International Journal of Mathematics and
Computer Applications Research (IJMCAR)
ISSN 2249-6955
Vol. 3, Issue 1, Mar 2013, 103-124

REVIEW OF TEN NON-TRADITIONAL OPTIMIZATION TECHNIQUES

S.ELIZABETH AMUDHINI STEPHEN[1] & JOE AJAY.A[2]

[1]Associate Professor of Mathematics, Karunya University, Coimbatore, Tamil Nadu, India

[2]Scholar, Karunya University, Coimbatore, Tamil Nadu, India

ABSTRACT

The Paper deals with the theory needed to understand ten nontraditional optimization techniques which are used to find the optimum values for problems. These methods are used to solve both linear and non linear optimization problems and the methods yield global solution. Although various optimization methods have been proposed in recent years, these have not yet been compared suitably. The methods were broadly reviewed.

KEYWORDS: Non Traditional Optimization, Genetic Algorithm, Simulated Annehealing, Pattern Search, Particle Swarm Optimization, Godlike, Fmincon, Differential Evolution, Direct Algorithm, LGO, Glccluster, Glcsolve

INTRODUCTION

The difficulties associated with using mathematical optimization on large-scale engineering problems have contributed to the development of alternative solutions. Linear programming and dynamic programming techniques, for example, often fail (or reach local optimum) in solving NP-hard problems with large number of variables and non-linear objective functions. To overcome these problems, researchers have proposed evolutionary-based algorithms for searching near-optimum solutions to problems. Evolutionary algorithms (EAs) are stochastic search methods that mimic the metaphor of natural biological evolution and/or the social behavior of species. To mimic the efficient behavior of these species, various researchers have developed computational systems that seek fast and robust solutions to complex optimization problems. The first evolutionary-based technique introduced in the literature was the genetic algorithms (GAs). GAs were developed based on the Darwinian principle of the 'survival of the fittest' and the natural process of evolution through reproduction. Based on its demonstrated ability to reach near-optimum solutions to large problems, the GAs technique has been used in many applications in science and engineering. Despite their benefits, GAs may require long processing time for a near optimum solution to evolve. Also, not all problems lend themselves well to a solution with Gas. In an attempt to reduce processing time and improve the quality of solutions, particularly to avoid being trapped in local optima, other EAs have been introduced during the past 10 years. In addition to various GA improvements, recent developments in EAs include four other techniques inspired by different natural processes: memetic algorithms. In this paper, the ten EAs presented are reviewed and a pseudo code for each algorithm is presented to facilitate its implementation. Guidelines are then presented for determining the proper parameters to use with each algorithm. A brief description of the ten algorithms is presented in the following subsections.

GENETIC ALGORITHM

Basics of Genetic Algorithm

Among the probability based (stochastic) search methods is the 'genetic algorithm' based on the principle of natural selection and the survival of the fittest. John Holland originally proposed genetic algorithms at the University of Michigan.

An initial 'population' is generated by random selection of the individual. Strings represent directly or indirectly the design variables in the objective function. Groups are formed initially at random to compose a family of strings, each

family containing a single set of parameters comprising a design. The fitness of each group is then evaluated and assessed against the objective function. The strings in the best families are given favorable weightings in a selection process, whereby pairs of strings are chosen and combined (mated) by a crossover process. It is useful also to introduce an element of mutation, whereby some bits are switched (0 to 1 or 1 to 0) to encourage the development of new genetic material. Through the prescription of a mutation, probability controls the incidence of mutation.

The cycle then continues into the next generation. The process is terminated when the specified maximum numbers of generations are reached. The effectiveness of genetic algorithm depends on a number of factors like population size, probability of crossover and probability of mutation.

Genetic algorithms are computationally simple but powerful in their search for improvement. In addition, they are not limited by the restrictive assumptions about search space, such as continuity on existence of derivatives. The success of the genetic algorithm owes much to the work of Professor David.E.Goldberg of the University of Illinois. Goldberg (1989) described the nature of genetic algorithms of choice by combining the survival of the fittest procedure with the structured, but randomized, exchange information to form canonical search procedure that is capable of addressing wide spectrum of problems.

- Genetic algorithms do not require problem specific knowledge to carry out search. For instance, calculus based search algorithms use derivative information to carry out a search. In contrast to this GA's are indifferent to problem specific information
- GA's work on coded design variables, which are finite length strings. These strings represent the artificial gene. GA's process successive populations of these artificial chromosomes in successive generations.
- GA's use population of points at a time in contrast to the single point approach by the traditional optimization methods, which means at a given time GA's process a number of designs.
- GA's use randomized operators in place of the usual deterministic ones.

Encoding Schemes

The fundamental to genetic algorithm structure is the coding mechanism for representing the optimization problem variables. The success and working of genetic algorithm rely on the type of coding variables. The type of coding or encoding scheme depends upon the nature of problem variables.

A large number of optimization problems have real valued continuous variables. A common method of encoding uses their integer representation. Each variable is first linearly mapped to an integer defined in a specific range and the integer is encoded using a fixed number of binary bits. The binary code corresponding to each integer can be easily computed. The binary codes of all variables are then concatenated to obtain a binary string. In the case of discrete variables, it is necessary to supply a list of values that the variables can take.

Binary Coding

Bit string encoding or binary coding is the most classic approach used by GA researchers due to its simplicity and tractability. In this coding, variables are represented as binary sub-strings consisting of 0's and 1's. The length of a sub-string depends on the accuracy required and data available. These sub-strings are decoded and the corresponding variable is read from the database. So each population represents one solution to the problem.

In the case of continuous variable, if, for example, four bit binary strings are used to decode a variable, then the sub-string (0000) is decoded to lower limit of variables, (1111) is decoded to upper limit of variables. Any other string is decoded to a value in the range between upper and lower limits using a suitably adopted special step size operator. There could be only 2^4 or 16 different strings possible because each bit position can take a value 0 or 1. So using a four bit binary sub-string, one can represent 16 available solutions.

In case of discrete variables, the values can be directly fed in and if 16 areas are available, four bit binary string can be used to represent the sixteen available areas. If the available sections are less than 16, then some of the most commonly used are duplicated variables. This can be achieved by experience. If it is more than 16, then sub-string length equal to five, which represent 32 sections, is used.

