I

PRELIMINARIES

COPYRIGHTED MATERIAL

1

A BRIEF INTRODUCTION TO EVOLUTIONARY AND OTHER NATURE-INSPIRED ALGORITHMS

Nasimul Noman

School of Electrical Engineering and Computer Science, Faculty of Engineering and Built Environment, The University of Newcastle, New South Wales, Australia
and
The Priority Research Centre for Bioinformatics, Biomarker Discovery and Information-Based Medicine, The University of Newcastle, New South Wales, Australia

Hitoshi Iba

Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo, Japan

1.1 INTRODUCTION

When we look at nature, everything seems to be working very systematically. All natural phenomena, ranging from molecular level to ecological level, and from individual level to population level, are functioning effectively. The flawless operation of various natural systems becomes possible due to some underlying governing rules.

From the beginning of human history, people have borrowed ideas and mimicked different natural processes in solving their daily-life problems. With the progress of

Evolutionary Computation in Gene Regulatory Network Research, First Edition.
Edited by Hitoshi Iba and Nasimul Noman.
© 2016 John Wiley & Sons, Inc. Published 2016 by John Wiley & Sons, Inc.

civilization, we started to analyze and understand the basic laws and fundamental mechanisms behind natural phenomena and imitate those in designing artificial systems. With the beginning of information era, researchers started to investigate these natural processes from the perspective of information processing. We started to mimic how information is stored, processed, and transferred in natural systems in developing new techniques for solving complex problems. Today, a broad field of research is involved in the design, development, and study of intelligent computational systems that are inspired by the mechanisms and principles (often highly simplified versions of those) observed in various natural processes.

Perhaps, the largest natural information processing system that we have studied most widely and understand reasonably is evolution. Evolution refers to the scientific theory that explains how biological hierarchy of DNA, cells, individuals, and populations slowly change over time and give rise to the fantastic diversity that we see around us. Through the evolutionary process, the changes taking place in an organism's genotypes give rise to optimized phenotypic behaviors. Therefore, evolution can be considered as a process capable of finding optimized, albeit not optimal, solutions for problems.

Evolutionary computation (EC) is a branch of computer science, dedicated to the study and development of search and optimization techniques which draw inspiration from Darwinian theory of evolution and molecular genetics. The incremental growth of the field resulted in algorithms with different flavors although all of them utilize the *in silico* simulation of natural evolution. Classically, the most prominent types of evolutionary computation are genetic algorithms (GA), genetic programming (GP), Evolutionary Strategy (ES) and Evolutionary Programming (EP). Although, at the beginning, each class of algorithms had their distinct characteristics, lately, because of hybridization and concept borrowing, it is difficult to categorize some new algorithms as a specific class of EC.

After natural evolution, the artificial intelligence community has been heavily influenced by the social behavior emerged, through information processing and sharing, among relatively simpler life forms. Social insects like ants, termites and bees exhibit remarkable intelligence in improving their way of life, for example, retrieval of food, reducing the threat of predator, division of labour, or nest building. They possess impressive problem-solving capabilities through collaboration and cooperation among fellow members which themselves have very limited intelligence. Many computational algorithms and problem-solving techniques, commonly known as swarm intelligence, have been developed by simulating the coordination and teamwork strategies in social insects.

Other than evolutionary computation and swarm intelligence, many other computational algorithms have been proposed which are inspired by different natural phenomenon such as immune systems of vertebrate, biological nervous systems, chemical systems, or the behavior of different animals such as bat, firefly, and cuckoo. There exist a lot of variation and differences among these algorithms in terms of problem representation and solution searching mechanism; however, the common connection among them is that all of these algorithms extract metaphor and inspiration from nature. These classes of algorithms are commonly known as nature-inspired

algorithms or bio-inspired algorithms. In this book, we will mostly focus on evolutionary computation and a few other swarm and nature-inspired algorithms; therefore, we will commonly refer to them as evolutionary computation.

Because of their robust and reliable search performance, these algorithms are preferred for solving many complex problems where traditional computational approaches are found to be inadequate. Gene regulatory networks (GRNs) are complex, nonlinear systems with incomplete understanding of their underlying mechanism at molecular level. Consequently, evolutionary and other nature-inspired algorithms are preferred as the computational approach in different research in GRN which is the topic of this book. Therefore, in this first chapter, we present a gentle introduction of evolutionary and other nature-inspired computation so that the readers can have a better understanding of the more advanced versions of these algorithms presented in subsequent chapters. After the generalized introduction, we also discuss relative advantages/disadvantages and application areas of these algorithms.

1.2 CLASSES OF EVOLUTIONARY COMPUTATION

1.2.1 Genetic Algorithms

Genetic algorithms, which are typical examples of evolutionary computation, have the following characteristics:

- Work with a population of solutions in parallel
- Express candidate solutions to a problem as a string of characters
- Use mutation and crossover to generate next-generation solutions

Elements comprising GAs are data representation (genotypes or phenotypes), selection, crossover, mutation, and alternation of generations. How to implement these elements is a significant issue that determines the search performance. Each element is explained below.

1.2.1.1 Data Representation Data structures in GAs are genotypes (GTYPE) and phenotypes (PTYPE). GTYPE corresponds to genes of organisms, and indicates strings expressing candidate solutions (bit strings with fixed length). Genetic operators, such as crossover and mutation which are discussed later, operate on GTYPE. The implementer can determine how to convert candidate solutions to strings. For instance, GTYPE may be a candidate solution converted into an array of concatenated integers.

On the other hand, PTYPE corresponds to individual organisms, and indicates candidate solutions to a problem based on interpretation of GTYPE. The fitness value that indicates the quality of a candidate solution is calculated using PTYPE.

1.2.1.2 Selection In GAs, individuals that adapt better to the environment leave many children and others are eliminated in line with Darwinian evolution theory.

JWST651-c01 JWST651-Iba Printer: Yet to Come December 29, 2015 15:21 Trim: 6.125in × 9.25in

Individuals that adapt to the environment are candidate solutions that score highly regarding the problem, and the fitness function determines the score. Various methods of selecting parent individuals that generate children comprising the next generation have been proposed. Among these, the roulette selection (each individual generates children with a probability proportional to its fitness value) and the tournament selection (a number of individuals are selected at random and the best individual is chosen as the parent, and this procedure is repeated as necessary) are frequently used.

The elite strategy (best individual always remains in the next generation) is often used in addition to these selection methods. This strategy does not reduce the fitness value of the best individual in subsequent generations (as long as the environment to be evaluated does not change). However, using the elite strategy too much in the initial stages of a search may lead to premature convergence, which means convergence to a local solution.

1.2.1.3 Crossover Crossover is an analogy of sexual reproduction, and is an operation that mates two parent individuals to generate new children. There are a number of crossover methods with different granularity when splitting individuals; examples are one-point crossover and uniform crossover.

One-point crossover selects a crossover point at random and switches parts of two parent individuals at this crossover point for generating children. Figure 1.1 is an example of a one-point crossover. The point between bits 3 and 4 is chosen as the crossover point, and two children are generated. Two-point crossover, where two crossover points are chosen and two switches are made, and multiple-point crossover with three or more crossover points are also possible.

Uniform crossover is the most refined crossover method where the parent value to inherit is determined for each bit. Hitchhiking is a problematic phenomenon regarding

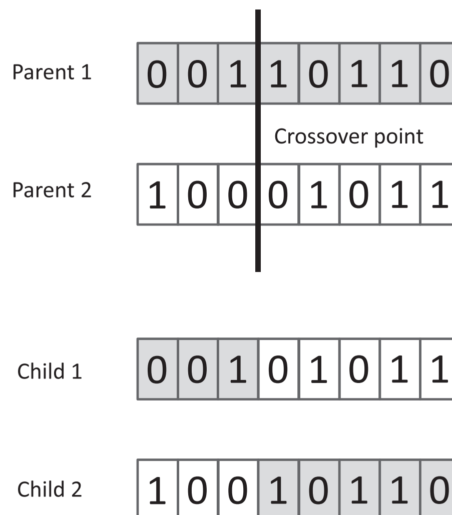

Figure 1.1 One-point crossover in a genetic algorithm.

