AMO - Advanced Modeling and Optimization, Volume 12, Number 2, 2010

Population based steepest descent method

S. D. Jabeen* and A. K. Bhunia

Department of Mathematics, The University of Burdwan, Burdwan-713104, India,

Abstract

In this paper, two approaches based on steepest descent method for solving unconstrained or bound constrained optimization problems having continuously differentiable objective functions have been proposed. In the first approach, an efficient heuristic method, viz. Genetic Algorithm is applied for finding the step length in each iteration of steepest descent method. Then, this idea has been extended to a set of multi-point approximations instead of single point approximation to avoid the convergence of the existing method at local optimum point of multi-modal objective functions and a new method (we call it as population based steepest descent method) has been proposed to find the global or close-to-global optima. Finally, to demonstrate the performance of both the proposed methods, several multi-dimensional standard test functions available in the literature have been solved. The results have been compared with the same of recently developed two hybrid algorithms with respect to different comparative factors like; functions evaluations, number of iterations, CPU time (computational time) and frequency of occurrence of close-to-global solution.

Keywords*: unconstrained optimization, steepest descent method and genetic algorithm.*

1. Introduction

During several decades, optimization is an active research area due to the introduction of competitive market situation as well as globalization of market. At present, in the different sectors of any country, most of the real life problems are of increasingly complex

Corresponding authors: *syed_sdj@yahoo.co.in* (S. D. Jabeen)* and math_ak*bhunia@buruniv.ac.in* (A. K. Bhunia)

optimization problems. Due to complexity of these problems, more efficient and powerful optimization algorithms are to be derived to obtain the global solution of them. Though the gradient based iterative method is a very oldest method, but it is still a useful optimization procedure. This method exploits the derivative information of a function and is usually faster search method. Among the gradient based methods, the steepest descent method due to Cauchy (1847) is one of the oldest and most widely known minimization schemes (Bazaraa et al., 1979) for unconstrained optimization of continuously differentiable functions.

According to the existing literature, there exist other several steepest descent procedures. Armijo (1996) developed a modified steepest descent method which automatically adapts the step size. M. N. Vrahates et al. (2000) derived a steepest descent method with adaptive step size using the local estimation of Lipschitz constant. Banzilai and Borwein (1988) proposed a gradient method using different strategies for choosing the step length. Yiu et al. (2004) proposed a hybrid descent method (HDM) for global optimization of multi-dimensional non-convex functions. This method employs gradient based techniques for local neighbourhood improvement and the simulated annealing technique to by-pass local solutions. Tsai et al. (2004) proposed a hybrid method [called Hybrid Taguchi genetic algorithm (HTGA)] to solve global numerical optimization problems with continuous variables.

This paper deals with two approaches based on steepest descent method to solve the unconstrained or bound constrained minimization problems having continuously differentiable objective functions. In the first approach, the step length in each iteration of this method is computed by a heuristic method, viz. Genetic Algorithm. Then this idea is incorporated to a set of multi-points approximations instead of single point approximation and a new method (called population based steepest descent method, PSDM) is proposed to find the global or close-to-global minima. Finally, to demonstrate the effectiveness of our proposed methods, several multi-dimensional test functions are solved and the results are compared with the Armijo's modified steepest descent methods, HTGA (Tsai et al., 2004) and HDM (Yiu et al., 2004).

2. Cauchy's steepest descent method and modifications

The Cauchy's steepest descent method is one of the oldest and most widely known simplest method for solving unconstrained minimization problems as follows:

$$\text{Minimize } f(x) \tag{1}$$

where $f:\Re^n \to \Re$ is continuously differentiable function. This is an iterative method which generates a sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$ belonging to the domain of definition of $f(x)$, for which $f\left(x^{(k)}\right) \to f^*$ as $k \to \infty$. The algorithm for solving Eq. (1) is as follows:

Algorithm 1

begin

input $x^{(0)}$ *, the initial approximation of x;*

set $K \leftarrow 1$*;*

repeat

compute $\nabla f\left(x^{(k-1)}\right)$*;*

obtain the optimal value of step length λ *by minimizing* $f\left(x^{(k-1)} - \lambda \nabla f\left(x^{(k-1)}\right)\right)$ *i.e.,* $g(\lambda)$ *and store in* $\lambda^{(k)}$*;*

assign $x^{(k)} \leftarrow x^{(k-1)} - \lambda \nabla f\left(x^{(k-1)}\right)$*;*

increase K by 1 i.e., $K \leftarrow K+1$*;*

until *termination criterion is satisfied;*

print the last approximation of x along with $f(x)$*;*

end

For the termination criterion mentioned above, any one condition of following can be used to terminate the iterative process in the **Algorithm 1**:

