Chapter 2
Evolutionary Computation

Abstract The basic approach to optimisation is to formulate a fitness function, which evaluates the performance of the fitness function and improves this performance by choosing from available alternatives. Most classical optimisation methods produce a deterministic sequence of trial solutions using the gradient or higher-order statics of the fitness function. However, such methods may converge to local optimal solutions. The evolutionary computation approach is a population-based optimisation process rooted on the model of organic evolution, which can outperform the classical optimisation methods for many engineering problems. The existing approaches to evolutionary computation include genetic algorithms, evolution strategies, evolutionary programming, genetic programming and so on, which are considerably different in their practical instantiations. The emphasis of this chapter is put on the biological background and basic foundations of genetic algorithm and evolutionary programming. As the principles of particle swarm optimisation are similar to that of evolutionary algorithms, the standard particle swarm optimisation algorithm and an improved particle swarm optimisation algorithm are also presented in this chapter.

2.1 The Evolutionary Algorithms of Computational Intelligence

2.1.1 Objectives of Optimisation

Before investigating the mechanics and power of evolutionary algorithms, which belong to the evolutionary approach of CI, it is necessary to outline the objective of optimising a function or a process, as in this book evolutionary algorithms are

W. H. Tang and Q. H. Wu, *Condition Monitoring and Assessment of Power Transformers Using Computational Intelligence,* Power Systems,
DOI: 10.1007/978-0-85729-052-6_2, © Springer-Verlag London Limited 2011

applied to solve engineering optimisation problems. Mathematical optimisation is the formal title given to the branch of computational science that seeks to answer the question "what is the best?", for problems in which the quality of any answer can be expressed as a numerical value. Such problems arise in all areas of mathematics, the physical, chemical and biological sciences, engineering, architecture and economics, and the range of techniques available to solve them is very wide. In simple words, optimisation concerns the minimisation or maximisation of a function. The conventional view about optimisation is presented well by Beightler, Philips and Wilde [1]:

> Man's longing for perfection finds expression in the theory of optimisation. It studies how to describe and attain what is the Best, once one knows how to measure and alter what is Good or Bad ... An optimisation theory encompasses the quantitative study of optima and methods for finding them.

The objective of an optimisation problem can be formulated as follows: find a combination of parameters (independent variables) which optimise a given quantity, possibly subject to some restrictions on allowed parameter ranges. The quantity to be optimised (maximised or minimised) is termed the objective function; the parameters which may be changed in the quest for the optimum are called control or decision variables; and the restrictions on allowed parameter values are known as constraints.

Generally speaking, an optimisation technique is mostly used to find a set of parameters, $x = [x_1, x_2, \ldots, x_n]$, which can in some way be defined as optimal. In a simple case this might be the minimisation or maximisation of some system characteristics that are dependent on x. In a more advanced formulation an objective function $f(x)$, to be minimised or maximised, might be subject to constraints in the form of equality constraints, inequality constraints and/or parameter bounds. For instance, optimisation of an engineering problem is an improvement of a proposed design that results in the best properties for minimum cost. In more elaborate problems encountered in engineering, there is a property to be made best (optimised) such as the weight or cost of a structure. Then there are constraints, such as the load to be handled, and the strength of steel that is available. Thus, optimisation seeks to improve performance towards some optimal points. There is a clear distinction between the process of improvement and the destination or optimum itself. However, attainment of the optimum is much less important for complex systems. It would be nice to be perfect, meanwhile, we can only strive to improve [1].

Conventionally, the general search and optimisation techniques are classified into three categories: enumerative, deterministic and stochastic (random). Although an enumerative search is deterministic, a distinction is made here as it employs no heuristics [2]. Common examples of each type are shown in Fig. 2.1 [1, 2]:

1. Enumerative schemes are perhaps the simplest search strategy. Within a defined finite search space, each possible solution is evaluated. However, it is easily seen this technique is inefficient or even infeasible when search spaces become large.

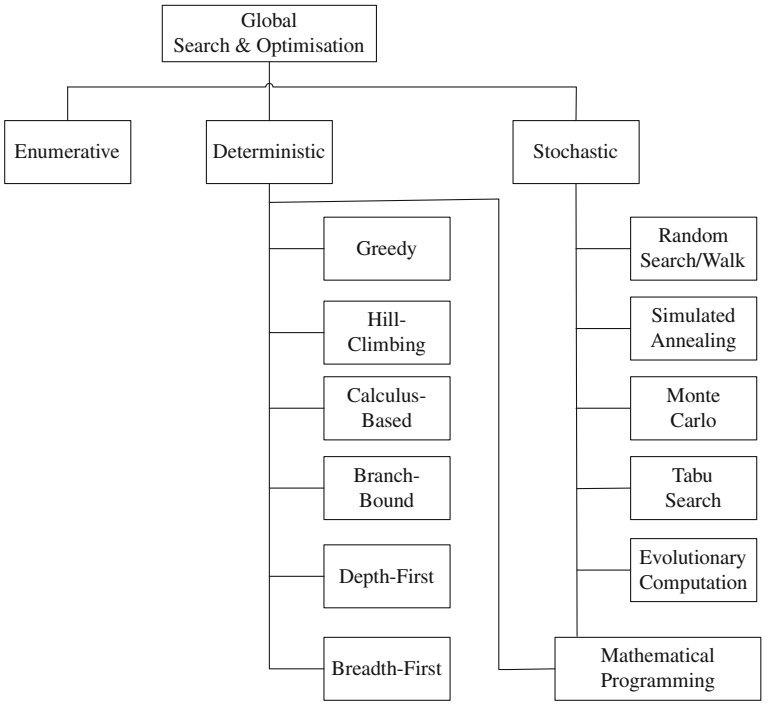

Fig. 2.1 Global search and optimisation techniques

2. Deterministic algorithms attempt to solve the inefficiency by incorporating problem domain knowledge. As many real-world problems are computationally intensive, some means of limiting the search space must be implemented to find acceptable solutions in acceptable time. Many of these are conceived as graph/tree search algorithms, e.g. the hill-climbing and branch-bound algorithms.
3. Random search algorithms have achieved increasing popularity, as researchers have recognised the shortcomings of calculus-based and enumerative schemes. A random search is the simplest stochastic search strategy, as it simply evaluates a given number of randomly selected solutions. A random walk is very similar amongst each other, except that the next solution evaluated is selected randomly using the last evaluated solution as a starting point.