JWST651-c01 JWST651-Iba Printer: Yet to Come December 29, 2015 15:21 Trim: 6.125in × 9.25in

Figure 1.2 Mutation in a genetic algorithm.

crossover in GAs, in which unnecessary bits existing around a good partial solution spread as parasites to the good partial solution regardless of whether the fitness value is good or not. In general, uniform crossover is considered to suppress hitchhiking.

1.2.1.4 Mutation Mutation corresponds to errors in gene reproduction in nature. In GAs, this operation changes one character in an individual after crossover (in a bit sequence, switches between 0 and 1). Figure 1.2 is one example. Crossover can, in principle, only search for combinations of existing solutions. Therefore, mutation is expected to increase the diversity of the population and broaden the search space by breaking part of a genotype. The reciprocal of the GTYPE length is often used as the mutation rate, which means that on average there is one mutation per genotype. Increasing the mutation rate diversifies the population, but the tradeoff is that there is a higher probability of destroying good partial solutions.

1.2.1.5 Algorithm Flow Summarizing the above, the flow in a GA is as follows.

1. Randomly generate strings (GTYPE) of the initial population.
2. Convert GTYPE to PTYPE and calculate the fitness value for all individuals.
3. Select parents based on the selection method.
4. Generate individuals of the next generation (children) using genetic operators.
5. Check termination conditions; return to 2 if termination conditions are not met.

Generation alternation is a procedure where children generated by operations such as selection, crossover, and mutation replace parent individuals to create the population of the next generation. Typical termination conditions are discovery of an individual with sufficient fitness value or iterating the algorithm for a predetermined number of generations. Instead, one may continue calculations for as long as possible while calculation resources exist, and finish when sufficient convergence is achieved or further improvement of the fitness value is not expected.

1.2.1.6 Extension of GA GTYPE has been explained as a string of fixed length, but improved GAs without this restriction have been proposed. Examples are real coded GA (a vector of real numbers is used as the genotype, see Section 1.2.1.7) and MessyGA where variable length strings are supported by pairing the position in the gene and its value. Genetic programming supporting tree structures, which is

explained in the next section, is one example of a variable length GA. Interactive genetic calculation (the user provides the fitness value to simulate breeding, which can be used when applying GAs to fields such as design and art where an objective function cannot be explicitly described) and multi-objective optimization (multiple objective functions are optimized simultaneously; see Section 1.2.6) have also been proposed and are known to be very effective when designing desirable targets.

1.2.1.7 Real Coded GA Function optimization, where a function is optimized in a continuous search space, is one of the important problems that frequently show up in real-world problems. Research on evolutionary computations for function optimization has a long history. Proposed methods are the bit-string GA where gene expressions are binary code or gray code, real coded GA where vectors of real numbers are used as gene expressions, evolution strategy (ES, see Section 1.2.3), differential evolution (DE, see Section 1.2.4), and meta evolutionary programming (meta-EP). This section describes crossover methods and generation alternation models for real coded GA that show good performance among evolutionary computation methods for function optimization.

Function optimization is a problem to find a set of $(x_1, \cdots, x_n)$ that minimizes or maximizes a function $f(x_1, \cdots, x_n)$ consisting of n continuous variables. Intuitively, this is a problem to find the highest or lowest point of the target function. Minimization problems are considered hereafter as these do not lose generality. A unimodal function has only one local solution that is also the global optimum solution in the search space, whereas a multimodal function has many local solutions. Generally speaking, multimodal functions are more difficult to optimize. When considering a function geometrically, there is "dependence between variables" if there are valleys that are not parallel to the coordinate axis, which means that multiple variables must be changed appropriately at the same time to improve the value of the function. Optimization is usually more difficult if the function has dependence between variables.

Design of methods to generate children, such as crossover and mutation, is the key to good performance when applying evolutionary computation methods to optimization problems. Beyer et al. [3] and Kita et al. [18] proposed guidelines for methods to generate children. Beyer et al.'s design guidelines consider dynamic environments where the form of the function changes with time; however, dependencies between variables are not taken into account. On the other hand, Kita et al.'s guidelines assume a static environment and can reflect dependencies between variables. The crossover design guidelines for real coded GAs by Kita et al. are described below.

Design guideline 1 (Inheritance of statistics): The distribution of children generated by crossover should inherit the average vector and the variance-covariance matrix of the parent distribution. In particular, inheritance of covariance is important in optimizing non-separable functions that have strong dependencies between variables. This means that children generated by crossover should have a similar distribution to that of parents.

Design guideline 2 (Generation of diverse solutions): The crossover procedure should be able to generate a population as diverse as possible within the constraint of "inheritance of statistics."

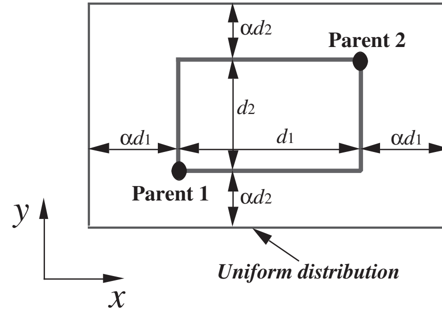

Figure 1.3 Schematic of BLX-α.

Design guideline 3 (Guarantee of robustness): To make the search more robust, the distribution of children should be slightly broader than one that satisfies the design guidelines.

Proposed crossover methods for real coded GA include the blend crossover (BLX-α) by Eshelman et al. [8] and unimodal normal distribution crossover (UNDX) by Ono et al. [21]. BLX-α generates children over a uniform distribution within a hyper-rectangle where each edge determined by parents is parallel to the coordinate axes (Figure 1.3). The algorithm of BLX-α is as follows.

1. Take two parent individuals x^1 and x^2.
2. Each component x_i^c of a child individual x^c is determined independently of each other using a uniform random number within the interval $[X_i^1, X_i^2]$. Here,

$$X_i^1 = \min(x_i^1, x_i^2) - \alpha d_i$$
$$X_i^2 = \max(x_i^1, x_i^2) + \alpha d_i$$
$$d_i = |x_i^1 - x_i^2|$$

Where x_i^1 and x_i^2 are the i-th component of x^1 and x^2, respectively, and α is a parameter.

On the other hand, UNDX generates children on or near a line connecting two parents using a normal distribution determined by these parents and a third parent. The UNDX algorithm is as follows.

1. Select three parents x^1, x^2, and x^3.
2. Find the center of parents x^1 and x^2, that is, $x^p = (x^1 + x^2)/2$.
3. Define the difference vector of parents x^1 and x^2 as $d = x^1 - x^2$.
4. The primary search line is defined as the line connecting parents x^1 and x^2, and the distance between parent x^3 and the primary search line is denoted as D.

JWST651-c01 JWST651-Iba Printer: Yet to Come December 29, 2015 15:21 Trim: 6.125in × 9.25in

5. Child x^c is generated using the formula

$$
\begin{aligned}
x^c &= x^p + \xi d + \sum_{i=1}^{n-1} \eta_i D e_i, \\
\xi &\sim N(0, \sigma_\xi^2), \quad \eta_i \sim N(0, \sigma_\eta^2).
\end{aligned}
\tag{1.1}
$$

   Here, n is the dimension of the search space, $N(0, \sigma^2)$ is a normal distribution with average 0 and variance σ^2, and e_i's are orthonormal basis vectors of the subspace normal to the primary search line.

System parameters of each crossover method can be determined to satisfy the above design guideline 1 (inheritance of statistics).

Crossover methods for real coded GAs can be combined with various selection methods. Generation alternation models for a single objective optimization using a single evaluation function include simple GA (SGA) by Goldberg [10], iterated genetic search (IGS) by Ackley [10], steady state (SS) by Syswerda [31] and elitist recombination (ER) by Thierens et al. [32]. Many engineering problems are formulated as multi-objective optimization problems that explicitly handle various evaluation functions in tradeoff relations (see Section 1.2.6). Combination with a generation alternation model that retains a high level of diversity is desirable for maximum crossover performance in real coded GAs for both single objective and multi-objective optimization.