Genetic algorithm basically consists of three parts:

- Coding and decoding variables into strings
- Evaluating the fitness of each solution string
- Applying genetic operations to generate the next generation of solution strings

The fitness of each string is evaluated by performing system analysis to compute a value of the objective function, if the solution violates constraints, the value of the objective function is penalized.

Power of Genetic Algorithm

Genetic algorithms derive their power from genetic operators. A simple genetic algorithm uses largely the three basic operators of:

- Reproduction
- Crossover
- Mutation

Reproduction

Reproduction is a process in which individual strings are copied according to their fitness value. The idea involved in this process is that, individuals having fitness values are picked and duplicated in the mating pool. After the design space is converted in terms of genetic space, the search is carried out for the optimization value of the objective function. The binary strings are decoded into real variables and the value of objective function is evolved. From this objective function, the fitness of the population is determined. Reproduction is the first operator applied on a population.

Crossover

Crossover is recombination operator, The aim of the crossover operator is to search the parameter space. It proceeds, in the following steps. First, the reproduction operator makes a match of two individual strings for mating. Then the cross-site is selected at random along the string length and the position values are swapped between the strings following the cross-site. For instance, let the two selected string in the mating pair be A=11111 and B=00000. If the random selection of the cross-site is 2, then the new strings following crossover would be A=11000 and B=00111. This is single site crossover. Though these operators may look simple, their combined action is responsible for much of GA's power. From the computer implementation view point, they involve only random number generation, string copying and partial string swapping. There exists the following type of crossover operators in GA.

- Single site crossover
- Two point crossover
- Multiple point crossover
- Uniform crossover
- Two dimensional crossover
- Flexible crossover.

Crossover Rate

In GA literature, the term crossover rate usually denoted by p_c is used to indicate the probability of crossover. Its probability varies from 0 to 1. This is used to GA in such way that out of the fixed population, some pairs are crossed. Typically for population size of 20 to 200 crossovers, rate ranges from 0.5 to 1. In the present work, two point cross over is used with crossover rate of 0.8, and Elite count as 2 since the work is compared with other techniques without any change.

Mutation

The mutation operator introduces new genetic structures in the population by randomly modifying some of the building blocks, thereby helping the search algorithm escape from local minima's trap. After crossing, strings are subjected to mutation. Mutation of a bit involves flipping it, i.e. changing 0 to 1 or vice versa. Just as p_c controls the probability of crossover, another parameter p_m (the mutation rate) gives a probability that a bit will be flipped. For example in the simple genetic material, if all the strings in the population have converged to 0 at a given position, and the optimal solution has a 1 in that position, then the crossover operator cannot generate a 1 in that position while a mutation can. Thus, mutation is simply an assurance policy against irreversible loss of genetic material.

Mutation rate is the probability of mutation, i.e. it is used to calculate the number of bits to be mutated. The mutation operator preserves the diversity among population, which is very important to the search. Mutation rates are similarly small in natural populations, leading us to conclude that mutation is approximately considered as a secondary mechanism of genetic algorithm adoption. Typically for population size varying from 20 to 200, mutation rates varying from 0.001 to 0.05 are used. In the present work, mutation rate of 0.001 is used

Simple Genetic Algorithm

The mechanism of a simple genetic algorithm is surprisingly simple, involves nothing more complex than copying strings and swapping partial strings. Simple genetic algorithm involves a population (or binary strings), a fitness function and genetic operators (reproduction, crossover and mutation.)

```
Simple Genetic Algorithm ( )
(   Initialize population;
    Evaluate population;
    While termination criterion not reached;
    {  Select solutions for next population;
       Perform crossover and mutation;
       Evaluate population;
    } )
```

Problems Misleading Genetic Algorithm to Local Optima

Genetic algorithms work by recombining low-level short schemata with above average fitness values to form higher order schemata. For chromosomes of length l in an alphabet A, a schema is a subset of the space A^l in which all the chromosomes share a particular set of defined values Schemata are represented as strings in the original alphabet extended with a wildcard symbol (e.g. *)

E.g. for A = {0, 1}, the schema $(1 * *)$ represents the chromosomes

{(100), (101), (110), (111)}

If the low order schemata (The order of a schema is the number of defined positions it has) schemata contain globally optimum solution, then the GA can locate it.

However, functions for which the low order high fitness value schemata do not contain the optimal string as an instance, the GA should converge to sub-optimal strings. Such functions are called “deceptive”. Recently, considerable research has focused on the analysis and design of deceptive functions.

Convergence of Genetic Algorithm

The selection of good strings in a population set and a random information exchange among good strings are simple and straightforward.. One criterion for convergence may be such that when a fixed percentage of columns and rows in the population matrix becomes the same, it can be assumed that convergence is attained. The fixed percentage may be 80% to 85%.

In genetic algorithm, as more generations proceeded there may not be much improvement in the population fitness and the best individual may not change for subsequent populations. As the generation progresses, the population gets filled by more fit individuals with only a slight deviation from the fitness of the best individual. We can specify some fixed number of generations after getting the optimum point to confirm there is no change in the optimum in the subsequent generations.

SIMULATED ANNEALING

Basics of Simulated Annealing

Simulated annealing (SA) is a generic probabilistic metaheuristic (iterative improvement method) for the global optimization problem of applied mathematics, namely locating a good approximation to the global optimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities).

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to dissociate from their initial positions and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends both on the difference between the corresponding function values and also on a global parameter T (called the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when T is large, but increasingly "downhill" as T goes to zero.

Overview

In the simulated annealing (SA) method, each point s of the search space is analogous to a state of some physical system, and the function E(s) to be minimized is analogous to the internal energy of the system in that state. The goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy.

The Basic Iteration

At each step, the SA heuristic (predicting the next value from the present value based on probability) considers some neighboring state S' of the current state S, and probabilistically decides between moving the system to state S' or staying in state S. These probabilities ultimately lead the system to move to states of lower energy. Typically, this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted.

The Neighbours of a State

The action taken to alter the solution in order to find neighbouring solutions is called "move" and different "moves" give different neighbours. These moves usually result in minimal alterations of the solution, in order to help an algorithm to optimize the solution to the maximum extent and also to retain the already optimum parts of the solution and affect only the suboptimum parts. Metaheuristic, optimize through the neighbourhood approach, they move through neighbours that are worse solutions than the current solution. Simulated Annealing in particular doesn't even try to find the best neighbour. The reason for this is that the search can no longer stop in a local optimum and in theory, if the metaheuristic can run for an infinite amount of time, the global optimum will be found.

