Research Journal of Applied Sciences, Engineering and Technology 3(6): 569-578, 2011
ISSN: 2040-7467

Received: May 24, 2011 Accepted: June 15, 2011 Published: June 25, 2011

A Survey on Meta-Heuristic Global Optimization Algorithms

[1]Mohammad Khajehzadeh, [1]Mohd Raihan Taha, [1]Ahmed El-Shafie and [2]Mahdiyeh Eslami
[1]Civil and Structural Engineering Department, University Kebangsaan Malaysia,
Bangi, 43600, Selangor, Malaysia
[2]Electrical Engineering Department, Anar Branch, Islamic Azad University, Anar, Iran

Abstract: Optimization has been an active area of research for several decades. As many real-world optimization problems become increasingly complex, better optimization algorithms are always needed. Recently, metaheuristic global optimization algorithms have become a popular choice for solving complex and intricate problems, which are otherwise difficult to solve by traditional methods. In the present study, an attempt is made to review the most popular and well known metaheuristic global optimization algorithms introduced during the past decades.

Key words: Global optimization, metaheuristic algorithm, swarm intelligence

INTRODUCTION

Optimization is ubiquitous and spontaneous process that forms an integral part of our day-to-day life. In the most basic sense, it can be defined as an art of selecting the best alternative among a given set of options. Optimization plays an important role in engineering designs, agricultural sciences, manufacturing systems, economics, physical sciences, pattern recognition and other such related fields. The objective of optimization is to seek values for a set of parameters that maximize or minimize objective functions subject to certain constraints (Rardin, 1997). A choice of values for the set of parameters that satisfy all constraints is called a feasible solution. Feasible solutions with objective function value(s) as good as the values of any other feasible solutions are called optimal solutions (Rardin, 1997). In order to use optimization successfully, we must first determine an objective through which we can measure the performance of the system under study. That objective could be time, cost, weight, potential energy or any combination of quantities that can be expressed by a single variable. The objective relies on certain characteristics of the system, called variable or unknowns. The goal is to find a set of values of the variable that result in the best possible solution to an optimization problem within a reasonable time limit. Normally, the variables are limited or constrained in some way.

Following the creation of the optimization model, the next task is to choose a proper algorithm to solve it. The optimization algorithms come from different areas and are inspired by different techniques. But they all share some common characteristics. They are iterative; they all begin with an initial guess of the optimal values of the variables and generate a sequence of improved estimates until they converge to a solution. The strategy used to move from one potential solution to the next is what distinguishes one algorithm from another. For instance, some of them use gradient based information and other similar methods (e.g., the first and second derivatives of these functions) to lead the search toward more promising areas of the design domain, whereas others use problem specific heuristics (e.g., accumulated information gathered at previous iterations). Broadly speaking, optimization algorithms can be placed in two categories: the conventional or deterministic methods and the modern heuristics or stochastic methods.

Conventional methods adopt the deterministic approach. During the optimization process, any solutions found are assumed to be exact and the computation for next set of solutions completely depends on the previous solutions found. That's why conventional methods are also known as deterministic optimization methods. In addition, these methods involve certain assumptions about the formulation of the objective functions and constraint functions. An important and widely used class of deterministic algorithms is gradient-based methods capable of choosing their search directions according to the derivative information of the objective functions. Conventional methods include algorithms such as linear programming, non linear programming, dynamic programming, Newton's method and others. Unfortunately, conventional optimization algorithms are not efficient at coping with demanding real world problems without derivate information. In other words, selection of the initial points for the deterministic

Corresponding Author: Mohammad Khajehzadeh, Civil and Structural Engineering Department, University Kebangsaan Malaysia, Bangi, 43600, Selangor, Malaysia

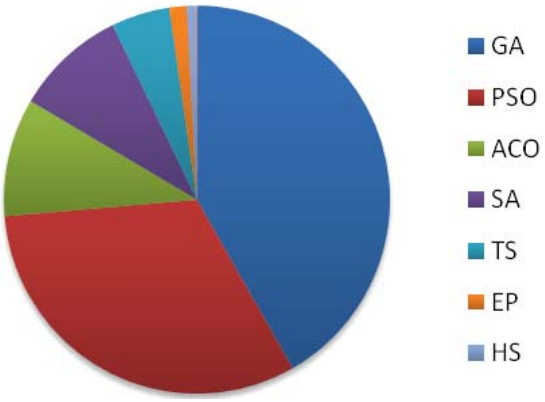


Fig. 1: Pie chart showing the publication distribution of the meta-heuristics algorithms

optimization methods has a decisive effect on their final results. However, a foresight of appropriate starting points is not always available in practice. One common strategy is to run the deterministic algorithms with random initialization numerous times and retain the best solution; however, this can be a time-consuming procedure. Therefore, in view of the practical utility of optimization problems there is a need for efficient and robust computational algorithms, which can numerically solve on computers the mathematical models of medium as well as large size optimization problem arising in different fields.

