: The iteration counter t increases by 1.

In spite of several advantages, DE still suffers from prematurity and/or stagnation. Hence a good volume of work in the literature has been devoted to overcome

its drawbacks, mainly from the perspectives of parameter control, operator design, population structure, and hybridization with other meta-heuristics. Many attempts have been made to improve DE performance by setting appropriate parameter values [45, 105, 112] or using parameter adaptation techniques [17, 78, 99] for scale factor F and crossover rate Cr. Also, lots of work focused on designing of new mutation operators, such as trigonometric mutation [39], "DE/current-to-$p$best" mutation [120], GPBX-α mutation [35], or mixing mutation operators [78, 99]. The population structure determines the way individuals share information with each other and many researchers investigate the population structure in DE, see [21, 35, 38]. Hybridization has become an attractive route in algorithm design due to its capability for handling quite complex problems. Therefore, significant work has been done on hybridizing DE with other meta-heuristics, see [7, 85, 91]. In addition to improving DE performance, DE has also been applied to various areas, like signal processing, controller design, planning, power systems, clustering, etc. For more details of DE, two most recently survey papers [27, 86], are recommended.

2.7 Conclusions

This chapter introduced the fundamental concepts and principles of several EA variants including GAs, QIEAs, ACO, PSO and DE. For each variant, we reviewed its history, detailed its algorithm and addressed its future research issues. The five variants will be used to design different types of membrane algorithms in the next chapter.

References

1. Bäck, T. 1996. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford: Oxford University Press.
2. Bäck, T., U. Hammel, and H. Schwefel. 1997. Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation* 1 (2): 3–17.
3. Bae, S.H., and B.R. Moon. 2004. Mutation rates in the context of hybrid genetic algorithms. In *Genetic and Evolutionary Computation (GECCO 2004)*. Lecture Notes in Artificial Intelligence, vol. 3103, ed. K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P.L. Lanzi, L. Spector, A.G.B. Tettamanzi, D. Thierens, and A. Tyrrell, 381–382. Berlin: Springer.
4. Bagchi, P., and S. Pal. 2011. Controlling crossover probability in case of a genetic algorithm. In *Information Technology and Mobile Communication (AIM 2011), Communications in Computer and Information Science*, vol. 147, ed. V.V. Das, G. Thomas, and F.L. Gaol, 287–290. Berlin: Springer.
5. Bennett, C.H., and D.P. DiVincenzo. 2000. Quantum information and computation. *Nature* 404: 247–255.
6. Birattari, M., P. Pellegrini, and M. Dorigo. 2007. On the invariance of ant colony optimization. *IEEE Transactions on Evolutionary Computation* 11 (6): 732–742.