2.1.2 Overview of Evolutionary Computation

Evolutionary computation techniques or evolutionary algorithms (EAs) work on a population of potential solutions in a search space. Through cooperation and competition amongst potential solutions, EAs can find optimal solutions more

quickly when applied to complex optimisation problems. In the last forty years, the growth of interest in heuristic search methods for optimisation has been quite dramatic. The most commonly used population-based EAs are motivated from the evolution of nature. The subject now includes GA [1], GP [3], evolutionary programming [4], evolution strategies [5], and most recently the concept of evolvable hardware [6]. These algorithms stemmed from the very basic description of biological systems and were derived with a simple understanding of genetic evolution, which have shown the capabilities in solving optimisation problems of complex systems. EAs are classified as stochastic search algorithms for global optimisation problems, which have found many engineering and industrial applications [7, 8].

Different from these evolution-motivated evolutionary computation techniques, a recently emerged evolutionary computation technique, namely PSO [9, 10], is motivated from simulations of social behaviours. PSO shares many similarities with evolutionary computation techniques such as GAs, which is initialised with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, potential solutions, called particles, fly through a problem space by following the current optimum particles. In general, compared with GAs, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. Recent studies of PSO indicate that although the standard PSO outperforms other EAs in early iterations, it does not improve the quality of solutions as the number of generations is increased. In [11], passive congregation, a concept from biology, was introduced to improve the search performance of the standard PSO. Simulation results show that this novel hybrid PSO outperforms the standard PSO on multi-model and high-dimensional optimisation problems.

All these related fields of research concerning GA, GP and PSO are often nowadays grouped under the heading of EAs, which is an offshoot of CI. EAs have been considered as general purpose parameter search techniques inspired by natural evolution models, which are appealing to many researchers in engineering. In this book, GA, GP and PSO are employed to identify model parameters and extract fault features for engineering problems. Detailed discussions on GA, GP and PSO are introduced in the following sections.

2.2 Genetic Algorithm

The application of GAs is one of the most important developments in the research field of EAs. GAs are excellent for quickly finding an approximate global maximum or minimum value, which explore a domain space with mutation and derive satisfactory results with selection and crossover. The two major problems using GAs are in converting a problem domain into genes (bit patterns) and creating an effective objective function.

2.2.1 Principles of Genetic Algorithms

GAs originated from the studies of cellular automata, conducted by John Holland and his colleagues at the University of Michigan. Holland's book, published in 1975 [12], is generally acknowledged as the beginning of the research of GAs.

Briefly, GAs require a natural parameter set of an optimisation problem to be coded as a finite-length string (analogous to chromosomes in biological systems) containing characters, features (analogous to genes), taken from some finite-length alphabets. For a binary GA, the binary alphabet that consists of only 0 and 1 is taken. Each feature is represented with different values (alleles) and may be located at different positions (loci). The total package of strings is called a structure or population (or, genotype in biological systems). A summary of the similarities between natural and artificial terminologies of GAs is given in Table 2.1.

A GA is generally recognised as a kind of optimisation method, which is different from the conventional optimisation techniques, e.g. gradients, Hessians and simulated annealing. GAs differ from the conventional optimisation algorithms in four aspects:

1. They work using an encoding scheme of control variables, rather than the variables themselves.
2. They search from one population of solutions to another, rather than from individual to individual.
3. They use only objective function information, not derivatives.
4. They employ probabilistic, not deterministic, rules, which do not require accurate initial estimates.

From the early 1980s the community of GA has experienced an abundance of applications, which spread across a wide range of disciplines. GAs have been applied to solve difficult problems with objective functions that do not possess nice properties such as continuity, differentiability, satisfaction of the Lipschitz Condition, etc. In recent years the furious development of GAs in sciences, engineering and business has lead to successful applications to optimisation problems, e.g. scheduling, data fitting and clustering and trend spotting. Particularly, GAs have been successfully applied to various areas in power systems such as power dispatch [13, 14, 15], reactive power planning [16, 17] and electrical machine design [18, 19].

Table 2.1 Comparison of natural and GA terminologies

Natural	GA
Chromosome	String
Gene	Feature
Allele	Feature value
Locus	String position
Genotype	Population
Phenotype	Alternative solution

2.2.2 Main Procedures of a Simple Genetic Algorithm

The GA used in this book is known as the simple genetic algorithm (SGA). The use of SGA requires the determination of five fundamental issues: chromosome representation, genetic operators making up the reproduction function, the creation of an initial population, termination criteria and an objective function. An SGA manipulates strings and reproduces successive populations using three basic genetic operators: selection, crossover and mutation [12]. The rest of this subsection describes each of these issues.

2.2.2.1 Solution Representation

In neo-Darwinism, we have a population of living organisms, i.e. the phenotype—coded by their deoxyribonucleic acid (DNA) and gene sequence—genotype. The genotype expresses its phenotype which competes in an environment. The competition drives the genotype to evolve a phenotype that performs best in an environment. The chromosomes found in living cells can be described as strings of many thousands of smaller units called alleles. There are only four different kinds of alleles. In the following example, we reduce the number of different kinds of alleles to 2 and the number of alleles in a chromosome to 10. Then, a simulated chromosome in an SGA scheme can be represented by a 10-digit binary number, e.g. 0010100111. The characteristic of an organism is determined by the particular sequence of alleles in its chromosomes. In this example, we parallel this concept by stating that the quality of any proposed binary number as a solution is determined by comparing it with an arbitrary ideal sequence which we are trying to find [20].

As mentioned previously, GAs are computer programs that employ the mechanics of natural selection and natural genetics to evolve solutions for solving an optimisation problem. In GAs there is a population of solutions encoded by a string. The representation of a possible solution as a string is essential to GAs as described in the above paragraph. A set of genes which corresponds to a chromosome in natural genetics is treated as a string in a GA. This algorithm, the most popular format of which is the binary GA, starts by setting an objective function based upon the physical model of a problem to calculate fitness values, and thereafter measures each binary coded string's strength with its fitness value. The stronger strings advance and mate with other stronger strings to produce offsprings. Finally, the best survives.