In the past few decades several global optimization algorithms have been developed that are based on the nature inspired analogy. These are mostly population based metaheuristics also called general purpose algorithms because of their applicability to a wide range of problems. Some popular global optimization algorithms include Evolution Strategies (ES) (Rechenberg, 1965), Evolutionary Programming (EP) (Fogel *et al*., 1966), Genetic Algorithms (GA) (Holland, 1975a, b), Genetic Programming (GP) (Smith, 1980), Simulated Annealing (SA) (Kirkpatrick *et al*., 1983), Artificial Immune System (AIS) (Farmer *et al*., 1986), Tabu Search (TS) (Glover, 1989; Glover, 1990), Ant Colony Optimization (ACO) (Dorigo, 1992), Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995), Harmony Search (HS) algorithm (Geem *et al*., 2001), Bee Colony Optimization (BCO) (Nakrani and Tovey, 2004), Gravitational Search Algorithm (GSA) (Rashedi *et al*., 2009), etc. Figure 1 shows the distribution of publications which applied the meta-heuristics methods to solve the optimization problem. This survey is based on ISI Web of Knowledge databases and included most of the papers that have been published during the past decade. Figure 1 shows that the GA and PSO are the most popular algorithms among the others.

These algorithms have proved their mettle in solving complex and intricate optimization problems arising in various fields. Generally, global optimization algorithms differ from traditional search and optimization paradigms in three main ways by:

- Utilizing a population of individuals (potential solutions) in the search domain. Most traditional optimization algorithms move from one point to another in the search domain using some deterministic rule. One of the drawbacks of this kind of approach is the likelihood of getting stuck at a local optimum. Evolutionary Computation (EC) paradigms, on the other hand, use a collection of points, called population. By inducing the evolutionary-like operators, they can generate a new population with the same number of members in each generation so that the probability of getting stuck is reduced.
- Using direct fitness information instead of function derivatives or other related knowledge. EC paradigms do not require information that is auxiliary to the problem, such as function derivatives, but only the value of the fitness function, as well as the constraints functions for constrained problems. Thus fitness is a direct metric of the performance of the individual population member on the function being optimized.
- Using probabilistic, rather than deterministic rules. These stochastic operators are applied to operations which lead the search towards regions where individuals are likely to find better values of the fitness function. So, the reproduction is often carried out with a probability which is dependent on the individual's fitness value.

A universal procedure can be described that can be implemented no matter the type of algorithm:

- Initializing the population in the search domain by seeding the population with random values.
- Evaluating the fitness for each individual of the population
- Generating a new population by reproducing selected individuals through evolutionary operations, such as crossover, mutation and so on.
- Looping to step 2 until stopping criteria are satisfied.

One can recognize two common aspects in the population-based heuristic algorithms: exploration and exploitation. The exploration is the ability of expanding search space, where the exploitation is the ability of finding the optima around a good solution. In premier iterations, a heuristic search algorithm explores the search

space to find new solutions. To avoid trapping in a local optimum, the algorithm must use the exploration in the first little iteration. Hence, the exploration is an important issue in a population-based heuristic algorithm. By lapse of iterations, exploration fades out and exploitation fades in, so the algorithm tunes itself in semi-optimal points. To have a high performance search, an essential key is a suitable tradeoff between exploration and exploitation. However, all the population-based heuristic algorithms employ the exploration and exploitation aspects but they use different approaches and operators. In other words, all search algorithms have a common framework. From a different point of view, the members of a population-based search algorithm pass three steps in each iteration to realize the concepts of exploration and exploitation: self-adaptation, cooperation and competition. In the self-adaptation step, each member (agent) improves its performance. In the cooperation step, members collaborate with each other by information transferring. Finally, in the competition step, members compete to survive. These steps have usually stochastic forms, and could be realized in different ways. These steps, inspired from nature, are the principle ideas of the population-based heuristic algorithms. These concepts guide an algorithm to find a global optimum. In this paper, we will give a concise introduction to the theory of some popular and well known global optimization algorithms.

GENETIC ALGORITHMS

Genetic algorithm is one of the most popular types of Evolutionary algorithms. To be more precise, it constitutes a computing model for simulating natural and genetic selection that is attached to the biological evolution described in Darwin's Theory which was first issued by Holland (1975a, b). In this computing model a population of abstract representations (called chromosomes or the genotype of the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem could result in better solutions, which are traditionally represented in binary form as strings comprising of 0 and 1 s with fixed-length, but other kinds of encoding are also possible which include real-values and order chromosomes. The program then will assign the proper number of bits and the coding.

Being a member of the family of evolutionary computation, the first step of GA is population initialization which is usually done stochastically. The GA usually uses three simple operators called selection, recombination (usually called crossover) and mutation. Selection is the step of a genetic algorithm in which a certain number of individuals is chosen from the current population for later breeding (recombination or crossover), the choosing rate is normally proportional to individual's fitness value. There are several general selection techniques. Tournament selection and fitness proportionality selection (also known as roulette-wheel selection) consider all given individuals. Other methods only choose those individuals with a fitness value greater than a given arbitrary constant. Crossover and mutation taken together is called reproduction. They are analogous to biological crossover and mutation respectively.