Acceptance Probabilities

The probability of making the transition from the current state S to a candidate new state S' is specified by an acceptance probability function P (e, e', T), that depends on the energies e = E(s) and e' = E(s') of the two states, and on a global time-varying parameter T called the temperature.

One essential requirement for the probability function P is that it must be nonzero when e' > e, meaning that the system may move to the new state even when it is worse (has a higher energy) than the current one. It is this feature that prevents the method from becoming stuck in a local minimum—a state that is worse than the global minimum, yet better than any of its neighbours.

On the other hand, when T goes to zero, the probability P(e,e',T) must tend to zero if e' > e, and to a positive value if e' < e. That way, for sufficiently small values of T, the system will increasingly favour moves that go "downhill" (to lower energy values), and avoid those that go "uphill". In particular, when T becomes 0, the procedure will reduce to the greedy algorithm—which makes the move only if it goes downhill.

In the original description of SA, the probability P(e,e',T) was defined as 1 when e' < e — i.e., the procedure always moved downhill when it found a way to do so, irrespective of the temperature. Many descriptions and implementations of SA still take this condition as part of the method's definition. However, this condition is not essential for the method to work, and one may argue that it is both counterproductive and contrary to its principle.

The P function is usually chosen so that the probability of accepting a move decreases when the difference e' − e increases—that is, small uphill moves are more likely than large ones. However, this requirement is not strictly necessary, provided that the above requirements are met.

Pseudo Code

The following pseudo code implements the simulated annealing heuristic as described above. It starts from a state S_0 and continues to either a maximum of kmax steps or until a state with energy of emax or less is found. In the process, the call neighbour(s) should generate a randomly chosen neighbour of a given state S; the call random() should return a random value in the range [0,1]. The annealing schedule is defined by the call temp(r), which should yield the temperature to use, given the fraction r of the time budget that has been expended so far.

```
s ← s0; e ← E(s)                        // Initial state, energy.
sbest ← s; ebest ← e                    // Initial "best" solution
k ← 0                                   // Energy evaluation count.
while k < kmax and e < emax             // While time left & not good enough:
  snew ← neighbour(s)                   // Pick some neighbour.
  enew ← E(snew)                        // Compute its energy.
  if P(e, enew, temp(k/kmax)) > random() then   // Should we move to it?
    s ← snew; e ← enew                  // Yes, change state.
  if enew < ebest then                  // Is this a new best?
    sbest ← snew; ebest ← enew          // Save 'new neighbour' to 'best found'.
  k ← k + 1                             // One more evaluation done
return sbest                            // Return the best solution found
```

General Outline

The algorithm chooses the distance of the trial depending on the current temperature (weightage parameter). If the new point is better than the current point, it becomes the next point. Systematically lowers the temperature, storing the best point found so far. Simulated Annealing reanneals after it accepts Reanneal-Interval points. Reannealing raises the temperature in each dimension and the algorithm stops when any one of the stopping criteria is met. If not, the steps are repeated from the beginning.

Acceptance Criteria for Neighbour

$$\frac{1}{1 + exp\left(\frac{\Delta}{\max(T)}\right)}$$

Where Δ = new objective value – old objective value

T_o = initial temp of component

T = current temperature

So smaller temperature leads to better acceptance as well as larger Δ leads to smaller acceptance.

The Temperature of the Trail point is given by: $T = T_o * 0.95k$

Where k is the iteration number until Reannealing

Reannealing

$$k_i = log\left(\frac{T_o}{T_i}\frac{max(S_j)}{S_i}\right)$$

Where:

T_o - Initial point Temperature

T - Temperature of trail

S_i - Gradient of i component of objective

K_i - Acceptance parameter

Efficient Candidate Generation

When choosing the candidate generator neighbour(), one must consider that after a few iterations of the SA algorithm, the current state is expected to have much lower energy than a random state. Therefore, as a general rule, one should skew the generator towards candidate moves where the energy of the destination state s' is likely to be similar to that of the current state. This heuristic (which is the main principle of the Metropolis-Hastings algorithm) tends to exclude "very good" candidate moves as well as "very bad" ones; however, the latter are usually much more common than the former, so the heuristic is generally quite effective.

Barrier Avoidance

When choosing the candidate generator neighbour() one must also try to reduce the number of "deep" local minima — states (or sets of connected states) that have much lower energy than all its neighbouring states. Such "closed catchment basins" of the energy function may trap the SA algorithm with high probability (roughly proportional to the number of states in the basin) and for a very long time (roughly exponential on the energy difference between the surrounding states and the bottom of the basin).

Cooling Schedule

The physical analogy that is used to justify SA assumes that the cooling rate is low enough for equilibrium at all times. Unfortunately, the relaxation time—the time one must wait for the equilibrium to be restored after a change in temperature-strongly depends on the "topography" of the energy function and on the current temperature. Therefore, in practice the ideal cooling rate cannot be determined beforehand, and should be empirically adjusted for each problem.

Restarts

Sometimes it is better to move back to a solution that was significantly better rather than always move from the current state. This process is called restarting of simulated annealing. To do this, we set S and e to sbest and ebest and perhaps restart the annealing schedule. The decision to restart could be based on several criteria. Notable among these include restarting based on a fixed number of steps, depending on whether the current energy is too high from the best energy obtained so far, restarting randomly etc.

Pattern Search

A direct search algorithm for numerical search optimization depends on the objective function only through ranking a countable set of function values. It does not involve the partial derivatives of the function and hence it is also

called non-gradient or zeroth order method. This methodology involves setting up of grids in the decision space and evaluating the values of the objective function at each grid point and thereby incrementing and decrementing the grid space to get the optimum point. The point which corresponds to the best value of the objective function is considered to be the optimum solution.

Patterns

A pattern is a set of vectors $\{v_i\}$ that the pattern search algorithm uses to determine which points to search at each iteration. The set $\{v_i\}$ is defined by the number of independent variables in the objective function, N, and the positive basis set.

Two commonly used positive basis sets in pattern search algorithms are the maximal basis, with 2N vectors, and the minimal basis, with N+ 1 vector.

With GPS, the collections of vectors that form the pattern are fixed-direction vectors. For example, if there are three independent variables in the optimization problem, the default for a 2N positive basis consists of the following pattern vectors:

$V_1 = [1\ 0\ 0]$ $V_2 = [0\ 1\ 0]$ $V_3 = [0\ 0\ 1]$ $V_4 = [-1\ 0\ 0]$ $V_5 = [0\ -1\ 0]$ $V_6 = [0\ 0\ -1]$

An N+1 positive basis consists of the following default pattern vectors.

