Chapter 2
Advanced Optimization Techniques

Many difficulties such as multi-modality, dimensionality and differentiability are associated with the optimization of large-scale problems. Traditional techniques such as steepest decent, linear programing and dynamic programing generally fail to solve such large-scale problems especially with nonlinear objective functions. Most of the traditional techniques require gradient information and hence it is not possible to solve non-differentiable functions with the help of such traditional techniques. Moreover, such techniques often fail to solve optimization problems that have many local optima. To overcome these problems, there is a need to develop more powerful optimization techniques and research is going on to find effective optimization techniques since last three decades.

Some of the well-known population-based optimization techniques developed during last three decades are: Genetic Algorithms (GA) [16] which works on the principle of the Darwinian theory of the survival-of-the fittest and the theory of evolution of the living beings; Artificial Immune Algorithms (AIA) [14] which works on the principle of immune system of the human being; Ant Colony Optimization (ACO) [10] which works on the principle of foraging behavior of the ant for the food; Particle Swarm Optimization (PSO) [20] which works on the principle of foraging behavior of the swarm of birds; Differential Evolution (DE) [35] which is similar to GA with specialized crossover and selection method; Harmony Search (HS) [15] which works on the principle of music improvisation in a music player; Bacteria Foraging Optimization (BFO) [27] which works on the principle of behavior of bacteria; Shuffled Frog Leaping (SFL) [12] which works on the principle of communication among the frogs, Artificial Bee Colony (ABC) [18] which works on the principle of foraging behavior of a honey bee; Biogeography-Based Optimization (BBO) [34] which works on the principle of immigration and emigration of the species from one place to the other; Gravitational Search Algorithm (GSA) [29] which works on the principle of gravitational force acting between the bodies and Grenade Explosion Method (GEM) [1] which works on the principle of explosion of

R. V. Rao and V. J. Savsani, *Mechanical Design Optimization Using Advanced Optimization Techniques*, Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-1-4471-2748-2_2, © Springer-Verlag London 2012

grenade. These algorithms have been applied to many engineering optimization problems and proved effective to solve some specific kind of problems.

All the above-mentioned algorithms are nature inspired population-based optimization methods, but they have some limitations in one or the other aspect. Due to this fact, more research is required to test algorithms for different problems to check their suitability for a wide variety of problems. Research is continued to enhance the existing algorithms to improve their performance. Enhancement is done either (a) by modifying the existing algorithms or (b) by hybridizing the existing algorithms. Enhancement due to modifications in the existing algorithms is reported in GA [22, 23, 28], PSO [5, 7, 25, 42], ACO [32, 45], ABC [19, 26], etc. Enhancement can also be done by combining the strengths of different optimization algorithms, known as hybridization of algorithms. Hybridization is an effective way to make the algorithm efficient and it combines the properties of different algorithms. Some of such hybridized algorithms can be found in Hui et al. [17], Wen [39], Ying [43], Yannis and Magdalene [41], Shahla et al. [31], Tung and Erwie [36], Dong et al. [8], etc.

Brief discussion of the algorithms, their modifications and hybridizations used in this book is presented in the following subsections.

2.1 Genetic Algorithm

Genetic Algorithm (GA) works on the theory of Darvin's theory of evolution and the survival-of-the fittest [16]. Genetic algorithms guide the search through the solution space by using natural selection and genetic operators, such as crossover, mutation and the selection.

GA encodes the decision variables or input parameters of the problem into solution strings of a finite length. While traditional optimization techniques work directly with the decision variables or input parameters, genetic algorithms usually work with the coding. Genetic algorithms start to search from a population of encoded solutions instead of from a single point in the solution space. The initial population of individuals is created at random. Genetic algorithms use genetic operators to create Global optimum solutions based on the solutions in the current population. The most popular genetic operators are (1) selection, (2) crossover and (3) mutation. The newly generated individuals replace the old population, and the evolution process proceeds until certain termination criteria are satisfied.

2.1.1 Selection

The selection procedure implements the natural selection or the survival-of-the fittest principle and selects good individuals out of the current population for generating the next population according to the assigned fitness. The existing selection operators can be broadly classified into two classes: (1) proportionate schemes, such as

roulette-wheel selection and stochastic universal selection and (2) ordinal schemes, such as tournament selection and truncation selection. Ordinal schemes have grown more and more popular over the recent years, and one of the most popular ordinal selection operators is tournament selection. After selection, crossover and mutation recombine and alter parts of the individuals to generate new solutions.

2.1.2 Crossover

Crossover, also called the recombination operator, exchanges parts of solutions from two or more individuals, called parents, and combines these parts to generate new individuals, called children, with a crossover probability. There are a lot of ways to implement a recombination operator. The well-known crossover operators include one-point crossover. When using one-point crossover, only one crossover point is chosen at random, for example let there be two parent string A_1 and A_2 as:

$$\begin{aligned} A_1 &= 1 \quad 1 \quad 1 \quad 1 \mid 1 \quad 1 \\ A_2 &= \mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \mid \mathbf{0} \quad \mathbf{0} \end{aligned} \tag{2.1}$$

Then, one-point crossover recombines A_1 and A_2 and yields two offsprings A_{-1} and A_{-2} as:

$$\begin{aligned} A_{-1} &= 1 \quad 1 \quad 1 \quad 1 \mid 1 \quad 1 \\ A_{-2} &= \mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \mid \mathbf{1} \quad \mathbf{1} \end{aligned} \tag{2.2}$$

2.1.3 Mutation

Mutation usually alters some pieces of individuals to form perturbed solutions. In contrast to crossover, which operates on two or more individuals, mutation operates on a single individual. One of the most popular mutation operators is the bitwise mutation, in which each bit in a binary string is complemented with a mutation probability. For example,

$$\begin{aligned} A &= 1 \quad 1 \quad 1 \quad 1 \mid 1 \quad 1 \\ A_{-1} &= \mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \mid \mathbf{0} \quad \mathbf{1} \end{aligned} \tag{2.3}$$

The step-by-step implementation of GA is explained as follows:

Step 1: Initialize GA parameters which are necessary for the algorithm. These parameters include population size which indicates the number of individuals, number of generations necessary for the termination criterion, crossover probability, mutation probability, number of design variables and respective ranges for the design variables. If binary version of GA is used then string length is also required as the algorithm parameter.

Step 2: Generate random population equal to the population size specified. Each population member contains the value of all the design variables. This value of design variable is randomly generated in between the design variable range specified. In GA, population means the group of individuals which represents the set of solutions.
Step 3: Obtain the values of the objective function for all the population members. The value of the objective function so obtained indicates the fitness of the individuals. If the problem is a constrained optimization problem then a specific approach such as static penalty, dynamic penalty and adaptive penalty is used to convert the constrained optimization problem into the unconstrained optimization problem.
Step 4: This step is for the selection procedure to form a mating pool which consists of the population made up of best individuals. The commonly used selection schemes are roulette-wheel selection, tournament selection, stochastic selection, etc. The simplest and the commonly used selection scheme is the roulette-wheel selection, where an individual is selected for the mating pool with the probability proportional to its fitness value. The individual (solution) having better fitness value will have more number of copies in the mating pool and so the chances of mating increases for the more fit individuals than the less fit ones. This step justifies the procedure for the survival of the fittest.
Step 5: This step is for the crossover where two individuals, known as parents, are selected randomly from the mating pool to generate two new solutions known as off-springs. The individuals from the population can go for the crossover step depending upon the crossover probability. If the crossover probability is more, then more individuals get chance to go for the crossover procedure. The simplest crossover operator is the single point crossover in which a crossover site is determined randomly from where the exchange of bits takes place. The crossover procedure is explained through Eqs. 2.1 and 2.2.
Step 6: After crossover, mutation step is performed on the individuals of population depending on the mutation probability. The mutation probability is generally kept low so that it does not make the algorithm unstable. In mutation, a random site is selected from the string of individuals and it is flapped as explained through Eq. 2.3.
Step 7: Best obtained results are saved using elitism. All elite members are not modified using crossover and mutation operators but can be replaced if better solutions are obtained in any iteration.
Step 8: Repeat the steps (from step 3) until the specified number of generations or termination criterion is reached.