The most important operator in GA is crossover which refers to the recombination of genetic information during sexual reproduction. The child shares in common with it parents many characteristics. Therefore, in GAs, the offspring has an equal chance of receiving any given gene from either one parent because the parents' chromosomes are combined randomly. To date, there are many crossover techniques for organisms which use different data structures to store themselves, such like One-point crossover, two-point crossover, Uniform Crossover as well as Half Uniform Crossover. The probabilities for crossovers vary according to the problem. Generally speaking, values between 60 and 80% are typical for one-point crossover as well as two-point crossover. Uniform crossovers work well with slightly lower probabilities on the other hand. The probability could also be altered during evolution. So a higher value might initially be attributed to the crossover probability. Then it is decreased linearly until the end of the run, ending with a value of one half or two thirds of the initial value. Additionally for real-value cases, the crossover operation could be expressed as:

$$y^1 = \lambda x^1 + (1-\lambda)x^2 \qquad (1)$$
$$y^2 = \lambda x^2 + (1-\lambda)x^1$$

where y^1 and y^2 are two descendants created by two given parents x^1 and x^2, and $\lambda = U[0, 1]$.

Mutation is the stochastic flipping of chromosome bits that occurs each generation which is used to maintain genetic diversity from one generation of a population of chromosomes to the next. Mutation occurs step by step until the entire population has been covered. Again, the coding used is decisive. In the case of binary chromosomes, it simply flips the bit while in real-valued chromosomes a noise parameter $N[0, \sigma]$ is added to the value at that position. σ could be chosen in such way that it decreases in time, e.g., $\sigma = 1/(1 + \sqrt{t})$, where t is the number of current iteration. The probability of mutation is normally kept as a low constant value for the entire run of the GA, such like 0.001. Despite of these evolutionary operators, in some cases a strategy called "elites" is used, where the best individual is directly copied to the next generation without undergoing any of the genetic

operators. In one single interaction, new parents are selected for each child and the process continues until a proper size of individuals for the new population is reached. This process ultimately results in the population of the new generation differing from the current one. Generally the average fitness of the population should be improved by this procedure since only the best organisms from the last generation are selected for breeding.

The implementation of the genetic algorithm is described as follows (Shyr, 2008):

Step 1: Initialization. The algorithm starts with a set of solutions (represented by chromosomes) called population. A set of chromosomes is randomly generated. Each chromosome is composed of genes.

Step 2: Evaluation. For every chromosome, its fitness value is calculated. Each chromosome's fitness value is analyzed one by one. Compared with the existing best fitness value, if one chromosome can generate better fitness, renew the values of the defined vector and variable with this chromosome and its fitness value; otherwise, keep their values unchanged.

Step 3: Selection. Population selection within the algorithm utilizes the principle of survival of the fittest, which is based on the Darwinian's concept of Natural Selection (Holland, 1975a, b). A random number generator is employed to generate random numbers whose values are between 0 and 1.

Step 4: Crossover. The crossover operation is one of the most important operations in GA. The basic idea is to combine some genes from different chromosomes. A GA recombines of bit strings by copying segments from chromosomes pairs.

Step 5: Mutation. Some useful genes are not generated during the initial step. This difficulty can be overcome by using the mutation approach. The basic mutation operator randomly generates a number as the crossover position and then changes the value of this gene randomly.

Step 6: Stopping criteria. Steps 2-5 are repeated until the predefined number of generations has been reached. The optimal solution can be generated after termination.

Simulated Annealing (SA): Based on the analogy between statistical mechanics and optimization, the SA is one of the most flexible techniques available for solving difficult optimization problems. The main advantage of the SA is that it can be applied to large-scale systems regardless of the conditions of differentiability, continuity, and convexity, which are usually required for conventional optimization methods. The SA was originally proposed by Metropolis in the early 1950s as a model of the crystallization process. The SA procedure consists of first mlting the system being optimized at a high temperature, and then slowly lowering the temperature until the system freezes and no further change occurs. At each temperature instant, the annealing must proceed long enough for the system to reach a steady state (Kirkpatrick *et al*., 1983).

The SA method actually mimics the behavior of this dynamical system to achieve the thermal equilibrium at a given temperature. It has the remarkable ability of escaping from the local minima by accepting or rejecting new solution candidates according to a probability function. In addition, the SA method requires little computational resource. The basic SA algorithm can be explained by the following steps:

- Specify initial temperature T_0, and initialize the solution candidate.
- Evaluate fitness E of the candidate.
- Move the candidate randomly to a neighboring solution.
- Evaluate the fitness of new solutions E'.
- If $E' \le E$, accept the new solution. If $E' > E$, accept the new solution with acceptance probability P.
- Decrease temperature T. The SA search is terminated, if T is close to zero. Otherwise, return back to Step 2.

As we can observe that the SA algorithm simulates the process of gradually cooling a metal/crystal until the energy of the system achieves the global minimum. Each configuration of the physical system and energy of the atoms correspond to the current solution found for the optimization problem and fitness of the objective function, respectively. The temperature T is used to control the whole optimization procedure. At each generation the candidate is updated with the random perturbation, and the improvement of its fitness is also calculated. If $E' \le E$, the moving change results in a lower or equivalent energy of the system, and this new solution can be accepted. Otherwise, the displacement is only accepted with the probability P:

$$P = e^{\frac{-(E'-E)}{T}} \tag{2}$$

The temperature is updated by:

$$T(k+1) = \lambda T(k), 0 < \lambda < 1 \tag{3}$$

where k is the number of generations, and λ is a given coefficient. As a matter of fact, the cooling schedule needs to be properly adjusted by modifying parameter λ. The main strength of the SA method is its capability of obtaining the global optimum with a great probability.