7. Biswas, A., S. Dasgupta, S. Das, and A. Abraham. 2006. A synergy of differential evolution and bacterial foraging algorithm for global optimization. *Neural Network World* 17 (6): 607–626.
8. Blum, C. 2005. Ant colony optimization introduction and recent trends. *Physics of Life Reviews* 2: 353–373.
9. Blum, C., and M. Dorigo. 2004. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 34 (2): 1161–1172.
10. Blum, C., and A. Roli. 2008. Hybrid metaheuristics: an introduction. In *Hybrid Metaheuristics: An Emerging Approach to Optimization, Studies in Computational Intelligence*, vol. 114, ed. C. Blum, M.J.B. Aguilera, A. Roli, and M. Sampels, 1–30. Berlin: Springer.
11. Blum, C., J. Puchinger, G.R. Raidl, and A. Roli. 2011. Hybrid metaheuristics in combinatorial optimization: a survey. *Applied Soft Computing* 11 (6): 4135–4151.
12. Boussa, I., J. Lepagnot, and P. Siarry. 2013. A survey on optimization metaheuristics. *Information Sciences* 237: 82–17.
13. Boozarjomehry, R.B., and M. Masoori. 2007. Which method is better for the kinetic modeling: decimal encoded or binary genetic algorithm? *Chemical Engineering Journal* 130 (1): 29–37.
14. Box, G.E.P. 1957. Evolutionary operation: a method for increasing industrial productivity. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 6 (2): 81–101.
15. Braune, R., S. Wagner, and M. Affenzeller. 2005. On the analysis of crossover schemes for genetic algorithms applied to the job shop scheduling problem. In *Proceedings of 9th World Multi-Conference on Systemics, Cybernetics and Informatics*, vol. 6, 236–241.
16. Bremermann, H.J. 1962. Optimization through evolution and recombination. In *Self-Organizing Systems*, ed. M.C. Yovits, G.T. Jacobi, and G.D. Goldstein. Washington DC: Spartan.
17. Brest, J., S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. 2006. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation* 10 (6): 646–657.
18. Burian, R. 1996. Underappreciated pathways toward molecular genetics as illustrated by Jean Brachet's cytochemical embryology. In *The Philosophy and History of Molecular Biology: New Perspectives*, ed. S. Sarkar, 67–85. Netherlands: Kluwer Academic Publishers.
19. Carvelli, L., and G. Sebastiani. 2011. Some issues of ACO algorithm convergence. In *Ant Colony Optimization: Methods and Applications*, ed. A. Ostfeld, 39–52. Croatia: InTech Press.
20. Černý, V. 1985. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45 (1): 41–51.
21. Cheng, J., G. Zhang, and F. Neri. 2013. Enhancing distributed differential evolution with multicultural migration for global numerical optimization. *Information Sciences* 247: 72–93.
22. Cheng, J., G.G. Yen, and G. Zhang. 2015. A many-objective evolutionary algorithm with enhanced mating and environmental selections. *IEEE Transactions on Evolutionary Computation* 19 (4): 592–605.
23. Cheng, J., G. Zhang, F. Caraffini, and F. Neri. 2015. Multicriteria adaptive differential evolution for global numerical optimization. *Integrated Computer-Aided Engineering* 22 (2): 103–117.
24. Clerc, M. 2006. Stagnation analysis in particle swarm optimization or what happens when nothing happens, Technical Report CSM-460, Department of Computer Science, University of Essex.
25. Clerc, M., and J. Kennedy. 2002. The Particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation* 6 (1): 58–73.
26. Darwin, C. 1859. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*. London: Murray.
27. Das, S., and P.N. Suganthan. 2011. Differential evolution: a survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation* 15 (1): 4–31.

28. Dorigo, M., and C. Blum. 2005. Ant colony optimization theory: a survey. *Theoretical Computer Science* 344: 243–278.
29. Dorigo, M., and L.M. Gambardella. 1997. Ant Colony System: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 1 (1): 53–66.
30. Dorigo, M., and T. Stutzle. 2004. *Ant Colony Optimization*. Scituate: Bradford Company.
31. Dorigo, M., M. Birattari, and T. Stützle. 2006. Ant colony optimization: artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine* 1: 28–39.
32. Dorigo, M., G. Caro, and L.M. Gambardella. 1999. Ant algorithms for distributed discrete optimization. *Artificial Life* 5 (2): 137–172.
33. Dorigo, M., V. Maniezzo, and A. Colorni. 1991. Positive feedback as a search strategy, Technical Report 01–016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.
34. Dorigo, M., V. Maniezzo, and A. Colorni. 1996. Ant System: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 26 (1): 29–41.
35. Dorronsoro, B., and P. Bouvry. 2011. Improving classical and decentralized differential evolution with new mutation operator and population topologies. *IEEE Transactions on Evolutionary Computation* 15 (1): 67–98.
36. Eiben, A.E., and J. Smith. 2003. *Introduction to Evolutionary Computing*. Berlin: Springer.
37. Eusuff, M.M., and K.E. Lansey. 2003. Optimization of water distribution network design using the shuffled frog leaping algorithm. *Journal of Water Resources Planning and Management* 129 (2): 210–225.
38. Falco, I.D., A.D. Cioppa, D. Maisto, U. Scafuri, and E. Tarantino. 2012. Improving classical and decentralized differential evolution. *Information Sciences* 207: 50–65.
39. Fan, H.Y., and J. Lampinen. 2003. A trigonometric mutation operator to differential evolution. *Journal of Global Optimization* 27 (1): 105–129.
40. Fogel, L., A. Owens, and M. Walsh. 1966. *Artificial intelligence through simulated evolution*. Chichester: Wiley.
41. Fraser, A.S. 1957. Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Sciences* 10 (4): 484–491.
42. Friedberg, R.M. 1958. A learning machine: Part I. *IBM Journal of Research and Development* 2 (1): 2–13.
43. Friedberg, R.M., B. Dunham, and J. North. 1959. A learning machine: Part II. *IBM Journal of Research and Development* 3 (3): 282–287.
44. Galaviz-Casas, J. 1998. Selection analysis in genetic algorithms. In *Progress in Artificial Intelligence (IBERAMIA 98)*, Lecture Notes in Artificial Intelligence, vol. 1484, ed. H. Coelho, 283–292. Berlin: Springer.
45. Gämperle, R., S.D. Müller, and P. Koumoutsakos. 2002. A parameter study for differential evolution. In *Proceedings of the Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, 293–298.
46. Glassner, A. 2001. Quantum computing, Part 2. *IEEE Computer Graphics and Applications* 21 (6): 86–95.
47. Glassner, A. 2001. Quantum computing, Part 3. *IEEE Computer Graphics and Applications* 21 (6): 72–82.
48. Glover, F. 1989. Tabu search-part I. *INFORMS Journal on Computing* 1 (3): 190–206.
49. Goldberg, D.E. 1989. *Genetic algorithms in search, optimization and machine learning*. Boston: Addison-Wesley Longman Publishing Co. Inc.
50. Grover, L.K. 1999. Quantum computation. In *Proceedings of the 12th International Conference on VLSI Design*, 548–553.
51. Gutjahr, W. 2000. A graph-based ant system and its convergence. *Future Generation Computer Systems* 16 (9): 873–888.
52. Gutjahr, W. 2008. First steps to the runtime complexity analysis of ant colony optimization. *Computers and Operations Research* 35 (9): 2711–2727.