2.2 Artificial Immune Algorithm

The immune system defends the body against harmful diseases and infections. B cells recognize the antigens which enter into the body. B cells circulate through the blood. Each antigen has a particular shape that is recognized by the receptors

present on the B cell surface. B cells synthesize and carry antibodies on their surfaces molecules that act like detectors to identify antigens. A B cell with better fitting receptors and binding more tightly the antigen replicate more and survive longer. This process of amplifying, by using proliferation, only those cells that produce a useful B cell type is called clonal selection [11, 21, 30, 38]. Clones are not perfect, but they are subjected to somatic permutations that result in children having slightly different antibodies from the parent. Clonal selection guarantees that only good B cells (i.e., with higher affinity with the antigen) can be cloned to represent the next generation [21]. However, clones with low affinity with antigen do not divide and will be discarded or deleted. Hence, the clonal selection enables the body to have sufficient numbers of antigen-specific B cells to build up an effective immune response. Mapping between the immune system and an optimization problem is done as follows. The immune response represents solutions and antigens represent the problem to solve. More precisely, B cells are considered as artificial agents that roam around and explore an environment. In other words, the optimization problem is described by an environment of antigens. The positive and negative selection mechanism is used to eliminate useless or bad solutions.

The AIA starts with a random population of antibodies [21]. Affinity of the antibody is decided from its objective function value. Select n highest antibodies to be cloned. These antibodies are cloned depending on its affinities. If the affinity is more for the particular antibody it will have more number of clones. It is calculated as

$$N_c = \sum_{i=1}^{n} \text{round}\left(\frac{\beta N}{i}\right) \tag{2.4}$$

where β is the multiplying factor controlling the number of clones and N is the total number of antibodies. These generate repertoire, which undergoes affinity maturation process as shown in Eq. 2.5, which is inversely proportional to its antigenic affinity. If the affinity is high the mutation rate is low.

$$x_{i,m} = x_i + A(\text{rand}[-1, 1])(x_{\max} - x_{\min}) \tag{2.5}$$

where, A is a factor depending on the affinity and decreases as affinity increases. Replace low affinity antibodies with new randomly generated antibodies given by Eq. 2.6

$$x_i = x_{\min} + \text{rand}(0, 1)(x_{\max} - x_{\min}) \tag{2.6}$$

The step-by-step implementation of AIA is explained as follows:

Step 1: Initialize AIA parameters which are necessary for the algorithm. These parameters include population size which indicates the number of individuals, number of generations necessary for the termination criterion, number of antibodies to be cloned, multiplying factor, repertoire rate, number of design variables and respective ranges for the design variables.

Step 2: Generate random population equal to the population size specified. Each population member contains the value of all the design variables. This value of design variable is randomly generated in between the design variable range specified. In AIA, population means the group of antibodies which represents the set of solutions.
Step 3: Obtain the values of the objective function for all the population members. The value of the objective function so obtained indicates antibody affinity. If the problem is a constrained optimization problem, then a specific approach such as static penalty, dynamic penalty and adaptive penalty is used to convert the constrained optimization problem into the unconstrained optimization problem.
Step 4: Select the n highest affinity antibodies from the population which comprises a new set of high affinity antibodies (Eq. 2.4). Clone the n selected antibodies independently and proportional to their affinities. This generates a group of clones. The higher the affinity, the higher the number of clones generated for each of the n selected antibodies.
Step 5: The group of clones undergoes affinity maturation process which is inversely proportional to its affinity (Eq. 2.5). A new set of solutions is generated consisting of matured clones. Determine the affinity of the matured clones. From this set of mature clones, reselect the highest affinity solutions. If the antigenic affinity of this solution is better than the previous iteration solution, then replace the population with the new one.
Step 6: Replace the lowest affinity antibodies from the population depending on the repertoire rate, by new antibodies using Eq. 2.6.
Step 7: Repeat the steps (from step 3) until the specified number of generations or termination criterion is reached.

2.3 Differential Evolution

The algorithm was first proposed by Storn and Price [35]. There are only three real control parameters in the algorithm. These are: (1) differentiation (or mutation) constant F, (2) crossover constant Cr and (3) size of population. The rest of the parameters are (a) dimension of problem S that scales the difficulty of the optimization task; (b) maximal number of generations (or iterations) G, which serves as a stopping condition in our case and (c) high and low boundary constraints, $x_{\max}$ and $x_{\min}$, respectively, that limit the feasible area. DE also starts with a set of random population which consist the initial solution to the problem. Mutant vector $v_{i,m}$ is generated from three different randomly chosen target vectors. This process can be mathematically written as [37],

$$v_{i,m} = x_{i,3} + F(x_{i,1} - x_{i,2}) \tag{2.7}$$

where, $v_{i,m}$ is the obtained mutant vector. In Eq. 2.7 the second term on RHS indicates the weighted difference of two randomly chosen target vectors. The

mutant vector is obtained by adding the third target vector to the weighted difference term. New trial vector $u_{i,tar}$ is obtained from the target vector and the mutant vector based on the crossover probability *Cr*. The scaling factor *F* is a user-supplied constant. Trial vector and the current target vector is compared and the best out of them is forwarded to the next generation. The optimal value of *F* for most of the functions lies in the range of 0.4–1.0 [35].

The step-by-step implementation of DE is explained as follows:

Step 1: Initialize DE parameters which are necessary for the algorithm. These parameters include population size which indicates the number of individuals, number of generations necessary for the termination criteria, crossover constant, mutation constant, number of design variables and respective ranges for the design variables.
Step 2: Generate random population equal to the population size specified. Each population member contains the value of all the design variables. This value of design variable is randomly generated in between the design variable range specified. In DE, population means the group of solutions.
Step 3: Obtain the values of the objective function for all the solutions. If the problem is a constrained optimization problem, then a specific approach such as static penalty, dynamic penalty and adaptive penalty is used to convert the constrained optimization problem into the unconstrained optimization problem.
Step 4: Choose three different target vectors. The chosen target vectors should be different from the current target vector. Obtain the mutant vector using Eq. 2.7. In Eq. 2.7, *F* indicates the mutation constant.
Step 5: Obtain trial vector based on the crossover constant. If the crossover constant is greater than the random number between 0 and 1, then the mutant vector becomes the trial vector; otherwise, the current target vector becomes the trial vector.
Step 6: Selection is done between the trial vector and the current target vector. If the objective function value of trial vector is better than the current target vector, then the trial vector enters the new population.
Step 7: Repeat the steps (from step 3) until the specified number of generations or termination criterion is reached.

2.4 Biogeography-Based Optimization

Biogeography-based optimization (BBO) is a population-based optimization algorithm inspired by the natural biogeography distribution of different species [34]. In BBO, each individual is considered as a "habitat" with a habitat suitability index (HSI). A good solution is analogous to an island with a high HSI, and a poor solution indicates an island with a low HSI. High HSI solutions tend to share their features with low HSI solutions. Low HSI solutions accept a lot of new features from high HSI solutions.

In BBO, each individual has its own immigration rate λ and emigration rate μ. A good solution has higher μ and lower λ and vice versa. The immigration rate and the emigration rate are functions of the number of species in the habitat. They can be calculated as follows

$$\lambda_k = I\left(1 - \frac{k}{n}\right) \tag{2.8}$$

$$\mu_k = E\left(\frac{k}{n}\right) \tag{2.9}$$

where, I is the maximum possible immigration rate; E is the maximum possible emigration rate; k is the number of species of the kth individual and n is the maximum number of species. In BBO, there are two main operators, the migration and the mutation.

2.4.1 Migration

Consider a population of candidate which is represented by design variable. Each design variable for particular population member is considered as suitability index variable (SIV) for that population member. Each population member is considered as individual habitat/Island. The objective function value indicates the HSI for the particular population member. Immigration and emigration rates are decided from the curve given in Simon [34]. The nature of the curve is assumed to be same for immigration and emigration but with opposite slopes, which behaves linearly. Value of S represented by the solution depends on its HSI. The emigration and immigration rates of each solution are used to probabilistically share the information between habitats. If a given solution is selected to be modified, then its immigration rate λ is used to probabilistically modify each SIV in that solution. If a given SIV in a given solution S_i is selected to be modified, then its emigration rates μ of the other solutions are used to probabilistically decide which of the solutions should migrate its randomly selected SIV to solution S_i. The above phenomenon is known as migration in BBO.