However, it usually uses a large number of generations to converge.

Tabu search: The Tabu Search (TS) is a metaheuristic global optimization method originally developed by Glover (Glover, 1989; Glover, 1990) for large combinatorial optimization tasks, and extended to continuous-valued function in (Cvijovi and Klinowski, 1995). It is different from the well-known hill-climbing local search techniques because the TS allows moves out of a current solution that makes the objective function worse in the hope that it eventually will achieve a better solution. TS is also different from the simulated annealing and genetic algorithms because the TS includes a memory mechanism. It has been shown in literatures (Al-Sultan, 1995; Cvijovi and Klinowski, 1995) that it is superior to SA and GA both in time required to obtain a solution and in the solution quality in solving some combinatorial optimization problems.

In order to solve a problem using TS, the following basic elements must be defined (Al-Sultan, 1995).

- Configuration is a solution or an assignment of values to variables.
- A *move* is a transition from one trial solution to another (i.e. a move is a procedure by which a new solution is generated from the current one).
- A *neighborhood* (set of candidate moves, or trial solutions) of the solution is the set of all possible moves out of a current configuration. Note that the actual definitions of the neighborhood depend on the particular implementation and the nature of problem.
- Tabu restrictions: in order to avoid a blind search, TS technique uses a prescribed problem specific set of constraints, known as tabu conditions. They are certain conditions imposed on moves which make some of them forbidden. These forbidden moves are known as tabu moves. It is done by forming a list of certain size that records these forbidden moves. This is known as tabu list. After a specified duration (tabu list size), they are removed from the list and are free to be visited again.
- Aspiration criteria: these are rules that override tabu restrictions, i.e. if a certain move is forbidden by tabu restriction, then the aspiration criterion, when satisfied, can make this move allowable.

Given the above basic components, the TS schemes can be described as follows:

- Start with a certain configuration, evaluate the objective function for that configuration.
- Follow a certain set of candidate moves. If the best of the moves is not tabu or if the best is tabu, but satisfies the aspiration criterion, then pick that move and consider it to be the new current configuration; otherwise, pick the best move that is not tabu and consider it to be the new current configuration.
- Repeat the procedure until some termination criteria are satisfied.

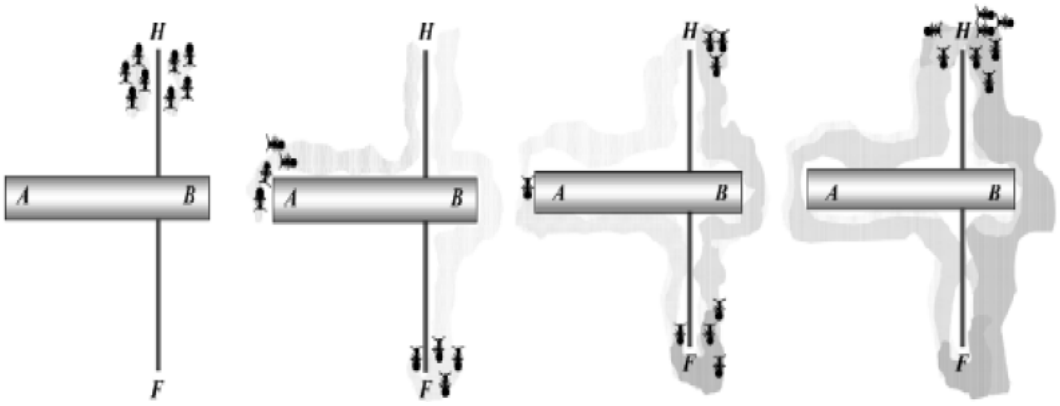
On termination, the best solution obtained so far is the solution obtained by the TS approach. Note that the move picked at certain iteration is put in the tabu list so that it is not allowed to be reversed in the next iterations. The tabu list has a certain size, and when the length of the tabu reaches that size and a new move enters that list, then the first move on the tabu list is freed from being tabu and the process continues i.e. the tabu list is circular.

ANT COLONY OPTIMIZATION

As the first successful optimization algorithm based on swarm intelligence, the Ant Colony Algorithm (ACO) was introduced by Dorigo in his Ph.D. Thesis (Dorigo, 1992). This algorithm is a technique for solving optimization problems that relies on probability. The inspiring source of ACO is the foraging behavior of ants in their search for food, i.e., over a period of time ants are able to determine the shortest path from their home to a food source. In nature, ants look for food randomly but having found food, on their return to the nest, they will lay down pheromone trails along the path which dissipate over time and distance. If other ants are attracted by the trails and find the food guided by the trails, the trails will be enhanced, because they leave the same thing when they go back. Over time, however, the pheromone trail starts to evaporate thus reducing its attractiveness. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have for evaporation. In the case of a short path, by contrast, pheromone density is kept at a high level because there is no time for evaporation before the laying down of a new layer of pheromones.