$V_1 = [1\ 0\ 0]$ $V_2 = [0\ 1\ 0]$ $V_3 = [0\ 0\ 1]$ $V_4 = [-1\ -1\ -1]$

Depending on the poll method choice, the number of vectors selected will be 2N or N+1. As in GPS, 2Nvectors consist of N vectors and their N negatives, while N+ 1 vector consist of N vectors and one that is the negative of the sum of the others.

Meshes

At each step, the pattern search algorithm searches a set of points, called a mesh, for a point that improves the objective function. The GPS and MADS algorithms form the mesh by

- Generating a set of vectors $\{d_i\}$ by multiplying each pattern vector v_i by a scalar Δ^m. Δ^m is called the mesh size.
- Adding the $\{d_i\}$ to the current point—the point with the best objective function value found at the previous step.

For example, using the GPS algorithm. suppose that:

- The current point is [1.6 3.4].
- The pattern consists of the vectors

  $V_1 = [1\ 0]$ $V_2 = [0\ 1]$ $V_3 = [-1\ 0]$ $V_4 = [0\ -1]$
- The current mesh size Δ^m is 4.

The algorithm multiplies the pattern vectors by 4 and adds them to the current point to obtain the following mesh.

$[1.6\ 3.4] + 4*[1\ 0] = [5.6\ 3.4]$

$[1.6\ 3.4] + 4*[0\ 1] = [1.6\ 7.4]$

[1.6 3.4] + 4*[-1 0] = [-2.4 3.4]

[1.6 3.4] + 4*[0 -1] = [1.6 -0.6]

The pattern vector that produces a mesh point is called its direction.

Polling

At each step, the algorithm polls the points in the current mesh by computing their objective function values. The algorithm stops polling the mesh points as soon as it finds a point whose objective function value is less than that of the current point. If this occurs, the poll is called successful and the point it finds becomes the current point at the next iteration.

The algorithm only computes the mesh points and their objective function values up to the point at which it stops the poll. If the algorithm fails to find a point that improves the objective function, the poll is called unsuccessful and the current point stays the same at the next iteration.

Expanding and Contracting

After polling, the algorithm changes the value of the mesh size Δ^m. The default is to multiply Δ^m by 2 after a successful poll and by 0.5 after an unsuccessful poll.

Pattern Search Algorithm

Pattern search finds a local minimum of an objective function by the following method, called polling. In this description, words describing pattern search quantities are in bold. The search starts at an initial point, which is taken as the current point in the first step:

- Generate a pattern of points, typically plus and minus the coordinate directions, times a mesh size, and center this pattern on the current point.
- Evaluate the objective function at every point in the pattern.
- If the minimum objective in the pattern is lower than the value at the current point, then the poll is successful, and the following happens:
  - the minimum point found becomes the current point.
  - the mesh size is doubled.
  - the algorithm proceeds to Step 1.
- If the poll is not successful, then the following happens:
  - the mesh size is halved.
  - if the mesh size is below a threshold, the iterations stop.
  - Otherwise, the current point is retained, and the algorithm proceeds at Step 1.

This simple algorithm, with some minor modifications, provides a robust and straightforward method for optimization. It requires no gradients of the objective function. It lends itself to constraints, too.

PARTICLE SWARM OPTIMIZATION

Overview of PSO

Particle swarm optimization (PSO) is a population-based stochastic approach for solving continuous and discrete optimization problems. In particle swarm optimization, agents, called particles, move in the search space of an optimization problem. The position of a particle represents a candidate solution to the optimization problem at hand. Each particle searches for better positions in the search space by changing its velocity according to rules originally inspired by behavioral models of bird flocking.

Particle swarm optimization (PSO) is by Dr. Eberhart and Dr. Kennedy in 1995,inspired by social behaviour of bird flocking or fish schooling.PSO shares many similarities with Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. The detailed information will be given in following sections. Compared to GA, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. PSO has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, and other areas.

Here the collective behavior of simple individuals interacting with their environment and each other is discussed. Someone called it as swarm intelligence. The particle swarm concept originated as a simulation of simplified social system. The original intent was to graphically simulate the choreography of bird of a bird block or fish school. However, it was found that particle swarm model can be used as an optimizer.

The Algorithm

Suppose in the following scenario, a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So the best strategy to find the food is to follow the bird which is nearest to the food. In PSO, each single solution is a "bird" in the search space. We call it "particle". All particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flight of the particles. The particles fly through the problem space by following the current optimum particles.

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest. When a particle takes part of the population as its topological neighbours, the best value is a local best and is called lbest.

After finding the two best values, the particle updates its velocity and positions with following equation (a) and (b).

*v[] = v[] + c1 * rand() * (pbest[] - present[]) + c2 * rand() * (gbest[] - present[]) (a)*

present[]=present[]+v[] (b)

v[] is the particle velocity, present[] is the current particle (solution). pbest [] and gbest[] are defined as stated before. rand () is a random number between (0,1). c_1 ,c_2 are learning factors. Usually $c_1 = c_2 = 2$.

The pseudo code of the procedure is as follows:

```
For each particle
   Initialize particle
END
```

Do

For each particle

Calculate fitness value

If the fitness value is better than the best fitness value (pBest) in history

set current value as the new pBest

End

Choose the particle with the best fitness value of all the particles as the gBest

For each particle

Calculate particle velocity according equation (a)

Update particle position according equation (b)

End

While maximum iterations or minimum error criteria is not attained

Particles' velocities on each dimension are clamped to a maximum velocity Vmax. If the sum of accelerations would cause the velocity on that dimension to exceed Vmax, which is a parameter specified by the user, then the velocity on that dimension is limited to Vmax.

PARAMETER CONTROL

One of the advantages of PSO is that PSO take real numbers as particles. It is not like GA, which needs to change to binary encoding, or special genetic operators have to be used. For example, we try to find the solution for $f(x) = x_1^2 + x_2^2+x_3^2$, the particle can be set as (x1, x_2, x_3), and fitness function is f(x). Then we can use the standard procedure to find the optimum. The searching is a repeat process, and the stop criteria are that the maximum iteration number is reached or the minimum error condition is satisfied.