Finally, evolution strategy, which is closely related to real coded GAs in the sense that real number vectors are used as gene expressions, is discussed. ES uses mutation as the main search operator in contrast to real coded GAs that instead use crossover. ES generates children based on a normal distribution around parent individuals, which is similar to some real coded GAs such as UNDX, UNDX-*m*, and extended normal distribution crossover (ENDX). However, ES codes evolution parameters, such as the standard deviation of the normal distribution, into the individual along with the decision variable to be optimized. The region where children are generated is adaptively derived through adaptation of the parameters by mutation. Correlated mutation proposed by Schwefel [27] uses a similar mechanism that considers dependencies between variables and tilts the axis of the normal distribution relative to the coordinate axes.

1.2.2 Genetic Programming

Genetic programming is an evolutionary computation method applicable to many problems and uses tree structures as the genotype. Programming languages such as LISP, relations between concepts, and many knowledge representations including mathematical expressions can be described using tree structures. As a result, GP can be used to apply evolutionary approaches to automatic code generation and problem solving by artificial intelligence. The basic idea of GP was originally proposed by John Koza et al. [19]. The main difference between GP and GAs is the expression of

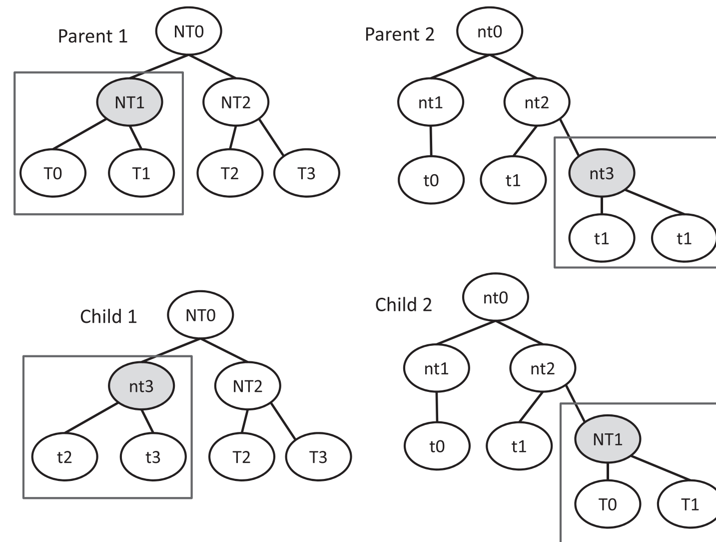

Figure 1.4 Crossover in genetic programming.

GTYPE and operator implementation; selection methods and generation alternation is the same. Data representation and genetic operators unique to GP are described below.

1.2.2.1 Data Representation GP generally expresses GTYPE, which are candidate solutions to a problem, as tree structures. Each node can be categorized into terminal symbols without arguments (corresponding to constants and variables) and nonterminal symbols with arguments (corresponding to functions). Design of GTYPE is carried out by defining usable symbols. As in GAs, the fitness value is obtained by converting each individual into PTYPE (for instance, the results after running code or the evaluated value of a mathematical expression).

1.2.2.2 Crossover Crossover in GP exchanges partial trees between two individuals. The node that would be the crossover point is selected at random in each individual, and partial trees beyond that node are exchanged to generate child individuals. Figure 1.4 is an example of a crossover. NT1 of parent 1 and nt3 of parent 2 are selected as crossover points, and children 1 and 2 are generated by exchanging partial trees beyond these points. However, repeating such simple crossover can lead to unnecessary expansion of tree size as the number of generations increases. This phenomenon is called “bloat” or “fluff,” which means to become “structurally complex.” The bloat is one factor that inhibits effective search using GP (see Section 1.2.2.4 for details).

1.2.2.3 Mutation Mutation in GP corresponds to replacement of one node by a randomly generated partial tree. Figure 1.5 shows an example of mutation. The

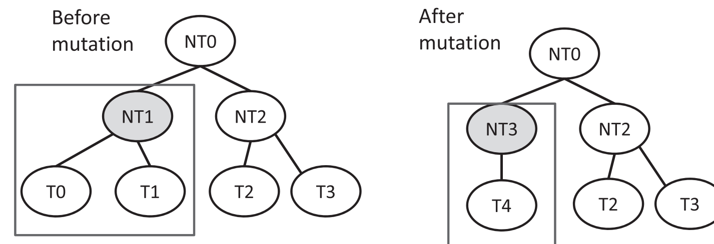

Figure 1.5 Mutation in genetic programming.

effect of mutation in GP is significantly influenced by the node undergoing mutation, thus care is necessary when selecting the node. Examples of mutation are changing a terminal symbol into another terminal symbol, replacing a nonterminal symbol with another nonterminal symbol with the same arguments, changing one nonterminal symbol into a terminal symbol (remove a partial tree), switching nodes in a GTYPE (inversion), and inserting or deleting a terminal symbol.

1.2.2.4 Extension of GP As a method to expand GP, the automatically defined function (ADF) that modularizes and reuses functions to streamline processing has been proposed.

Normal GP can only search combinations of nonterminal and terminal symbols; therefore, the size of GTYPE tends to increase in complex systems (the bloat phenomenon as mentioned above). ADF retains two tree structures per individual, that is, the function definition tree (ADF tree) and the evaluation tree (standard GTYPE). Modularization is achieved by reusing subroutine functions defined in the ADF tree within the evaluation tree. The ADF tree contains dedicated nodes that define functions and arguments, and the evaluation tree takes in functions defined in the ADF tree as nonterminal symbols. Crossover is carried out between ADF trees and between evaluation trees.

Bloat is one of the most persistent issues hindering the efficiency of GP searches. It would cause the following problems:

1. The large programs are difficult for people to understand.
2. The programs require much time and memory space to run.
3. Complicated programs tend to be inflexible and difficult to adapt to general cases, so that they are not very robust.

The following approaches are currently being used to control bloat:

1. Set maxima for tree depth and size. Try to avoid creating tree structures exceeding these upper limits by means of crossover, mutation, etc. This is the easiest way to impose such controls, but the user needs to have a good understanding of the problem at hand, and needs heuristics in order to choose the appropriate settings for maxima.

2. Incorporate program size in the fitness value calculations, that is, penalize large programs for being large. This is called "parsimony." More robust assessment standards using MDL (minimum description length) have been proposed (see Ref. [13] for details).
3. Suppress the tree length by adjustments to genetic operators. For instance, Langdon proposed a homologous crossover or a size-fair crossover to control the tree growth [20]. Other methods to suppress bloat include size-dependent crossover (attempts wherever possible to crossover partial trees of similar size) and depth-dependent crossover (bias crossover such that large partial trees are more likely to be exchanged [15]).

1.2.3 Evolution Strategy

Some research groups in Europe (especially in Germany) have been working on concepts similar to GAs for a long time, that is, "evolution strategy". One leader in ES is Ingo Rechenberg [24]. ES in its early days differed from GAs in the following two ways:

1. Mutation is used as the main operator.
2. Real number expressions are handled.

Individuals in ES are expressed as a pair of real number vectors, $(\vec{x}, \vec{\sigma})$. Here, $\vec{x}$ is a position vector in the search space and $\vec{\sigma}$ is a standard deviation vector. Mutation can be expressed as

$$\vec{x}^{t+1} = \vec{x}^{t} + N(\vec{0}, \vec{\sigma}), \tag{1.2}$$

where $N(\vec{0}, \vec{\sigma})$is a random number from a Gaussian distribution of average $\vec{0}$ and standard deviation $\vec{\sigma}$.

ES in its early days carried out search using a population consisting of one individual. A child ($\vec{x}^{t+1}$ in the above equation) generated by mutation can become a member of the new population (become the parent of the next generation) only when its fitness value is better than that of the parent ($\vec{x}^{t}$).