Figure 2 shows a simple example of how real ants find a shortest path. Let H be home, F be food source, and A-B be an obstacle in the route.

- at time t = 0, ants choose left and right side paths uniformly in their search for food source.
- at time t = 1, ants which have chosen the path F-B-H reach the food source earlier and are retuning back to their home, whereas ants which have chosen path H-A-F are still halfway in their journey to the food source.
- at time t = 2, since ants move at approximately constant speed, the ants which chose the shorter, right side path (H-B-F) reach the home faster, depositing more pheromone in H-B-F route.



(a) t = 0 (b) t = 1 (c) t = 2 (d) t = 4

Fig. 2: Illustration of ant colony principle, how real ants find shortest path in their search for food

- at time t = 4, pheromone accumulates at a higher rate on the shorter path (H-B-F), which is therefore, automatically preferred by the ants and consequently all ants will follow the shortest path. The darkness of shade is approximately proportional to the amount of pheromone deposited by ants.

The most important operations in ACO are arc selection and Pheromone Update, which constitute the foundations of the behavior of ant colonies. Arc selection describes that an ant will move from node *i* to node *j* with probability $P_{i,j}$, which is defined as:

$$P_{i,j} = \frac{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}{\sum \tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}} \quad (4)$$

where $\tau_{i,j}$ represents the amount of pheromone in arc (i, j) and $\eta_{i,j}$ the desirability of arc (i, j), α and β are two adjustable parameters that control the relative weight of trail intensity and desirability.
Pheromone can be updated in:

$$\tau_{i,j} = \rho\tau_{i,j} + \Delta\tau_{i,j} \quad (5)$$

where $\tau_{i,j}$ is the amount of pheromone in given arc (i, j), ρ is the rate of pheromone evaporation and Δ is the amount of pheromone deposited, typically given by:

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/Lk & if\ \ ant\ k\ travels\ on\ arc(i,j) \\ 0 & otherwise \end{cases} \quad (6)$$

where L_k is the cost of the *k*th ant's tour (typically length). The ACO is an efficient solution to a large variety of dynamical optimization problems (Dorigo and Stützle, 2004).

Particle swarm optimization: The Particle Swarm Optimization (PSO) algorithm belongs to category of the evolutionary computation for solving global optimization problems. Its concept was initially proposed by Kennedy as a simulation of social behavior and the PSO algorithm was first introduced as an optimization method in 1995 by Kennedy and Eberhart (Eberhart and Kennedy, 1995; Kennedy and Eberhart, 1995). PSO is a well known and popular search strategy has gained widespread appeal amongst researchers and has been shown to offer good performance in a variety of application domains, with potential for hybridization and specialization. It is a simple and robust strategy based on the social and cooperative behavior shown by various species like flock of bird, school of fish etc. PSO and its variants have been effectively applied to a wide range of benchmark as well as real life optimization problems.

The PSO is derived from a simplified version of the flock simulation. It also has features that are based upon human social behavior (their cognitive ability). The PSO is initialized with a population of random solutions and the size of the population is fixed at this stage and is denoted as s. Normally, a search space should first be defined, e.g., like a cube of the form $[x_{min}, x_{max}]^D$ for a *D* dimensional case. Each particle is distributed randomly in the search region according to a uniform distribution which it shares in common with other algorithms of stochastic optimization. The position of any given particle in the search space is a vector representing a design variable for the optimization problem, which is also called a potential solution. In addition, each particle has a velocity. This constitutes a major difference to other stochastic algorithms (e.g., GA). Here, the velocity is a vector that functions much like an operator that guides the particle to move from its current position to another potential improved place. All the particles' velocities are updated in every iteration. In order to describe the PSO conveniently, some symbols need to be defined:

$X_i(t)$: The position of particle *i* on time step *t*, i.e., it represents a Cartesian coordinates describing particle *i*'s position in solution space.

$V_i(t)$: The velocity of particle *i* on time step *t*, i.e., it represents particle *i*'s moving direction and its norm $\| V_i(t)\|$ is corresponding step size.

$p_i(t)$: The best personal position of the particle *i* discovered so far, which is updated as follows:

$$p_i(t+1) = \begin{cases} p_i(t) & if\ f(X_i(t)) \geq f(p_i(t)) \\ X_i(t) & if\ f(X_i(t)) < f(p_i(t)) \end{cases} \quad (7)$$

where *f* is the fitness function to be minimized.

$b(t)$: The so far discovered best position of particle *i* compared with its neighbors defined as:

$$\begin{aligned} & b(t) \in \{X_0(t),....,X_{ns}(t)\} \\ & \left| f(b(t)) = \min\left\{f(X_0(t)),....,f(X_{ns}(t))\right\}\right. \end{aligned} \quad (8)$$

where n_s is the total number of particles in the swarm.