2.4.2 Mutation

In nature a habitat's HSI can change suddenly due to apparently random events (unusually large flotsam arriving from a neighboring habitat, disease, natural catastrophes, etc.). This phenomenon is termed as SIV mutation, and probabilities of species count are used to determine mutation rates. This probability mutates low HSI as well as high HSI solutions. Mutation of high HSI solutions gives them the chance to further improve. Mutation rate is obtained by using following Eq. 2.10.

$$m(S) = m_{\max}\left(1 - \frac{P_s}{P_{\max}}\right) \tag{2.10}$$

where, $m_{\max}$ is a user-defined parameter called mutation coefficient.

The step-by-step procedure about the implementation of BBO is explained as follows:

Step 1: Initialize BBO parameters which are necessary for the algorithm. These parameters include population size which indicates the number of habitats/islands, number of generations necessary for the termination criterion, maximum immigration and emigration rates, mutation coefficient, number of design variables and respective ranges for the design variables.
Step 2: Generate random population equal to the population size specified. Each population member contains the value of all the design variables. This value of design variable is randomly generated in between the design variable range specified. Every design variable in the population indicates SIVs for that respective population member (Habitat).
Step 3: Obtain the value of objective function for all population members. The value of objective function so obtained indicates the HSI for that Habitat (population member). If the problem is a constrained optimization problem, then a specific approach such as static penalty, dynamic penalty and adaptive penalty is used to convert the constrained optimization problem into the unconstrained optimization problem.
Step 4: Map the value of HSI to obtain the species count. High species count is allotted to the population member having high HSI for maximization optimization problem. If the optimization problem is of minimization type then low HSI member is given high species count.
Step 5: Modify the population using the migration operator considering its immigration and emigration rates. If a given solution is selected to be modified, then its immigration rate λ is used to probabilistically modify each suitability index variable (SIV) in that solution. If a given SIV in a given solution Si is selected to be modified, then its emigration rates μ of the other solutions are used to probabilistically decide which of the solutions should migrate the randomly selected SIV to solution S_i. Pseudo code for migration is given as follows.
Select H_i with probability proportional to λ_i (H_i is any solution vector)

```
If H_i is selected
    For j = 1 to n (n is population size)
        Select H_j with probability proportional to μ_i
        If H_j is selected
            Randomly select an SIV σ from H_j
            Replace a random SIV in H_i with σ
        end
    end
end
```

Step 6: Modify population using mutation operator. Calculate probability of existence from the value of immigration and emigration rates as explained earlier. Also calculate the mutation rate considering the user-defined mutation coefficient and probability of existence. The pseudo code for mutation is given as follows:

For j = 1 to m (m is number of design variables)

Use λ_i and μ_i to compute the probability P_i
Select SIV $H_i(j)$ with probability proportional to P_i and mutation rate
If $H_i(j)$ is selected

Replace $H_i(j)$ with a randomly generated SIV

end

end

Step 7: Best obtained results are saved using elitism. All elite members are not modified using migration and mutation operators but can be replaced if better solutions are obtained in any iteration.
Step 8: Repeat the steps (from step 3) until the specified number of generations or termination criterion is reached.

2.5 Particle Swarm Optimization

Particle swarm optimization (PSO) is an evolutionary computation technique developed by Kennedy and Eberhart [20]. It exhibits common evolutionary computation attributes including initialization with a population of random solutions and searching for optima by updating generations. Potential solutions, called particles, are then "flown" through the problem space by following the current optimum particles. The particle swarm concept was originated as a simulation of a simplified social system. The original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock. Each particle keeps track of its coordinates in the problem space, which are associated with the best solution (fitness) it has achieved so far. This value is called '*pBest*'. Another "best" value that is tracked by the global version of the particle swarm optimization is the overall best value and its location obtained so far by any particle in the population. This location is called '*gBest*'. The particle swarm optimization concept consists of, at each step, changing the velocity (i.e. accelerating) of each particle toward its '*pBest*' and '*gBest*' locations (global version of PSO). Acceleration is weighted by a random term with separate random numbers being generated for acceleration toward '*pBest*' and '*gBest*' locations. The updates of the particles are accomplished as per the following Eqs. 2.11 and 2.12.

$$V_{i+1} = w^{*} V_i + c_1^{*} r_1^{*} (pBest_i - X_i) + c_2^{*} r_2^{*} (gBest_i - X_i) \tag{2.11}$$

$$X_{i+1} = X_i + V_{i+1} \tag{2.12}$$

Equation 2.11 calculates a new velocity (V_{i+1}) for each particle (potential solution) based on its previous velocity, the best location it has achieved ('*pBest*') so far,

and the global best location ('*gBest*'), the population has achieved. Equation 2.12 updates individual particle's position (X_i) in solution hyperspace. The two random numbers 'r_1' and 'r_2' in Eq. 2.11 are independently generated in the range [0, 1]. It is observed from Eq. 2.11 that the LHS indicates the velocity term and the RHS has three terms: the first term contains the multiplication of w and V_i, where w is the constant parameter and V_i is the velocity term which indicates the correct dimension as that of LHS, the second and third terms indicate the rate of change of position toward $pBest_i$ and $gBest_i$ from the current position X_i respectively and so both the terms are to be multiplied by $1/\Delta t$, where Δt indicates the time step value. To simplify the algorithm and to reduce the algorithm parameters, the value of Δt is assumed to be unity. Moreover, in Eq. 2.12 the second term on RHS is to be multiplied by Δt, which reduces the term to match the dimension of the position (X_{i+1}) on LHS. So, the Eqs. 2.11 and 2.12 are the final equations after assuming the value of Δt as unity.

The acceleration constants 'c_1' and 'c_2' in Eq. 2.11 represent the weighting of the stochastic acceleration terms that pull each particle toward '*pBest*' and '*gBest*' positions. 'c_1' represents the confidence the particle has in itself (cognitive parameter) and 'c_2' represents the confidence the particle has in swarm (social parameter). Thus, adjustment of these constants changes the amount of tension in the system. Low values of the constants allow particles to roam far from target regions before being tugged back, while high values result in abrupt movement toward, or past through target regions [9]. The inertia weight 'w' plays an important role in the PSO convergence behavior since it is employed to control the exploration abilities of the swarm. The large inertia weights allow wide velocity updates allowing to globally explore the design space while small inertia weights concentrate the velocity updates to nearby regions of the design space. The optimum use of the inertia weight "w" provides improved performance in a number of applications. The effect of w, c_1 and c_2 on convergence for standard numerical benchmark functions was provided by Bergh and Engelbrecht [4].

Particle's velocities on each dimension are confined to a maximum velocity parameter $V_{\max}$, specified by the user. If the sum of accelerations would cause the velocity on that dimension to exceed $V_{\max}$, then the velocity on that dimension is limited to $V_{\max}$.

Unlike genetic algorithm, PSO algorithm does not need complex encoding and decoding process and special genetic operator. PSO takes real number as a particle in the aspect of representation solution and the particles update themselves with internal velocity. In this algorithm, the evolution looks only for the best solution and all particles tend to converge to the best solution.

The step-by-step implementation of PSO is explained as follows:

Step 1: Initialize PSO parameters which are necessary for the algorithm. These parameters include population size which indicates the number of individuals, number of generations necessary for the termination criterion, cognitive constant, social constant, variation of inertia weight, maximum velocity, number of design variables and respective ranges for the design variables.

Step 2: Generate random population equal to the population size specified. Each population member contains the value of all the design variables. This value of design variable is randomly generated in between the design variable range specified. In PSO, population means the group of birds (particles) which represents the set of solutions.
Step 3: Obtain the values of the objective function for all the population members. For the first iteration, value of objective function indicates the *pBest* for the respective particle in the solution. Identify the particle with best objective function value which identifies as *gBest*. If the problem is a constrained optimization problem, then a specific approach such as static penalty, dynamic penalty and adaptive penalty is used to convert the constrained optimization problem into the unconstrained optimization problem.
Step 4: Update the velocity of each particle using Eq. 2.11. Check for the maximum velocity. If the velocity obtained using Eq. 2.11 exceeds the maximum velocity, then reduce the existing velocity to the maximum velocity.
Step 5: Update the position of the particles using Eq. 2.12. Check all the design variables for the upper and lower limits.
Step 6: Obtain the value of objective function for all the particles. The new solution replaces the *pBest* if it has better function value. Identify the *gBest* from the population. Update the value of inertia weight if required.
Step 7: Best obtained results are saved using elitism. All elite members are not modified using crossover and mutation operators but can be replaced if better solutions are obtained in any iteration.
Step 8: Repeat the steps (from step 4) until the specified number of generations or termination criterion is reached.