Quantitative research on ES is more feasible than on GAs because the former is not affected by crossover, and the effect of the mutation rate has been mathematically analyzed. For example, theorems regarding convergence have been proven. In addition, the "$\frac{1}{5}$ rule," that is, "let the probability that a mutation succeeds be $\frac{1}{5}$; if this value is larger (smaller) than $\frac{1}{5}$, increase (reduce) $\vec{\sigma}$." In practice, the probability that a mutation succeeds in the last k generations, $\varphi(k)$, is observed and mutation is controlled such that

$$\vec{\sigma}^{t+1} = \begin{cases} c_d \times \vec{\sigma}^{t}, & \text{if } \varphi(k) < 1/5, \\ c_i \times \vec{\sigma}^{t}, & \text{if } \varphi(k) > 1/5, \\ \vec{\sigma}^{t}, & \text{if } \varphi(k) = 1/5. \end{cases} \tag{1.3}$$

JWST651-c01 JWST651-Iba Printer: Yet to Come December 29, 2015 15:21 Trim: 6.125in × 9.25in

In particular, Schwefel adopted $c_d = 0.82$ and $c_i = 1/0.82$. The intuitive meaning of this rule is: "if successful, continue searching with bigger steps; otherwise, reduce the step size".

ES was later extended to be a search method employing a population of multiple individuals. In addition to the mutation operator mentioned above, the crossover operator and the average operator (an operator that takes the average of two parent vectors) were introduced. Unlike GAs, ES uses the following two selection methods.

1. $(\mu + \lambda) - ES$
   A parent population with μ individuals generates λ children. μ individuals are selected from a total of $(\mu + \lambda)$ individuals to be the parents in the next generation.
2. $(\mu, \lambda) - ES$
   A parent population with μ individuals generates λ children ($\mu < \lambda$). μ individuals are selected from λ individuals to be the parents in the next generation.

In general, $(\mu, \lambda) - ES$ is considered to perform better in environments that change with time and in problems with noise.

ES has been applied to many optimization problems, and recently is being applied to problems other than real number problems.

1.2.4 Differential Evolution

Differential evolution [30] is one category of evolutionary computation that derives an approximate solution to optimization problems. DE is known to be effective in providing algorithms for various problems such as nonlinear problems, non-differentiable problems, and multimodal problems.

Individuals in DE are real number vectors (points in search space). The flow of this method is outlined below (see Figures 1.6 and 1.7).

Step 1: Input random numbers in each individual (vector) to generate the initial population. Here, the number of elements in the population is N and each individual is denoted as $\vec{x_i}$ $(i = 0, 1, \cdots, N-1)$.

Step 2: Three individuals are randomly chosen from the solution set and are labeled as $\vec{x_{r1}}, \vec{x_{r2}}, \vec{x_{r3}}$ $(r1, r2, r3 \in \{0, 1, \cdots, N-1\}$ and $r1 \neq r2 \neq r3)$. The individual after mutation, $\vec{v_i}$, is generated as

$$\vec{v_i} = \vec{x_{r1}} + F \times (\vec{x_{r2}} - \vec{x_{r3}}) \qquad (F \text{ is a constant}). \tag{1.4}$$

This is repeated N times to generate N individuals $\vec{v_0}, \cdots, \vec{v_{N-1}}$.

JWST651-c01 JWST651-Iba Printer: Yet to Come December 29, 2015 15:21 Trim: 6.125in × 9.25in

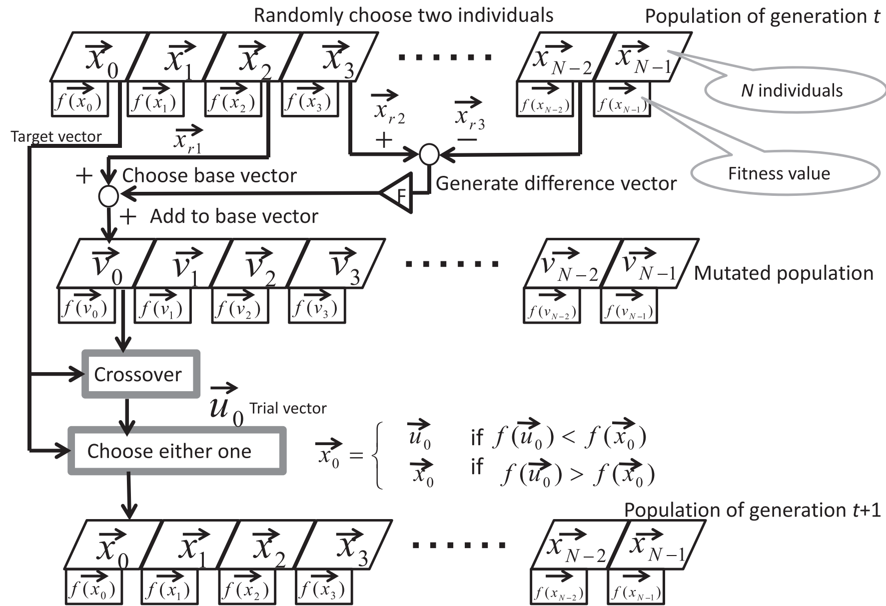

Figure 1.6 Generation alternation in differential evolution (Reprinted with permission from Ref. [23]).

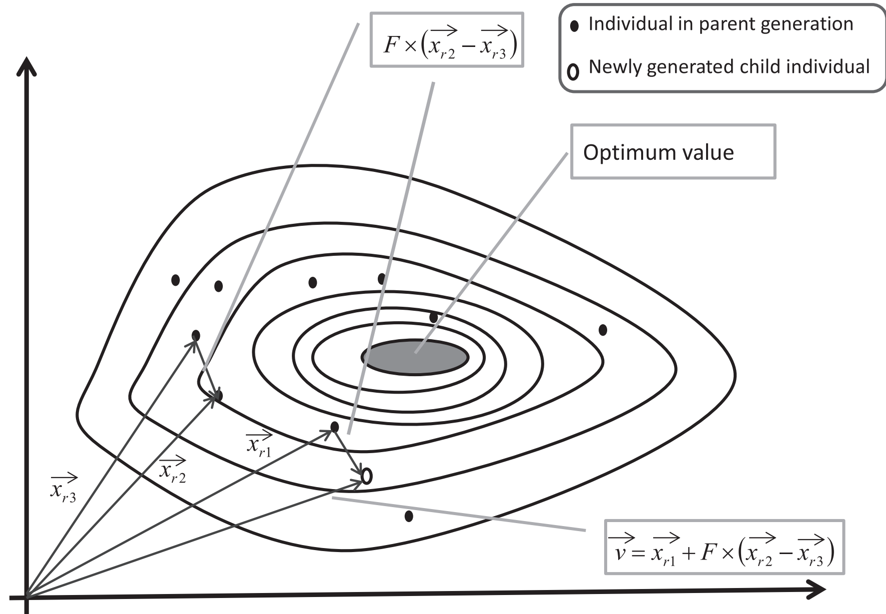

Figure 1.7 Crossover and mutation in differential evolution (Reprinted with permission from Ref. [30]).

Step 3: Generate a child population $\vec{u}_i$ from the parent population $\vec{x}_i$. The elements in $\vec{u}_i$ are selected from elements in $\vec{x}_i$ and $\vec{v}_i$ based on the crossover rate CR.

$$u_{i,j} = \begin{cases} x_{i,j} & \text{if } rand \geq CR, \\ v_{i,j} & \text{if } rand < CR. \end{cases} \tag{1.5}$$

Here, $u_{i,j}, x_{i,j}, v_{i,j}$ are the j-th element of the i-th individual (vector) $\vec{u}_i, \vec{x}_i, \vec{v}_i$, and *rand* is a random number within the interval [0,1]. As a result, the elements in $\vec{u}_i$ contain the elements of both $\vec{x}_i$ and $\vec{v}_i$.

Step 4: Evaluate the child population $\vec{u}_i$ generated in **Step**3 and the parent population $\vec{x}_i$, and decide which solution to adopt.

$$\vec{x}_i = \begin{cases} \vec{x}_i & \text{if } fit(\vec{x}_i) > fit(\vec{u}_i), \\ \vec{u}_i & \text{if } fit(\vec{x}_i) < fit(\vec{u}_i). \end{cases} \tag{1.6}$$

Here, *fit*() is the evaluation function and $fit(x)$ is the evaluated value of x.