53. Han, K.H., and J.H. Kim. 2000. Genetic quantum algorithm and its application to combinatorial optimization problem. In *Proceedings of IEEE Congress on Evolutionary Computation*, 1354–1360.
54. Han, K.H., and J.H. Kim. 2002. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Transactions on Evolutionary Computation* 6 (6): 580–593.
55. Herrera, F., M. Lozano, and A.M. Sanchez. 2003. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems* 18 (3): 309–338.
56. Hey, T. 1999. Quantum computing: an introduction. *Computing and Control Engineering Journal* 10 (3): 105–112.
57. Hinterding, R. 1999. Representation, constraint satisfaction and the knapsack problem. In *Proceedings of IEEE Congress on Evolutionary Computation*, 1286–1292.
58. Holland, J.H. 1975. *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.
59. Iba, H., and N. Noman. 2011. *New frontier in evolutionary algorithms: theory and applications*. London: Imperial College Press.
60. Jin, Y. 2005. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing* 9: 3–12.
61. Karaboga, D. 2005. An idea based on honey bee swarm for numerical optimization, Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department.
62. Katayama, K., H. Hirabayashi, and H. Narihisa. 2003. Analysis of crossovers and selections in a coarse-grained parallel genetic algorithm. *Mathematical and Computer Modelling* 38 (11–13): 1275–1282.
63. Kaya, M. 2011. The effects of two new crossover operators on genetic algorithm performance. *Applied Soft Computing* 11 (1): 881–890.
64. Kennedy, J. 1999. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1931–1938.
65. Kennedy, J., and R. Eberhart. 1996. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, 69–73.
66. Kennedy, J., and R.C. Eberhart. 2001. *Swarm Intelligence*. San Francisco: Morgan Kaufmann Publishers Inc.
67. Kennedy, J., and R. Mendes. 2002. Population structure and particle swarm performance. In *Proceedings of IEEE International Conference on Evolutionary Computation*, 1671–1676.
68. Kicinger, R., T. Arciszewski, and K. De Jong. 2005. Evolutionary computation and structural design: a survey of the state-of-the-art. *Computers and Structures* 83 (23–24): 1943–1978.
69. Kin, S. 1965. Computer solutions of the traveling salesman problem. *Bell System Technical Journal* 44 (10): 2245–2269.
70. Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220 (4598): 671–680.
71. Koza, J.R. 1992. *Genetic programming: on the programming of computers by means of natural selection*. Cambridge: MIT Press.
72. Koza, J.R. 1994. *Genetic programming II: automatic discovery of reusable programs*. Cambridge: MIT Press.
73. Langdon, W.B., and R. Poli. 2002. *Foundations of genetic programming*. Berlin: Springer.
74. Larrañaga, P., and J.A. Lozano (eds.). 2002. *Estimation of distribution algorithms: a new tool for evolutionary computation*. Boston: Kluwer Academic Publishers.
75. Li, L., H. Peng, J. Kurths, Y. Yang, and H.J. Schellnhuber. 2014. Chaos-order transition in foraging behavior of ants. *Proceedings of the National Academy of Sciences of the United States of America* 111 (23): 8392–8397.
76. Li, L.X., Z.J. Shao, and J.X. Qian. 2002. An optimizing method based on autonomous animate: fish swarm algorithm. In *Proceeding of System Engineering Theory and Practice*, 32–38.