Now the initial PSO could be now denoted as (Shi and Eberhart, 1998):

$$\begin{aligned} V_i(t+1) = {} & w \times V_i(t) + c_1 \times Rand(.) \times \\ & [p_i(t) - X_i(t)] + c_2 \times rand(.) \times [b(t) - X_i(t)] \end{aligned} \quad (9)$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (10)$$

where *w* is an inertia weight, *Rand*(.) and *rand*(.) are two independent random numbers selected in each step according to a uniform distribution in a given interval [0, 1] and c_1 and c_2 are two constants which are equal to 2 in this initial version. The random number was multiplied by 2 to give it a mean of 1, so that particles would overshoot the target about half the time. This inertia weight can either be a constant or a dynamically changed value. Essentially, this parameter controls the exploration of the search space, so that a high value allows particles to move with large velocities in order to find the global optimum neighborhood in a fast way and a low value can narrow the particles' search region and facilitate local search. Research has been taken into account in order to find a suitable set of *w*. So far there are two common strategies to choose the value of *w*:

Linear strategy: The value of *w* is decreased from a higher initial value (typically 0.9) to a lower value (typically 0.4) linearly as the iteration number increases. It has shown good performance in some applications. As the value of *w* is decreased, the search model of particles is also transformed from an exploratory mode to an exploitive mode. The main disadvantage of this strategy is that once the inertia weight is decreased, the swarm loses its ability to search new areas because it is impossible to recover its exploration mode.

Random strategy: the value of *w* comprises two parts, a constant (typically 0.5) and a random value distributed in [0, 0.5]. The constant part ensures the particles' basic search ability within an exploitive mode, the random part ensures that the particle can shift its search mode between exploratory mode and exploitative mode randomly. Using this strategy, particles can search the design domain more flexibly and widely.

Equation (9) clearly shows that the particle's velocity can be updated in three situations: The first one is known as the momentum part, meaning that the velocity cannot change abruptly from the velocity of the last step; and that it could be scaled by a constant as in the modified versions of PSO. The second one is called "memory" part and describes the idea that the individual learns from its flying experience. The last one is known as the "cognitive" part which denotes the concept that particles learn from their group flying experience because of collaboration. According to the description above, the whole workflow of the standard particle swarm optimization is shown as follows:

Step 1: Define the problem space and set the boundaries, i.e., equality and inequality constraints.

Step 2: Initialize an array of particles with random positions and their associated velocities inside the problem space.

Step 3: Check if the current position is inside the problem space or not. If not, adjust the positions so as to be inside the problem space.

Step 4: Evaluate the fitness value of each particle.

Step 5: Compare the current fitness value with the particles' previous best value (*pbest*). If the current fitness value is better, then assign the current fitness value to *pbest*.

Step 6: Determine the current global minimum among particle's best position.

Step 7: If the current global minimum is better than *gbest*, then assign the current global minimum to *gbest*.

Step 8: Change the velocities according to Eq. (9).

Step 9: Move each particle to the new position according to Eq. (10) and return to Step 3.

Step 10: Repeat Step 3- Step 9 until a stopping criteria is satisfied.

And here is the corresponding pseudo code of PSO algorithm:

Harmony search algorithm: Geem *et al*. (2001) developed a harmony search meta-heuristic algorithm which is based on natural musical performance process of searching for a perfect state of harmony such as during jazz improvisation. Jazz improvisation searches to find musically pleasing harmony (a perfect state) as determined by an aesthetic standard, just as the optimization process seeks to find a global solution (a perfect state) as determined by an objective function. The pitch of each musical instrument determines the aesthetic quality, just as the objective function value is determined by the set of values assigned to each decision variable. Generally, the optimization procedure based on harmony search algorithm may be illustrated as follows:

Step1. Initialize the optimization problem and algorithm parameters: The HS parameters include the number of solution vectors in the harmony memory or the Harmony Memory Size (HMS), Harmony Memory Considering Rate (HMCR), Pitch-Adjusting Rate (PAR), and the maximum number of improvisations (*K*), or termination criterion. These parameters are selected depending on the problem.

Step2. Initialize the harmony memory (HM): In this step, the harmony memory (HM) matrix as shown in Eq. (11) is randomly generated and sorted by the values of the objective function.

$$HM = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_N^1 \\ x_1^2 & x_2^2 & \cdots & x_N^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{HMS} & x_2^{HMS} & \cdots & x_N^{HMS} \end{bmatrix} \begin{matrix} \Rightarrow f\left(X^1\right) \\ \Rightarrow f\left(X^2\right) \\ \Rightarrow \vdots \\ \Rightarrow f\left(X^{HMS}\right) \end{matrix} \quad (11)$$

where $X^1, X^2, \ldots, X^{HMS}$ and $f(X^1), f(X^2), \ldots, f(X^{HMS})$ show each solution vector for design variables and the corresponding objective function value, respectively.

Step3. Improvise a new harmony from the HM: A new harmony vector $[x^{nh}]=[x_1^{nh}, x_2^{nh}, \ldots, x_N^{nh}]$ is improvised from either the HM or the entire section list. The new harmony vector is generated by three rules: memory consideration, pitch adjustment, and random generation.