Step 5: Repeat **Step**2∼4 for a fixed number of generations, and output the most valuable individual from the final set of solutions as the optimum solution.

Conventional GAs crossover vectors of two individuals and children obtained by crossover are included in the next generation regardless of their fitness values. Mutation occurs at a fixed parameter value (mutation rate); hence, the amount of mutation does not differ between early generations and later generations near convergence.

In contrast, DE crossovers one individual with (one individual + scaled difference vector of two individuals). Crossover involving a difference vector instead of just position vectors of individuals allows a higher possibility of obtaining children in regions with high fitness value. Faster convergence of the population can be attained because a generated child individual is retained only if it is better than its parent individual. Moreover, mutation in DE is based on the difference vector of individuals; thus the amount of mutation changes depending on the population. As a result, the amount of mutation is large in early generations and becomes smaller in generations near convergence. In other words, evolution progresses effectively and setting of mutation parameters is unnecessary for mutation because the amount of mutation is automatically adjusted.

1.2.5 Swarm Intelligence

Many scientists have tried, using various methods, to reproduce collective behavior in groups of ants, birds, and fish on a computer. Reynolds and Heppner, who have simulated the motion of birds, are well known among such scientists. Reynolds was strongly attracted by the beauty of flocks of birds [25] while Heppner was interested in rules hidden in flocks of birds that instantly gather and scatter. These two researchers had the insight to focus on unpredictable motion of birds. The motion is microscopically very simple, resembling that of cellular automata, but

macroscopically is very complex and chaotic. The effect of interactions between individuals has a huge influence in their models as they emphasized the rule that a bird wants to keep an optimum distance between itself and other individuals when considering the overall motion of birds in a flock.

Reynolds' CG animation consists of agents called boids. Each boid determines its motion by combining three vectors, which are (1) the force to move away from the closest neighbor or obstacle, (2) the force to move toward the center of the flock, and (3) the force to move toward the target position. Various patterns of motion can be obtained by adjusting the coefficients used in combining vectors. Complex motion as a whole group emerges when each individual acts based on simple action principles. Technology related to boids is currently widely used for special effects in movies and for animation.

1.2.5.1 Ant Colony Optimization Simple models on the behavior of ants have provided new ideas regarding routing, agents, and distributed control. Applications of ant behavior models have been the focus of many papers and are being established as a research field.

Marching of ants is a cooperative behavior that can be explained by the pheromone trail model. Many cooperative behaviors as a group, such as ant marches, are observed in colonies of ants, and have strongly attracted the interest of entomologists and behavioral scientists. During collecting activities, many types of ants leave a trail of chemical substance when moving from food to their nest, and ants searching for food move along trails that other ants made, if any exists. The chemical substance, which ants generate in their bodies, is called a pheromone.

Ant colony optimization (ACO) is a method that uses the pheromone trail model, for instance, to solve the traveling salesman problem (TSP) [7]. In TSP, there are a number of cities located in different places on a map, and the aim is to look at all of the paths that go through every city exactly once and return to the starting point (called a Hamiltonian cycle or path) and determine the shortest route. There is no efficient algorithm that will solve the traveling salesman problem; in all cases, an exhaustive investigation is required in order to find the optimum solution. Consequently, as the number of cities grows, we see a dramatic leap in the complexity of the problem. This is called a "combinatorial explosion," and is an important issue (an NP-complete problem) in the field of computer science.

ACO optimizes the travel path through the following algorithm:

1. Place ants randomly in each city.
2. Ants move to the next city. The destination is probabilistically determined based on pheromones and given information. Cities already visited are excluded.
3. This procedure is repeated until all cities are visited.
4. Ants completing one loop drop pheromones according to the path length.
5. Return to 1 if a satisfactory solution has not been found.

The length of the path between each city (d_{ij}) and the amount of pheromones on the path are stored in a table, and ants have knowledge about its surroundings. Ants

then probabilistically determine the next city to visit. The probability that an ant k at a city i chooses a city j as the next destination, $p_{ij}^k(t)$, is obtained using the reciprocal of the distance $1/d_{ij}$ and the amount of pheromone $\tau_{ij}(t)$ as follows:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)] \cdot [d_{ij}]^\alpha}{\sum_{h \in J_i^k} [\tau_{ij}(t)] \cdot [d_{ij}]^\alpha}. \tag{1.7}$$

Here, J_i^k is the set of all cities that ant k can move to from city i. The setting that ants are more likely to select paths with more pheromone reflects positive feedback from past searches and incorporates the heuristic that ants are more likely to select shorter paths. As shown above, information unique to each problem can be adequately reflected in ACO.

The pheromone table is updated using the following two equations. Here, $Q(k)$ is the reciprocal of the length of the loop that ant k found.

$$\Delta\tau_{ij}(t) = \sum_{k \in A_{ij}} Q(k) \tag{1.8}$$

$$\tau_{ij}(t+1) = (1-\rho)\dot{\tau}_{ij}(t) + \Delta\tau_{ij}(t). \tag{1.9}$$

The amount of pheromone to be added to each path is inversely proportional to the length of the loop that an ant found. The score of all ants that passed through a path is reflected in the path. Here, A_{ij} is the set of all ants that passed through the path from city i to city j. Negative feedback to avoid local minima is provided as the pheromone evaporation coefficient. In other words, the pheromone in each path evaporated with a fixed probability (ρ), thereby discarding past information.

The ACO has been applied to, and demonstrated to be effective in combination optimization problems such as the TSP and network routing problems.

1.2.5.2 Particle Swarm Optimization Particle swarm optimization (PSO) was introduced by Eberhart and Kennedy in 1995 [17]. The PSO algorithm was inspired by social behavior, and is closely related to code that simulates the collective behavior of birds and fish (for example, of boids by Reynolds). In contrast to GAs that perform genetic operations, PSO decides the next move based on the motion of itself and its neighbors.

The basic PSO proposed by Kennedy et al. consists of many individuals (particles) moving around in a multi-dimensional space and can be applied to real number problems [17]. Each individual remembers its position vector (x_i), velocity vector (v_i), and the position where that individual had its maximum fitness value (p_i). In addition, the position where the group as a whole had its maximum fitness value (p_g) is shared in each individual.

The velocity is updated in each individual based on the best position as a whole and for itself that was found over the generations. The velocity is obtained by

$$v_i = \chi(\omega v_i + \phi_1 \cdot (p_i - x_i) + \phi_2 \cdot (p_g - x_i)).$$

The coefficients used here are the convergence coefficient χ (random number between 0.9 and 1.0) and the decay coefficient ω. In addition, ϕ_1 and ϕ_2 are random numbers equal to or smaller than 2 that are unique to each individual and dimension. The maximum velocity V_{max} is used when the velocity exceeds a given limit. In this way, a search can be performed while keeping individuals in the search space.

The position of each individual is updated in each generation according to the equation

$$x_i = x_i + v_i.$$

Unlike GAs, PSO does not require complex operations such as mutation and crossover, and the structure is very simple. There is theoretical research to derive appropriate values for PSO parameters through mathematical analysis of stability and convergence. PSO is known to give performance comparable to GAs in function optimization properties. Active research is under way for improving the performance of PSO and PSO is being applied to many real-world problems such as power grids and disease diagnosis.

1.2.5.3 Bee Algorithms Bees, together with ants, are well known as social insects. Honey bees can be categorized into three types:

- Employed bees
- Onlooker bees
- Scout bees

Employed bees fly around feeding grounds that they memorize and convey information about food to onlooker bees. Onlooker bees use information from employed bees to selectively find the best food in the feeding ground. When information on a feeding ground becomes too old, employed bees throw away the information, become scout bees, and move to find a new feeding ground. The objective of a beehive is to find the most efficient feeding ground. It is considered that in general about half of the bees in the hive are employed bees, 10–15% are scout bees, and the rest are onlooker bees.