2.5.1 Modifications in PSO

PSO suggested by Kennedy and Eberhart [20] had no inertia factor term in the algorithm. It was first suggested by Shi and Eberhert [33] and was shown that PSO performs better with introduction of inertia weight factor term. Many research works were reported for the variation of w to increase the performance of PSO. Shi and Eberhart [33] suggested linear variation of weight factor by using following expression:

$$w = ((\max w - \min w)^{*}(\max iter - \text{cur}iter)/\max iter) + \min w \tag{2.13}$$

where, max$w$ and min$w$ are the maximum and minimum value of weight factor (w) respectively; maxiter is the maximum number of generations and curiter is the current iteration of the algorithm. Generally max$w$ is taken as 0.9 and min$w$ as 0.4. Xiaohui et al. [40] suggested random weight factor as:

$$w = 0.5 + 0.5^{*}(\text{rand}) \tag{2.14}$$

where, rand is any random number from 0 to 1. Yong et al. [44] presented Chaotic descending inertia weight. The strategy for the logistic mapping changes inertia weight as:

$$w = ((\max w - \min w)^{*}(\max iter - \text{cur}iter)/\max iter) + \min w^{*}(zr) \quad (2.15)$$

where, zr = 4 * (rand) * (1−rand). Chaotic descending inertia weight is also applied to the inertia weight suggested by Xiaohui et al. [40] as

$$w = 0.5^{*}(zr) + 0.5^{*}(\text{rand}) \quad (2.16)$$

where, rand is any random number between 0 and 1.

So, it is observed that there is a significant role of inertia weight for the performance of PSO. Experimentation is carried out in this book to suggest a new inertia weight for the PSO so as to increase its performance. A new variation of inertia weight variation is suggested in this book to increase the success rate for finding the global solution in a few iterations. This saves computation time and less number of function evaluations will be required to arrive at the optimum solution. The procedure to alter the weight factor is explained below.

Set the initial value for the weight (generally 0.9)
Start loop
Set New$w$ = w
Perform PSO operation
w_factor = New$w$/max$iter$
Set w = w–w_factor
End loop

The above variation of weight factor follows a nonlinear behavior and it depends on the value of initial weight and maximum number of generations. PSO with the above inertia weight factor is referred to as PSO_M_1 in this book. Moreover, Chaotic descending inertia weight suggested by Yong et al. [44] is also incorporated in the modified inertia weight. Chaotic descending inertia weight changes the value of New$w$=w as New$w$=w*(zr). PSO with modified inertia weight and chaotic descending inertia weight is referred to as PSO_M_2 in this book.

2.6 Artificial Bee Colony Algorithm

Artificial Bee Colony (ABC) algorithm is an optimization algorithm based on the intelligent foraging behavior of honey bee swarm. The colony of artificial bees consists of three groups of bees: employed bees, onlookers and scouts [3, 18]. An employed bee searches the destination where food is available. They collect the food and return back to its origin, where they perform waggle dance depending on the amount of food available at the destination. The onlooker bee watches the dance and follows the employed bee depending on the probability of the available food.

So, more onlooker bees will follow the employed bee associated with the destination having more amount of food. The employed bee whose food source becomes abandoned behaves as a scout bee and it searches for the new food source. This principle of foraging behavior of honey bee is used to solve optimization problems by dividing the population into two parts consisting of employed bees and onlooker bees. An employed bee searches the solution in the search space and the value of objective function associated with the solution is the amount of food associated with that solution. Employed bee updates its position by using Eq. 2.17 and it updates new position if it is better than the previous position, i.e. it follows greedy selection.

$$v_{ij} = x_{ij} + R_{ij}(x_{ij} - x_{kj}) \tag{2.17}$$

where, v_{ij} is the new position of employed bee, x_{ij} is the current position of employed bee, k is a random number between *(*1, (*population size)/2*) $\neq$ i and j = 1, 2,…, *Number of design variables.* R_{ij} is a random number between (−1, 1).

An onlooker bee chooses a food source depending on the probability value associated with that food source, p_i, calculated by using Eq. 2.18.

$$p_i = \frac{F_i}{\sum_{n=1}^{N/2} F_n} \tag{2.18}$$

where, F_i is the fitness value of the solution i and *N/2* is the number of food sources which is equal to the number of employed bees.

Onlooker bees also update its position by using Eq. 2.17 and also follow greedy selection. The Employed bee whose position of the food source cannot be improved for some predetermined number of cycles than that food source is called abandoned food source. That employed bee becomes scout and searches for the new solution randomly by using Eq. 2.19.

$$x_i^j = x_{\min}^j + \text{rand}\ (0, 1)(x_{\max}^j - x_{\min}^j) \tag{2.19}$$

The value of predetermined number of cycles is an important control parameter of the ABC algorithm, which is called "*limit*" for abandonment. The value of limit is generally taken as *Number of employed bees*Number of design variables.*

The step-by-step implementation of ABC is explained as follows:

Step 1: Initialize ABC parameters which are necessary for the algorithm. These parameters include population size which indicates the number of individuals, number of generations necessary for the termination criterion, number of employed bees, number of onlooker bees, limit, number of design variables and respective ranges for the design variables.

Step 2: Generate random population equal to the number of employed bees (generally number of employed bees are half of the population size) specified. Each population member contains the value of all the design variables. This value of design variable is randomly generated in between the design variable range

specified. In ABC, population means the group of honey bees which represents the set of solutions.
Step 3: Obtain the values of the objective function for all the population members. The objective function value in ABC indicates the amount of nectar for the food source. If the problem is a constrained optimization problem, then a specific approach such as static penalty, dynamic penalty and adaptive penalty is used to convert the constrained optimization problem into the unconstrained optimization problem.
Step 4: Update the value of employed bees using Eq. 2.17. Obtain the value of objective function. If the new solution is better than the existing solution, replace the existing solution with the new one. This step indicates the greedy selection procedure for the employed bee phase.
Step 5: Onlooker bees proportionally choose the employed bees depending on the amount of nectar found by the employed bees. Mathematically, for the onlooker bee phase, the solution from the employed bee phase is chosen proportionally based on its objective function value (Eq. 2.18).
Step 6: Update the value of onlooker bees using Eq. 2.17. Obtain the value of objective function. If the new solution is better than the existing solution, replace the existing solution with the new one. Identify the abundant solutions using the limit value. If such solutions exist then these are transformed into the scout bees and the solution is updated using Eq. 2.19.
Step 7: Repeat the steps (from step 4) until the specified number of generations or termination criterion is reached.

2.6.1 Modifications in ABC

As suggested by Karaboga [18], ABC modifies the solution by using the following Eq. 2.20:

$$v_{ij} = x_{ij} + R_{ij}\left(x_{ij} - x_{kj}\right) \tag{2.20}$$

where, R_{ij} is uniformly distributed random number between –1 and 1. Modification in ABC is carried out by changing Eq. 2.20. Uniformly distributed random number is replaced by normally distributed random number with mean equal to zero and svariance equal to one. And also the expression $(x_{ij} - x_{ik})$ is replaced by $(x\text{best}_j - x_{ij})$. Here, $x\text{best}_j$ is the best solution from the population at any particular iteration. The reason for this modification is that, in the Eq. 2.20 the solution tries to move toward any random solution (x_{ik}) and there is no guarantee for the x_{ik} to be better than x_{ij}. So solution can move toward worst solution also, which may require more computational time to reach the optimum solution. By replacing x_{ik} with $x\text{best}_j$, the solution will try to move toward the best solution in every iteration which will lead to optimum solution with less computational effort.