In the memory consideration process, the value of the first design variable (x_1^{nh}) for the new vector is chosen from any value in the specified HM range $[x_1^1, x_1^2, \ldots, x_1^{HMS}]$. Values of the other decision variables (x_i^{nh}) are chosen by the same rationale. Here, the possibility that a new value will be chosen is indicated by the HMCR parameter, which varies between 0 and 1 as follows:

$$x_i^{nh} \leftarrow \begin{cases} x_i^{nh} \in \left\{x_i^1, x_i^2, \ldots, x_i^{HMS}\right\} & if\ r \le HMCR \\ x_i^{nh} \in X_i & if\ r > HMCR \end{cases} \quad (12)$$

where *r* is a random number from interval [0,1]. At first, a random number (*r*) is generated. If this random number is less than the HMCR value, the *i*-th design variable of the new design $[x^{nh}]$ is selected from the current values stored in the *i*-th column of HM. If *r* is higher than HMCR, the *i*-th design variable of the new design $[x^{nh}]$ is selected from the entire section list X_i.

After that, every component of the new harmony vector, $[x^{nh}] = [x_1^{nh}, x_2^{nh}, \ldots, x_N^{nh}]$, evaluated by memory consideration is examined to determine whether it should be pitch adjusted. This procedure uses the PAR parameter. PAR investigates better design in the neighboring of the current design and is applied as follows:

$$Pitch\ adjusting\ decision\ x_i^{nh} \leftarrow \begin{cases} yes\ if\ \ r \le PAR \\ no\ \ if\ \ r > PAR \end{cases} \quad (13)$$

A random number (*r*) uniformly distributed over the interval [0, 1] is generated for x_i^{nh}. If this random number is less than the PAR, x_i^{nh} is replaced with its neighboring values. If this random number is not less than the PAR, x_i^{nh} remains the same. If the pitch adjustment decision for x_i^{nh} is Yes, the pitch adjustment value of x_i^{nh} for continuous design variable is:

$$x_i^{nh} \leftarrow x_i^{nh} + \alpha \quad (14)$$

where α is the value of bw.u(-1,1); bw is an arbitrary distance bandwidth for the continuous variable; and u(−1, 1) is a uniform distribution between -1 and 1.

Step4. Fitness measure and update the harmony memory: In this Step, if the new harmony vector, $[x^{nh}]=[x_1^{nh}, x_2^{nh}, \ldots, x_N^{nh}]$, is better than the worst design in the HM in terms of the objective function value, the new design is included in the HM, and the existing worst harmony is excluded from the HM. The HM is sorted again by the objective function value.

Step 5. Termination criterion: Repeat steps 2 and 3 until the termination criterion is satisfied.

Gravitational search algorithm: Gravitational Search Algorithm (GSA) is a newly developed stochastic search

algorithm based on the law of gravity and mass interactions. In GSA, the search agents are a collection of masses which interact with each other based on the Newtonian gravity and the laws of motion, completely different from other well-known population-based optimization method inspired by swarm behaviors in nature (such as PSO, GA or ant colony search algorithm). In GSA, agents are considered as objects and their performance are measured by their masses. All of the objects attract each other by the gravity force, while this force causes a global movement of all objects towards the objects with heavier masses (Rashedi *et al*., 2009). The heavy masses correspond to good solutions of the problem. In other words, each mass presents a solution, and the algorithm is navigated by properly adjusting the gravitational and inertia masses. By lapse of time, the masses will be attracted by the heaviest mass which it presents an optimum solution in the search space.

To describe the GSA, consider a system with N agents (masses), the position of the ith agent is defined by:

$$X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n) \text{ for } i = 1, 2, \ldots, N \quad (15)$$

where x_i^d presents the position of ith agent in the dth dimension and n is the space dimension.

According to Newton gravitation theory, at a specific time t, the force acting on the ith mass from the jth mass is defined as:

$$F_{ij}^d(t) = G(t)\frac{Mi(t) \times Mj(t)}{\left\| X_i(t), X_j(t) \right\|_2}\left(x_j^d(t) - x_i^d(t)\right) \quad (16)$$

where M_i and M_j are masses of agents, G(t) is the gravitational constant at time t. M_i is calculated through comparison of fitness as follows:

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \quad (17)$$

where;

$$m_i(t) = \frac{fit(t) - worst(t)}{best(t) - worst(t)} \quad (18)$$

where $fit_i(t)$ represent the fitness value of the agent i at time t, $best(t)$ is the best fitness of all agent and $worst(t)$ is the worst fitness of all agents.

For the ith agent, the randomly weighted sum of the forces exerted from other agents:

$$F_i^d(t) = \sum_{j \neq i} rand_j \, F_{ij}^d(t) \quad (19)$$

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

Hence, by the law of motion, the acceleration of the agent i at time t and in direction d, $a_i^d(t)$, is given as follows:

$$a_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \quad (20)$$

Then, the searching strategy on this concept can be described by following equations:

$$v_i^d(t+1) = randi \times v_i^d(t) + a_i^d(t) \quad (21)$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \quad (22)$$

where x_i^d, v_i^d and a_i^d represents the position, velocity and acceleration of ith agent in dth dimension, respectively. $rand_i$ is a uniform random variable in the interval [0, 1]. This random number is applied to give a randomized characteristic to the search.