Employed bees waggle-dance (figure-of-eight dance) to convey information to onlooker bees. An employed bee that finds flower nectar or pollen and returns to the nest does a figure-of-eight dance to indicate the direction of the feeding ground to other bees. The direction opposite to gravity corresponds to the direction of the sun and the direction of a straight-line waggle corresponds to the direction of the feeding ground. In other words, bees indicate the angle between the direction of the sun and the direction of the feeding ground to other bees by expressing the angle between the opposite of gravity and the direction of the straight-line waggle. The speed of the waggle represents the distance to the food, and a faster waggle means that the food is nearer. Communication using a similar dance is used to convey the position of a new nest in addition to pollen or the position of water.

JWST651-c01 JWST651-Iba Printer: Yet to Come December 29, 2015 15:21 Trim: 6.125in × 9.25in

Karaboga proposed the artificial bee colony (ABC) optimization algorithm based on the above behavior [16]. The ABC algorithm is a collective search method that mimics food collection by bees. One benefit of the ABC algorithm is the small number of control parameters compared to GAs and the PSO.

The artificial group of bees in the ABC algorithm is separated into employed bees, onlooker bees, and scout bees. N solutions to a problem with d dimensions are generated as feeding grounds. Each employed bee is assigned to a feeding ground $\vec{x}_i$ and finds a new feeding ground $\vec{v}_i$ using the operator

$$v_{ij} = x_{ij} + \text{rand}(-1, 1) \times (x_{ij} - x_{kj}). \tag{1.10}$$

Here, $k \in \{1, 2, \cdots, N\}$, $k \neq i$, and $j \in \{1, 2, \cdots, d\}$ is a randomly chosen index. v_{ij} is the j-th element of vector $\vec{v}_i$. In other words, $\vec{v}_i = (v_{i1}, v_{i2}, v_{i3}, \cdots, v_{id})^{\mathrm{T}}$ and $\vec{x}_i = (x_{i1}, x_{i2}, x_{i3}, \cdots, x_{id})^{\mathrm{T}}$. If the new position is outside the domain, the position is moved to the allowed range. The obtained $\vec{v}_i$ is compared to $\vec{x}_i$, and the better feeding ground is adopted.

In contrast to employed bees, onlooker bees search the feeding ground further using equation (1.10) to select better food. The choosing scheme is based on feedback from employed bees. If a feeding ground cannot be improved for a number of iterations, the feeding ground is abandoned and the bee that was assigned to that feeding ground becomes a scout and reassociates itself with a new feeding ground that is chosen via some principles (in classical ABC it is random initialization).

ABC algorithm is one of the new swarm algorithms that has exhibited very good search performances comparable to many other established algorithms in EC such as DE or PSO.

1.2.5.4 Learning Classifier Systems The classifier system (CS) is a typical example where a GA is applied to machine learning, and has been studied by many researchers such as Holland. Machine learning has two objectives, that is, learning of knowledge in complex systems and generation of appropriate output. A CS uses a GA to enhance and generate rule-based knowledge in achieving these objectives.

Machine learning using a GA is called genetic-based machine learning (GBML). The cognitive system level-1 (CS1) of Holland and Reitman is a famous early example of GBML [11]. Holland and coworkers used this system to learn how to search in maze problems. Smith later developed learning system one (LS1) [28]. LS1 was applied to maze searches and poker game strategy learning, and its effectiveness has been demonstrated.

GBML with its origin in these two systems led to two approaches, that is, the Michigan and Pittsburgh approaches. The difference between these approaches is whether the number of rule sets is just one or more than one. The Michigan approach is based on CS1, where one rule is considered as one individual, and is the main type of CSs.

Machine learning differs from optimization which searches for a solution close to the optimum solution. Instead, machine learning generates new structures and obtains

a coordinated set of rules while incorporating given information. Therefore, GBML must consider the following.

1. New rules are continuously generated, and good rules remain while bad rules are discarded.
2. Good rules generated during the learning process are not destroyed in later learning.
3. The number of rules is not limited, and retaining of all necessary rules is possible.
4. Similar rules are sorted out to generate a rule set with little redundancy.

CSs use the following symbol in learning to achieve a creative rule generation mechanism:

```
If <<condition>> then <<action>>
```

This is the same as the production rule often used in expert systems. The rules in CSs consist of a condition that can be expressed with a string of 0, 1, and # (don't care) and an action that can be expressed with a string of 0 and 1. Here, # is a string that matches both 0 and 1. Providing an external message from the environment to a system that learned data in this format results in simultaneous booting of many rules in the system that gives a corresponding output. In other words, the action part of rules where the input message matches the condition part is executed. The following paragraphs describe the key characteristics of Michigan approach and Pittsburgh approach.

(1) Michigan approach

Each rule, called a classifier (CF), corresponds to one individual in the Michigan approach. The system needs a strengthening functionality that provides a "strength" parameter to each CF in addition to functionality to execute learned CFs. Here, strength is a measure of the reliability of CFs in CSs. Moreover, a functionality to generate new classifiers is necessary, and a GA is used in this generation process. The following is an explanation of the functionalities.

- Execution functionality: Searches for a CF that corresponds to input data (state) from the environment and outputs an action resulting from this CF to the environment. The CF that is selected is determined from the strength obtained using past usefulness.
- Strengthening functionality: Observes changes in the environment caused by CF execution and updates the strength of the CF. If the result is good, the CF is determined to be effective and the strength of the executed CF increases. In contrast, the strength decreases if the result is bad. The system is strengthened to be centered on good CFs by repeating this process. Proposed learning methods to update the strength include the bucket brigade algorithm and the profit- sharing plan.

- Generation functionality: The types of CFs in a system are finite and the influence on the environment is limited. Generation of new CFs is necessary to increase the kinds of action to the environment. However, random generation of CFs would require too much time before a useful CF was generated. Therefore, genetic operations are carried out using a GA where CF is the individual and the strength is the fitness function. New individuals generated by the GA replace individuals in the previous population with low strength or high similarity. CFs are generated after every few steps to reflect the strengthening of CFs based on a combination of many actions to the new CFs.

(2) Pittsburgh approach

The Pittsburgh approach considers one individual as a set of rules that comprise one function, and can consider genetic operations for each rule, which are each regarded as one unit. Consequently, GAs can be implemented more directly and therefore more easily. The following are the characteristics of the Pittsburgh approach.

- Rules are considered as genes; thus crossover means recombination of rules and mutation means conversion into a different rule. As a result, the rule strengthening functionality is unnecessary, and the implemented GA will be natural and easy to handle.
- The gene length (number of rules that make up one individual) is variable, not fixed, because many rules comprise one individual. Therefore, some tricks may be necessary for the gene structure in the GA, such as making the apparent length the same for all genes.
- A large number of rules are necessary to avoid premature convergence. As a result, the number of individuals and the number of rules that each individual contains are very large. Furthermore, evaluation of each individual needs to be carried out for the rule set rather than a simple sum of evaluated values of each rule comprising the individual. Consequently, learning takes time.

Reports of methods that improve CF systems include the zeroth level CS system (ZCS) by Wilson [33], the CS system based on accuracy (XCS) by Wilson [34], and the anticipatory CS system by Stolzmann [29].

1.2.5.5 Artificial Immune System The objective of using algorithms mimicking biological systems, such as neural networks and GAs, in engineering applications is mainly to leverage the adaptability and flexibility of natural systems. The same is true for immune system algorithms that are mainly used to achieve diversity. Details of immune systems in organisms are given in Ref. [22]. This section mainly discusses antibody reactions and outlines the mechanisms that are important for engineering applications.

Foreign bodies such as germs and viruses entering from outside the body have antigens, which are “non-self” markers that do not exist in the body. Immune reactions are caused by detection of antigens. T-cells, which are a type of lymphocyte, identify cells that have been changed by antigens and give commands to B-cells by secreting

interleukin (IL). B-cells are another type of lymphocyte and secrete antibodies that react only to a specific antigen.

