Metaheuristic Optimization

The word ‘‘heuristic” is Greek and means ‘‘to know”, ‘‘to find”, ‘‘to discover” or ‘‘to guide an investigation”. Heuristic refers to experience-based techniques for problem solving, learning, and discovery that give a solution which is not guaranteed to be optimal. A metaheuristic technique is a problem independent higher level heuristic technique that can be employed to solve many hard optimization problems.

Two major components of any metaheuristic algorithm are intensification and diversification, or exploitation and exploration. Diversification means to generate diverse solutions so as to explore the search space on a global scale, while intensification means to focus the search in a local region knowing that a current good solution is found in this region.

The use of metaheuristic algorithms in real applications has increased from last few years. The metaheuristic algorithms are mainly based on mimicry (imitation) of social behavior of various plants, animals, and biological systems. All metaheuristic algorithms possess some common characteristics. These are

• These algorithms are nature inspired algorithms and mimic the laws of nature in the inner structure to solve complex engineering design problems.

• These algorithms possess stochastic nature, this nature is simulated by the incorporation of the random component in the search process and in other decision-making points.

• These algorithms are derivative-free algorithms, this fact makes these algorithms flexible, simple and less time-consuming.

• These algorithms possess several parameters, which have to be fitted according to the nature of the problem.

The metaheuristic algorithms are classified into three main categories namely:

1. Physics postulates based algorithms
2. Evolutionary computing based algorithms
3. Swarm Intelligence (SI) based algorithms

In physics postulates based algorithms, concepts of physics are employed to derive the optimization process. Examples of these algorithms are simulated annealing, gravitational search algorithm, Big Bang-Big Crunch Algorithm, Black Hole Algorithm and many more algorithms fall in this category. These algorithms are based on the laws of physics.

In category 2, the algorithm which is widely used is Genetic Algorithm (GA) and is based on survival of the fittest Darwinian Theory. Other examples of these algorithms are Evolution Strategy (ES), Evolutionary Programming (EP) , Differential Evolution and Biogeography Based Optimization (BBO). The flow of these algorithms starts with the initial set of solution and successive improvements in runs with the incorporation of different operators.

In category 3, the algorithms which mimic the behavior of swarms, school of fish, birds and based on cognitive and social intelligence. In this category, Particle Swarm Optimization (PSO) is the most popular technique and experimented algorithm. This algorithm is based on follow the leader philosophy. Particles (search agents) search for the solution in multidimensional search space and update the position and velocity with

the help of cognitive and social intelligence. Many experiments have been done in order to improve the performance of this optimizer. Other algorithms, in this category are: Bat Algorithm, Spider Monkey Optimization Algorithm, Cuckoo Search Optimization Algorithm, Fruit Fly Optimization, Firefly Optimization and many more.

1. Physics postulates based algorithms

1.1. Simulated annealing technique

Simulated Annealing (SA), was independently introduced by Kirkpatrick, Gela and Vecchi in 1982,1983 and Cerny in 1985. Annealing , physically, refers to the process of heating up a solid to a high temperature followed by slow cooling achieved by decreasing the temperature of the environment in steps At each step the temperature is maintained constant for a sufficient period of time to reach thermal equilibrium.

Simulated annealing is a powerful optimization technique which exploits the resemblance between a minimization process and the cooling of molten metal. The physical annealing process is simulated in the simulated annealing (SA) technique for the determination of global or near-global optimum solutions for optimization problems. Simulated annealing technique was originally inspired by the formation of crystals in solids during cooling. The method itself has a direct analogy with thermodynamics, specifically with the way that metal cools and anneals. At high temperatures, molecules move freely with respect to one another. If it is cooled slowly, thermal mobility is restricted. The atoms are often able to line themselves up and form a pure crystal that is completely regular. This crystal is the state of minimum energy for the system, which would correspond to the optimal solution in a mathematical optimization problem. However, if a liquid metal is cooled quickly, i.e. quenched, it does not reach a minimum energy state but a somewhat higher energy state corresponding, in the mathematical sense, to a suboptimal solution found by iterative improvement. The defect free crystals, i.e. minimum energy solids, are more likely to be formed under a slow cooling process. The two main features of the simulated annealing process are (1) the transition mechanism between states and (2) the cooling scheme. The SA technique simulates the procedure of gradually cooling a metal, until the energy of the system reaches the globally minimum value. Beginning with a high temperature, a metal is slowly cooled, so that the system is in thermal equilibrium at every stage. At high temperatures, the metal is in liquid phase and the atoms of the system are randomly arranged. By gradually cooling the metal, the system becomes more organized, until it finally reaches a “frozen” ground state, where the energy of the system has reached the globally minimum value.

Although the Simulated Annealing has the disadvantage of taking long CPU time, it has other strong features. These features include:

1) It could find a high quality solution that does not strongly depend on the choice of the initial solution.
2) It does not need a complicated mathematical model of the problem under study.
3) It can start with any given solution and try to improve it. This feature could be utilized to improve a solution output from other suboptimal or heuristic methods.
4) It has been theoretically proved to converge to the optimum solution.
5) It does not need large computer memory.

By making an analogy between the annealing process and the optimization problem, a great class of combinatorial optimization problems can be solved following the same procedure of transition from equilibrium state to another, reaching minimum energy of the system This analogy can be stated as: Solutions in the combinatorial optimization problem are equivalent to states or configurations of the physical system. The cost of a solution is equivalent to the energy of a state. A control parameter is introduced to play the role of the temperature in the annealing process.

The flow chart of simulated annealing method is shown in Fig. 1.

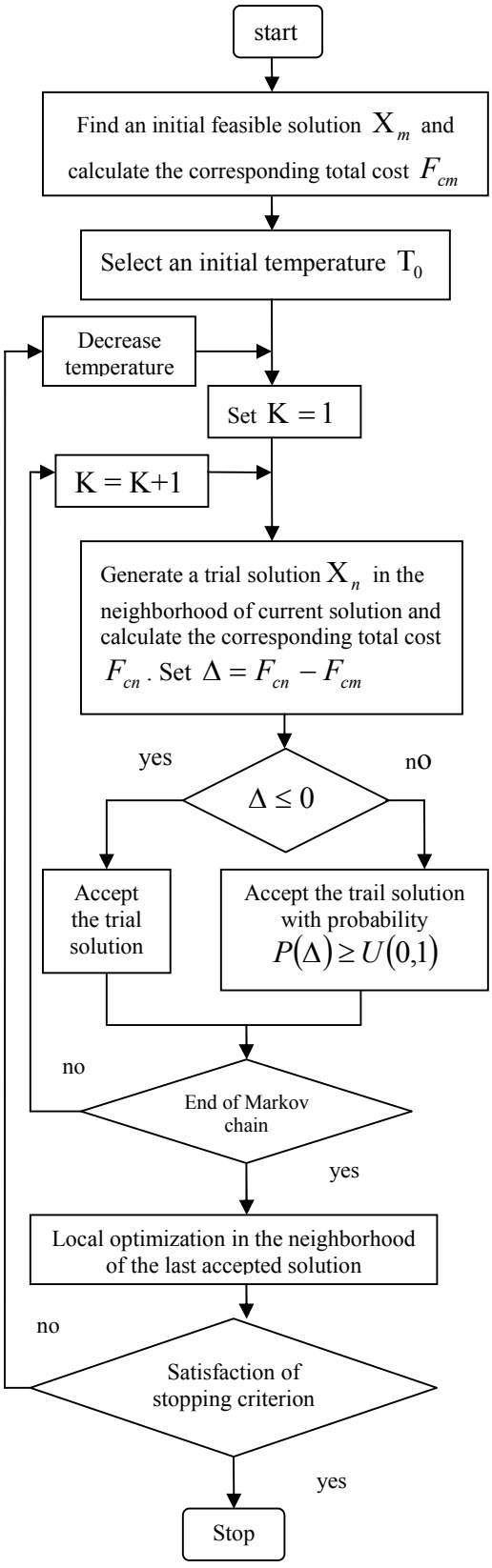

Fig. 1. Flow chart of simulated annealing

1.2. Gravitational Search Algorithm

Law of gravity

The gravitation is the tendency of masses to accelerate towards each other. Gravity acts between separated particles without any intermediary and without any delay. According to Newton's law of gravity, each particle attracts every other particle with a 'gravitational force'. The gravitational force, which acts between two particles, is directly proportional to the product of their masses and inversely proportional to the square of the distance between them.

$$F = G \times \left(\frac{\mathrm{M}_1 \mathrm{M}_2}{R^2} \right) \qquad (1)$$

where F is the magnitude of the gravitational force, G is the gravitational constant, M_1 and M_2 are the mass of the first and second particles, respectively, and R is the distance between the two particles.

Newton's second law says that when a force F is applied to a particle, its acceleration a depends only on the force and its mass M.

$$a = \frac{F}{\mathrm{M}} \qquad (2)$$

Therefore based on (1) and (2), it is clear that there is an attracting gravitational force among all particles of the universe, where the effect of bigger and the closer particle is high. The concept is presented in Fig. 1. An increase in the distance between two particles means reduction of gravitational force between them. In this figure, F_{1j} is the force that acts on M_1 from M_j and F_1 is the overall force that acts on M_1 and causes the acceleration vector a_1.