2.2.2.2 Selection Function

The selection of individuals to produce successive generations plays an extremely important role in GAs. This selection is based on the string fitness according to the

"survival of the fittest" principle. A reproduction operator copies individual strings according to their fitness values. The strings with higher fitness values tend to have a higher probability of contributing one or more offsprings to the next generation. A selection method is required, which chooses individuals in relation to their fitness. It can be deterministic or stochastic, and the roulette wheel selection (RWS) method used in this study is discussed as the following [1, 20].

To reproduce, a simulated weighted roulette wheel is spun as many times as a population size. The selection for an individual, i, is stochastic and proportional to its fitness, f_i. It requires fitness values to be positive numbers, $f_i > 0, \forall i$, as each individual occupies a slice of a pie (hence a biased roulette wheel): f_i is the ith element of the total fitness $-\sum_{i=1}^{N} f_i$, where N is the population size. The probability of individual i to be selected is $P(i) = f_i / \sum_{i=1}^{N} f_i$. A uniformly distributed random number, R, is generated: $R \in U[0, 1]$. If R is between the cumulative probabilities of the ith and $(i + 1)$th individuals, then i is selected. This is repeated for the required number of replacements (usually N) for the next step.

2.2.2.3 Crossover Function

Nature modifies its code by crossing over sections of chromosomes and mutating genes, and GAs borrow this idea for this artificial algorithm. Once two parents have been selected for crossover, a crossover function combines them to create two new offsprings. The crossover operator operates in two steps following reproduction. First, each member in the newly reproduced string group is matched with another at random with a high probability p_c. Secondly, each pair of strings performs crossover with an exchange of each end part of strings at a certain position to generate a pair of new strings. An example of one point crossover is given below [20].

If a simulated weighted coin toss rejects crossover for a pair, then both solutions remain in a population unchanged. However, if it is approved, then two new solutions are created by exchanging all the bits following a randomly selected locus on the strings. For example, if crossover after position 5 is proposed between solutions 1100111010 and 1000110001, the resulting offsprings are 1100110001 and 1000111010, which replace their parents in the population.

2.2.2.4 Mutation Function

The mutation operator flips the code of certain digits of binary coded strings randomly with a small probability. For instance, if every solution in a population has 0 as the value of a particular bit, then a number of crossover operations may produce a solution with a 1 at a particular bit. This process could prevent strings from loss of useful genetic information, which usually results from frequent reproduction and crossover operations. In general, every bit of each solution is

potentially susceptible to mutation. Each bit is subjected to a simulated weighted coin toss with a probability of mutation p_m, which is usually very low (of the order of 0.01 or less). If mutation is approved, the bit changes its value (in the case of binary coding from 1 to 0 or from 0 to 1).

2.2.2.5 Initialisation and Termination Criteria

GAs must be provided with an initial population as indicated previously. The most common method is to randomly generate solutions for the entire population. Since GAs can iteratively improve existing solutions, the initial population can be seeded with potentially good solutions, with the remainder of the population being randomly generated solutions. GAs move from generation to generation selecting and reproducing parents until a termination criterion is met. The most frequently used termination criterion is a specified maximum number of generations. Another termination criterion involves population convergence criteria. In general, GAs force much of an entire population to converge to a single solution. When the sum of deviations amongst individuals becomes smaller than a specified threshold, the algorithm can be terminated. The algorithm can also be terminated due to a lack of improvement in the best solution over a specified number of generations. Alternatively, a target value for an evaluation measure can be established based upon some arbitrarily acceptable thresholds. Moreover, several termination strategies can be employed in conjunction with each other.

2.2.2.6 Fitness Function

For engineering problems, GAs are usually employed to optimise model parameters, so that outputs of a model have a good agreement with reference values, subject to the minimal requirement that a function can map a population into a partially ordered set. A fitness evaluation function is independent of a GA, which depends on a particular problem to be optimised. In a simple term, the fitness function is the driving force behind a GA. A fitness function is called from a GA to determine the fitness of each solution string generated during a search. A fitness function is unique to the optimisation of the problem at hand; therefore, when a GA is used for a different problem, a fitness function must be formulated to determine the fitness of individuals. For many problems, a fitness value is normally determined by the absolute error produced by a GA individual with respect to a given reference value. The closer this error to zero, the better the individual.

Suppose o denotes the desired signal raw and the output raw of a GA individual is p. In general, the fitness can be calculated using an error fitness function or an objective function:

$$f = \frac{1}{n}\sum_{j=1}^{n}|p(j) - o(j)| \quad \text{or} \quad f = \frac{1}{n}\sum_{j=1}^{n}\sqrt{(p(j) - o(j))^2} \tag{2.1}$$

or a squared error fitness function:

$$f = \frac{1}{n}\sum_{j=1}^{n}(p(j) - o(j))^2, \tag{2.2}$$

where n is the number of output target samples.

2.2.3 Implementation of a Simple Genetic Algorithm

The SGA used in this book is implemented in binary coding, and its computation process is listed in Fig. 2.2:

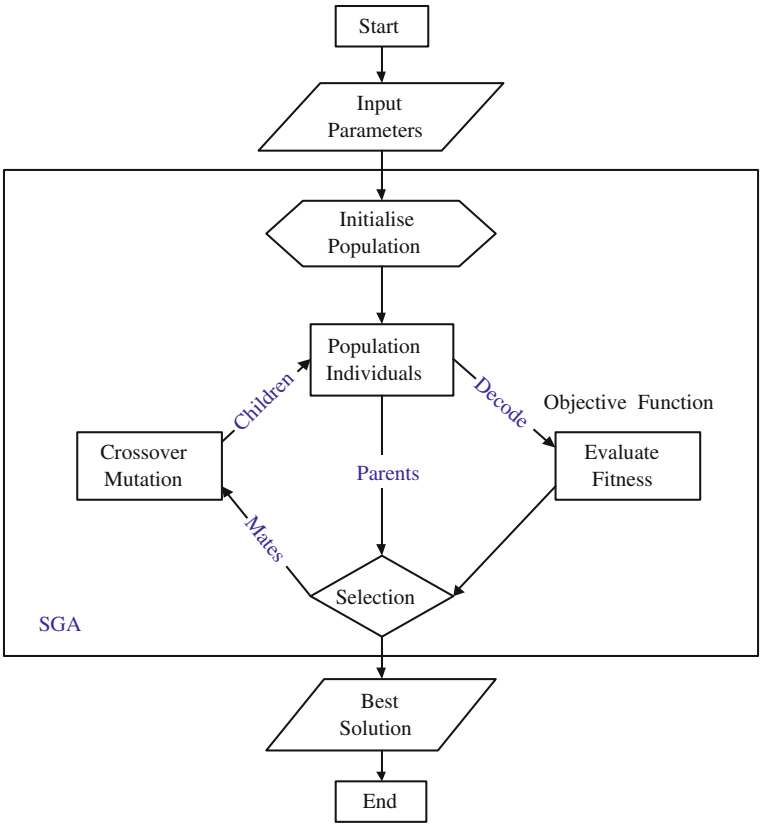

Fig. 2.2 A basic computation process of SGA

1. Initialise a population.
2. Evaluate the fitness of every string in the population.
3. Keep the best string in the population.
4. Make a selection from the population at random.
5. Crossover on selected strings with a probability p_c.
6. Mutation on selected strings with a probability p_m.
7. Evaluate the fitness of every string in a new population.
8. Make elitism.
9. Repeat (4) to (8) until a termination criterion is met.