2.7 Harmony Elements Algorithm

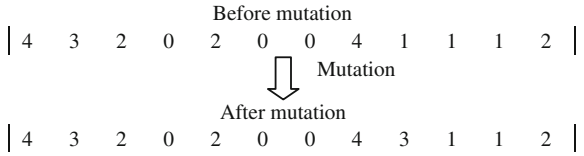
According to Chinese philosophy, the five kinds of substances (wood, fire, earth, metal and water) are essential things in the daily life of mankind. Among the five elements, there exist the relations of generation and restriction [24, 46]. The order of generation is: wood generates fire, fire generates earth, earth generates metal, metal generates water and water, in its turn, generates wood. Relationship of restriction for the five elements works in the following order: wood restricts earth, earth water, water fire, fire metal and metal wood. So, they oppose each other and at the same time cooperate with each other, thus a relative balance is maintained between generation and restriction, to ensure normal growth and development of things in nature.

Harmony elements algorithm follows the generation and restriction rules between the elements of the string. It starts the procedure with a random population. Like GA, each individual in the population is made up of string which represents the design variables. Dissimilar to GA, the algorithm initializes the solutions as strings of 0s, 1s, 2s, 3s and 4s to represent 'earth', 'water', 'wood', 'fire' and 'metal', five elements, respectively. Population is modified according to generation and restriction rules to reach its harmonious state.

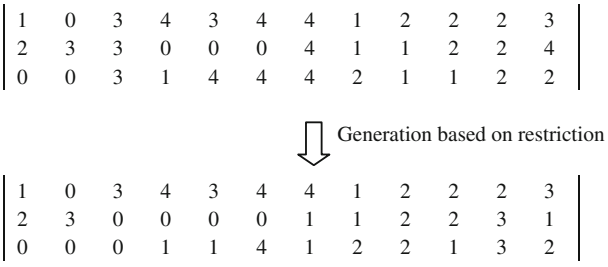
2.7.1 Modifications in HEA

Harmony Elements Algorithm starts with a random population. Population size, length of each individual string, number of input variables, upper bound and lower bound of input variables are to be initialized at the start of the algorithm. The individual strings will consist of 0s, 1s, 2s, 3s and 4s. Each number corresponds to an element. In this book the initial population matrix is denoted by Q. The basic version of the algorithm reported by Cui and Guo [6] generates random population equal to the population size. Here only one-fifth of the total population size is randomly generated; rest is generated from Q following the generation rule of five elements. This reduces the functional evolutional by 4*population size*number of generations. Four different matrices A, B, C and D are generated from matrix Q by following generation rule (Generation rule: 2 create 3, 3 create 0, 0 create 4, 4 create 1 and 1 create 2.). The above procedure helps to maintain initial harmonious state in the population. The basic algorithm generates one random matrix E equal to the population size to maintain the diversity in the population. Modification is incorporated by introducing the mutation operator to reduce the function evaluations by 1*population size*number of generations. Mutation is incorporated depending on the specified probability. Generally probability for the mutation is very low. The purpose of mutation is to maintain the diversity within the population so that algorithm does not get trapped in local optima. Mutation is carried out by changing the element in the string at randomly selected site. The above procedure for the mutation is same as that in GA. Mutation in HEA is shown as

follows:



There is no bound to follow generation and restriction rule for the mutation. Decode each individual string and evaluate the fitness values of matrix Q, A, B, C and D. Fitness value corresponds to the value of objective function. Rank individual strings in matrix Q, A, B, C and D by fitness value. Check all the matrices for the restriction rule. As all the matrices are ranked and rearranged, the first row of the matrix represents the best string so far. For instance, the best individual string is Q (1, j), for matrix Q. If other individual strings Q (i+1, j) is restricted by Q (1, j), then it is modified by replacing the element following the generation rule. If Q (i, j) and Q (i+1, j) are same, then both of them are modified according to generation rule. (i indicates the population member and j indicated the element number in the string). The restriction rule in HEA is given as follows:



Merge all the matrices A, B, C, D and Q into one matrix. Decode each individual string and evaluate the fitness values of matrix and then rank individual strings of matrix by fitness value. The so obtained matrix will be the new matrix Q for the next iteration.

The step-by-step procedure about the implementation of HEA is explained as follows:

Step 1: Initialize HEA parameters which are necessary for the algorithm to proceed. These parameters include population size, number of generations necessary for the termination criterion, string length, number of design variables, design variable ranges, and mutation rate.

Step 2: Generate initial population considering all the design variables. All the design variables are initially coded using the string consisting of 0, 1, 2, 3 and 4. Strings for each design variables are combined to form a single string. Initial population is created considering the combined strings which gives matrix Q as

Matrix Q: Randomly generated population

$$\begin{array}{cccccccccccccc}
1 & 0 & 3 & 4 & 3 & 4 & 4 & 1 & 2 & 2 & 2 & 3 & . & . \\
2 & 3 & 3 & 0 & 0 & 0 & 4 & 1 & 1 & 2 & 2 & 4 & . & . \\
0 & 0 & 3 & 1 & 4 & 4 & 4 & 2 & 1 & 1 & 2 & 2 & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & & \\
. & . & . & . & . & . & . & . & . & . & . & . & &
\end{array}$$

Step 3: All other matrices A, B, C and D are created from matrix Q following the generation rule (Generation rule: 2s create 3s, 3s create 0s, 0s create 4s, 4s create 1s, 1s create 2s) as given below.

Matrix Q: Randomly generated population

$$\begin{array}{cccccccccccc}
1 & 0 & 3 & 4 & 3 & 4 & 4 & 1 & 2 & 2 & 2 & 3 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

Generation rule
Corresponding Matrix A: **Created from Matrix Q**

$$\begin{array}{cccccccccccc}
2 & 4 & 0 & 1 & 0 & 1 & 1 & 2 & 3 & 3 & 3 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

Generation rule
Corresponding Matrix B: **Created from Matrix A**

$$\begin{array}{cccccccccccc}
3 & 1 & 4 & 2 & 4 & 2 & 2 & 3 & 0 & 0 & 0 & 4 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

Generation rule
Corresponding Matrix C: **Created from Matrix B**

$$\begin{array}{cccccccccccc}
0 & 2 & 1 & 3 & 1 & 3 & 3 & 0 & 4 & 4 & 4 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

Generation rule
Corresponding Matrix D: **Created from Matrix C**

$$\begin{array}{cccccccccccc}
4 & 3 & 2 & 0 & 2 & 0 & 0 & 4 & 1 & 1 & 1 & 2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

After the creation of all the matrices, all the design variables are decoded and mapped for the considered design variable range.

Step 4: Mutation is carried out considering the mutation probability. Mutation consists of changing one of the bits randomly in the chromosome and so it leads to maintain the diversity in the population and it also does not allow the algorithm to get trapped at local optima. Mutation phenomenon is explained in the earlier

section. After performing mutation operation all the matrices are arranged in the ascending order of their objective function value as all the considered problems are for the minimization.
Step 5: Apply generation based on restriction rule for all the matrices and then merge all the matrices to form a single matrix which is the total population size considered. After merging all the matrices they are again arranged in the ascending order. From this arranged matrix one-fifth of the population is selected starting from row one and gives again matrix Q for the next generation. This leads to the best population members obtained in the particular generation. This completes one generation.
Step 6: Repeat the steps (from step 3) until the specified termination criterion is reached.

2.8 Hybrid Algorithms

For the population-based optimization methods, the terms exploration and exploitation have been playing an important role in describing the working of an algorithm. Use of existing information is known as 'exploitation'. Generation of new solutions in the search space is termed as 'exploration'. As exploitation and exploration are the opposing forces, its balance is required for the algorithm to search for the global optimum solutions. Any selection procedure in the algorithm is generally characterized as exploitation because the fitness (information) of the individuals is used to determine whether or not an individual should be exploited. So, exploration and exploitation are two important aspects in the population-based optimization algorithms. However, different algorithms employ different operators for exploration and exploitation.