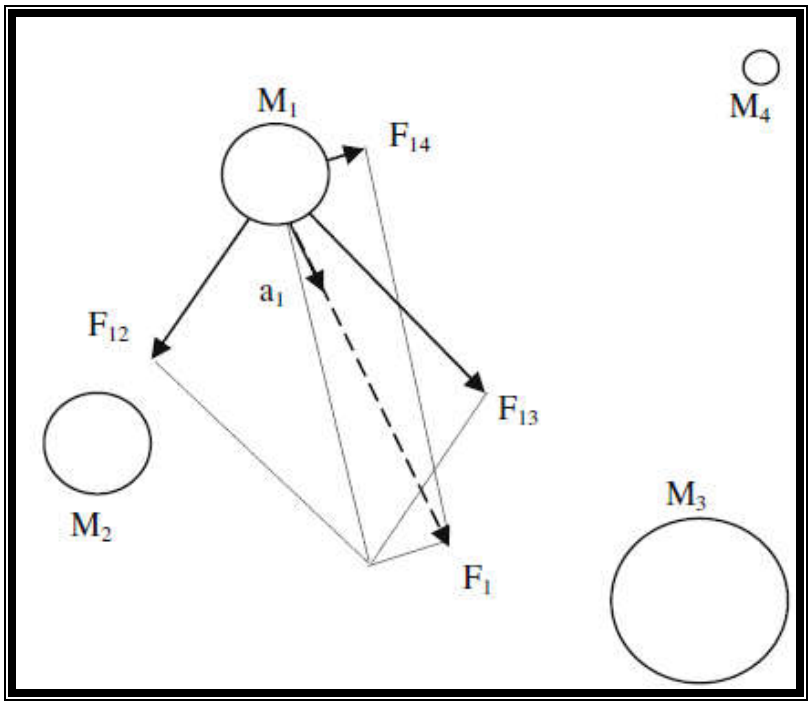

Fig. 1. Interaction of bodies in an isolated system

The actual value of the 'gravitational constant' depends on the actual age of the universe. This is represented as follows

$$G(t) = G(t_0) \times \left(\frac{t_0}{t} \right)^{\beta}, \quad \beta < 1 \quad (3)$$

where $G(t)$ is the value of the gravitational constant at time t. $G(t_0)$ is the value of the gravitational constant at the beginning, that is, at time t_0. Owing to the effect of decreasing gravity, equation (3) gives the decrease of the gravitational constant $G(t)$ with the age.
There are three kinds of masses defined in theoretical physics.

Active gravitational mass: M_a is a measure of the strength of the gravitational field owing to a particular object. Gravitational field of an object with large active gravitational mass is stronger than the object with less active gravitational mass.

Passive gravitational mass: M_P is a measure of the strength of an object's interaction with the gravitational field. Within the same gravitational field, an object with a larger passive gravitational mass experiences a larger force than an object with a smaller passive gravitational mass.

Inertial mass: M_i is a measure of an object's resistance to change its state of motion when a force is applied. An object with large inertial mass changes its motion more slowly than an object with small inertial mass.

As per aspects mentioned above, Newton's laws can be rewritten in the following ways:
The gravitational force, F_{ij} exerted by mass j on mass i is proportional to the product of the active gravitational of mass j and passive gravitational of mass i, and inversely proportional to the square distance between them.
Acceleration a_i is proportional to F_{ij} and inversely proportional to inertia mass M_{ii} of mass i. More precisely, one can rewrite (1) and (2) as follows

$$F_{ij} = G \times \left(\frac{\mathrm{M}_{aj}\mathrm{M}_{\mathrm{P}i}}{R^2} \right) \quad (4)$$

$$a_i = \frac{F_{ij}}{\mathrm{M}_{ii}} \quad (5)$$

where M_{aj} and M_{pi} are active gravitational mass of particle j and passive gravitational mass of particle i, respectively. M_{ii} represents the inertial mass of particle i.
Theoretically inertial mass, passive gravitational mass and active gravitational mass are distinct; however, no experiment has ever clearly demonstrated any difference between them.

The theory of general relativity is based on the assumption that inertial and passive gravitational masses are equivalent. Standard general relativity is based on the fact that inertial mass and active gravitational mass are the same. This equivalence is sometimes called the strong equivalent principle.

.

Esmat Rashedi et al. have proposed the Gravitational Search Algorithm (GSA) in 2009, which is based on the Newtonian Laws of Gravity involving the masses of bodies and the gravitational forces acting between them. In the algorithm, the performance of the objects is measured by their masses. Using gravitational force masses communicate with each other. The heavy masses correspond to good solutions and move more slowly than lighter ones. This step improves the exploitation ability of the algorithm. In GSA, mass of each object has four specifications: position, inertial mass, active gravitational mass and passive gravitational mass. Each mass is responsible to find out a solution of the optimization problem and its position corresponds to a solution of the problem. The algorithm is navigated by properly adjusting the gravitational and inertia masses that are determined using a fitness function.
Masses obey the following laws:

Law of gravity: Each particle attracts every other particle and the gravitational force between two particles is directly proportional to the product of their masses and inversely proportional to the distance R between them. After performing experiment several times, it has been observed that R provides better results than R^2. Therefore the term R^2 has been changed to R in (1).

Law of motion: The present velocity of any mass is equal to the sum of the fraction of its previous velocity and the variation in the velocity. Variation in the velocity is the acceleration of any mass which is equal to the force acted on the system divided by its inertial mass.

The position of any agent (mass) i, consisting of N number of masses may be defined as

$$\mathrm{X}_i = \left(x_i^1, x_i^2, ..., x_i^d, ..., x_i^n\right) \quad (6)$$

where x_i^d represents the position of the i th agent (mass) in the d th dimension and n is the dimension of the search space.

At any specific time 't', the force acting on mass 'i' because of mass 'j' may be represented as

$$F_{ij}^d(t) = G(t) \times \left(\frac{\mathrm{M}_{aj}(t) \times \mathrm{M}_{pi}(t)}{R_{ij}(t) + \varepsilon} \right) \times \left(x_j^d(t) - x_i^d(t) \right) \quad \mathbf{(7)}$$

where M_{aj} is the active gravitational mass related to agent j, $\mathrm{M}_{\mathrm{P}i}$ is the passive gravitational mass related to agent i, $G(t)$ is gravitational constant at time t and ε is a constant term whose magnitude is very small. $R_{ij}(t)$ is the Euclidian distance between two agents i and j that can be represented as

$$R_{ij}(t) = \left\| x_i(t), x_j(t) \right\|^2 \quad (8)$$

The total force that acts on agent i in a dimension d is a randomly weighted sum of d th components of the forces exerted (applied) from other agents

$$F_i^d(t) = \sum_{\substack{j=1 \\ j \neq i}}^{\mathrm{N}} rand_j \times F_{ij}^d(t) \quad (9)$$

where $rand_j$ is a random number in the interval [0, 1].

According to Newton's law of motion, the acceleration $a_i^d(t)$ of the agent i at time t, in the d th direction, is given by

$$a_i^d(t) = \frac{F_i^d(t)}{\mathrm{M}_{ii}(t)} \qquad (10)$$

where M_{ii} is the inertial mass of the i th agent.

Furthermore, the updated velocity of an agent is considered as sum of the fraction of its current velocity and its acceleration. Therefore its position and velocity could be calculated as follows

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t) \quad (11)$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \qquad (12)$$

The random number $rand_i$ gives a randomized characteristic to the search.

The gravitational constant G is initialized at the beginning and reduces with time to control the search accuracy. In other words, G is a function of the initial value G_0 and time t

$$G(t) = G(G_0, t) \qquad (13)$$

Value of G is modified as follows

$$G(t) = G_0 \times e^{-\frac{\alpha t}{Iter_{\max}}} \quad (14)$$

where G_0 is starting value of gravitational constant and $Iter_{\max}$ is the total number of iterations (the total age of system). α is a constant term.