The relation between antigens, lymphocytes, and antibodies is called "relation between keys and keyholes." An antigen invading from outside the body selects the lymphocyte that is the closest match. This lymphocyte becomes active and triggers an immune reaction. This mechanism is called clonal selection, and is used, in combination with a GA, to determine the number of individuals in the next generation that is proportional to its fitness value.

Reactions to antigens encountered in the past are memorized, and swift reaction and repression is possible in subsequent invasions. This is known as immunological memory and is used to memorize good solutions in case-based reasoning (CBR) and evolutionary computation methods. Considering diversity at this time allows searching for diverse solutions and suppression of the number of cases.

There is a limit to the number of antibodies and lymphocytes that can exist in the body while the number of possible antigens is infinite. Moreover, attacking of cells comprising the body must be avoided. The immune system enables both identification of self and non-self as well as retention of diversity.

Somatic mutation in antibody genes is a mechanism to improve the fitness value to an antigen by causing abnormally frequent mutations in a portion of an antibody gene. Affinity maturation is a similar mechanism that is incorporated into evolutionary computations in the form of a step to improve the fitness value (identification capability) of a given individual.

Negative selection is the mechanism in which T-cells generated in bone marrow are sent to the thymus, undergo reaction tests against self-derived cells, and those that did not react are selected. Negative selection is applied to detection of computer viruses and anomalies. Here, normal data (packets and logs) are kept as self-data and a population of detectors is obtained by negative selection that does not react to self-data. This procedure can be used to check and detect viruses.

The immune network is a network that assumes identification between antibodies. This explains why antibodies can stay for a long time in a body as immunological memory long after the corresponding antigen is removed from the body and why a diverse set of antibodies can always be retained. This mechanism has one of the largest number of engineering applications, and can be applied to systems that consist of many elements, including multi-agent systems. Typical examples are coordinated control between agents, detection of abnormal processes, and information visualization through active propagations between keywords.

Artificial immune systems (AIS) are a class of computationally intelligent systems inspired by the principles and processes of the above-mentioned vertebrate immune system. AIS has been successfully applied in a number of areas (see Refs. [4, 5] for details).

1.2.6 Multi-Objective EA's

The design of engineering systems must address many needs at the same time, such as enhancement of functionalities and reliability, improvement of user-friendliness, and reduction of manufacturing costs. Multi-objective optimization problems (MOPs)

are characterized by the requirement of optimization of multiple objectives simultaneously. In other words, multiple objective functions $\boldsymbol{f} = (f_1, \cdots, f_m)$ are considered which will be minimized simultaneously as

$$\text{(MOP)} \quad \min_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \cdots, f_m(\boldsymbol{x})) \quad \boldsymbol{x} \in \mathcal{F}.$$

Here, $\boldsymbol{x}$ is the decision variable, which is a vector, and $\mathcal{F}$ is the feasible region. Objective functions typically have tradeoff relations and a decision variable $\boldsymbol{x}$ that minimizes all objective functions does not necessarily exist.

The concept of "dominance" is introduced in multi-objective optimization. For two solutions $\boldsymbol{x}_1$ and $\boldsymbol{x}_2 \in \mathcal{F}$, $\boldsymbol{x}_1$ dominates $\boldsymbol{x}_2$ if $f_k(\boldsymbol{x}_1) \leq f_k(\boldsymbol{x}_2)$ for all $k = 1, \cdots, m$ and $f_k(\boldsymbol{x}_1) < f_k(\boldsymbol{x}_2)$ for at least one $k = 1, \cdots, m$.

A "Pareto optimal solution" or "non-inferior solution" $\boldsymbol{x}$ is a reasonable solution to a MOP that is not dominant over any other solution. In general, multiple Pareto optimal solutions exist and the entire set of such solutions is called the Pareto optimal set. Therefore, the objective of solving a MOP is to obtain the Pareto optimal set or to appropriately sample solutions in the Pareto optimal set.

The multi-objective GA (MOGA) is a well-researched method to search a number of Pareto optimal solutions to a MOP at the same time by leveraging a GA that searches many points using a solution set. Although standard GAs use a single objective function as the standard for selection and elimination to solve optimization problems, MOGAs need to address a number of requirements in searching the Pareto optimum set:

1. Retain solutions closer to the Pareto optimal set while eliminating distant solutions.
2. The solution set should not be concentrated in a part of the Pareto optimal set but instead be spread out as much as possible.
3. New solutions should be efficiently obtained using crossover and selection from the group of solutions in the Pareto optimal set.

The following measures regarding the above issues are taken when designing the algorithm:

1. To search Pareto optimal solutions, selection and elimination are carried out using "dominance" relations within solutions in the solution population. For instance, Goldberg [10] and Fonseca [9] proposed the Pareto ranking method where solutions in the solution population are ranked based on dominance. The vector evaluated GA (VEGA) by Schaffer [26] and the Pareto tournament strategy by Horn et al. [12] are also demonstrated to be effective.
2. Methods to consider the local density of individuals during selection and elimination are included to disperse solutions in the Pareto optimal set. In other words, the number of other solutions near an individual could be evaluated as the solution density and be reflected in selection and elimination.

3. Generally speaking, good solutions from a GA cannot be obtained from a crossover of very different solutions. This becomes very problematic in MOGAs because the solution population is scattered over the Pareto optimal set. Consequently, one might make efforts such as placing crossover mates close to each other. However, there has been little general discussion regarding this point because the design of the crossover operation in a GA depends heavily on the problem.

MOGAs addressing these problems are rapidly being sorted out. Ref. [6] provides examples of how such algorithms are actually designed.

1.3 ADVANTAGES/DISADVANTAGES OF EVOLUTIONARY COMPUTATION

Evolutionary computations offer some unique advantages over the traditional algorithms for searching and optimization. The classical optimization algorithms such as Quasi-Newtons method, conjugate gradient methods, etc. are often iterative algorithms that can be effective in solving linear, quadratic, convex, or unimodal problems. Often, many of these algorithms have additional requirements such as continuity and/or differentiability of the search space for their working principle. Unfortunately, most of the problems in real life are very complex, nonlinear, non-convex, non-separable, and multi-modal. Often, we do not have a very good understanding of the search spaces or rather knowledge of their continuity or differentiability. Therefore, traditional approaches are often not suitable for searching the optimal solution of these problems.

Generally, evolutionary computation can work in poorly understood search problems with limited or almost no specific knowledge about the search space. Usually, by virtue of their parallel search mechanism, these algorithms show superior performances for multi-modal, nonlinear, non-separable, and non-convex search spaces compared to classical algorithms. One big advantage of these algorithms is their scalability—these algorithms can be readily applied to really large dimensional problems. Another advantage of EC over traditional search algorithms is they are capable of delivering multiple competing solutions which is often desirable in real- world problem solving but not possible in case of most of the traditional algorithms that generally utilize single-point search strategy. EC can also exhibit very good performance in optimizing noisy search spaces and can search with imperfect models as well, which makes them valuable in real-life scenarios because real world is noisy and we often have to work with approximate models for many complex systems such as biological systems. Another advantage of evolutionary algorithms is that they can generate multiple tradeoff solutions by optimizing multiple competitive criteria, which is very useful for practical applications. The parallel nature of EC is an inherent advantage for these algorithms in terms of designing computationally efficient methods.

Although evolutionary algorithms have many benefits when it comes to solving critical problems, they are not free of demerits. The main criticism against EC is that

it cannot guarantee finding the optimum solution in a finite amount of time. EC can only guarantee quasi-optimal solution which is particularly useful with very large and complex problems where the optimal solution is unknown. The second shortcoming of these algorithms is that many of these algorithms need to tune various search parameters without proper guidance on how to set them for unknown problems. Today, many adaptive strategies have been incorporated in different algorithms where these parameters can be automatically adjusted online based on the algorithms' search performance. EC is often blamed for utilizing too much exploration; hence it is computationally expensive due to its population-based search approach. However, parallel implementation and other sophisticated approaches like surrogate assistance are used to overcome this limitation. Nevertheless, according to no-free-lunch (NFL) theorem [35], there does not exist any algorithm that has superior performance compared to some other algorithm in solving all optimization problems in general. Therefore, EC cannot be claimed to be superior/inferior to some other algorithm, but they certainly have advantages and limitation over some specific classes of problems.