A stepwise algorithm for the PSO technique is given below:

Step 1: Initialize the particles of the swarm with random position and velocity vectors.
Step 2: Evaluate the fitness of each particle using its current position x_i.
Step 3: Compare the performance of each particle to its best performance so far.
If fitness (x_i) < fitness (pbest) then (since objective is to minimize)
fitness (pbest) = fitness (x_i)
pbest = x_i
Step 4: Compare the performance of each particle to the global best particle.
If fitness (x_i) < fitness (gbest) then
fitness (gbest) = fitness (x_i)
gbest = x_i
Step 5: Change the velocity of the particle according to Equation (5.5)
Step 6: Move each particle to a new position using Equation (5.6)
Step 7: If the stopping criteria or maximum iteration is not satisfied, go to Step 2
Step 8: Stop.

The number of particles: the typical range is 20 - 40. Actually for most of the problems, 10 particles are large enough to get good results.

Dimension of particles: It is determined by the problem to be optimized, Range of particles: It is also determined by the problem to be optimized, one can specify different ranges for different dimension of particles.

V_{max}: it determines the maximum change one particle can take during one iteration. Learning factors: c_1 and c_2 usually equal to 2. However, other settings were also used in different papers. But usually c_1 equals to c_2 and ranges from

[0, 4] The stop condition: the maximum number of iterations the PSO execute and the minimum error requirement. The maximum number of iterations is to be set. This stop condition depends on the problem to be optimized.

GODLIKE- HYBRID (GLOBAL OPTIMUM DETERMINATION BY LINKING AND INTERCHANGING KINDRED EVALUATOR)

The GODLIKE algorithm was written as an attempt to improve the robustness of the meta-heuristic algorithms. GODLIKE stands for Global Optimum Determination by Linking and Interchanging Kindred Evaluators and this is exactly what it does. It uses all four algorithms simultaneously (Linking), and after convergence, or exceeding certain predefined limits, it takes random members from each population and inserts then into random other populations (Interchanging) before continuing the optimization The *interchange*-operator indeed destroys part of the convergence properties of either of the algorithms it uses. By interchanging individuals between populations, GODLIKE introduces *immigrants* into the populations that can provide alternative good solutions to the ones already being explored by one of the algorithms.

Originally, the aforementioned meta-heuristic algorithms were intended for problems with m = 1 (referred to as single-objective optimization). Many problems can indeed be stated as a single objective problem:

find min F(x) subject to

G(x) < 0 H(x) = 0

lb < x < ub

and x = [x1; x2; : : : ; xN] as before.

Where each x_{ij} is taken within the preset boundaries [lb] and [ub] (the constraints G(x) < 0 and H(x) = 0 are then usually added to F(x) in the form of penalty functions). The objective function F(x) is evaluated for each member in this population, and a new population is created based on the function values of the initial population, and a certain degree of randomness. The four algorithms do this as follows:

GA (Based on Natural Evolution)

As stated in Genetic algorithm

DE (Based on Globalized Pseudo-Derivatives)

- Randomly select three individuals from the population, say 3, 7 and 15.These individuals will function as the base vector and differentiation vectors, respectively.
- The ith individual is created according to the rule

  if rnd < Cr

  ind = pop(3; :) + F(pop(7; :) ¡ pop(15; :))

  else

  ind = pop(i; :)

  end

  where rnd is a random number, Cr is the crossover probability, and F the constant of differentiation, usually a random number in [¡1; 1].
- Do this until P new individuals have been created.

- Evaluate the objective function for all these new individuals. If a new individuals is found to have a better function value than its spawning solution, it will become part of the new population. Otherwise, the original vector is inserted. Taking the difference between the two differentiation vectors is very much like taking the derivative.

ASA (Based on the Laws of Thermodynamics)

- Randomly perturb every individual in the population. The so-called Bolz-mann generating scheme accomplishes this:

  ind = ind + sqrt(T) ¤ randn(1; dimensions);

  with randn() random numbers from the standard normal distribution, and T the current temperature.

- Evaluate the objective function for all new individuals.
- Accept or reject new individuals into the next population. If the value of the objective function is lower than before the perturbation, always accept it. If it is higher, accept it according to the probabilistic rule accept if
- rnd < exp ((E_0 - E_p)/T)where E_0 - Ep is the difference in objective function values before (E0) and after (Ep) the perturbation and T is the current temperature.
- At every new iteration the temperature is first decreased according to a cooling schedule. Usually, this cooling schedule has the form Tnew = c ¢ Told;

  where 0 < c < 1 is a constant. This form will decrease the temperature logarithmically, just as it would in physical system undergoing cooling1.

Traditionally, for this method, individual trial solutions are called atoms or particles, to reject the method's underlying philosophy - as the temperature drops, the atoms literally freeze into low-energy states (low function values). But before they freeze, they have the ability to move to higher energy states, with a certain probability (step 3). This is what makes ASA also a global optimizer, in the sense that it is not greedy as to only accept lower function values, but also explores regions behind high-energy barriers. Originally, Simulated Annealing was built around a single solution (the initial condition). However, this method is easily rewritten into a population-based method (just use N randomly generated initial conditions).

PSO (Based on Swarm Intelligence)

The last step is the crux of the algorithm. Updating velocities in this fashion will steer every particle into a direction that was found to be good by its neighbours (social learning), a direction found to be good by all individuals combined (cooperative), and a direction that each individual found to be good in the past (nostalgia). This gives the particles (the traditional name for individuals) a type of behaviour reminiscent of a swarm of insects around a good food reserve - most swarm around it, having a feeding frenzy (local optimization), while others remain swarming in a relatively large area around it (localized global search), and sometimes there are the true explorers going to completely new areas (global search).

FMINCON (NON LINEAR OPTIMIZATION)

Fmincon uses Active Set Algorithm

Overview

In constrained optimization, the general aim is to transform the problem into an easier sub problem that can then be solved and used as the basis of an iterative process. A characteristic of a large class of early methods is the translation of the constrained problem to a basic unconstrained problem by using a penalty function for constraints that are near or beyond the constraint boundary. In this way, the constrained problem is solved using a sequence of parameterized unconstrained optimizations, which in the limit (of the sequence) converge to the constrained problem. These methods are now considered relatively inefficient and have been replaced by methods that have focused on the solution of the Karush-Kuhn-Tucker (KKT) equations. The KKT equations are necessary conditions for optimality for a constrained optimization problem. If the problem is a so-called convex programming problem, that is, f(x) and $G_i(x)$, i = 1,...,m, are convex functions, then the KKT equations are both necessary and sufficient for a global solution point.