Gravitational and inertia masses are simply calculated by the fitness evaluation. A heavier mass means a more efficient agent. Assuming the equality of the gravitational

and inertia masses, the values of masses are calculated using the map of fitness. The gravitational and inertial masses are updated by the following equations

$$\mathrm{M}_{ai} = \mathrm{M}_{\mathrm{P}i} = \mathrm{M}_{ii} = \mathrm{M}_i, \quad i = 1, 2, ..., \mathrm{N} \qquad (15)$$

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \qquad (16)$$

$$\mathrm{M}_i(t) = \frac{m_i(t)}{\sum_{j=1}^{\mathrm{N}} m_j(t)} \qquad (17)$$

where $fit_i(t)$ represents the fitness value of the agent i at time t, and $worst(t)$ and $best(t)$ are defined as follows:

For a minimization problem

$$best(t) = \min_{j \in \{1,2,...,\mathrm{N}\}} \left[fit_j(t) \right] \qquad (18)$$

$$worst(t) = \max_{j \in \{1,2,...,\mathrm{N}\}} \left[fit_j(t) \right] \qquad (19)$$

For a maximization problem

$$best(t) = \max_{j \in \{1,2,...,\mathrm{N}\}} \left[fit_j(t) \right] \qquad (20)$$

$$worst(t) = \min_{j \in \{1,2,...,\mathrm{N}\}} \left[fit_j(t) \right] \qquad (21)$$

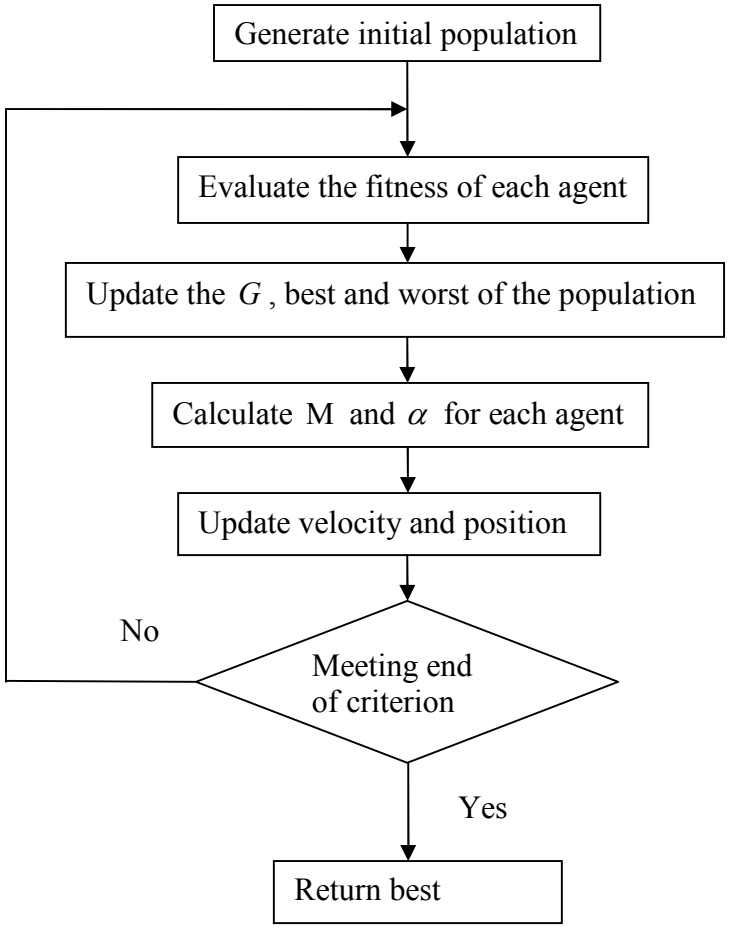

Fig. 2. Flowchart of GSA

2. Evolutionary computation

Evolutionary computing techniques also called evolutionary algorithms (EAs) mimic evolutionary processes encountered in nature. EAs are inspired by Darwin's evolutionary theory. The common conceptual base of these methods is to evolve a population of candidate solutions by simulating the main processes involved in the evolution of genetic material of organism (living being) populations, such as natural selection and biological evolution. EAs can be characterized as global optimization algorithms. Their population-based nature allows them to avoid getting trapped in a local optimum and consequently provides a great chance to find global optimal solutions.

In general, every EA starts by initializing a population of candidate solutions (individuals). The quality of each solution is evaluated using a fitness function. A selection process is applied at each iteration of the EA to produce a new set of solutions (population). The selection process is biased toward the most promising traits (qualities) of the current population of solutions to increase their chances of being included in the new population. At each iteration (generation), the individuals are evolved through a

predefined set of operators, like mutation and recombination. This procedure is repeated until convergence is reached. The best solution found by this procedure is expected to be a near-optimum solution.

Mutation and recombination are the two most frequently used operators and are referred to as evolutionary operators. The role of mutation is to modify an individual by small random changes to generate a new individual. Its main objective is to increase diversity by introducing new genetic material into the population, and thus avoid local optima. The recombination (or crossover) operator combines two, or more, individuals to generate new promising candidate solutions. The main objective of the recombination operator is to explore new areas of the search space.

Every iteration of the algorithm corresponds to a generation, where a population of candidate solutions to a given optimization problem, called individuals, is capable of reproducing and is subjected to genetic variations followed by the environmental pressure that causes natural selection (survival of the fittest). New solutions are created by applying recombination, that combines two or more selected individuals (the so-called parents) to produce one or more new individuals (the children or offspring). Mutation allows the appearance of new traits (qualities) in the offspring to promote diversity. Biologically, "mutation" means a sudden change in the gene characteristics of a chromosome. The fitness (how good the solutions are) of the resulting solutions is evaluated and a suitable selection strategy is then applied to determine which solutions will be maintained into the next generation or iteration. As a termination condition, a predefined number of generations or iterations (or function evaluations) of simulated evolutionary process is usually used, or some more complex stopping criteria can be applied.

A main issue in the application of EAs to a given optimization problem is to determine the values of the control parameters of the algorithm that will allow the efficient exploration of the search space, as well as its effective exploitation. Exploration enables the identification of regions of the search space in which good solutions are located. On the other hand, exploitation accelerates the convergence to the optimum solution. Inappropriate choice of the parameter values can cause the algorithm to become greedy or very explorative and consequently the search of the optimum can be hindered. For example, a high mutation rate will result in much of the space being explored, but there is also a high probability of losing promising solutions; the algorithm has difficulty in converging to an optimum due to insufficient exploitation.

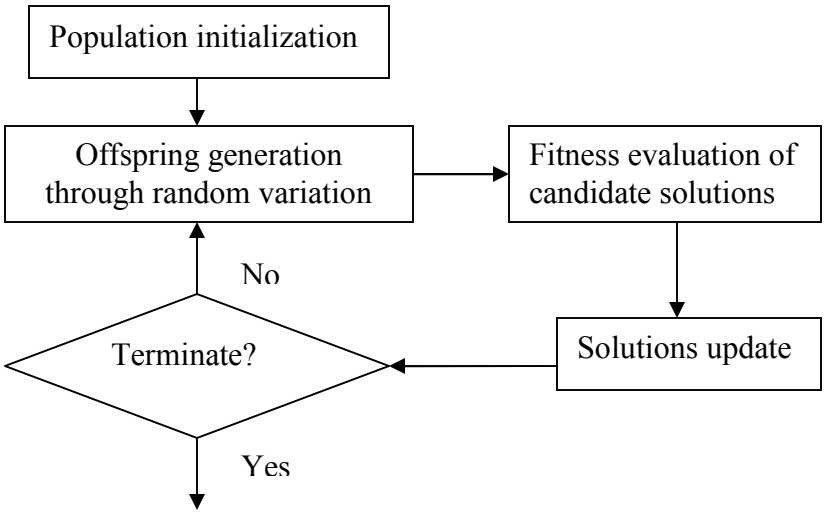

Fig. 3. A general flowchart of typical EAs

Evolutionary computing techniques (also called Evolutionary Algorithms (EAs)) include genetic algorithm, evolutionary programming and differential evolution

2.1. Genetic Algorithm

Genetic algorithm (GA) was originally developed in the early 1970s at the University of Michigan by John Holland and his students. GA is a global optimization technique and has become a candidate for many optimization applications due to its flexibility and efficiency. GA is a stochastic search algorithm based on the mechanics of nature (e.g. natural selection, survival of the fittest) and natural genetics. One of the advantages of GA is using stochastic operators instead of deterministic rules to search a solution. GA hops randomly from point to point, thus allowing it to escape from local optimum. Another attractive property of GA is that it searches for many optimum points in parallel. Genetic algorithm is considered to be robust method because no restriction on the solution space is made during the process.

It combines the adaptive nature of the natural genetics. By simulating "the survival of the fittest" of Darwinian evolution among chromosome structures, the optimal chromosome (solution) is searched by randomized information exchange. In every generation, a new set of artificial chromosomes is created. While randomized, GA is not a simple random walk. It efficiently exploits historical information to speculate (consider) on new search points with expected improved performance. The three prime operators associated with the GA are reproduction, crossover, and mutation.

This method has proved useful in domains that are not well understood, or for search spaces which are too large to be efficiently searched by standard methods. Each individual in the population represents a possible solution to the problem. A ‘fitness’ value, derived from the problem’s objective function, is assigned to each member of the population. Individuals that represent better solutions are awarded higher fitness values, thus enabling them to survive more generations.

Individuals for producing offspring are chosen using a selection strategy after evaluating the fitness value of each individual in the population. Some of the popular selection schemes are roulette-wheel selection, tournament selection, ranking selection etc.
After crossover, individuals are subjected to mutation. Mutation introduces some randomness into the search to prevent the optimization process from getting trapped into local optima. It is usually considered as a secondary genetic operator that performs a slight perturbation to the resulting solutions. The replacement (survivor selection) uses the fitness value to identify the individuals to maintain as parents for successive generations and is responsible to assure the survival of the fittest individuals.