In ABC, a new solution vector is calculated using the current solution and a randomly chosen solution from the population indicates the explorative ability of the algorithm. Moreover, a fitness-based probabilistic selection scheme is used in the ABC algorithm which indicates the exploitation tendency of the algorithm. ABC also has the diversification controlled by the random selection process in the scout bees phase which makes ABC escape from local minima. However, in ABC, a greedy selection scheme is applied between the new solution and the old one and the better one is preferred for inclusion in the population which once again indicates the exploitation tendency of the algorithm. In PSO, a new position vector is calculated using the particle's current and best solution and the swarm's best solution. In PSO, the new solution is replaced with the old one without considering which one is better. So, PSO has only explorative tendency and it lacks the exploitation ability. In DE, the existing solution is updated by the difference of the two existing solutions which is weighted by a constant scaling factor, while in ABC it is weighted by a random step size. So, DE also possesses the explorative ability like ABC for updating the solutions. DE also has explicit crossover and also employs greedy selection between

the current solution and a new solution. The crossover and greedy selection indicate the exploitation tendency of the DE algorithm. GA also uses both exploration and exploitation of the solutions. The crossover and the mutation operators indicate the exploration ability of the GA algorithm. The selection scheme employed in GA algorithm indicates its exploitation tendency as the information of the individuals is used for the further processes of the algorithm. BBO works by exchanging the design variables from one solution to the other based on the immigration and emigration rate which is similar to the crossover procedure of the GA and so it indicates the explorative ability of the algorithm. But, the mutation process in BBO uses the probability of the solution to decide whether the solution is to be mutated or not. So, mutation process in BBO indicates the exploitation tendency of the algorithm.

It is observed from the above discussion that all the described algorithms have different exploration and exploitation ability. ABC updates the solution in the employed bee phase and generates the new solution by exploration and the greedy selection process is done on the new solution by exploitation. Furthermore, the solutions are exploited by using the proportional selection in the onlooker bee phase and again the new solutions are generated in the onlooker bee phase by exploration. Moreover, the exploration in the employed bee phase and the onlooker bee phase is similar as it is using the similar mathematical expression. Motivated by this point, it is decided to investigate the onlooker bee phase by using some other mathematical expression for the exploration. ABC uses three different exploitation mechanisms (two for the greedy selection and one for the proportional selection) in the onlooker bee phase. The investigation is also carried out to reduce the exploitation in ABC by removing the proportional selection of the onlooker bee. The exploration of the onlooker bee is replaced by the exploration mechanisms of PSO, DE, BBO and GA separately which results in the four different hybrid algorithms.

All hybrid algorithms are developed by keeping ABC as the common algorithm. The four different hybrid algorithms that are developed are HPABC (Hybrid Particle swarm based Artificial Bee Colony), HBABC (Hybrid Biogeography-based Artificial Bee Colony), HDABC (Hybrid Differential evolution based Artificial Bee Colony) and HGABC (Hybrid Genetic algorithm based Artificial Bee Colony). All the developed hybrid algorithms start with the employed bee phase of ABC and then the onlooker bee phase is replaced by the searching mechanism of other algorithms. All the hybrid algorithms are discussed as follows:

2.8.1 HPABC

Both ABC and PSO are good at exploring the search space. HPABC is developed to combine the advantages of both ABC and PSO. HPABC starts with the initial population and updates the solution by following the searching mechanism of the employed bees in ABC. The solutions obtained after the employed bee phase follows the mechanism of particle swarm optimization. The pseudo code for HPABC is given below:

START

Initialize Population size, number of generations, value of w, c_1 and c_2, V_{max} and range of design variables.
Generate the initial population and evaluate the fitness for each individual

For i =1 to number of generations

For i = 1 to *Population size*

Produce new solutions for the employed bees and evaluate them (Eq. 2.17)
Replace new solution if it is better than the previous one

End

For i = 1 to Population size

Calculate the velocity of each solution (Eq. 2.11)
Check the obtained velocity for the limit (V_{max})
Produce new solutions (Eq. 2.12)
Replace new solution if it is better than the previous

End
End
STOP

It is observed from the above pseudo code that there is only a little increase in the computational effort of HPABC as compared to basic ABC. However, HPABC eliminates the proportional selection for the onlooker bees and also the scout bees. Solution is updated after the employed bee phase by following the search mechanism of particle swarm optimization and hence it combines the strength of both the algorithms.

2.8.2 HBABC

ABC is good at exploring the search space and locating the region of global minimum. On the other hand, BBO has a good exploitation searching tendency for global optimization. Based on these considerations, in order to maximize the exploration and the exploitation a HBABC approach is proposed which combines the strength of ABC and BBO. The pseudo code for HBABC is given below:

START

Initialize Population size, number of generations, immigration rates, emigration rates, mutation rate and range of design variables.
Generate the initial population and evaluate the fitness for each individual

For i = 1 to number of generations

For i = 1 to *Population size*

Produce new solutions for the employed bees and evaluate them (Eq. 2.17)

Replace new solution if it is better than the previous one

End

For each individual, map the fitness to the number of species

Calculate the immigration rate λ_i and the emigration rate μ_i for each individual X_i

For i = 1 to *Population size*

Select X_i with probability proportional to λ_i

if rand(0, 1) $< \lambda_i$

For j = 1 to N

Select X_j with probability proportional to μ_j

if rand(0, 1) $< \mu_j$

Randomly select a variable σ from X_j

Replace the corresponding variable in X_i with σ

Endif

Endif

End

Replace new solution if it is better than the previous one

End

End

STOP

It is observed from the above pseudo code that there is only a little increase in the computational effort of HBABC as compared to basic ABC. However, HBABC eliminates the proportional selection for the onlooker bees and also the scout bees. Solution is updated after the employed bee phase by following the search mechanism of Biogeography-based optimization and hence it combines the strength of both the algorithms.

2.8.3 HDABC

ABC and DE have different searching capability and the searching mechanism. Both the algorithms are good at exploring the search space. HDABC is developed to combine the advantages of both ABC and DE. HDABC starts with the initial population and updates the solution by following the searching mechanism of the employed bees in ABC. The solutions obtained after the employed bee phase follows the mechanism of differential evolution. The pseudo code for HDABC is given below.

START

Initialize Population size, number of generations, value of F, and C and range of design variables.

Generate the initial population and evaluate the fitness for each individual

For i = 1 to number of generations

For i = 1 to *Population size*

Produce new solutions for the employed bees and evaluate them (Eq. 2.17)

Replace new solution if it is better than the previous one

End

For i = 1 to *Population size*

Generate mutant vector by using three randomly selected solutions (Eq. 2.7)

Generate trial vector based on crossover probability

If trial vector is better than the current target vector, replace the current solution with the trial solution.

End

End

STOP

It is observed from the above pseudo code that there is only a little increase in the computational effort of HDABC as compared to basic ABC. However, HDABC eliminates the proportional selection for the onlooker bees and also the scout bees. Solution is updated after the employed bee phase by following the search mechanism of differential evolution and hence it combines the strength of both the algorithms.

2.8.4 HGABC

ABC and GA are also having different searching capability and the searching mechanism. ABC is good in the exploration of the search space while GA uses both exploration and exploitation for finding the solution. HGABC is developed to combine the advantages of both ABC and DE. HGABC also starts with the initial population and updates the solution by following the searching mechanism of the employed bees in ABC. The solutions obtained after the employed bee phase follows the mechanism of genetic algorithm to further enhance the solution. The pseudo code for HGABC is given below.

START

Initialize Population size, number of generations, crossover probability, mutation probability and range of design variables.

Generate the initial population and evaluate the fitness for each individual

For i=1 to number of generations

For i = 1 to *Population size*

Produce new solutions for the employed bees and evaluate them (Eq. 2.17)

Replace new solution if it is better than the previous one

End

For i = 1 to Population size

Update solutions by using crossover according to crossover probability
Update solutions by using mutation according to mutation probability
Replace solutions if it is better than the existing

End

End

STOP

It is observed from the above pseudo code that there is only a little increase in the computational effort of HGABC as compared to basic ABC. However, HGABC eliminates the proportional selection for the onlooker bees and also the scout bees. Solution is updated after the employed bee phase by following the search mechanism of genetic algorithm and hence it combines the strength of both the algorithms.

2.9 Shuffled Frog Leaping Algorithm

The shuffled frog leaping algorithm is an algorithm based on memetic meta-heuristic. It was brought forward and developed by Eusuff et al. [13]. This algorithm uses the mode of memetic evolvement among frog subgroups in local exploration. The algorithm uses the shuffled strategy and allows the message changing in local exploration. The shuffled frog leaping algorithm combines the advantages of memetic evolvement algorithm and particle swarm optimization (PSO). The algorithm changes message not only in the local exploration but also in the global exploration. So, the local and the global are combined well in the SFLA. The local search makes memetic to transfer among the individuals and the shuffled strategy makes memetic to transfer among the global. As genetic algorithm (GA) and particle swarm optimization (PSO), the shuffled frog leaping algorithm (SFLA) is an optimization algorithm based on colony.