Referring to GP, the Kuhn-Tucker equations can be stated as

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla G_i(x^*) = 0$$

$$\lambda_i G_i(x^*) = 0, \quad i = 1, 2, \ldots, m_e \quad \lambda_i \geq 0, \quad i = m_e + 1 \ldots m$$

In addition to the original constraints where the equation is given by

$$\min_x f(x),$$

$$G_i(x) = 0 \quad i=1,\ldots,m_c, \quad G_i(x) \leqq 0 \quad i=m_c+1,\ldots,m,$$

Where x is the vector of length n design parameters, $f(x)$ is the objective function, which returns a scalar value, and the vector function $G(x)$ returns a vector of length m containing the values of the equality and inequality constraints evaluated at x.

An efficient and accurate solution to this problem depends not only on the size of the problem in terms of the number of constraints and design variables but also on characteristics of the objective function and constraints. The Nonlinear Programming (NP) problem in which the objective function and constraints can be nonlinear functions of the design variables. A solution of the NP problem generally requires an iterative procedure to establish a direction of search at each major iteration. This is usually achieved by the solution of an LP, a QP, or an unconstrained sub problem.

The first equation describes a cancelling of the gradients between the objective function and the active constraints at the solution point. For the gradients to be cancelled, Lagrange multipliers (λ_i, i = 1,...,m) are necessary to balance the deviations in magnitude of the objective function and constraint gradients. The 'active-set' algorithm is not a large-scale algorithm

Global-Search

The Global Search applies to problems with smooth objective and constraint functions. Global Search work is done by starting a local solver, such as fmincon, from a variety of start points by scatter search mechanism. Generally the start points are random.

Steps

Step I. Run fmincon from x_0:

Global Search runs fmincon from the start point we give in the problem structure. If this run converges, Global Search records the start point and end point for an initial estimate on the radius of a basin of attraction. Furthermore, Global Search records the final objective function value for use in the score function. The score function is the sum of the objective function value at a point and a multiple of the sum of the constraint violations. So a feasible point has a score equal to its objective function value.

Step II. Generate Trial Points

Global Search uses the scatter search algorithm to generate a set of trial points. Trial points are potential start points. For a description of the scatter search algorithm the trial points have components in the range $(-1e^{(4+1)}, 1e^{(4+1)})$. This range is not symmetric about the origin so that the origin is not in the scatter search.

Step III. Obtain Stage 1 Start Point, Run

Global Search evaluates the score function StageOnePoints, trial points. It then takes the point with the best score and runs fmincon from that point. GlobalSearch removes the set of NumStageOnePoints trial points from its list of points to examine.

Step IV. Initialize Basins, Counters, Threshold

The local Solver Threshold is initially the smaller of the two objective function values at the solution points. The solution points are the fmincon solutions starting from x0 and from the Stage 1 start point. The GlobalSearch heuristic assumption is that basins of attraction are spherical. The initial estimate of basins of attraction for the solution point from x0 and the solution point from Stage 1 are spheres centered at the solution points. The radius of each sphere is the distance from the initial point to the solution point. These estimated basins can overlap. There are two sets of counters associated with the algorithm. Each counter is the number of consecutive trial points that:

- Lie within a basin of attraction. There is one counter for each basin.
- Have score function greater than localSolverThreshold.
- All counters are initially 0.

Step V. Begin Main Loop

GlobalSearch repeatedly examines a remaining trial point from the list, and performs the following steps. It continually monitors the time, and stops the search if the elapsed time exceeds MaxTime seconds.

Step VI. Examine Stage 2 Trial Point to see if fmincon Runs Call the trial point p. Run fmincon from p if the following conditions hold:

- p is not in any existing basin. The criterion for every basin i is: $|p - center(i)| > DistanceThresholdFactor \times radius(i)$. DistanceThresholdFactor is an option (default value 0.75).
- Score (p) < localSolverThreshold.
- (optional) p satisfies bound and/or inequality constraints. This test occurs if you set the Start Points. To Run property of the GlobalSearch object to 'bounds' or 'bounds-ineqs'.

Case I: Fmincon Runs

1. Reset Counters set the counters for basins and threshold to 0.

2. Update Solution Set, if fmincon runs starting from p, it can yield a positive exit flag, which indicates convergence. In that case, GlobalSearch updates the vector of GlobalOptimSolution objects. Call the solution point xp and the objective function value f_p. There are two cases:

• For every other solution point x_q with objective function value f_q,

$|x_q - x_p| > TolX \times \max(1,|x_p|)$ Or $|f_q - f_p| > TolFun \times \max(1,|f_p|)$.

In this case, GlobalSearch creates a new element in the vector of GlobalOptimSolution objects.

• For some other solution point x_q with objective function value f_q, $|x_q - x_p| <= Tol_X \times \max(1,|x_p|)$ And $|f_q - f_p| <= TolFun \times \max(1,|f_p|)$.

In this case, GlobalSearch regards x_p as equivalent to xq. The GlobalSearch algorithm modifies the GlobalOptimSolution of x_q by adding p to the cell array of X_0 points.

3. Update Basin Radius and Threshold. If the exit flag of the current fmincon run is positive:

- Set threshold to the score value at start point p.
- b. Set basin radius for x_p equal to the maximum of the existing radius (if any) and the distance between p and x_p.

Case II: Fmincon does not Run

1. Update Counters Increment the counter for every basin containing p. Reset the counter of every other basin to 0.Increment the threshold counter if score(p) >= localSolverThreshold. Otherwise, reset the counter to 0.
2. React to Large Counter Values for each basin with counter equal to MaxWaitCycle, multiply the basin radius by 1 – BasinRadiusFactor. Reset the counter to 0. If the threshold counter equals MaxWaitCycle, increase the threshold:

   new threshold = threshold + PenaltyThresholdFactor × (1 + abs(threshold)). Reset the counter to 0.
3. Report to Iterative Display Every 200th trial point creates one line in the GlobalSearch iterative display.

Step VII. Create Global Optim Solution

After reaching Max Time seconds or running out of trial points, Global Search creates a vector of Global Optim Solution objects. Global search orders the vector by objective function value, from lowest (best) to highest (worst).