Due to difficulties of binary representation when dealing with continuous search space with large dimensions, the proposed approach has been implemented using real-coded genetic algorithm (RCGA). The simulated Binary Crossover (SBX) and polynomial mutation are explained as follows.

Simulated Binary Crossover (SBX) operator
The procedure of computing child populations c_1 and c_2 from two parent populations y_1 and y_2 under SBX operator as follows:
1. Create a random number *u* between 0 and 1.
2. Find a parameter γ using a polynomial probability distribution as follows:

$$\gamma = \begin{cases} (u\gamma)^{1/(\eta_c+1)}, & \text{if} \quad u \le \dfrac{1}{\gamma} \\ (1/(2-u\gamma))^{1/(\eta_c+)}, & \text{otherwise} \end{cases} \tag{22}$$

where $\gamma = 2 - \delta^{-(\eta_c+1)}$ and $\delta = 1 + \dfrac{2}{y_2 - y_1}\min[(y_1 - y_l),(y_u - y_2)]$

Here, the parameter y is assumed to vary in $[y_l, y_u]$. Here, the parameter η_c is the distribution index for SBX and can take any non-negative value. A small value of η_c allows the creation of child populations far away from parents and a large value restricts only near-parent populations to be created as child populations.

3. The intermediate populations are calculated as follows:

$$c_{p1} = 0.5[(y_1 + y_2) - \gamma(|y_2 - y_1|)]$$
$$c_{p2} = 0.5[(y_1 + y_2) + \gamma(|y_2 - y_1|)] \tag{23}$$

Each variable is chosen with a probability p_c and the above SBX operator is applied variable-by-variable.

Polynomial Mutation Operator

A polynomial probability distribution is used to create a child population in the vicinity of a parent population under the mutation operator. The following procedure is used:

1. Create a random number u between 0 and 1.
2. Calculate the parameter δ as follows:

$$\delta = \begin{cases} \left[2u + (1-2u)(1-\phi)^{(\eta_m+1)}\right]^{\frac{1}{(\eta_m+1)}} - 1, & \text{if } u \le 0.5 \\ 1 - \left[2(1-u) + 2(u-0.5)(1-\phi)^{(\eta_m+1)}\right]^{\frac{1}{(\eta_m+1)}}, & \text{otherwise} \end{cases} \quad (24)$$

where $\varphi = \dfrac{\min\left[(c_p - y_l), (y_u - c_p)\right]}{(y_u - y_l)}$

The parameter η_m is the distribution index for mutation and takes any non-negative value.

3. Calculate the mutated child as follows:

$c_1 = c_{p1} + \delta(y_u - y_l)$

$c_2 = c_{p2} + \delta(y_u - y_l)$

The perturbance in the population can be adjusted by varying η_m and p_m with generations as given below:

$$\eta_m = \eta_{m\min} + gen \quad (25)$$

$$p_m = \frac{1}{D} + \frac{gen}{gen_{\max}}\left(1 - \frac{1}{D}\right) \quad (26)$$

where $\eta_{m\min}$ is the user defined minimum value for η_m, p_m is the probability of mutation, and D is the number of decision variables

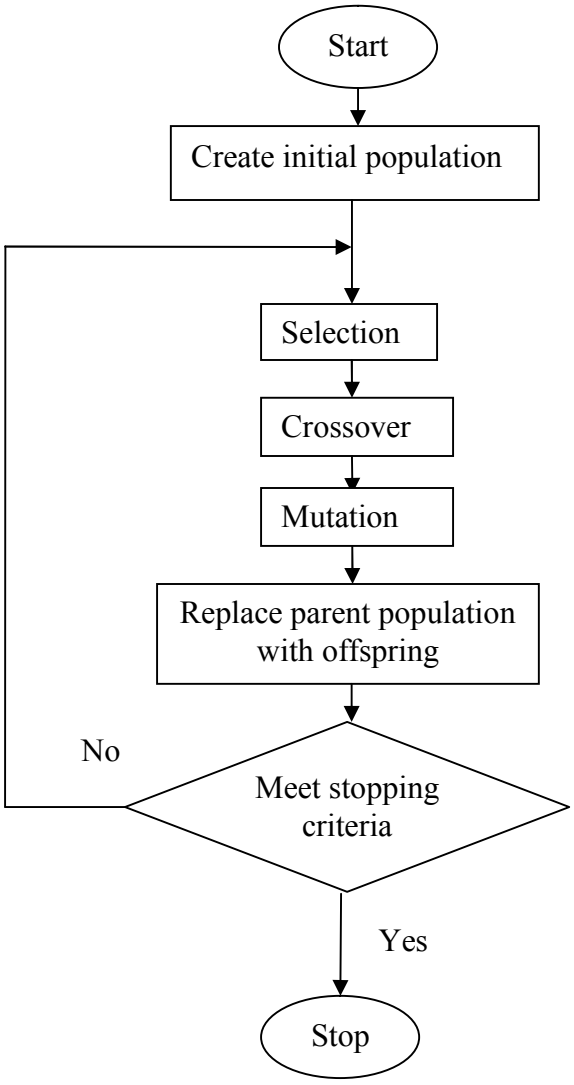

Figure 4. Flowchart of genetic algorithm

2.2. Evolutionary Programming

Evolutionary Programming (EP) was first presented in the 1960s by L. J. Fogel as an evolutionary approach to artificial intelligence. Later, in the early 1990s, EP was reintroduced by D. Fogel to solve more general tasks including prediction problems, numerical and combinatorial optimization, and machine learning.

Evolutionary programming is a probabilistic search technique. Evolutionary Programming seeks the optimal solution of an optimization problem by evolving a population of candidate solutions over a number of generations or iterations. The main stages of this technique are initialization, creation of off-spring by mutation and competition and selection. In contrast to the GA, the EP does not rely on any kind of recombination or crossover. The mutation is the only operator used to generate offspring. It generates the initial parent vectors distributed uniformly in intervals within the limits. A new population is formed from an existing population through the use of a mutation

operator. This operator perturbs each component of every solution in the population by a random amount to produce new solutions. The degree of optimality of each of the new candidate solutions or individuals is measured by its fitness which can be defined as a function of the objective function of the problem. Through the use of a competition scheme, individuals in population compete with each other. The winning individuals will form a resultant population which is regarded as the next generation. For optimization to occur, the competition scheme must be such that the more optimal solutions have a greater chance of survival than the poorer solutions. Through this the population evolves towards the global optimal point. The survivor selection process (replacement) is probabilistic and is based on a stochastic tournament selection.
In the case of standard EP, the normally distributed random mutation is applied. In fast evolutionary programming (FEP) Cauchy-distribution mutation is applied. In improved fast evolutionary programming (IFEP), two offspring are created from each parent, one using a Gaussian distribution, and the other using the Cauchy distribution.
The EP technique is iterative and the process is terminated by a stopping rule. The rule widely used is either (a) stop after a specified number of iterations or (b) stop when there is no appreciable change in the best solution for a certain number of iterations.
EP is different from other optimization methods in the following features

1. EP searches from a population of points, not single point. The population can move over hills and across valleys. EP can therefore discover a globally optimal point. As the computation for each individual in the population is independent of others, EP has inherent parallel computation ability.
2. EP uses payoff (fitness or objective functions) information directly for the search direction, not derivatives. EP therefore can deal with non-smooth, non-continuous and non-differentiable functions that are the real-life optimization problems.
3. EP uses probabilistic transition rules to select generations, not deterministic rules, so they can search a complicated and uncertain area to find the global optimum.

The procedure can be stated as:

Initialization: The initial population (X_i) of control variables chosen randomly from the set of uniformly distributed control variables ranging over their maximum and minimum limits can be stated as:

$$x_{i,j} \sim U\left(x_j^{\min}, x_j^{\max}\right), \quad j \in n, i \in \mathrm{N_P} \tag{27}$$

where n is the number of decision variables in a population, $\mathrm{N_P}$ is the population size; $x_{i,j}$ signifies the initial j th variable of the i th population ; $x_j^{\min}$ and $x_j^{\max}$ are the minimum and maximum limits of the j th decision variable. $U\left(x_j^{\min}, x_j^{\max}\right)$ signifies a uniform random variable ranging over $\left[x_j^{\min}, x_j^{\max}\right]$.

Compute the fitness f_i of each population and the maximum fitness $f_{\max}$.

Mutation: Each parent population (X_i) enduring mutation creates the mutated population (X_{i+m}) stated as:

$$x_{i+m,j} = x_{i,j} + \sigma_j \times N(0,1), j \in n, i \in N_P, m \in N_P \quad (28)$$

$$\sigma_j = \beta \times \frac{f_i}{f_{\max}} \times \left(x_j^{\max} - x_j^{\min}\right), j \in n \quad (29)$$

where $N(0,1)$ signifies Gaussian random variable with mean 0 and standard deviation 1; $f_{\max}$ is the maximum fitness of the last iteration and β is the scaling factor. Compute the fitness of each mutated population.