(i) When the change in function value in two consecutive iterations is very small i.e.,

$$\left|\frac{f_{new}^{(k)} - f_{old}^{(k)}}{f_{old}^{(k)}}\right| \le \varepsilon_1$$

(ii) When the norm of the gradient of *f* is very small i.e.,

$$\left\|\nabla f\right\| \le \varepsilon_2$$

(iii) When the change in the vector in two consecutive iterations is very small i.e.,

$$\left|x_{new}^{(k)} - x_{old}^{(k)}\right| \le \varepsilon_3$$

where $\varepsilon_i (i=1,2,3)$ being very small positive numbers.

In the Cauchy's method, the main task is to find the optimal step length for getting the better approximations of the decision variables in each iteration. In the k-th iteration this step length is computed by solving another optimization problem as follows:

$$\text{Minimize } f\left(x^{(k-1)} - \lambda \nabla f\left(x^{(k-1)}\right)\right),$$

where $x^{(k-1)}$ being the *(k*-1*)* th approximation

i.e, Minimize $g(\lambda)$ where $g(\lambda) = f\left(x^{(k-1)} - \lambda \nabla f\left(x^{(k-1)}\right)\right)$ (2)

A necessary and sufficient condition for λ to be optimal in Eq. (2) is that $g'(\lambda) = 0$ which is a nonlinear equation and can be solved by any method like Newton-Raphson, Regula-Falsi, fixed point iteration method, etc. The main disadvantages for using this method is to find out the location of root in the iteration of descent method. To overcome this difficulty, Armijo (1996) proposed an alternative approach for finding an optimal step length. It is known as modified steepest descent method which is presented in **Algorithm 2**.

Algorithm 2

begin

input $x^{(0)}$*, the initial approximation;*

set $k \leftarrow 0$ *;*

compute the objective function value $f\left(x^{(k)}\right)$ *and* $\nabla f\left(x^{(k)}\right)$*;*

while *(termination criterion not satisfied)* ***do***

set $\lambda_{(k)} \leftarrow \lambda^0$*,* λ^0 *being a known arbitrary large initial step length;*

compute $y^{(k)} = x^{(k)} - \lambda_{(k)} \nabla f\left(x^{(k)}\right)$*;*

repeat

$\lambda_{(k)} \leftarrow \lambda_{(k)} / 2$ *;*

compute $y^{(k)}$*, where* $y^{(k)} \leftarrow x^{(k)} - \lambda_{(k)} \nabla f\left(x^{(k)}\right)$*;*

until $f\left(y^{(k)}\right) - f\left(x^{(k)}\right) \le -\frac{\lambda_{(k)}}{2}\left\|\nabla f\left(x^{(k)}\right)\right\|^2$ *;*

$x^{(k)} \leftarrow y^{(k)}$*,* $k \leftarrow k+1$ *;*

end while

print $x^{(k)}$*,* $f\left(x^{(k)}\right)$ *and k, the number of iterations at which the solution is found;*

end

In this method, a large step length λ^0 is set up to find out the optimal or close-to-optimal value of λ in each iteration. Then halving this length consecutively the optimal step length is obtained but, how much the initial step length should be set is unknown. In this connection, it is to be noted that for higher value of λ^0, the inner loop for calculation of best λ^0 in each iteration of the algorithm will be executed more. On the other hand, for smaller value of λ^0, the outer loop be executed more to extract an appropriate step length. So, alternatively this difficulty can be overcome by applying a heuristic method for finding the best found step length. Among the heuristic methods Genetic Algorithm (GA) may be used to overcome the difficulties raised in both the methods of Bazaraa et al., (1979) due to Cauchy and Armijo (1996).

3. GA-based approach for step length computation

To find the optimal or near to optimal value of the step length using GA, an initial population $p(0)$ having N individuals/chromosomes is first created. The gene of these individuals/chromosomes represents the step length (λ) with intuitive step length values which is chosen randomly from a predefined range. The fitness value of each chromosome is computed from the fitness function which is defined as $f\left(x-\lambda\nabla f\left(x\right)\right)$ where x denotes the approximate solution vector $x = (x_1, x_2, x_3,\ldots, x_m) \in S$ and S is the n dimensional box constraints i.e., $S = \{x = (x_1, x_2, x_3,\ldots, x_m) \in R^n: a_i \leq x_i \leq b_i, i = 1, 2,\ldots, m\}$. After the evaluation of fitness value of each chromosome, reproduction/ selection operator selects the above average chromosomes (sometimes multiple copies of better chromosomes) for the next generation. Then highly fitted chromosomes take place in the crossover operation and produce offspring exchanging some of the genetic materials of the parents. Mutation is then applied by altering the gene in order to prevent the premature convergence. The evolution process continues till the maximum number of generation is reached. The algorithm of this heuristic method for step length computation is as follows.