DIFFERENTIAL EVOLUTION (NUMERICAL OPTIMIZATION METHOD)

Differential evolution (DE) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality.

DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized. DE optimizes a problem by maintaining a population of candidate solutions and by creating new candidate solutions. This is done by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand.

Algorithm

A basic variant of the DE algorithm works by having a population of candidate solutions (called agents). These agents are moved around in the search-space by using simple mathematical formulae to combine the positions of existing agents from the population. If the new position of an agent is an improvement, it is accepted and forms part of the population, Otherwise, the new position is simply discarded. The process is repeated.

Formally, let f: $\mathbb{R}^n \rightarrow \mathbb{R}$ be the fitness or cost function which must be minimized. The function takes a candidate solution as an argument in the form of a vector of real numbers and produces a real number as output which indicates the fitness of the given candidate solution. The gradient of 'f' is not known. The goal is to find a solution 'm' for which $f(m) \leq f(p)$ for all p in the search-space, which would mean 'm' is the global minimum. Maximization can be performed by considering the function h = (-f) instead.

Let $x \in \mathbb{R}^n$ designate a candidate solution (agent) in the population. The basic DE algorithm can then be described as follows:

- Initialize all agents 'x' with random positions in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following: For each agent 'x' in the population do:
- Pick three agents a, b, and c from the population at random, they must be distinct from each other
    - Pick a random index $R \in \{1.., n\}$, where the highest possible value 'n' is the dimensionality of the problem to be optimized.
- Compute the agent's potentially new position $y = [y_1, ..., y_n]$ by iterating over each $i \in \{1, ..., n\}$ as follows:
    - Pick $r_i \sim U(0,1)$ uniformly from the open range (0,1)
    - If $(i=R)$ or $(r_i<CR)$ let $y_i = a_i + F(b_i - c_i)$, otherwise let $y_i = x_i$
    - If $(f(y) < f(x))$ then replace the agent in the population with the improved candidate solution, that is, set x = y in the population.
- Pick the agent from the population that has the lowest fitness and return it as the best found candidate solution.

Note that $F \in [0,2]$ is called the differential weight and $CR \in [0,1]$ is called the crossover probability, both these parameters can be selected by the practitioner along with the population size $NP > 3$, see below.

Parameter Selection

The choice of DE parameters F, CR and NP can have a large impact on optimization performance. Selecting the DE parameters that yield good performance has therefore been the subject of much research.

DIRECT ALGORITHM

The name DIRECT stands for DIviding RECTangles, but also captures the fact that it is a direct search technique. DIRECT is a sampling algorithm. That is, it requires no knowledge of the objective function gradient. Instead, the algorithm samples points in the domain, and uses the information it has obtained to decide where to search next. A global search algorithm like DIRECT can be very useful when the objective function is a black box" function or simulation.

Description of DIRECT Algorithm in Steps

Step 1: Initialization

Sample the centre point of the entire space. If the centre is feasible, set x_{min} equal to the centre point and f_{min} equal to the objective function value at this point. Set s_j=0 for j=0, 1, 2... m; t_i =0 for i= 1,...,n; and neval=1 (function evaluation counter). Set maxeval equal to the limit on the number of function evaluations (stopping criterion)

Step 2: Select Rectangles

Compute the c_j values using the current values of s_0 and s_j, j=1,…, m. If a feasible point has not been formed, select the rectangle that minimizes the rate of change required to bring the weighted constraint violations to zero. On the other hand, if feasible point has been found, identify the set of rectangles that participate in the in the lower envelope of the h_r (f^*) functions for some $f^* \leq f_{min} - \epsilon$. A good value for ϵ is $\epsilon = \max (10^{-4} f_{min}, 10^{-8})$.

Let S be the set of selected rectangles

Step 3: Choose any rectangle $r \in S$.

Step 4: Trisect and sample rectangle 'r'. Choose a splitting dimension by identifying the set of long sides of rectangles r and then choosing the long side with the smallest t_i value. If more than one side is tied for the lowest t_i value, choose the one with the lowest dimensional index. Let 'i' be the resulting splitting dimension. Note that a "long side" is defined as a side that has been split upon the least and, if integer, has a positive range. Trisect rectangle r along dimension i and increment t_i by one. Sample the midpoint of the left third, increment neval by one, and update x_{min} and $f_{min.}$ If neval = maxeval, go to step 7. Otherwise, sample the midpoint of the right third, increment neval by one, and update x_{min} and f_{min} (note that there might not be a right child when trisecting on an integer variable). Update the s_j's, j=0, 1, 2… m. If all n variables are integer, check whether a child rectangle has been reduced to a single point and, if so, delete it from further consideration. Go to Step 5.

Step 5: Update S. Set S =S – {r}. If S is not empty, go to Step 3. Otherwise go to step6.

Step 6: Iterate. Report the result of this iteration, and then go to Step 2.

Step 7: Terminate. The search is complete. Report X_{min} and f_{min} and Stop.

The following solvers are based on the Direct Algorithm but they differ from each other:

- LGO - Does not use Lipchitz constant.
- glcCluster – Uses Direct Algorithm based local solver and Clustering algorithm for generating search points.
- glcSolve – Global solver based on Direct Algorithm.

Lipchitz Global Optimisation (LGO)

TOMLAB LGO nonlinear constrained global optimization solver

function Result = lgoTL (Prob)

LGO solves continuous global constrained nonlinear problems of the type:

Min f(x) subject to $x\_L \leq x \leq x\_U$, $g(x) \leq 0$, $h(x) = 0$

Where x, x_L, x_U are n-vectors and [g(x);h(x)] is an m-vector of linear and/or nonlinear functions.

The standard TOMLAB global constrained nonlinear (glc) problem definition is:

min f(x) subject to

x_L ≤ x ≤ x_U, n variable bounds

b_L ≤ A*x ≤ b_U, m1 linear constraint

c_L ≤ c(x) ≤ c_U, nonlinear constraints

where x, x_L, x_U are n-vectors, b_L,b_U are m1-vectors, A is a $m_1$xn matrix,

c_L,c_U are m_2-vectors and c(x) is an m_2-vector of nonlinear functions.

LGO does NOT treat integer variables, which is possible to define with the glc

This is done by the separation of constraints with both upper and lower bounds

glcCluster

glcCluster solves general constrained mixed integer global optimization. glcCluster is a hybrid algorithm, that is using one of the following DIRECT algorithms: glcDirect (default), (Step 1). Step 2 is an adaptive clustering algorithm to find a suitable number of clusters, where the best point in each cluster is then used as an initial point for a local search (Step 3). The 4th step is to run the DIRECT algorithm once again, to possible improve. If the DIRECT algorithm improves the best point, a local search is finally made as Step 5 with the new best point(s) as starting points.