Competition and selection: Total $2N_P$ combined populations have to contend with each other to acquire their aperture for the next iteration. The weight value w_i of the i th population can be stated as:

$$w_i = \sum_{t=1}^{N_t} w_{i,t} \quad (30)$$

where N_t is the number of competition created randomly; $w_{i,t}$ is 1 or 0 based on the i th population fitness (f_i) is lower than that of the fitness (f_r) or equal to or more than that of the fitness (f_r) of a randomly assortment (r th) population in the combined $2N_P$ populations. The value of $w_{i,t}$ can be stated as:

$$w_{i,t} = \begin{cases} 1 & \text{if } f_i < f_r \\ 0 & \text{otherwise} \end{cases} \quad (31)$$

The $2N_P$ populations are graded in a descending order in proportion to their weight value w_i and the first N_P populations are chosen for the next iteration.

Figure 5 shows the flowchart of evolutionary programming.

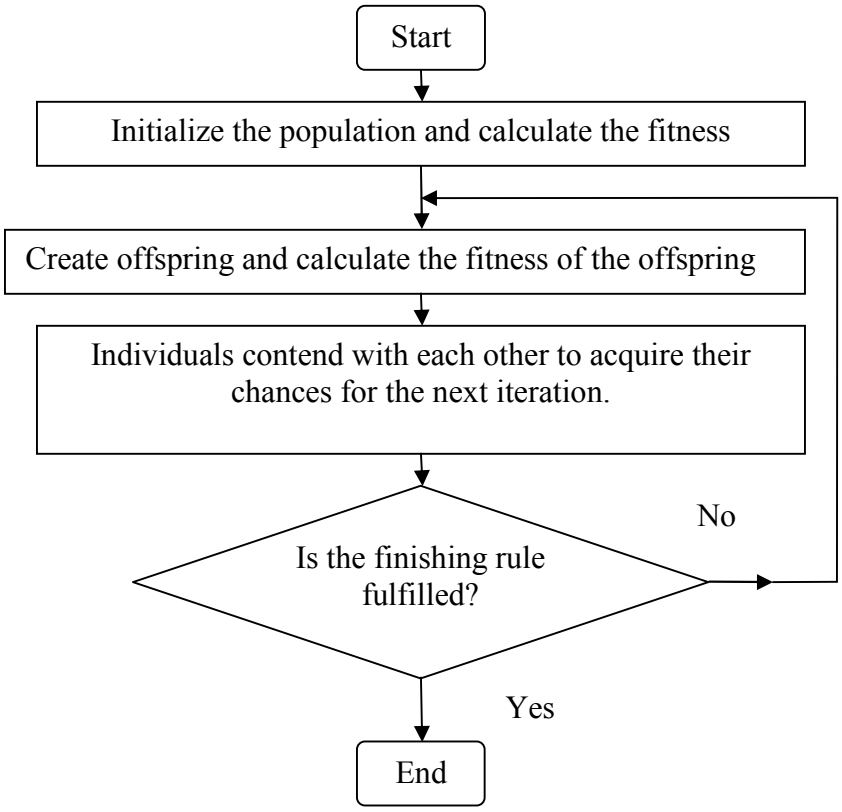

Fig. 5. Flowchart of evolutionary programming

2.3. Differential Evolution

Differential Evolution (DE) was originally proposed by Price and Storn in 1995-1997 at Berkley. DE is a population-based stochastic parallel direct search method for optimization problems over a continuous domain. DE is simple, significantly faster, precise and robust.

The basic idea of DE is to adapt the search during the evolutionary process. At the start of the evolution, the perturbations are large since parent populations are far away from each other. As the evolutionary process matures, the population converges to a small region and the perturbations adaptively become small. As a result, DE performs a global exploratory search during the early stages of the evolutionary process and local exploitation during the mature stage of the search. The optimization process in DE is carried out with three basic operations: mutation, crossover and selection.

Like any evolutionary algorithm, a population of candidate solutions for the optimization problem to be solved is arbitrarily initialized. For each generation of the evolutionary process, new individuals are created by applying crossover and mutation. The fitness of the resulting solutions is evaluated. Each individual (target individual) of the population competes against a new individual (trial individual) to determine which one will be maintained into the next generation. The trial individual is created by recombining the target individual with another individual created by mutation (called mutant individual).

In DE the fittest of an offspring competes one-to-one with that of corresponding parent which is different from other evolutionary algorithms. This one-to-one competition gives rise to faster convergence rate. Differential evolution exploits the differences of randomly sampled pairs of objective vectors for its mutation process. Consequently the variation between vectors will outfit the objective function toward the optimization process and therefore provides efficient global optimization capability. The key parameters of control in DE are population size, scaling factor and crossover rate. Price and Storn gave the working principle of DE with simple strategy in. Later on, they suggested ten different strategies of DE. Strategy-7 (DE/rad/1/bin) is the most successful and widely used strategy.

1. Initialization

The initial population of $\mathrm{N_P}$ vectors is randomly selected based on uniform probability distribution for all variables to cover the entire search uniformly. Each individual X_i is a vector that contains as many parameters as the problem decision variables D. Random values are assigned to each decision parameter in every vector according to:

$$\mathrm{X}_{ij}^{0} \sim U\left(\mathrm{X}_{j}^{\min}, \mathrm{X}_{j}^{\max}\right) \tag{32}$$

where i=1,…., $\mathrm{N_P}$ and j=1,…., D; $\mathrm{X}_{j}^{\min}$ and $\mathrm{X}_{j}^{\max}$ are the lower and upper bounds of the j th decision variable; $U\left(\mathrm{X}_{j}^{\min}, \mathrm{X}_{j}^{\max}\right)$ denotes a uniform random variable ranging over $\left[\mathrm{X}_{j}^{\min}, \mathrm{X}_{j}^{\max}\right]$. X_{ij}^{0} is the initial j th variable of i th population. All the vectors should satisfy the constraints. Evaluate the value of the cost function $f\left(\mathrm{X}_{i}^{0}\right)$ of each vector.

2. Mutation

DE generates new parameter vectors by adding the weighted difference vector between two population members to a third member. For each target vector X_{i}^{g} at g th generation the noisy vector $\mathrm{X}_{i}^{/g}$ is obtained by

$$\mathrm{X}_{i}^{/g} = \mathrm{X}_{a}^{g} + F\left(\mathrm{X}_{b}^{g} - \mathrm{X}_{c}^{g}\right), \qquad i \in \mathrm{N_P} \tag{33}$$

where X_{a}^{g}, X_{b}^{g} and X_{c}^{g} are selected randomly from $\mathrm{N_P}$ vectors at g th generation and $a \neq b \neq c \neq i$. The scaling factor (F), in the range $0 < F \leq 1.2$, controls the amount of perturbation added to the parent vector. The noisy vectors should satisfy the constraint.

3. Crossover

Perform crossover for each target vector X_{i}^{g} with its noisy vector $\mathrm{X}_{i}^{/g}$ and create a trial vector $\mathrm{X}_{i}^{//g}$ such that

$$X_i^{//g} = \begin{cases} X_i^{/g} \text{ , if } \rho \le C_R \\ \\ X_i^{g} \text{ , otherwise} \end{cases}, \qquad i \in N_P \tag{34}$$

where ρ is an uniformly distributed random number within [0, 1]. The crossover constant (C_R), in the range $0 \le C_R \le 1$, controls the diversity of the population and aids the algorithm to escape from local optima.

4. Selection

Perform selection for each target vector, X_i^g by comparing its cost with that of the trial vector, $X_i^{//g}$. The vector that has lesser cost of the two would survive for the next generation.

$$X_i^{g+1} = \begin{cases} X_i^{//g} \text{ , if } f\left(X_i^{//g}\right) \le f\left(X_i^{g}\right) \\ \\ X_i^{g} \text{ , otherwise} \end{cases}, \qquad i \in N_P \tag{35}$$

The process is repeated until the maximum number of generations or no improvement is seen in the best individual after many generations.