The SFLA is a combination of determinacy method and random method. The determinacy strategy allows the algorithm to exchange messages effectively. The

randomicity ensures the algorithm's flexibility and robustness. The SFLA progresses by transforming frogs (solutions) in a memetic evolution. In this algorithm, individual frogs are not so important; rather, they are seen as hosts for memes and described as a memetic vector. In the SFLA, the population consists of a set of frogs (solutions) that is partitioned into subsets referred to as memeplexes. The different memeplexes are considered as different cultures of frogs, each performing a local search. Within each memeplex, the individual frogs hold ideas, which can be influenced by the ideas of other frogs, and evolve through a process of memetic evolution. After a defined number of memetic evolution steps, ideas are passed among memeplexes in a shuffling process. The local search and the shuffling processes continue until defined convergence criteria are satisfied.

The algorithm begins the random selection of frog groups. The frog groups are divided to some subgroups. These subgroups can accomplish the local exploration independently and in different directions. A frog of each subgroup can affect others in the subgroup. So, they undergo the memetic evolution. The memetic evolution improves the individual memetics quality and strengthens the executive ability to goal. It is possible to increase the good frog's weight and to decrease the bad frog's weight for a good goal. When some memetics accomplish the evolution, the frog subgroups are shuffled. The memetics are optimized in global scope and produce some new frog subgroups by mixture mechanism.

The shuffling enhances the quality of memetics which are affected by the different subgroups. The local exploration and the global exploration are shuffled until the end of the constringency condition. The balance strategy between the global message exchange and local deep search makes the algorithm to jump out the local extremum easily (Yue et al., [45]). The flowchart of SFLA algorithm is shown in Fig. 2.1.

2.10 Grenade Explosion Algorithm

The idea of the presented algorithm is based on observation of a grenade explosion, in which the thrown pieces of shrapnel destruct the objects near the explosion location. Le is the length of explosion along each coordinate, in which the thrown piece of shrapnel may destruct the objects. The loss caused by each piece of shrapnel is calculated. A high value for loss per piece of shrapnel in an area indicates there are valuable objects in that area. To make more loss, the next grenade is thrown where the greatest loss occurs. Although the objects near grenade's location are more likely to be damaged, the probability of destruction is still kept for farther objects by choosing a high value for Le. This process would result in finding the best place for throwing the grenades, even though shrapnel cannot discover this area in early iterations. The loss caused by destruction of an object is considered as the fitness of the objective function at the object's location. Suppose that X is the current location of a grenade and $X = \{X_m\}$, $m = 1, 2, \ldots, n$. n is the

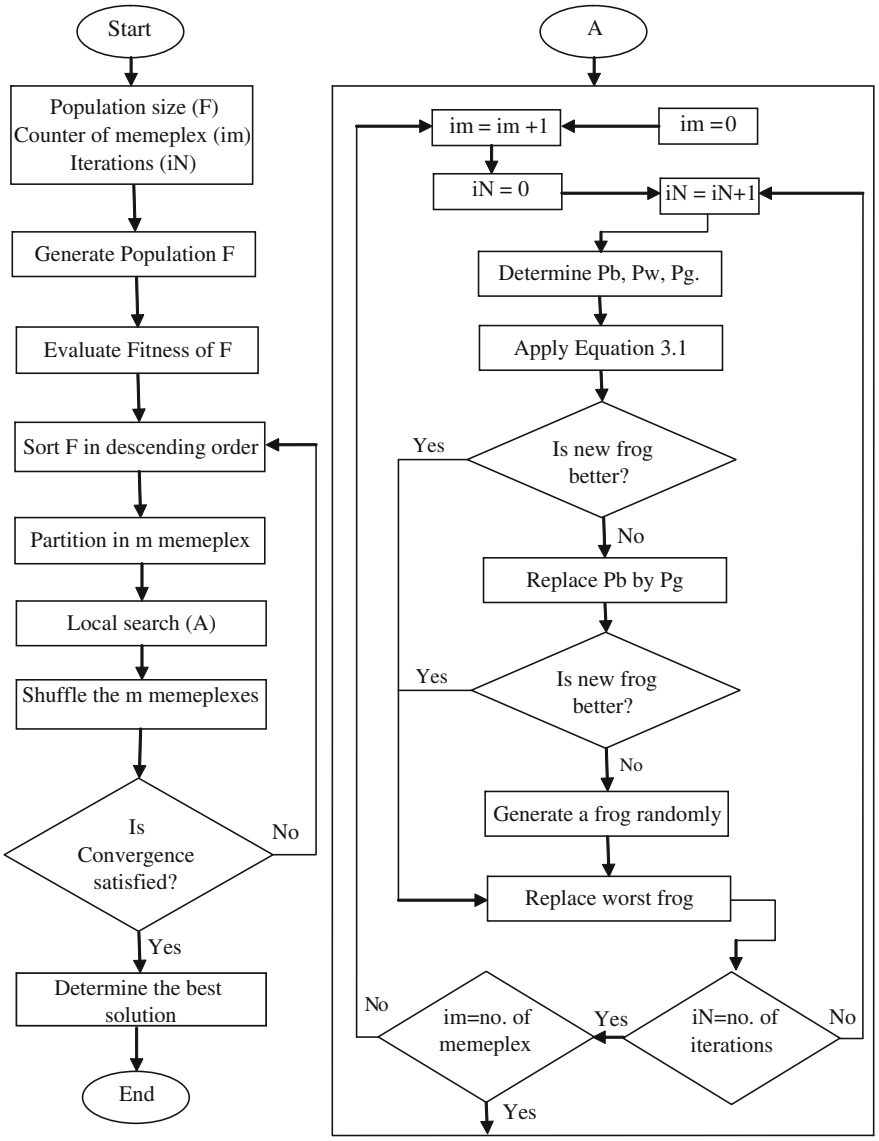


Fig. 2.1 Flowchart of SFLA (from [2]; reprinted with permission from Springer Science+ Business Media)

search space dimension and is equal to the number of independent variables. Now N_q pieces of shrapnel are produced by the grenade explosion and destruct objects that are in X'_j location (from [1]; reprinted with permission from Elsevier):

$$X'_j = \{X_m + \text{sign}(r_m) * p| r_m| * Le\}, j = 1, 2. \ldots . N_q \quad (2.21)$$

where r_m is a uniformly distributed random number in $[-1, 1]$ and p is a constant. A high value for p lets pieces of shrapnel search the region near the exploded grenade more accurately, while a low one lets them to explore farther regions better.

Considering Eq. 2.21, it is obvious that exploration for more valuable items performs in an n-dimensional cubic space extended 2Le units along each coordinate and the grenade is located at the center of this cube. To use this algorithm, an independent variable range is scaled to $[-1, 1]$. Using Eq. 2.21, some produced shrapnel may collide to objects outside the feasible space. To increase the convergence rate and exploration of near-boundary regions more accurately, such a collision location is transported to a new location inside the feasible region according to the following scheme:

$$\text{If } X'_j \text{ doesn't belong to } [-1, 1]^{\mathrm{n}},\ \left(B'_j = X'_{\mathrm{j}}/\text{Largest component of } X'_j \text{ in value}\right) \tag{2.22}$$

$$B''_j = r'_j * \left(B'_j - \mathrm{X}\right) + \mathrm{X} \tag{2.23}$$

$$j = 1 \text{ to Nq (shrapnel number) and } 0 < r'_{\mathrm{j}} < 1 (\text{random number})$$

where, X'_j is the collision location outside the feasible space and $\mathrm{B}''_{\mathrm{j}}$ is the new location inside the feasible space. One of the special concepts of this algorithm is the agent's territory radius (R_{t}), which means an agent (in this algorithm agents are grenades) does not let other agents come closer than a specific distance, which is specified by R_{t}. When several agents are exploring the feasible space, a high value for this parameter makes sure that grenades are spread quite uniformly in the feasible space and the whole space is being explored. While a low value for this parameter lets the grenades get closer to search the local region all together, a higher value for the explosion range makes it possible to explore farther regions (better exploration), while a lower one lets the grenades focus on the region nearby (better exploitation). The value of exponent p determines the intensity of exploration. This parameter is updated based on the value of Tw:

$$P = \max\{1/n, \log(R_{\mathrm{t}}/\mathrm{Le})/\log(\mathrm{Tw})\} \tag{2.24}$$

where Tw is the probability that a produced piece of shrapnel collides an object in an n-dimension hyper-box which circumscribes the grenade's territory.