Algorithm 3

begin

set $t \leftarrow 0$ *;*

create an initial population $p(t)$ *(at t-th generation) with N individuals, whose genes are the step length* λ *chosen randomly ;*

evaluate the fitness of each individual of the initial population $p(t)$ *;*

find the best individual and its gene value from $p(t)$ *;*

repeat

(i) $t \leftarrow t+1$*;*

(ii) select $p(t)$ *from* $p(t-1)$ *by any selection process;*

(iii) produce offspring using genetic operators like crossover and mutation and remove the corresponding parents those who have taken part in the said operation;

(iv) evaluate the fitness value of each individual of the improved population $p(t)$ *;*

(v) find the best individual and its gene value from $p(t)$ *;*

until *termination condition is satisfied*

find the best found value of the gene (λ *));*

end

To implement the above algorithm, the following three GA operators have been considered.

(i) Exponential ranking selection operator.

(ii) Multi-parent whole arithmetical crossover operator with variable probability rate.

(iii) Non-uniform mutation operator with variable probability rate.

The primary objective of selection operator is to emphasize on the above average solutions and eliminate the below average solutions from the population for the next generation. The popularly known selection operators are ranking selection, roulette wheel selection, truncation selection, tournament selection and stochastic universal sampling selection, etc. In our work, we have applied exponential ranking selection operator.

From the mating pool, two or more chromosomes are then selected at random and crossed to reshuffle the genetic material and create better offspring. The crossover can be done in many different ways using different crossover operators. In our work, we have used multi parent whole arithmetical crossover with a variable probability rate which is a decreasing function of population age. Initially, it takes the higher prescribed value [say, $p_c(0)$], then decreases consecutively to obtain the lowest final value [say, $p_c(m\_gen)$]. Hence the variable probability rate $p_c(t)$ at the t-th generation of population has the following form;

$$p_c(t) = p_c(0)exp(-\alpha t)$$

where $\alpha = \frac{1}{m\_gen} \log\left(\frac{p_c(m\_gen)}{p_c(0)} \right)$

clearly, $0 < p_c(t) < 1$ as $0 < p_c(0) < 1$ and $0 < \exp(-\alpha t) < 1$ and it will be constant when $p_c(\text{m\_gen}) = p_c(0)$.

In the *t*-th generation, the different steps for crossover operation are given below.

Step-1: *Find the rounding off integral value of the product of p_c (t) and p_size and store it in N.*

Step-2: *Generate a random number r in [0, 1].*

Step-3: *Select randomly N number of chromosomes for crossover operation.*

Step-4: *Again, select three chromosomes from the earlier selected N chromosomes and arrange themin descending order according to their fitness values.*

Step-5: *Produce two offspring, keeping the chromosome with higher fitness value as same. Amongthese two offspring, one will be generated by the convex combination of the first two chromosomes with higher weightage of first parent. Similarly, the other one will be produced by the convex combination of all the three parent chromosomes taking higher weightage of first two parents.*

After crossover operation, the offspring will be as follows:

$$\lambda_1^* = \lambda_1$$

$$\lambda_2^* = \beta\lambda_1 + (1-\beta)\lambda_2$$

$$\lambda_3^* = \gamma\lambda_1 + \delta\lambda_2 + (1-\gamma-\delta)\lambda_3$$

where α, β, γ *are real numbers in (0,1) and* $\beta > 0.5$, $\gamma + \delta < 1$, $\gamma > \delta > 1-\gamma-\delta$

Step-6: Repeat the steps 4 and 5 for either N/3 times (if N is divisible by 3) or (N/3+1) times.

Step-7: Stop.

It is to be noted that in every generation there is an improvement in the quality of offspring. As this operation is done several times, at the end of each generation highly energetic offspring are created.

Another genetic operator used in the algorithm is the mutation operator. This operator gently sharpens the selected chromosomes to bring diversity among the population avoiding local convergence. In the existing GA literature, the popularly known mutation operators are uniform mutation, whole mutation, boundary mutation, exponential mutation, non-uniform mutation, etc. In our work, we have used the non-uniform mutation operator with the varying

rate of probability lying between [p_m (m_gen), p_m (0)], where both p_m(m_gen) and p_m (0) are prescribed. Initially it takes higher value and then decreases consecutively in the following form.

$$p_m(t) = p_m(0)exp(-\beta t)$$

where $\beta = \frac{1}{m\_gen} \log\left[\frac{p_m(m\_gen)}{p_m(0)}\right]$, t being the generation number.