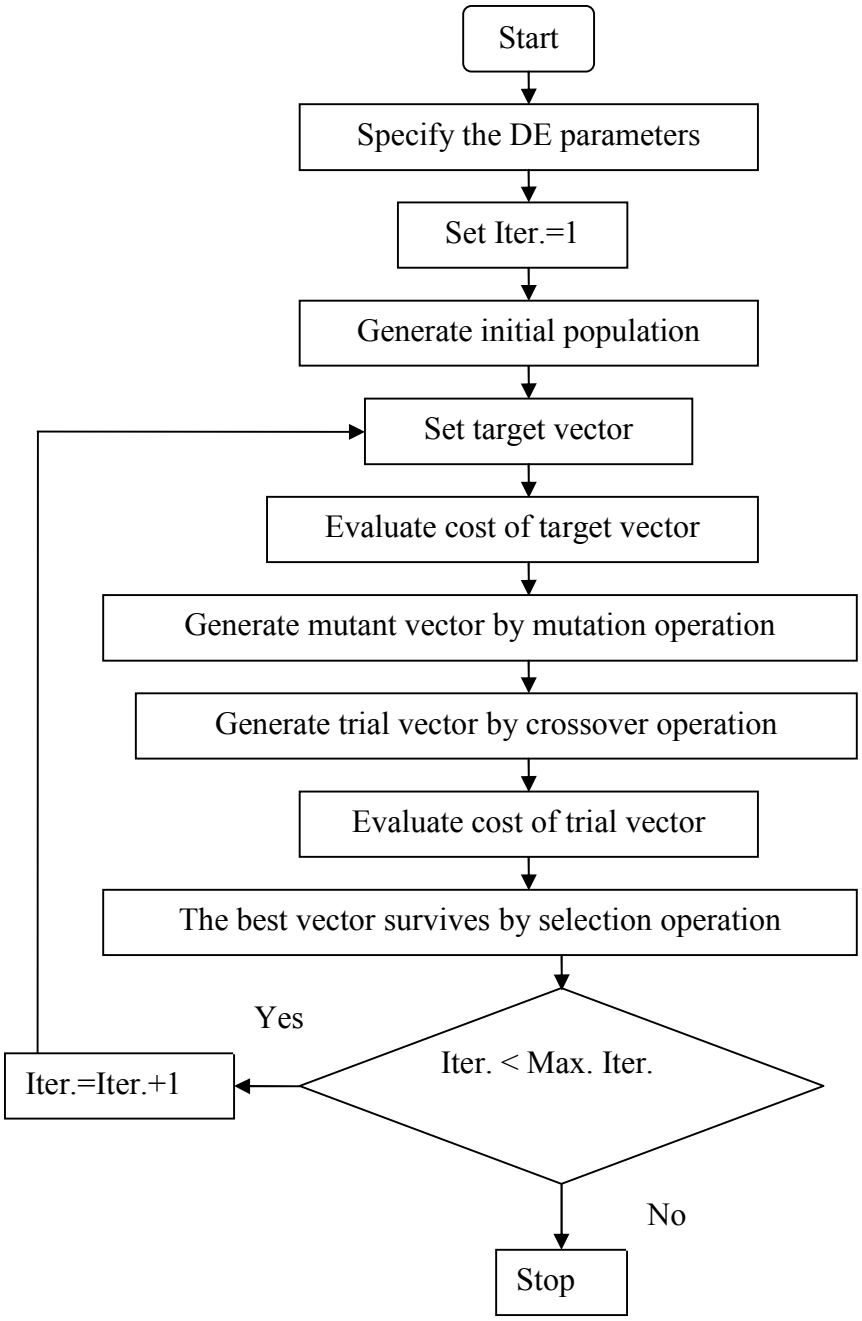

Fig. 6. Flowchart of Differential Evolution

In differential evolution, scaling factor (S_F), crossover constant (C_R), and population size (N_P) are the three control parameters. Proper selection of control parameters is important for the performance of the algorithm. The scaling factor controls the amount of perturbation in the mutation process. Its value lies in the range of [0, 2]. Lower value of S_F results premature convergence while higher value of S_F tends to slow down convergence speed. The crossover constant whose value is in the range of [0, 1], controls

the diversity of the population. The diversity for searching the solution space depends on the population size. However, large population slows down convergence speed.

The optimal control parameters are problem dependent. Generally, parameter tuning is used to select control parameters. Through testing, parameter tuning adjusts the control parameters until the best settings are determined.

The choice of numerical values for the three control parameters , , and highly depends on the problem under consideration. In fact, the population size does not need to be fine-tuned and just a few typical values can be tried according to the preestimated complexity of the given problem. Between other two parameters, C_R is usually more sensitive to problems with different characteristics, e.g., the unimodality and multimodality, while F is closely related to the convergence speed.
The proper choice of C_R can lead to successful optimization performance while a wrong choice may deteriorate the performance. In fact, good values of C_R generally fall into a small range for a given problem, with which the algorithm can perform consistently well. Therefore, we consider gradually adjusting the range of C_R values for a given problem according to previous C_R values that have generated trial vectors successfully entering the next generation.

3. Swarm intelligence

Swarm Intelligence (SI) was introduced by Gerardo Beni and Jing Wang in 1989. It is the study of the collective behavior of different natural system which consists of number of agents working together.
Swarm Intelligence (SI) is an innovative (novel or new) distributed intelligent paradigm (model) for solving optimization problems. It takes inspiration from the collective behavior of a group of social insect colonies and of other animal societies. SI is typically made up of a population of simple agents interacting locally with one another and with their environment. These entities (creatures) with very limited individual capability can jointly (cooperatively) perform many complex tasks necessary for their survival. Although there is normally no centralized control structure dictating how individual agent should behave, local interactions between such agents often lead to the emergence of global and self-organized behavior.
Several optimization algorithms inspired by the metaphors (descriptions) of swarming behavior in nature are proposed. Ant colony optimization, particle swarm optimization, bacterial foraging algorithm, cuckoo search algorithm, group search optimization, bee colony optimization, firefly algorithm and biogeography-based optimization are examples to this effect.

3.1. Particle Swarm optimization

Particle Swarm Optimization (PSO) was introduced in 1995 by James Kennedy and Russell Eberhart as a global optimization technique. It is a population-based self-adaptive stochastic search technique with reduced memory requirement. The PSO method is becoming very popular due to its simplicity of implementation and ability to quickly converge to a reasonably good solution.

PSO is motivated by social behavior of organisms such as fish schooling and bird flocking. In PSO, individuals called particles change their positions or states with time and fly around in a multidimensional search space. During flight, each particle adjusts its position according to its own experience, and the experience of neighboring particles, making use of the best position encountered by itself and its neighbors. The swarm (group or crowd or flock) direction of a particle is defined by the set of particles neighboring the particle and its history experience.
In the multidimensional space where the optimal solution is sought, each particle in the swarm is moved toward the optimal point by adding a velocity with its position. The velocity of a particle is influenced by three components, namely, inertial, cognitive and social. The inertial component simulates the inertial behavior of the bird to fly in the previous direction. The cognitive component models the memory of the bird about its previous best position and the social component models the memory of the bird about the best position among the particles. The particles move around the multidimensional search space until they find the food i.e. optimal solution.
The particle swarm optimization starts with the random initialization of a population of individuals called particles in the search space. Each particle is a candidate solution to the problem, and is represented by a velocity, a location in the search space and has a memory which helps it in remembering its previous best position. However, unlike in other evolutionary optimization methods, in PSO there is no direct recombination of genetic material between individuals during the search. The PSO algorithm works on the social behavior of particles in the swarm. PSO finds the global best solution by simply adjusting the trajectory of each individual toward its own best location and toward the best particle of the entire swarm at each time step (generation).
Each particle moves based on its previous velocity, the particle's location at which the best fitness so far has been achieved, and the population global location at which the best fitness so far has been achieved. The particles have a tendency to move toward better search areas over the course of a search process.

[It has a well balanced mechanism with flexibility to enhance both global-searching ability at the beginning and a local search near the end of the run. Two commonly used PSOs are the global version and the local version. This is achieved by dynamically adjusting an inertia weight over the course of a PSO run. The global version of PSO converges fast, but with the potential to converge to a local minimum, while the local version of PSO might have more chances to find better solutions slowly.]

After each iteration the new velocity and hence the new position of each particle is updated on the basis of a summated influence of each particle's present velocity, distance of the particle from its own best performance, achieve so far during the search process and the distance of the particle from the leading particle, i.e. the particle which at present is globally the best particle producing till now the best performance.

Let x and v denote a particle position and its corresponding velocity in a search space, respectively. Therefore, the i th particle is represented as $x_i = (x_{i1}, x_{i2}, ..., x_{id})$ in the d dimensional space. The best previous position of the i th particle is recorded and represented as $pbest_i = (pbest_{i1}, pbest_{i2}, ..., pbest_{id})$. The index of the best particle

among all the particles in the group is represented by the $gbest_d$. The rate of the velocity for the particle i is represented as $v_i = (v_{i1}, v_{i2}, ..., v_{id})$. The modified velocity and position of each particle can be calculated using the current velocity and the distance from $pbest_{id}$ to $gbest_d$ as shown in the following formulas:

$$v_{id}^{(k+1)} = w * v_{id}^{(k)} + c_1 * rand(\ ) * (pbest_{id} - x_{id}^k) + c_2 * rand(\ ) * (gbest_d - x_{id}^k) \quad (36)$$

$$x_{id}^{(k+1)} = x_{id}^k + v_{id}^{(k+1)},\ i = 1,2,..., \mathrm{N_P},\ d = 1,2,..., \mathrm{N}_g \quad (37)$$

where
$\mathrm{N_P}$: number of particles in the swarm
N_g: number of members in a particle
k: pointer of iterations
w: inertia weight factor
c_1, c_2: acceleration constant
$rand(\ )$: uniform random value in the range [0,1]
$v_i^{(k)}$: velocity of a particle i at iteration k, $v_d^{\min} \le v_{id}^k \le v_d^{\max}$
x_i^k: current position of a particle i at iteration k

In the above procedures, the parameter $v^{\max}$ determined the resolution, with which regions are to be searched between the present position and the target position. If $v^{\max}$ is too high, particles might fly past good solutions. If $v^{\max}$ is too small, particles may not explore sufficiently beyond local solutions.
The constants c_1 and c_2 represent the weighting of the stochastic acceleration terms. Constant c_1 pulls the particle towards local best position (*pbest*) whereas c_2 pulls it towards the global best position (*gbest*). Low values allow particle to roam far from the target regions before being tugged back. On the other hand, high values result in abrupt movement toward or past, target regions. Hence, the acceleration constants c_1 and c_2 were often set to be 2.0 according to past experiences.
Suitable selection of inertia weight w provides a balance between global exploration and local exploitation thus requiring less number of iterations to find a sufficiently optimal solution. A larger inertial weight is used during initial exploration and its value is gradually reduced as the search proceeds. As originally developed, w often decreases linearly from about 0.3 to –0.2 during a run. In general, the inertia weight w is set according to the following equation:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{iter_{\max}} \times iter \quad (38)$$

where $iter_{\max}$ is the maximum number of iterations and $iter$ is the current number of iterations.