To increase the global search ability, a high value for R_{t} should be chosen at the beginning ($R_{t-\mathrm{initial}}$) and reduced gradually to let the grenades search the probable global minimum location found altogether for a precise answer. A simple method to reduce R_t is given by Eq. 2.25.

$$R_t = \left(\mathrm{R}_{\mathrm{t-initial}}/\ \mathrm{R}_{\mathrm{rd}}^{\text{iteration number/total iterations}}\right) \tag{2.25}$$

The value of R_{rd} is set before the algorithm starts. This parameter represents the ratio of the value of R_t in the first iteration to its value in the final iteration. Furthermore, Le is reduced according to the following equation:

$$\text{Le} = (\text{Le initial})^{\text{m}}(\text{Rt})^{1-m}, \quad 0 \leq m \leq 1 \tag{2.26}$$

which indicates that Le is reduced more slowly than R_t during the iterations in order to save the global search ability. m can be constant during the algorithm, or reduced from a higher value to a lower one.

The next chapter presents the applications of many existing optimization algorithms to the design optimization of mechanical elements.

References

1. Ahrari A, Atai A (2010) Grenade Explosion Method-A novel tool for optimization of multimodal functions. Appl Soft Comput 10(4):1132–1140
2. Amiri B, Fathian M, Maroosi A (2009) Application of shuffled frog leaping algorithm on clustering. Int J Adv Manuf Technol 45:199–209
3. Basturk B, Karaboga D (2006) An artificial bee colony (ABC) algorithm for numeric function optimization. IEEE Swarm Intelligence Symposium, 12–14 May ,Indianapolis
4. Bergh F, Engelbrecht AP (2006) A study of particle swarm optimization particle trajectories. Inf Sci 176:937–971
5. Cai X, Cui Y, Tan Y (2009) Predicted modified PSO with time-varying accelerator coefficients. Int J Bio-Inspired Comput 1:50–60
6. Cui YH, Guo R (2008) Harmony elements algorithmHarmony elements algorithm. http://www.mathworks.com/matlabcentral/fileexchange/21963-harmony-element-algorithm
7. Cui H, Turan O (2010) Application of a new multi-agent hybrid co-evolution based particle swarm optimisation methodology in ship design. Comput-Aided Des 2:1013–1027
8. Dong HK, Ajith A, Jae HC (2007) A hybrid genetic algorithm and bacterial foraging approach for global optimization. Inf Sci 177:3918–3937
9. Dong Y, Tang J, Xu B, Wang D (2005) An application of swarm optimization to nonlinear programming. Comput Math Appl 49:1655–1668
10. Dorigo M (1992) Optimization, learning and natural algorithms. PhD Dissertation, Politecnico di Milano, Italy
11. Emma H, Jon T (2008) Application areas of AIS: the past, the present and the future. Appl Soft Comput 8:191–201
12. Eusuff M, Lansey E (2003) Optimization of water distribution network design using the shuffled frog leaping algorithm. J Water Resour Plan Manag ASCE 129:210–225
13. Eusuff M, Lansey K, Pasha F (2006) Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. Eng Optim 38(2):129–154
14. Farmer JD, Packard N, Perelson A (1986) The immune system, adaptation and machine learning. Physica 22:187–204
15. Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimization algorithm: Harmony Search. Simul, the Soc for Model and Simul Int 76(2):60–68
16. Holland J (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
17. Hui L, Zixing C, Yong W (2010) Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Appl Soft Comput 10:629–640
18. Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, Turkey

19. Karaboga D, Akay B (2010) A modified artificial bee colony (ABC) algorithm for constrained optimization problems. Appl Soft Comput doi:10.1016/j.asoc.2010.12.001
20. Kennedy J, Eberhart RC (1995) Particle swarm optimization. Proceedings IEEE International Conference on Neural Networks, Piscataway, 1942–1948
21. Leandro NC, Fernando JVZ (2002) Learning and optimization using the clonal selection principle. IEEE Trans Evol Comput Spec Issue Artif Immune Sys 6(3):239–251
22. Li R, Chang X (2006) A modified genetic algorithm with multiple subpopulations and dynamic parameters applied in CVAR model. Comput Intell for Model, Control and Autom, Sydney, p 151
23. Liu J, Tang LA (1999) Modified genetic algorithm for single machine scheduling. Comput Ind Eng 37:43–46
24. Maciocia G (2005) The foundations of chinese medicine. Elsevier, London
25. Montalvo I, Izquierdo J, Perez-Garcia R, Herrera M (2010) Improved performance of PSO with self-adaptive parameters for computing the optimal design of water supply systems. Eng Appl Artif Intell 23:727–735
26. Mouti FSA, Hawary MEE (2009) Modified artificial bee colony algorithm for optimal distributed generation sizing and allocation in distribution systems. IEEE Electr Power and Energy Conf (EPEC), Montreal, pp 1–9
27. Passino KM (2002) Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst Mag 22:52–67
28. Preechakul C, Kheawhom S (2009) Modified genetic algorithm with sampling techniques for chemical engineering optimization. J Ind and Eng Chem 15:101–107
29. Rashedi E, Nezamabadi-pour H, Saryazdi S (2009) GSA: a gravitational search algorithm. Inf Sci 179:2232–2248
30. Rodin V, Benzinou A, Guillaud A, Ballet P, Harrouet F, Tisseau J, Le Bihan J (2004) An immune oriented multi-agent system for biological image processing. Pattern Recogn 37:631–645
31. Shahla N, Mohammad EB, Nasser G, Mehdi HA (2009) A novel ACO–GA hybrid algorithm for feature selection in protein function prediction. Expert Sys Appl 36:12086–12094
32. Shen Q, Jiang J, Tao J, Shen G, Yu R (2005) Modified ant colony optimization algorithm for variable selection in QSAR modeling: QSAR Studies of Cyclooxygenase Inhibitors. J Chem Inf Model 45:1024–1029
33. Shi Y, Eberhart RC (1998) A modified particle swarm optimization. Proceedings the International Conference on Evolutionary Computer, Anchorage, pp 69–73
34. Simon D (2008) Biogeography-based optimization. IEEE Trans Evol Comput 12:702–713
35. Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11:341–359
36. Tung Y, Erwie Z (2008) A hybrid genetic algorithm and particle swarm optimization for multimodal functions. Appl Soft Comput 8:849–857
37. Vitaliy F (2006) Differential evolution–in search of solutions. Springer, New York
38. Wang X, Gao XZ, Ovaska SJ (2004) Artificial immune optimization methods and applications–a survey. IEEE Int Conf Sys Man Cybern 4:3415–3420
39. Wen YL (2010) A GA–DE hybrid evolutionary algorithm for path synbook of four-bar linkage. Mech Mach Theory 45:1096–1107
40. Xiaohui H, Eberhart RC, Shi Y (2003) Engineering optimization with particle swarm. Proceedings of swarm intelligence symposium, West Lafayette, pp 53–57
41. Yannis M, Magdalene M (2010) Hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem. Comput Oper Res 37:432–442
42. Yildiz AR (2009) A novel particle swarm optimization approach for product design and manufacturing. Int J Adv Manuf Technol 40:617–628
43. Ying PC (2010) An ant direction hybrid differential evolution algorithm in determining the tilt angle for photovoltaic modules. Expert Sys Appl 37:5415–5422

44. Yong F, Yong MY, Wang AX (2007) Comparing with chaotic inertia weights in particle swarm optimization. Proceedings the Sixth International Conference on Machine Learning and Cybernetics, Hong Kong, pp 19–22
45. Yue H, Gu G, Liu H, Shen J, Zhao J (2009) A modified ant colony optimization algorithm for tumor marker gene selection. Genomics Proteomics Bioinforma 7:200–208
46. Zhang EQ (1992) Basic theory of traditional chinese medicine. Shanghai University of Traditional Medicine, Shanghai

http://www.springer.com/978-1-4471-2747-5

Mechanical Design Optimization Using Advanced Optimization Techniques

Rao, R.V.; Savsani, V.J.

2012, XII, 320 p., Hardcover
ISBN: 978-1-4471-2747-5