77. Liang, J.J., A.K. Qin, P.N. Suganthan, and S. Baskar. 2006. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation* 10 (3): 281–285.
78. Mallipeddi, R., P.N. Suganthan, Q.K. Pan, and M.F. Tasgetiren. 2011. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing* 11 (2): 1679–1696.
79. Martin, J.L.F.V., and M.S. Sanchez. 2002. Does crossover probability depend on fitness and hamming differences in genetic algorithms? In *Artificial Neural Networks (ICANN 2002)*, Lecture Notes in Computer Science, vol. 2415, ed. J.R. Dorronsoro, 389–394. Berlin: Springer.
80. Mendes, R., J. Kennedy, and J. Neves. 2004. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation* 8 (3): 204–210.
81. Milton, J., P. Kennedy, and H. Mitchell. 2005. The effect of mutation on the accumulation of information in a genetic algorithm. In *AI 2005: Advances in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, vol. 3809, ed. S. Zhang, and R. Jarvis, 360–368. Berlin: Springer.
82. Mitchell, M., and C.E. Taylor. 1999. Evolutionary computation: an overview. *Annual Review of Ecology and Systematics* 30: 593–616.
83. Moore, M., and A. Narayanan. 1995. Quantum-inspired computing, Department of Computer Science, University Exeter, Exeter, U.K.
84. Narayanan, A., and M. Moore. 1996. Quantum-inspired genetic algorithms. In *Proceedings of IEEE International Conference on Evolutionary Computation*, 61–66.
85. Neri, F., and V. Tirronen. 2008. On memetic differential evolution frameworks: a study of advantages and limitations in hybridization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2135–2142.
86. Neri, F., and V. Tirronen. 2010. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review* 33 (1): 61–106.
87. Neshat, M., G. Sepidnam, M. Sargolzaei, and A.N. Toosi. 2014. Artificial fish swarm algorithm: a survey of the state-of-the-art, hybridization, combinatorial and indicative applications. *Artificial Intelligence Review* 42 (4): 965–997.
88. Nielsen, A.M., and I.L. Chuang. 2000. *Quantum computation and quantum information*. Cambridge: Cambridge University Press.
89. Okabe, T. 2007. Theoretical analysis of selection operator in genetic algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 4676–4683.
90. Olsson, A. 2011. *Particle swarm optimization: theory, techniques and applications, engineering tools, techniques and tables*. Nova Science Publishers, Incorporated.
91. Omran, M.G.H., A.P. Engelbrecht, and A. Salman. 2009. Bare bones differential evolution. *European Journal of Operational Research* 196 (1): 128–139.
92. Osaba, E., R. Carballedo, F. Diaz, E. Onieva, I. de la Iglesia, and A. Perallos. 2014. Crossover versus mutation: a comparative analysis of the evolutionary strategy of genetic algorithms applied to combinatorial optimization problems. *The Scientific World Journal* 2014. Article ID 154676, 22 p.
93. Passino, K.M. 2002. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine* 22 (3): 52–67.
94. Pelikan, M., D.E. Goldberg, and F.G. Lobo. 2002. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications* 21 (1): 5–20.
95. Pilato, C., D. Loiacono, F. Ferrandi, P.L. Lanzi, and D. Sciuto. 2008. High-level synthesis with multi-objective genetic algorithm: a comparative encoding analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 3334–3341.
96. Poli, R., J. Kennedy, and T. Blackwell. 2007. Particle swarm optimization-an overview. *Swarm Intelligence* 1 (1): 33–57.
97. Price, K. 1994. Genetic annealing. *Dr. Dobb's Journal* 127–132.
98. Price, K., R.M. Storn, and J.A. Lampinen. 2005. *Differential evolution: a practical approach to global optimization (Natural Computing Series)*. New York: Springer.