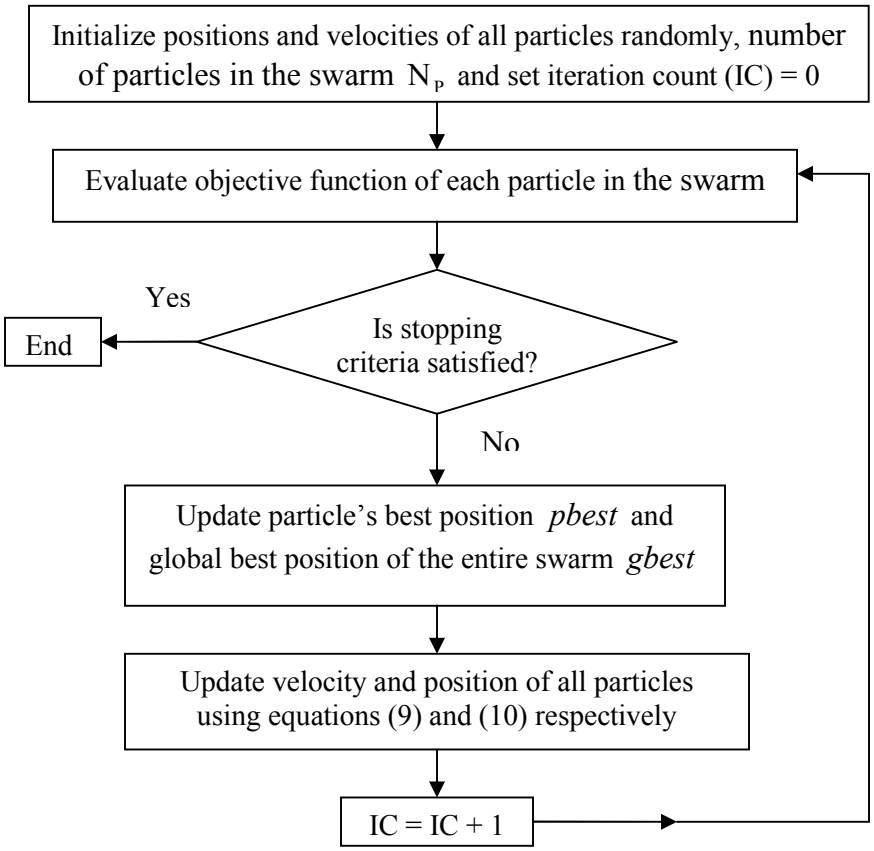

Fig. 7. Flowchart of particle swarm optimization

3.2. Cuckoo Search Algorithm

Cuckoo search is a nature-based meta-heuristic search technique originally proposed by Yang and Deb in 2009. It is inspired from the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds of other species.

Cuckoo Breeding Behavior

Cuckoos are fascinating birds, not only because of beautiful sounds they make, but also because of their aggressive reproduction strategy. Some species like the ani & Guira cuckoos lay their eggs in communal nests and remove others eggs so that the probability of hatching of their own eggs increases (Payne et al 2005). Some cuckoos like new world brood parasitic tapera lays eggs of such color and pattern which matches with some specific host birds. This reduces the probability of its eggs being abandoned and increases the chances of its reproduction. The time of laying eggs of some cuckoo species is quite amazing. Parasitic cuckoos often choose a nest where the host bird just laid its own eggs. After the cuckoo chick hatches, its first instinct (nature) will be evicting (throwing out) the host eggs by blindly propelling (forcing) out the eggs of the nest. This increases its chance of getting more share of food. There are three basic types of brood parasitism: intraspecific brood parasitism, cooperative breeding, and nest takeover. If a host bird

finds eggs in its nest which are not its own, it can do two things, it may either throw away the alien egg from its nest or abandon the nest and build a new nest elsewhere.

Lévy Flight

L´evy Flight is a random walk of step lengths having direction of the steps as isotropic and random. It is very useful in stochastic measurements and simulations of random and pseudo-random phenomena. In practical world, it is seen that when sharks and other predators cannot procure food, they abandon Brownian motion (i.e. the random motion seen in swirling gas molecules) and adopt Lévy flight. Lévy flight is a mix of long trajectories and short random movements. Birds and other animals such as insects seem to follow Lévy flights when searching for food. A recent study by Reynolds and Frye shows that fruit flies or Drosophila melanogaster; explore their landscape using a series of straight flight paths punctuated by a sudden 90 degrees turn, leading to a L´evy-flight-style intermittent scale free search pattern. Studies on human behavior such as the Ju'hoansi hunter-gatherer foraging patterns also show the typical feature of L´evy flights. Even light can be related to L´evy flights.

Assumptions

Before understanding Cuckoo Search Algorithm as described by Yang and Deb, three idealized rules are kept in mind.

1) Each Cuckoo lays one egg at a time and dumps it to a randomly chosen nest.
2) The best nests with high quality of eggs will carry over to the next generations.
3) The number of available host nests is fixed and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in [0,1]$.

Steps of Cuckoo Search Algorithm

Step 1: Initialization

A population of N_P host nests is represented by $X = [X_1, X_2, ..., X_{N_P}]^T$ where each nest $X_i = [x_{i1}, x_{i2}, ..., x_{ij}, .., x_{iN}]$. The individual is initialized by:

$$x_{ij} = x_{ij}^{\min} + rand_1 * (x_{ij}^{\max} - x_{ij}^{\min})$$

where $rand_1$ is a uniformly distributed random number between 0 and 1.

Step 2: Generation of new solution via Levy flights

The new solution is calculated based on the previous best nests via Levy flights. In the proposed method, the optimal path for the Levy flights is calculated by Mantegna's algorithm. The new solution for each nest is calculated as follows:

$$X_i^{new} = Xbest_i + \alpha \times rand_2 \times \Delta X_i^{new}$$

where α > 0 is the updated step size and $rand_2$ is a normally distributed stochastic number. ΔX_i^{new} is calculated as follows:

$$\Delta X_i^{new} = v \times \frac{\sigma_x(\beta)}{\sigma_y(\beta)} \times \left(X best_i - Gbest\right)$$

$$v = \frac{rand_x}{\left|rand_y\right|^{1/\beta}}$$

where $X best_i$ is the best value of nest i and $Gbest$ is the best nest of all nests in the population. $rand_x$ and $rand_y$ are two normally distributed stochastic variables with standard deviation $\sigma_x(\beta)$ and $\sigma_y(\beta)$ given by

$$\sigma_x(\beta) = \left[\frac{\Gamma(1+\beta) \times \sin\left(\frac{\Pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{\left(\frac{\beta-1}{2}\right)}}\right]^{1/\beta}$$

$$\sigma_y(\beta) = 1$$

where β is the distribution factor $0.3 \le \beta \le 1.99$ and $\Gamma(.)$ is the gamma distribution function.
The newly obtained solution should be satisfied according to its lower and upper limits.

Step 3: Alien egg discovery and randomization
The action of discovery of an alien egg in a nest of a host bird with the probability of p_a also creates a new solution for the problem similar to the Levy flights. The new solution due to this action can be found out in the following way.

$$X_i^{dis} = X best_i + K \times \Delta X_i^{dis}$$

where K is the updated coefficient determined based on the probability of a host bird to discover an alien egg in its nest:

$$K = \begin{cases} 1 & if rand_3 < p_a \\ 0 & otherwise \end{cases}$$

The increased value ΔX_i^{dis} is determined by

$$\Delta X_i^{dis} = rand_3 \times \left[randp_1\left(X best_i\right) - randp_2\left(X best_i\right)\right]$$

where $rand_3$ is the distributed random number in [0, 1]. $randp_1(\mathrm{X}best_i)$ and $randp_2(\mathrm{X}best_i)$ are the random perturbation for positions of nests in $\mathrm{X}best_i$.