The algorithm has the following five phases:

Phase 0: Do local search starting with the column vector given in Prob.x_0

(if not [] or all 0s) and each of the points given as columns in Prob.X0 (if defined). Also start from adjusted centre point. Best point (xMin, fMin) is input to DIRECT routine in Phase 1.

Phase 1: Run DIRECT solver maxFunc1 function value trials

If DIRECT never finds a feasible point, it will run warm starts with maxFunc1 function evaluations until totally maxFunc3 evaluations. If DIRECT never finds a feasible point, and maxFunc is reached then glcCluster will stop

Phase 2: Apply a clustering algorithm on all sampled points by DIRECT

The algorithm finds a set of N points clusters. The point with the lowest function value in each cluster is selected. it uses clustering algorithm togenerate the points

Phase 3: Do local search with of the N point best cluster points as initial starting value. Most probably the local search will find the global optimum, if there are not too many local minima

Phase 4: If the best point in the local searches in Phase 3 is better than the best point found in the global search in Phase 1, this new point (xMin, fMin) is added as an input to the DIRECT solver and a warm start run with DIRECT doing maxFunc2 function trials is done.

Phase 5: Apply clustering algorithm as in Phase 2, Select points which are not too close to previous initial points.

Phase 6: Local search from each of the nPnt2 points with the best function value If the local search improves the best point found, then this point (xMin, fMin) could be used as an input for the next start

glcSolve

glcSolve Solves general constrained mixed integer global optimization problems. glcSolve implements the algorithm DIRECT by Donald R. Jones presented in the paper "DIRECT", Encyclopedia of Optimization, Kluwer Academic Publishers, 2001. The algorithm is expanded to handle nonlinear and linear equalities, and linear inequalities.

glcSolve solves problems of the form:

min f(x)

x

subject to $x\_L \le x \le x\_U$; $b\_L \le A(x) \le b\_U$; $c\_L \le c(x) \le c\_U$ x(i) integer, for i in I

Recommendation: Put the integers as the first variables. Put low range integers before large range integers. Linear constraints are specially treated. Equality constraints are added as penalties to the objective. Weights are computed automatically, assuming f(x) scaled to be roughly 1 at optimum. Otherwise scale f(x).

glcSolve will set f =100000 and not compute f(x) for any x where the linear constraints are not feasible. Any nonlinear constraints are computed.

CONCLUSIONS

In this paper, ten Nontraditional optimization algorithms are explained in detail. These include: Genetic Algorithm, Simulated Annealing, Pattern Search, Particle Swarm optimization, GODLIKE, Fmincon, Differential evolution, Direct algorithm, LGO, glcCluster, glcSolve. A brief description of each method is presented along with a pseudo code to facilitate their implementation. Also presented were the nontraditional optimization methods to solve Non Linear optimization Techniques which are handy to solve all types of engineering problems which are complicated. All the ten methods can be used to solve the problems and comparative results can be obtained in terms of success rate, solution quality and in terms of processing time.

GA	SA	PS	PSO	G-L	Fmincon	DE	LGO	GlcClu	GlcSol
If the maximum generations is reached (100).	Max. Time reached.	Mesh Tolerance: 10^{-6}.	Max.Generation = 200.	Max.FunEvals = 10^{-5}.	Function Tolerance: 10^{-6}	Max.Iterations=200	If the current best solution did not improve for the last iters	Function Tolerance =10^{-7}	Max.Iterations is exceeded > No. of variables * 1000.
If maximum time is reached (∞).	The avg change in value of the objective function is < 10^{-6}.	Max. Iteration: 100 * No. of Variables.	Max. Time Limit= ∞.	Max. Iterations = 20.	Max.Iterations > 400	Max.Value of function reached= 10^{-6}	Program execution time limits > 600 seconds.	Tolerance of Variables = 10^{-5}	Max.function evaluations > No. of variables * 2000 .
If average change in function value < 10^{-6}.	Max. Iterations are reached.	Max. Function Evaluation: 2000 * No. of Variables.	Average change in fitness value= 10^{-6}	Min. Iterations = 2.	Max.Time: Inf.		first local search ends, if the fun difference is < 10^{-6}	Maximum Function count = 10000	If the difference of objective function is < 10^{-6}
	If the no. of fun evaluations reached.	Max. Time Limit: Inf.	Time Limit = ∞.	Function Tolerance = 10^{-4}	Max Wait cycle: 20		If max. constrain violation exceeds	Maximum Iterations = 10000	

Table-Contd.,

GA	SA	PS	PSO	G-L	Fmincon	DE	LGO	GlcClu	GlcSol
	If the best obj func value is < or = the value of Objective limit it is stopped.	Function Tolerance: 10^{-6}	Function Tolerance= 10^{-6}	Total. Iterations = 15.					

General stopping criteria for ten nontraditional optimization techniques

REFERENCES

1. Tarek Hegazyb, Donald Grierson, Emad Elbeltagi, "Comparison among five evolutionary-based optimization algorithms," *Advanced Engineering Informatics*, vol. 19, pp. 43-53, January 2005, Vol.
2. John Tsitsikils Dimitris Bertismas, "Simulated Annealing," *Stastical Science*, 1993.
3. Jian Zhong Zhou,Ning LU, Hui Quin & You lin Yaoyao He, "Differential Evolution Algorithm combined with chaotic pattern search," *KYBERNETIKA*, p. Vol 46, 2010.
4. Global Optimization Toolbox Users' guide.
5. I C. D.IRTTUNEN AND B. E. STUCKMAN, D. R. JONES, "Lipschitzian Optimization Without the Lipschitz Constant," *JOURNAL OF OPTIMIZATION THEORY AND APPLICATION*, pp. Vol79, No1, October, 1993.
6. Mattias Bj orkman and Kenneth Holmstrom, "Global Optimization Using the DIRECT Algorithm in Matlab," *Advanced Modeling and Optimization*, pp. Volume 1, Number 2, 1999.
7. Marcus M. Edvall1, Tomlab Models, November 6, 2006.
8. Kenneth and Marcus.M Edvall, Users guide for tomlab, May5,2010.