99. Qin, A.K., V.L. Huang, and P.N. Suganthan. 2009. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation* 13 (2): 398–417.
100. Rao, R.V., V.J. Savsani, and D.P. Vakharia. 2011. Teaching learning-based optimization: a novel method for constrained mechanical design optimization problems. *Computer Aided Design* 43 (3): 303–315.
101. Ratnaweera, A., S.K. Halgamuge, and H.C. Watson. 2004. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation* 8 (3): 240–255.
102. Rechenberg, I. 1973. *Evolutionsstrategie: optimierung technischer systemenach prinzipien der biologischen evolution*. Stuttgart: Frommann-Holzboog.
103. Reynolds, R.G. 1994. An Introduction to cultural algorithms. *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, 131–139. World Scientific Publishing.
104. Ronald, S. 1997. Robust encodings in genetic algorithms: a survey of encoding issues. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, 43–48.
105. Rönkkönen, J., S. Kukkonen, and K.V. Price. 2005. Real-parameter optimization with differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 506–513.
106. H. Schwefel, H. 1975. Evolutionsstrategie und numerische optimierung. Ph.D. dissertation, Technische Berlin, Germany.
107. Schwefel, H. (ed.). 1995. *Evolution and optimum seeking*. New York: A Wiley-Interscience publication.
108. Shi, Y., and R.C. Eberhart. 1998. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, 69–73.
109. Shi, Y., and R.C. Eberhart. 1999. Empirical study of particle swarm optimization. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, 101–106.
110. Simon, D. 2008. Biogeography-based optimization. *IEEE Transactions on Evolutionary Computation* 12 (6): 702–713.
111. Simon, D. 2013. *Evolutionary optimization algorithms: biologically-inspired and population-based approaches to computer intelligence*. New York: Wiley.
112. Storn, R., K. Price. 1995. Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces, Technical Report TR-95-012, Berkeley, CA.
113. Storn, R., and K. Price. 1997. Differential evolution-a simple and efficient heuristic for global numerical optimization. *Journal of Global Optimization* 11 (4): 341–359.
114. Stützle, T., and H.H. Hoos. 2000. MAX-MIN ant system. *Future Generation Computer Systems* 16 (8): 889–914.
115. Volná, E. 2013. *Introduction to soft computing*. Bookboon.com.
116. Wang, X., G. Zhang, J. Zhao, H. Rong, F. Ipate, and R. Lefticaru. 2015. A modified membrane-inspired algorithm based on particle swarm optimization for mobile robot path planning. *International Journal of Computers, Communications and Control* 10 (5): 732–745.
117. Watts, D.J., and S.H. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393: 440–442.
118. Yang, X.S. 2008. *Nature-inspired metaheuristic algorithms*. Frome: Luniver Press.
119. Zhang, G. 2011. Quantum-inspired evolutionary algorithms: a survey and empirical study. *Journal of Heuristics* 17: 303–351.
120. Zhang, J., and A. Sanderson. 2009. JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation* 13 (5): 945–958.
121. Zhang, G., J. Cheng, and M. Gheorghe. 2011. A membrane-inspired approximate algorithm for traveling salesman problems. *Romanian Journal of Information Science and Technology* 14 (1): 3–19.
122. Zhang, G., M. Gheorghe, L. Pan, and M.J. Pérez-Jiménez. 2014. Evolutionary membrane computing: a comprehensive survey and new results. *Information Sciences* 279: 528–551.
123. Zhang, G., F. Zhou, X. Huang, J. Cheng, M. Gheorghe, F. Ipate, and R. Lefticaru. 2012. A novel membrane algorithm based on particle swarm optimization for solving broadcasting problems. *Journal of Universal Computer Science* 18 (13): 1821–1841.

http://www.springer.com/978-3-319-55987-2

Real-life Applications with Membrane Computing

Zhang, G.; Pérez-Jiménez, M.J.; Gheorghe, M.

2017, XII, 355 p. 148 illus., 54 illus. in color., Hardcover
ISBN: 978-3-319-55987-2