It must be pointed out that the gravitational constant $G(t)$ is important in determining the performance of GSA and is defined as a function of time t:

$$G(t) = G_0 \times \exp\left(-\beta \times \frac{t}{t_{\max}}\right) \quad (23)$$

where G_0 is the initial value, β is a constant, t is the current iterations, t_{max} is the maximum iterations. The parameters of maximum iteration t_{max}, population size N, initial gravitational constant G_0 and constant β control the performance of GSA (N, G_0, β and t_{max}).

According to the description above, the whole workflow of the gravitational search algorithm is shown as follows:

Step 1: Define the problem space and set the boundaries, i.e., equality and inequality constraints.
Step 2: Initialize an array of masses with random positions.
Step 3: Check if the current position is inside the problem space or not. If not, adjust the positions so as to be inside the problem space.
Step 4: Evaluate the fitness value of agents.
Step 5: Update $G(t)$, $best(t)$, $worst(t)$ and $M_i(t)$ for $i = 1, 2, \ldots, N$.
Step 6: Calculation of the total force in different directions and acceleration for each agent.
Step 7: Change the velocities according to Eq. (21).
Step 8: Move each agent to the new position according to Eq. (22) and return to Step 3.
Step 9: Repeat Step 4 to Step 8 until a stopping criteria is satisfied.

In (Rashedi *et al*., 2009), GSA has been compared with some well known heuristic search methods. The high performance of GSA has been confirmed in solving various nonlinear functions. As an excellent optimization algorithm, GSA has the potential to solve a broad range of optimization problems.

CONCLUSION

Most of the real world optimization problems often involve large scale nonlinear optimization. In the past, many optimization techniques used to find optimal solutions were constrained by the complexities of non-linear relationships in the model formulation and increase in the number of state variables and constraints. For this reason, recently many heuristic and metaheuristic global optimization algorithms have been proposed and the use of these algorithms for solving optimization problems has become more practical in the last few years. In this paper attempts were made to present a concise description on the well known existing global optimization methods, which providing a solution to many real worlds complex optimization problems that are difficult to be tackled using the traditional gradient based methods, due to their nature that implies discontinuities of the search space, non differentiable objective functions, imprecise arguments and function values.

REFERENCES

Al-Sultan, K.S., 1995. A tabu search approach to the clustering problem. Pattern Recogn., 28(9): 1443-1451.

Cvijovi, D. and J. Klinowski, 1995. Taboo search: An approach to the multiple minima problem. Science, 267(5198): 664.

Dorigo, M. and V. Maniezzo, 1992. Optimization, Learning and Natural Algorithms. Ph.D. Thesis, Politecnico di Milano, Italy.

Dorigo, M. and T. Stützle, 2004. Ant Colony Optimization. The MIT Press, Cambridge, MA.

Eberhart, R. and J. Kennedy, 1995. A new optimizer using particle swarm theory. Sixth International Symposium on Micro Machine and Human Science Nagoya, Japan, pp: 39-43.

Farmer, J.D., N.H. Packard and A.S. Perelson, 1986. The immune system, adaptation, and machine learning. Physica D., 22(1-3): 187-204.

Fogel, L.J., A.J. Owens and M.J. Walsh, 1966. Artificial Intelligence Through Simulated Evolution. John Wiley, New York.

Geem, Z.W., J.H. Kim and G. Loganathan, 2001. A new heuristic optimization algorithm: Harmony search. Simulation, 76(2): 60.

Glover, F., 1989. Tabu search-part I. ORSA J. Comput., 1(3): 190-206.

Glover, F., 1990. Tabu search-part II. ORSA J. Comput., 2(1): 4-32.

Holland, J., 1975a. An introduction with application to biology, control and artificial intelligence Adaptation in Natural and Artificial System. MIT Press, Cambridge, MA.

Holland, J.H., 1975b. Adaptation in natural and artificial systems. The University of Michigan Press, AnnArbor, Michigan.

Kennedy, J. and R. Eberhart, 1995. Particle swarm optimization. IEEE International Conference on Neural Networks Perth, Australia, pp: 1942-1948

Kirkpatrick, S., C.D. Gelatt and M.P. Vecchi, 1983. Optimization by simulated annealing. Science, 220(4598): 671-680.

Nakrani, S. and C. Tovey, 2004. On honey bees and dynamic server allocation in internet hosting centers. Adapt. Behav., 12(3-4): 223.

Rardin, R.L., 1997. Optimization in Operations Research. Prentice Hall, New Jersey, USA.

Rashedi, E., H. Nezamabadi-Pour and S. Saryazdi, 2009. GSA: A gravitational search algorithm. Inform. Sci., 179(13): 2232-2248.

Rechenberg, I., 1965. Cybernetic solution path of an experimental problem, Royal Aircraft Establishment, Library Translation no. 1122.

Shi, Y. and R. Eberhart, 1998. A modified particle swarm optimizer. Proceedings of the IEEE Congress on Evolutionary Computation, Piscataway, pp: 69-73.

Shyr, W.J., 2008. Introduction and Comparison of Three Evolutionary-Based Intelligent Algorithms for Optimal Design. Third 2008 International Conference on Convergence and Hybrid Information Technology, pp: 879-884.

Smith, S.F., 1980. A learning system based on genetic adaptive algorithms. Ph.D. Thesis, University of Pittsburgh.