The crossover probability, p_c, and the mutation probability, p_m, the size of the population and the maximum number of generations are usually selected as "a priori".

2.3 Genetic Programming

2.3.1 Background of Genetic Programming

GP, established by Koza in his fundamental book [3] upon the concepts of GAs, has become one of the most applied techniques in evolutionary computation.

The main difference between GP and GA is the representation of individuals in a population. Whilst GA encodes solution variants into fixed-length strings (chromosomes), GP has no such requirements, since a tree-structured (or hierarchical) representation of GP individuals holds an ability to evolve individual structures during a learning process, i.e. dynamically vary its size, shape and values. GP produces mathematical expressions as solutions. According to Langdon [21]:

> Genetic programming is a technique, which enables computers to solve problems without being explicitly programmed.

A complete GP process is typically a GA and repeats its operation sequence as listed in Sect. 2.2.3 [3, 22]. In order to run GP, several preliminary procedures are required to be undertaken [22]:

1. Determination of terminals and functions.
2. Definition of a fitness function.
3. Choosing GP parameters such as a population size, a maximum individual size, crossover and other probabilities, a selection method and termination criteria (e.g. maximum number of generations).

The population of GP individuals, being constructed as tree-structured expressions, is undergone by a procedure of fitness evaluation, which represents the individual survivability during a selection procedure. Then the fittest individuals, being chosen as parents for performing genetic operations, produce

offsprings constituting new generations of a population. The process continues until a given termination criterion is met or simply a certain generation number is reached. The finally survived individual is treated as a variant of a desired solution. The tree structure of individuals allows GP to vary its size and shape, thereby, achieving a high efficiency in searching of a solution space with respect to what GAs are able to do [3, 22].

2.3.2 Implementation Processes of Genetic Programming

2.3.2.1 Terminals and Functions

GP generates an expression as a composition of functions from a function set and terminals from a terminal set. The choice of functions and terminals, which are collectively referred as nodes, plays an important role in GP since they are the building blocks of GP individuals.

Terminals correspond to the inputs of GP expressions, whether they are constants, variables or zero-argument functions that can be executed. Regarding tree-structured (or hierarchical) representations of individuals, terminals end a branch of a tree. In order to improve GP performance, an ephemeral random constant can also be included as a terminal [3].

Functions are chosen to be appropriate to a problem domain, which may be presented by arithmetic operations, standard mathematical, logical and domain-specific functions, programming functions and statements. In this book, only the mathematical functions listed in Table 2.2 are adopted for feature extraction using GP after numerous GP trials with different sets of functions are utilised.

2.3.2.2 Population Initialisation

The initial generation of a population of GP individuals for later evolution is the first step of a GP process. In general, the size of a newly initialised or reproduced

Table 2.2 A function set for feature extraction using GP

Symbolic name	No. of arguments	Description
Add, Sub	2	Addition, substraction
Mul, Div	2	Multiplication, division
Power	2	Involution
Sqr, Abs	1	Square, absolute value
SqrtAbs	1	Square root of absolute value
Exp, Ln	1	Exponent, natural logarithm
Sin, Cos	1	Sine, cosine
Arctan, Not	1	Arc tangent, invertor

GP individual is bounded by the maximum depth of a tree, i.e. by the maximum total number of nodes in the tree.

The depth of a node is the minimal number of nodes that must be traversed to reach from the root of the tree to the selected node and, correspondingly, the maximum depth is the largest depth being permitted between the root node and the outmost terminals of an individual [22].

In most cases, the initialisation of GP tree structures is implemented using the full or grow methods [3]. The grow method creates an irregular shape tree structure due to random selections of its nodes, whether it is a function or a terminal (except the root node being only a function). Thus, the maximum depth of a tree could not be reached until the terminal node is appeared, concluding the tree branch. As an example in Fig. 2.3a a tree-structured GP individual of a maximum depth of 4, calculating the following expression:

$$a(b - c) + \sin b, \tag{2.3}$$

is presented being initialised with the grow method. The terminals are variables a, b and c, whereas arithmetical functions +, − and *sin* are the functions.

On the other hand, the full method generates tree structures by choosing only functions to build nodes in a tree branch until it reaches a maximum depth. Then only terminals are chosen. As a result, each branch of the tree is of the full maximum depth [22]. For instance, the tree in Fig 2.3b, representing the following expression:

$$ab - (b + c), \tag{2.4}$$

is initialised using the full method with a maximum depth of 3.

The ramped half-and-half method has also been devised in order to enhance the population diversity by combining both the full and grow methods [3]. Given the maximum depth d, a GP population is divided equally amongst individuals to be initialised having maximum depths 2, 3, …, $d - 1$, d. For each depth group, half

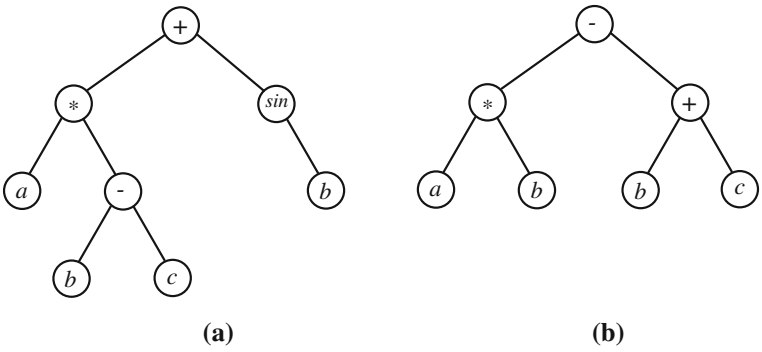

Fig. 2.3 Tree-structured GP expressions

of the individuals are generated using the grow method, and the other half by using the full method.