1.4 APPLICATION AREAS OF EC

Because of their robust and reliable performance in solving complex and odd problems, EC found numerous applications in diverse domains: engineering, science, biology, architecture, arts, music, design, transportation, etc. Almost in every field where we need to solve difficult optimization problems, EC has been successfully used. EC draws researchers' attention through its success in solving different planning problems in the form of routing and scheduling tasks. Different kinds of optimization problems arise in engineering design that ranges from the filter design for digital systems to gearbox or accelerator design for automobiles, to blades, turbine or to engine design for aircrafts. Numerous applications of EC exist in structural engineering, architectural design, environmental engineering, geotechnical and water resource engineering. Today, another broad application area of EC is biological and medical science. In the field of biological sciences, EC is a preferred technique for data analysis, classification, pattern recognition, reverse engineering, and model optimization. In medicine and pharmacology, EC is used for diagnosis, disease data classification, drug design, optimal therapy design for complex disease, etc. In the post-genome era, an increasing surge is observed in analysis and interpretation of the enormous amount of data that is being generated by different studies. EC has also been utilized for solving many problems in finance and economics such as investment planning, market forecasting, etc. Another major application area of EC is control problems where it is applied for fault diagnosis, stability analysis, structure and parameter identification for controllers, etc. Several applications of EC have been observed in robotics which vary from robotic motion planning to automatic learning of cooperation among robots. Besides, EC has found application in other fields as well, such as agriculture, climatology, environmental science and ecology, geo and hydro science, etc.

1.5 CONCLUSION

In this chapter, we have presented a very brief introduction to various algorithms that come under the broad umbrella of evolutionary computation. This introduction is incomplete in every sense —we did not cover many major algorithms in this field such as adaptive evolutionary algorithms, cellular evolutionary algorithms, memetic algorithms (MAs), estimation of distribution algorithms, (EDA) etc.; there are hundreds of variants for each of these algorithms which are not included; parameter setting and relative merits/demerits of each class of algorithms were not discussed. In fact these discussions are not within the purpose and scope of this introduction. The aim of this much generalized introduction is to prepare a background for novice readers to this branch of computation so that they can easily follow the specialized variants of some of these algorithms applied to various GRN research presented in this book.

REFERENCES

1. Ackley, D., “A connectionist machine for genetic hillclimbing,” *The Springer International Series in Engineering and Computer Science*, vol. 28, 1987.
2. Aldana, M., Balleza, E., Kauffman, S., and Resendiz, O., “Robustness and evolvability in genetic regulatory networks,” *Journal of Theoretical Biology*, vol. 245, no. 3, pp. 433–448, 2006.
3. Beyera, H.-G., Schwefela, H.-P., and Wegener, I., “How to analyse evolutionary algorithms,” *Theoretical Computer Science*, vol .287, no. 1, 2002.
4. Castro, de L. N. and Zuben, F. J. V., “Artificial Immune Systems: Part II - A Survey of Applications,” Technical Report DCA-RT 02/00 (2000).
5. Dasgupta, D. (Ed.), *Artificial Immune Systems and Their Applications*, Springer-Verlag, 1999.
6. Deb, K., *Multi-Objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons, 2001.
7. Dorigo, M. and Caro, G. Di, “The ant colony optimization meta-heuristic,” in *New Ideas in Optimization* (Corne, D., Dorigo, M., and Glover, F. (Eds.)), McGraw-Hill, pp. 11–32, 1999.
8. Eshelman, L. J. and Schaffer, J. D., “Real-coded genetic algorithms and interval-schemata,” *Foundations of Genetic Algorithms 2*, pp. 187–202, 1993.
9. Fonseca, C. M. and Fleming, P. J., “Genetic algorithms for multi-objective optimizations,” *Proceedings of International First Conference on Genetic Algorithms and Their Applications* C pp. 93–100, 1985D.
10. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Boston, MA, 1989.
11. Holland, J. H. and Reitman, J. S., “Cognitive systems based on adaptive algorithms,” in *Pattern-Directed Inference Systems* (Waterman, D. A. and Hayes-Roth, F. (Eds.)), New York, Academic Press, pp. 313–329, 1978.

12. Horn, J., Nafpliotis, N., and Goldberg, D. E., "A niched Pareto genetic algorithm for multi-objective optimization," *Proceedings of the First IEEE Conference on Evolutionary Computation* C pp. 82–87, 1994D.
13. Iba, H., deGaris, H., and Sato, T., "Genetic programming using a minimum description length principle," in *Advances in Genetic Programming* (Kinnear, Jr. K., (Ed.)), MIT Press, pp. 265–284, 1994.
14. Iba, H., deGaris, H., and Sato, T., "Numerical approach to genetic programming for system identification," *Evolutionary Computation*, vol. 3, no. 4, pp. 417–452, 1996.
15. Ito, T., Iba, H., and Sato, S., "Depth-dependent crossover for genetic programming," *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation* , pp. 775–780, 1998.
16. Karaboga, D., "An idea based on honey bee swarm for numerical optimization," Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
17. Kennedy, J. and Eberhart, R., "Particle swarm optimization," *Proceedings of IEEE the International Conference on Neural Networks*, 1995.
18. Kita, H., Ono, I., and Kobayashi, S., "Theoretical analysis of the unimodal normal distribution crossover for realcoded genetic algorithms," *Proceedings of ICEC'98*, pp. 529–534, 1998.
19. Koza, J., *Genetic Programming, On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.
20. Langdon, W. B., "Size fair and homologous tree crossovers for tree genetic programming," *Genetic Programming and Evolvable Machines*, vol. 1, no. 1–2, pp. 95–119, 2000.
21. Ono, I. and Kobayashi, S., "A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover," *Proceedings of 7th ICGA*, pp. 246–253, 1997.
22. Perelson, A. S. and Weisbuch, G., "Immunology for physicists," *Reviews of Modern Physics*, vol. 69, no. 4, 1997.
23. Price, K. V., Storn, R. M., and Lampinen, J. A., *Differential Evolution - A Practical Approach to Global Optimization*, Springer, 2005.
24. Rechenberg, I., *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog Verlag, Stuttgart, 1973.
25. Reynolds, C. W., "Flocks, herds and schools: a distributed behavioral model," *Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.
26. Schaffer, J. D., "Multiple objective optimization with vector evaluated genetic algorithms," *Proceedings of First International Conference on Genetic Algorithms and Their Application* C pp. 93–100, 1995.
27. Schwefel, H.-P., *Numerical Optimization of Computer Models*, John Wiley & Sons, Inc., New York, NY, 1981.
28. Smith, S. F., "Flexible learning of problem solving heuristics through adaptive search," *Proceedings eighth International Conference on Artificial Intelligence*, pp. 422–425, 1983.
29. Stolzmann, W., *Antizipative Classifier Systems [Anticipatory Classifier Systems]*, Osnabrueck, Germany: Shaker Verlag, Aachen, Germany, 1997.
30. Storn, R. and Price, K., "Differential Evolution – A simple and efficient heuristic for global optimization over continuous spaces," *Journal Global Optimization*, pp. 341–359, 1997.

31. Syswerda, G., “A study of reproduction in generational and steady-state genetic algorithms,” *Foundations of Genetic Algorithms*, pp. 94–101, 1991.
32. Thierens, D. and Goldberg, D., “Elitist recombination: an integrated selection recombination GA,” *Proceedings of the First IEEE Conference on volutionary Computation*, pp. 508–512, 1994.
33. Wilson, S. W., “A zeroth level classifier system,” *Evolutionary Computation*, vol. 2, no. 1, pp. 1–18, 1994.
34. Wilson, S. W., “Classifier fitness based on accuracy,” *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.
35. Wolpert, D.-H. and Macready, W.-G.,“No free lunch theorems for optimization.”*IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.