Step 4: Stopping criteria

The above algorithm is stopped when the number of iterations reaches the predefined value. The flowchart of CSA is shown in Fig. 8.

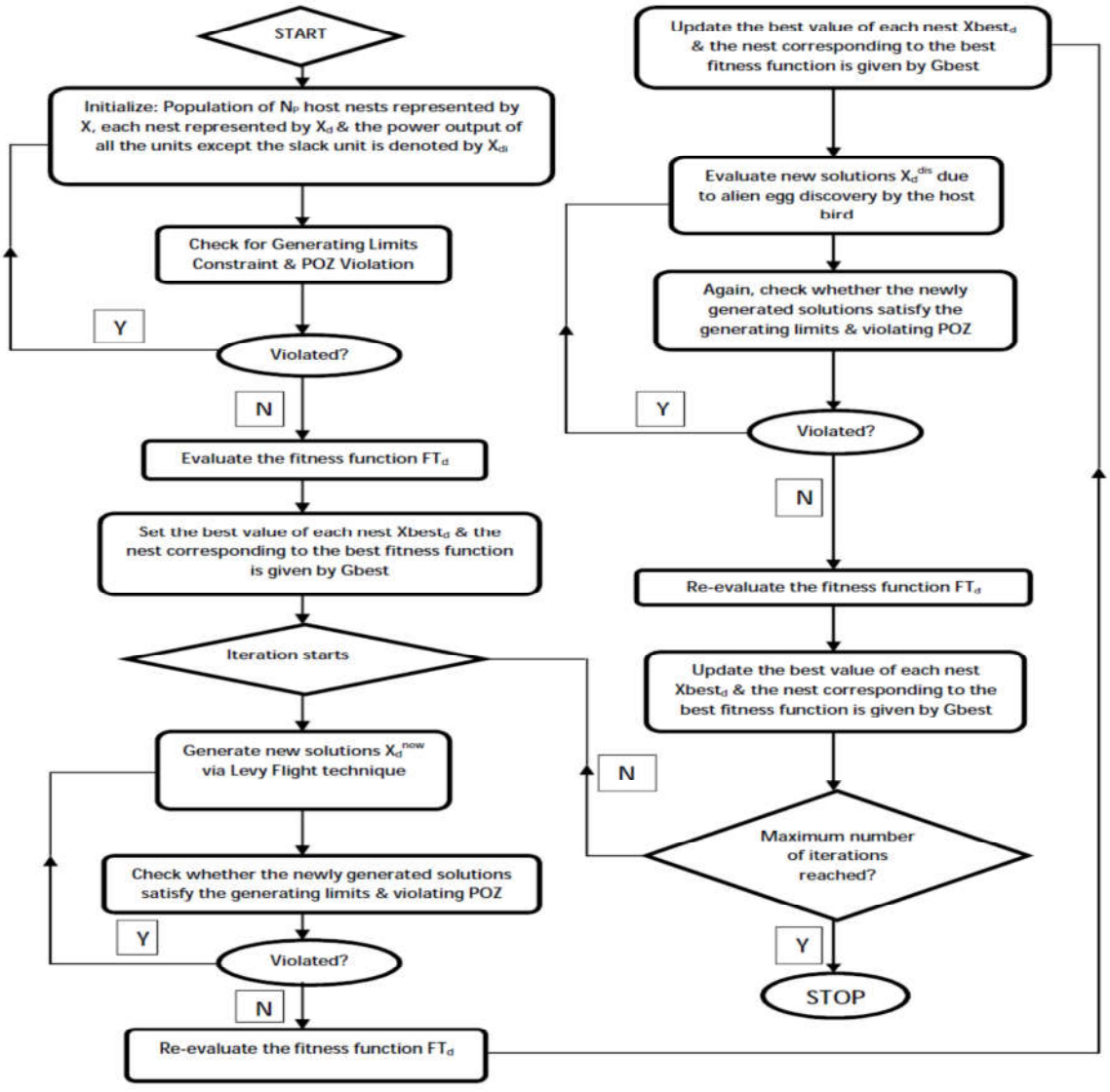

Fig.8. Flowchart for Cuckoo Search Algorithm

3.3. Artificial Bee Colony Optimization

Artificial bee colony optimization (ABCO) was originally developed by Karaboga in 2005. This algorithm mimics the food foraging behavior of honey bees. In ABCO, the

colony of bees consists of two groups, scout and employed bees. The scout bees search for a new food source and the employed bees try to find a food source within the neighborhood of the food source in their memories.

Fig. 9 shows the flowchart of Bee Colony Optimization algorithm. The algorithm starts with n_s scout bees randomly distributed in the search space. The nectar amounts of sites visited by n_s scout bees are calculated. Sites (m) that have the highest nectar amounts are chosen for neighborhood search. Recruit n_b bees for each selected site to explore neighborhood search. The nectar amounts of all $(n_b \times m)$ sites are calculated. Select m sites which have the highest nectar amounts from $(n_b \times m)$ sites to form the next bee population.

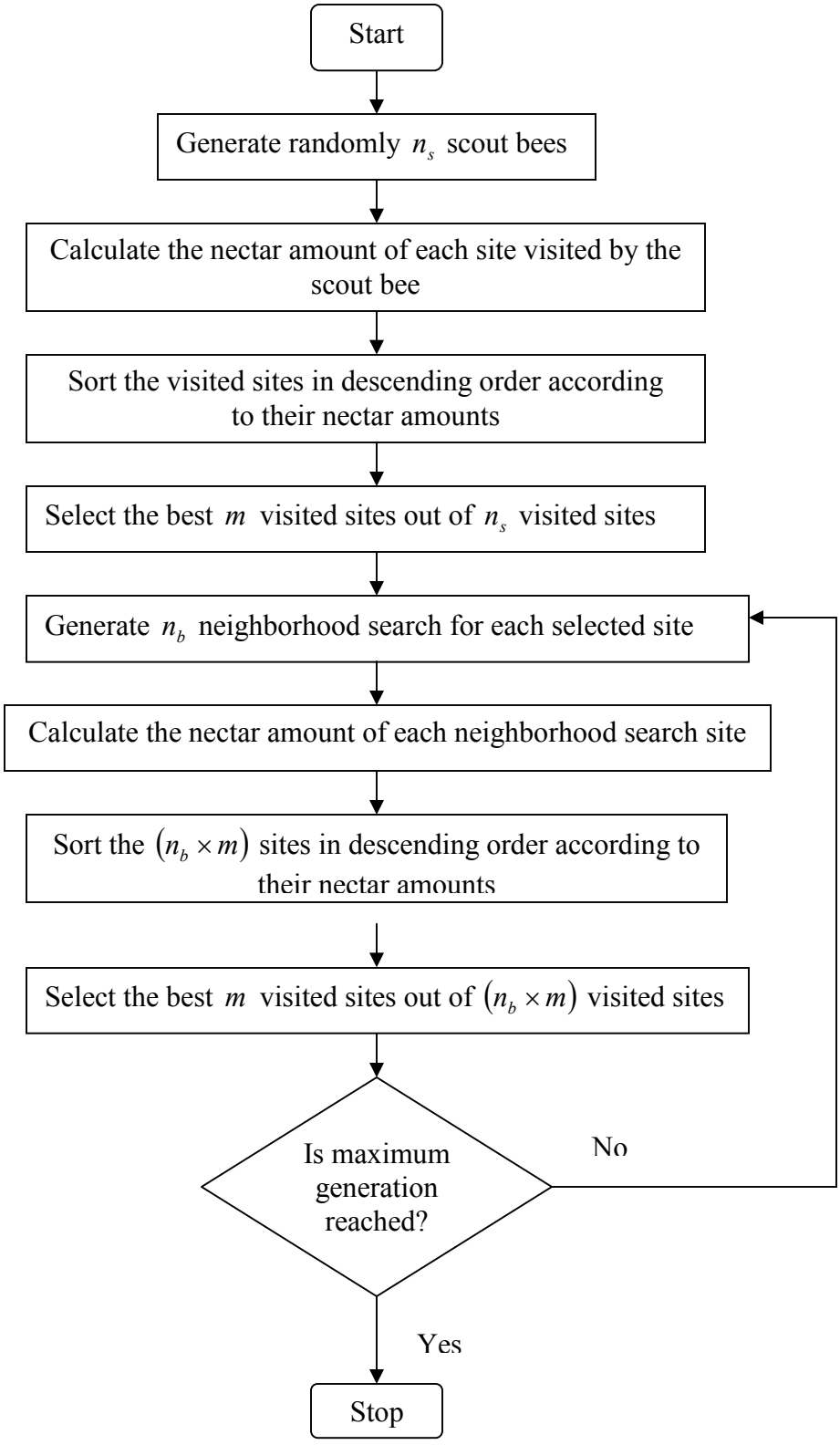

Fig. 9. Flowchart of Artificial Bee Colony Optimization