2.3.2.3 Genetic Operators

In GP the populations of hundreds and thousands of expressions are generally bred using the Darwinian principle of survival and reproduction of the fittest. Similar to GAs, three genetic operators, i.e. crossover, mutation and reproduction, are employed for this breeding, which are appropriate for generating a new offspring population of individual expressions from an initial population.

The crossover operation is used to create new offspring individuals from two parental ones selected by exchanging of the subtrees between parental structures as shown in Fig. 2.4. These offspring individuals are generally with different sizes and shapes to their parents [22, 23]. Mutation is operated on only one individual by replacing a subtree at a randomly selected node of an individual by a randomly generated subtree as shown in Fig. 2.5. The reproduction operation makes a direct copy of the best individual from the parental population and places it into the offspring population [22].

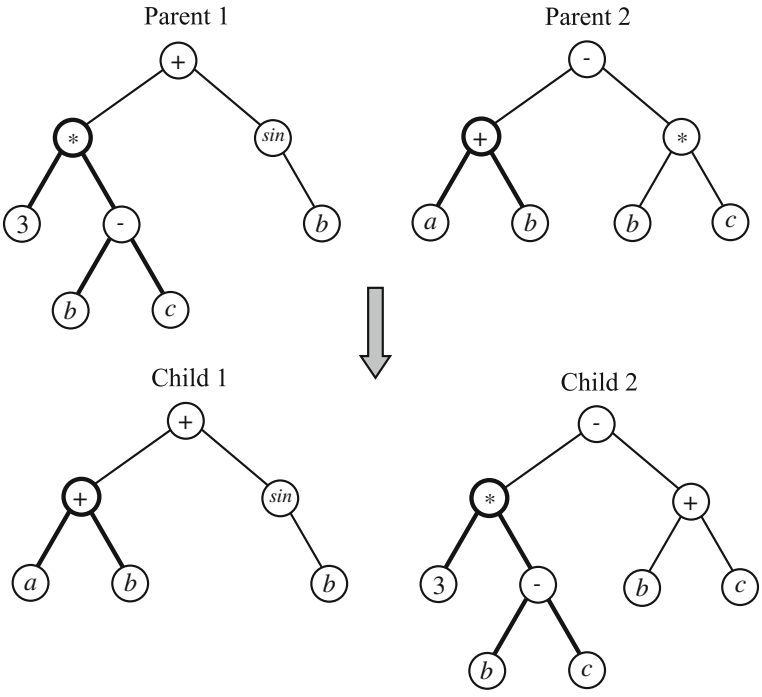

Fig. 2.4 Crossover of tree-structured GP expression

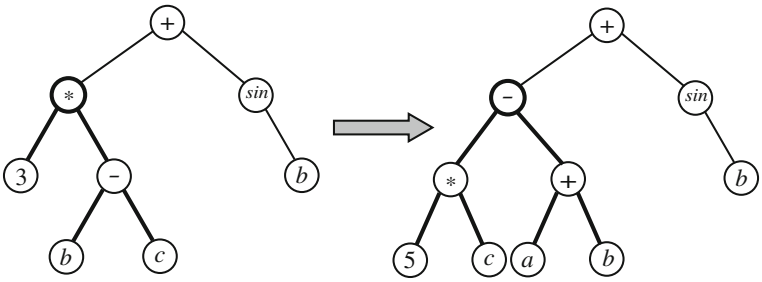

Fig. 2.5 Mutation of tree-structured GP expression

2.3.2.4 Fitness Function

Each individual expression in a population is evaluated in order to quantify how well it performs in a particular problem environment, which is represented by a fitness value. For a two-category classification problem, the Fisher's discriminant ratio (FDR) criterion is usually utilised as a fitness function. FDR is based on the maximisation of between-class scatter over the within-class scatter [24, 25]. Thus, each GP individual is evaluated according to its ability to separate particular classes of data by using the following equation [25]:

$$\text{fitness} = \frac{(\mu_1 - \mu_2)^2}{(\sigma_1^2 + \sigma_2^2)} - pN, \tag{2.5}$$

where μ_1, μ_2 and σ_1^2, σ_2^2 denote the mean values and variances of the two categories to be separated, respectively. p is a small value, e.g. 0.0005, which is introduced as a penalty to the fitness function depending on the number of nodes N of each evaluated individual. This allows a GP program to control the increase in the size of GP individuals and, hence, the production of more simple solutions [26]. Consequently, a GP individual with a larger fitness value is considered to be more accurate in two-category discrimination.

2.3.2.5 Selection Procedure

The selection of individuals to produce successive generations plays an extremely important role in GP. There are various fitness-based selection methods, amongst which the tournament selection is recognised as the mainstream method for a GP selection procedure [22]. The tournament selection operates on subsets of individuals in a population. A randomly chosen number of individuals, defined by the tournament size, form a subset, where a selection competition is performed. Best individuals from the subsets are then passed to the next level, where the competition is repeated. The tournament selection allows to adjust the selection pressure [27], which is an objective measure to the characterise convergence rate of the selection, i.e. the smaller the tournament size, the lower the pressure [22].

2.4 Particle Swarm Optimisation

The standard particle swarm optimiser (SPSO) is a population-based algorithm that was invented by Kennedy and Eberhart [9], which was inspired by the social behaviour of animals such as fish schooling and bird flocking. Similar to other population-based algorithms, such as GAs, SPSO can not only solve a variety of difficult optimisation problems but also has shown a faster convergence rate than other EAs for some problems [10]. Another advantage of SPSO is that it has very few parameters to adjust, which makes it particularly easy to implement.

Angeline [28] pointed out that although SPSO may outperform other EAs in early iterations, its performance may not be competitive as the number of generations is increased. Recently, investigations have been undertaken to improve the performance of SPSO. Løvbjerg et al. [29] presented a hybrid PSO model with breeding and subpopulations. Kennedy and Mendes [30] studied the impacts of population structures to the search performance of SPSO. Other investigations on improving SPSO's performance were undertaken using the cluster analysis [31] and the fuzzy adaptive inertia weight [32]. SPSO has been used to tackle various engineering problems as presented in [33].

The foundation of SPSO is stemmed on the hypothesis that social sharing of information amongst conspecifics offers an evolutionary advantage [9], and the SPSO model is rooted on the following two factors [9]:

1. The autobiographical memory, which remembers the best previous position of each individual (P_i) in a swarm.
2. The publicised knowledge, which is the best solution (P_g) found currently by a population.

Therefore, the sharing of information amongst conspecifics is achieved by employing the publicly available information P_g, shown in Fig. 2.6. There is no information sharing amongst individuals except that P_g broadcasts the information to the other individuals. Therefore, a population may lose diversity and is more

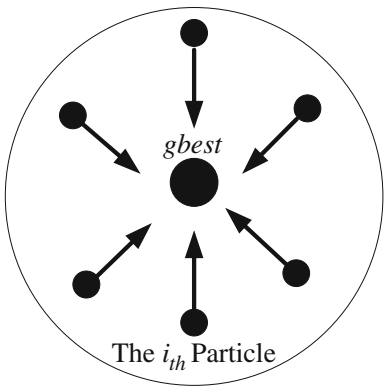

Fig. 2.6 Interaction between particles and the best particle *gbest*

likely to confine the search around local minima if committed too early in the search to the global best found so far.

To overcome this weakness, ideas from biology science have been borrowed to avoid early convergence and biologists have proposed four types of biological mechanisms that allow animals to aggregate into groups: passive aggregation, active aggregation, passive congregation and social congregation [34]. There are different information sharing mechanisms inside these forces. It is found that the passive congregation model is suitable to be incorporated in the SPSO model. Inspired by this observation, a hybrid model of PSO with passive congregation is presented in this book [11].

2.4.1 Standard Particle Swarm Optimisation

The population of SPSO is called a swarm and each individual is called a particle. For the ith particle at iteration k, it has the following two attributes.

1. A current position in an N-dimensional search space $X_i^k = (x_{i,1}^k, \ldots, x_{i,n}^k, \ldots, x_{i,N}^k)$, where $x_{i,n}^k \in [l_n, u_n], 1 \leq n \leq N, l_n$ and u_n are the lower and upper bounds for the nth dimension, respectively.
2. A current velocity V_i^k

$$V_i^k = (v_{i,1}^k, \ldots, v_{i,n}^k, \ldots, v_{i,N}^k)$$

which is clamped to a maximum velocity

$$V_{\max}^k = (v_{\max,1}^k, \ldots, v_{\max,n}^k, \ldots, v_{\max,N}^k).$$

In each iteration, the swarm is updated by the following equations [9]:

$$V_i^{k+1} = \omega V_i^k + c_1 r_1 (P_i^k - X_i^k) + c_2 r_2 (P_{\mathrm{g}}^k - X_i^k) \tag{2.6}$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \tag{2.7}$$

where P_i is the best previous position of the ith particle (also known as *pbest*) and P_g is the global best position amongst all the particles in the swarm (also known as *gbest*). They are given by the following equations:

$$P_i = \begin{cases} P_i & : \quad f(X_i) \geq P_i \\ X_i & : \quad f(X_i) < P_i \end{cases} \tag{2.8}$$

$$P_{\mathrm{g}} \in \{P_0, P_1, \ldots, P_m\} | f(P_{\mathrm{g}}) = \min(f(P_0), f(P_1), \ldots, f(P_m)) \tag{2.9}$$

where f is the objective function, m is the number of particles, r_1 and r_2 are the elements from two uniform random sequence on the interval $[0, 1] : r_1 \sim U(0, 1)$; $r_2 \sim U(0, 1)$ and ω an inertia weight which is typically chosen in the range of [0,1]. A larger inertia weight facilitates the global exploration and a smaller inertia weight tends to facilitate the local exploration to fine-tune the current search area [35]. Therefore, the inertia weight ω is critical for SPSO's convergence behaviour. A suitable value of ω usually provides a balance between global and local exploration abilities and consequently results in a better optimum solution. c_1 and c_2 are acceleration constants, which also control how far a particle moves in a single iteration. The maximum velocity $V_{\max}$ is set to be half of the length of the search space.

2.4.2 Particle Swarm Optimisation with Passive Congregation

It is mentioned that SPSO is inspired by social behaviours such as spatial order, more specially, aggregation such as bird flocking, fish schooling, or swarming of insects. Each of these cases has stable spatio-temporal integrities of a group of organisms: the group moves persistently as a whole without losing the shape and density.

For each of these groups, different biological forces are essential for preserving the group's integrity. Parrish and Hamner [34] proposed mathematical models of the spatial structure of animal groups to show how animals organise themselves. In these models, aggregation sometimes refers to a grouping of the organisms by non-social, external and physical forces. There are two types of aggregation: passive aggregation and active aggregation. Passive aggregation is a passive grouping by physical processes. One example of passive aggregation is the dense aggregation of plankton in open water, in which the plankton are not attracted actively to the aggregation but are transported passively there via physical forces such as water currents. Active aggregation is a grouping by attractive resources, such as food or space, with each member of the group recruited to a specific location actively. Congregation, which is different from aggregation, is a grouping by social forces, which is the source of attraction, in the group itself. Congregation can be classified into passive congregation and social congregation. Passive congregation is an attraction of an individual to other group members but where there is no display of social behaviour. Social congregations usually happen in a group where the members are related (sometimes highly related). A variety of inter-individual behaviours are displayed in social congregations, necessitating active information transfer [34]. For example, ants use antennal contacts to transfer information about an individual identity or a location of resources [36].

From the definitions above, the third part of Eq. 2.6: $c_2 r_2 (P_{\mathrm{g}}^k - X_i^k)$ can be classified as either active aggregation or passive congregation. Since P_{g} is the best solution a swarm has found so far, which can be regarded as the place with most

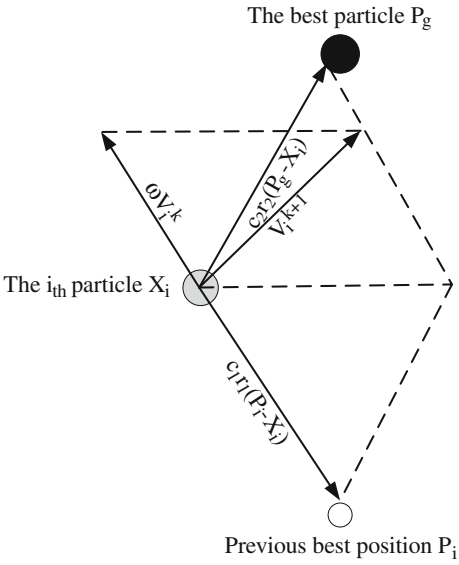

Fig. 2.7 Search direction of the *i*th particle in SPSO

food, we argue that it is better to classify $c_2r_2(P_g^k - X_i^k)$ as active aggregation. From a biology point of view, the sharing of information amongst conspecifics is achieved by employing the publicly available information *gbest*. There is no information sharing amongst individuals except that *gbest* gives out the information to the other individuals. Therefore, for the *i*th particle, the search direction is only affected by 3 factors as shown in Fig. 2.7: the inertia velocity ωV_i^k, the best previous position *pbest*, and the position of global best particle *gbest*. The population is more likely to lose diversity and confine the search around local minima. From our experiment results, the performance of SPSO is not sufficiently good enough for high-dimensional and multi-model optimisation problems.

It has been discovered that in spatially well-defined congregations, such as fish schools, individuals may have low fidelity to a group because the congregations may be composed of individuals with little to no genetic relation to each other [37]. Schooling fish are generally considered as a "selfish herd" [38], in which each individual attempts to take the sweeping generalisation advantage from group living, independent of the fates of neighbours [39]. In these congregations, information may be transferred passively rather than actively [40]. Such asocial types of congregations can be referred to as passive congregation. As SPSO is inspired by fish schooling, it is, therefore, natural to ask if a passive congregation model can be employed to improve the performance of SPSO. Here, we do not consider other models such as passive aggregation, because SPSO is not aggregated passively via physical processes. Furthermore, social congregation usually happens when group fidelity is high, i.e. the chance of each individual meeting any of the others is high [41]. Social congregations frequently display a division of labour. In a social insect colony, such as an ant colony, large tasks are

accomplished collectively by groups of specialised individuals, which is more efficient than performing sequentially by unspecialised individuals [42]. The concept of labour division can be employed by data clustering, sorting [43] and data analysis [44].

Group members in an aggregation can react without direct detection of incoming signals from an environment, because they can get necessary information from their neighbours [34]. Individuals need to monitor both environment and their immediate surroundings, such as the bearing and speed of their neighbours [34]. Therefore, each individual in an aggregation has a multitude of potential information from other group members that may minimise the chance of missed detection and incorrect interpretations [34]. Such information transfer can be employed in the model of passive congregation. Inspired by this perception, and to keep the model simple and uniform with SPSO, a hybrid PSO with passive congregation is proposed:

$$V_i^{k+1} = \omega V_i^k + c_1 r_1 (P_i^k - X_i^k) + c_2 r_2 (P_g^k - X_i^k) + c_3 r_3 (R_i^k - X_i^k) \tag{2.10}$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \tag{2.11}$$

where R_i is a particle randomly selected from the swarm, c_3 is the passive congregation coefficient and r_3 is a uniform random sequence in the range (0,1): $r_3 \sim U(0,1)$. The interactions between individuals of PSOPC are shown in Fig. 2.8, and the pseudo code for implementing PSOPC is illustrated in Table 2.3.

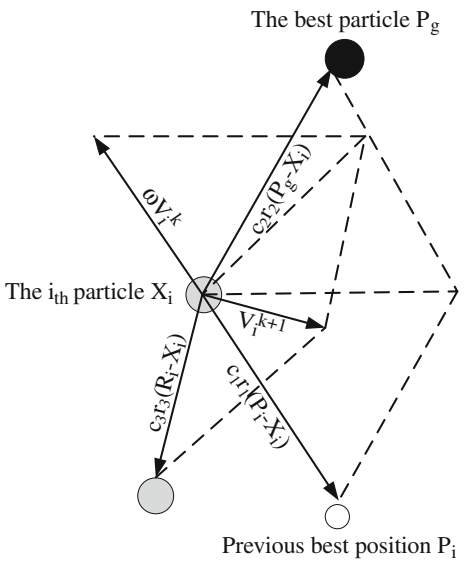

Fig. 2.8 Search direction of the *i*th particle in PSOPC

Table 2.3 Pseudo code for the implementation of PSOPC

Set $k = 0$;	
Randomly initialise positions;	
Randomly initialise velocities;	
WHILE (the termination conditions are not met)	
FOR (each particle i in the swarm)	
Check feasibility:	Check the feasibility of the current particle. If X_i^k is outside the feasible region, then reset X_i to the previous position X_i^{k-1};
Calculate fitness:	Calculate the fitness value $f(X_i)$ of the current particle;
Update *pbest*:	Compare the fitness value of *pbest* with $f(X_i)$. If $f(X_i)$ is better than the fitness value of *pbest*, then set *pbest* to the current position X_i;
Update *gbest*:	Find the global best position of the swarm. If $f(X_i)$ is better than the fitness value of *gbest*, then *gbest* is set to the position of the current particle X_i;
Update R_i:	Randomly select a particle from the swarm as R_i;
Update velocities:	Calculate velocities V_i using Eq. 2.10;
Update positions:	Calculate positions X_i using Eq. 2.11;
END FOR	
Set $k = k + 1$;	
END WHILE	

2.5 Summary

This chapter presents a brief introduction to evolutionary computation and its constitutive algorithms in order to provide a necessary background for the work discussed in later chapters. The basics of three EAs, i.e. GA, GP and PSO, are described, which are employed for identifying model parameters and evaluating fault features. First, the principles of GA are described, as well as the implementation procedures of an SGA. Then, the foundation of GP is presented, including the definition of terminals and functions, genetic operators, population initialisation and selection procedures. Finally, the standard PSO algorithm is introduced, followed by a description of an improved PSO algorithm.

References

1. Goldberg D (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Publishing Company, Inc., USA
2. Coello CAC, Veldhuizen DAV, Lamont GB (2002) Evolutionary algorithms for solving multi-objective problems. Kluwer Academic Publishers, New York
3. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge

4. Fogel LJ (1994) Evolutionary programming in perspective: the top-down view. In: Zurada JM, Marks II RJ, Robinson CJ (eds) Computational intelligence: imitating life. IEEE Press, Piscataway
5. Rechenberg I (1994) Evolution strategy. In: Zurada JM, Marks II RJ, Robinson C (eds) Computational intelligence: imitating life. IEEE Press, Piscataway
6. Reeves CR, Rowe JE (2003) Genetic algorithms—principles and perspectives (a guide to GA theory). Kluwer Academic Publishers, London
7. Miettinen K (1999) Evolutionary algorithms in engineering and computer science. JohnWiley and Sons Inc., UK
8. Bäck T (1996) Evolution algorithms in theory and practice. Oxford University Press, NY
9. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: IEEE international conference on neural networks, vol 4. IEEE Press, NY, pp 1942–1948
10. Kennedy J, Eberhart RC (2001) Swarm intelligence. Morgan Kaufmann Publishers, San Francisco
11. He S, Wu QH, Wen JY, Saunders JR, Paton RC (2004) A particle swarm optimizer with passive congregation. BioSystems 78(1–3):135–147
12. Holland J (1975) Adaptation in natural and artificial systems. University of Michigan Press, USA
13. Wu QH, Cao YJ, Wen JY (1998) Optimal reactive power dispatch using an adaptive genetic algorithm. Int J Electr Power Energy Syst 20(8):563–569
14. Walters DC, Sheble GB (1993) Genetic algorithm solution of economic dispatch with valve point loading. IEEE Trans Power Syst 8(3):1325–1332
15. Wu QH, Ma JT (1995) Power system optimal reactive dispatch using evolutionary programming. IEEE Trans Power Syst 10(3):1243–1249
16. Iba K (1994) Reactive power optimization by genetic algorithm. IEEE Trans Power Syst 9(2):685–692
17. Lee KY, Bai X, Park YM (1995) Optimization method for reactive power planning using a genetic algorithm. IEEE Trans Power Syst 10(4):1843–1850
18. Ma JT, Wu QH (1995) Generator parameter identification using evolutionary programming. Int J Electr Power Energy Syst 17(6):417–423
19. Zhao Y, Edwards RM, Lee KY (1997) Hybrid feedforward and feedback controller design for nuclear steam generators over wide range operation using genetic algorithm. IEEE Trans Energy Conv 12(1):100–106
20. Michalewicz Z (1994) Genetic algorithms + data structures = evolution programs. AI series. Springer-Verlag, New York
21. Langdon WB (1998) Genetic programming and data structures: genetic programming + data structures = automatic programming. Kluwer Academic Publishers, Boston
22. Banzhaf W, Nordin P, Keller RE, Francone FD (1998) Genetic programming-an introduction: on the automatic evolution of computer program and its applications. Morgan Kaufmann Publishers Inc., San Francisco
23. Koza JR (1994) Genetic programming: automatic discovery of reusable programs. MIT Press, Cambridge
24. Guo H, Jack LB, Nandi AK (2005) Feature generation using genetic programming with application to fault classification. IEEE Trans Syst Man Cybernet B: Cybernet 35(1):89–99
25. Theodoridis S, Koutroumbas K (2003) Pattern recognition, 2nd edn. Academic Press, London
26. Zhang L, Jack LB, Nandi AK (2005) Fault detection using genetic programming. Mech Syst Signal Process 19:271–289
27. Bäck T, Fogel DB, Michalewicz Z (2000) Evolutionary computation 1: basic algorithms and operators. Institute of Physics Publishing Ltd., Bristol
28. Angeline P (1998) Evolutionary optimization versus particle swarm optimization: philosophy and performance difference. In: Proceedings of the evolutionary programming conference, San Diago, USA

29. Løvbjerg M, Rasmussen T, Krink K (2001) Hybrid particle swarm optimizer with breeding and subpopulations. In: Proceedings of the third genetic and evolutionary computation conference (GECCO-2001), vol 1. pp 469–476
30. Kennedy J, Mendes R (2002) Population structure and particle swarm performance. In: Proceedings of the 2002 congress on evolutionary computation CEC2002. IEEE Press, pp 1671–1676
31. Kennedy J (2000) Stereotyping: improving particle swarm performance with cluster analysis. In: Proceedings of the IEEE international conference on evolutionary computation. pp 1507–1512
32. Shi Y, Eberhart RC (2001) Fuzzy adaptive particle swarm optimization. In: Proceedings of the IEEE international conference on evolutionary computation. pp 101–106
33. Yoshida H, Kawata K, Fukuyama Y, Takayama S, Nakanishi Y (2000) A particle swarm optimization for reactive power and voltage control considering voltage security assessment. IEEE Trans Power Syst 15(4):1232–1239
34. Parrish JK, Hamner WM (1997) Animal groups in three dimensions. Cambridge University Press, Cambridge
35. Shi Y, Eberhart RC (1998) Parameter selection in particle swarm optimization. Evolutionary programming VII (1998). Lecture notes in computer science, vol 1447. Springer, NY, pp 591–600
36. Gordon DM, Paul RE, Thorpe K (1993) What is the function of encounter pattern in ant colonies? Anim Behav 45:1083–1100
37. Hilborn R (1991) Modelling the stability of fish schools: exchange of individual fish between schools of skipjack tuna (Katsuwonus pelamis). Canad J Fish Aqua Sci 48:1080–1091
38. Hamilton WD (1971) Geometry for the selfish herd. J Theor Biol 31:295–311
39. Pitcher TJ, Parrish JK (1993) Functions of shoaling behaviour in teleosts. Pitcher TJ (ed) Behaviour of teleost fishes. Chapman and Hall, London, pp 363–439
40. Magurran AE, Higham A (1988) Information transfer across fish shoals under predator threat. Ethology 78:153–158
41. Alexander RD (1974) The evolution of social behaviour. Annu Rev Ecol Syst 5:325–383
42. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press, USA
43. Deneubourg JL, Goss S, Franks N, Sendova-Franks A, Detrain C, Chretien L (1991) The dynamics of collective sorting: robot-like ant and ant-like robot. In: Proceedings of the first conference on simulation of adaptive behavior: from animals to animals. pp 356–365
44. Lumer E, Faieta B (1994) Diversity and adaptation in population of clustering ants. In: Proceedings of the third international conference on simulation of adaptive behavior: from animals to animals. pp 499–508

http://www.springer.com/978-0-85729-051-9

Condition Monitoring and Assessment of Power Transformers Using Computational Intelligence

Tang, W.H.; Wu, Q.H.

2011, XVIII, 202 p., Hardcover
ISBN: 978-0-85729-051-9