Clearly, the variable mutation rate is a proper fraction as $0 < p_m(0) < 1$ and $0 < \exp(-\beta t) < 1$ and it will be uniform when $p_m(m\_gen) = p_m(0)$. This mutation operator is dependent on the age of the population. If the element (gene) V_{ik} of chromosome V_i is selected for this operation and the domain of V_{ik} is $[l_k, u_k]$ where l_k and u_k are the lower and upper bounds of the variable corresponding to the gene V_{ik}, then the new value of V_{ik} is represented by

$$V'_{ik} = \begin{cases} V_{ik} + \Delta(t, u_k - V_{ik}), & \text{if the random digit is 0} \\ V_{ik} - \Delta(t, V_{ik} - l_k), & \text{if the random digit is 1} \end{cases}$$

where k$\in \{1,2,3,...,n\}$ and $\Delta(t, y)$ returns a value in the range [0,y].

In our experiment, we have used

$$\Delta(t, y) = yr\left(1 - \frac{t}{m\_gen}\right)^b$$

where r is a random real number in [0,1], t, the current generation and b(>0) (non- uniform mutation parameter), a constant.

Using earlier mentioned advanced genetic algorithm for computation of step length, we have modified the existing steepest descent method and proposed a method named as Genetic Algorithm based Steepest Descent Method (GASDM). The algorithm of this method is as follows:

Algorithm 4

begin

set $k \leftarrow 0$ *;*

create an initial approximate solution $x^{(k)}$ *randomly from the search domain;*

calculate the objective function value $f\left(x^{(k)}\right)$ *and its gradient* $\nabla f\left(x^{(k)}\right)$*;*

while *(termination criterion not satisfied)* ***do***

find the step length $\lambda^{(k)}$ *from* ***Algorithm 3****;*

improve the solution by the iterative formula $x^{(k+1)} = x^{(k)} - \lambda^{(k)}\nabla f\left(x^{(k)}\right)$ *and calculate the improved function value* $f\left(x^{(k+1)}\right)$*;*

set $x^{(k)} \leftarrow x^{(k+1)}$, $k \leftarrow k+1$ *;*

end while

print $f\left(x^{(k+1)}\right)$, $x^{(k+1)}$ *and k, the number of iterations at which the solution is obtained;*

end

4. Population based steepest descent method (PSDM)

The solution found from **Algorithm 4** may or may not always converge to the global solution; the reason behind this is that, the method is sensitive to the initial approximation. The idea of single-point approximation search has been extended to a multi-point approximation search called population based steepest descent method (PSDM). The multiple approximations produce a multiple search paths from among which at least one converges to the global optimum. This method always consternates on the point which has higher precession among all sequences generated differently at each iteration. The algorithm of this method is given below.

Algorithm 5

begin

set $k \leftarrow 0$ *;*

create an initial approximation (population) $x^{(k)}$*, a set of individuals/chromosomes* $x_j^{(k)}$ (j = 1, 2,..., m) *whose each component/gene can be generated randomly from the search domain(in case of unconstrained optimization problems, a large space is considered as search domain);*

compute the function values $f\left(x_j^{(k)}\right)$ *for all j;*

find the best value of f from all $f\left(x_j^{(k)}\right)$ *along with* $x_j^{(k)}$ *and store it in* $f_{old}^{(k)}$ *and* $X_{old}^{(k)}$ *respectively;*

set $k \leftarrow k+1$*;*

while *(termination criterion not satisfied)* ***do***

$j \leftarrow 1$*;*

repeat

(i) find the best found value of step length λ *using* ***Algorithm 3*** *and store this value in* $\lambda_i^{(k)}$*;*

(ii) compute $x_j^{(k)} = x_j^{(k-1)} - \lambda_i^{(k-1)} \nabla f\left(x_j^{(k)}\right)$ *and* $f\left(x_j^{(k)}\right)$*;*

(iii) $j \leftarrow j+1$*;*

until $j = m$*;*

find the best value of f *from all* $f\left(x_j^{(k)}\right)$ *along with* $x_j^{(k)}$ *and store it in* $f_{new}^{(k)}$ *and* $X_{new}^{(k)}$ *respectively;*

assign $f_{old}^{(k)} \leftarrow f_{new}^{(k)}$, $x_{old}^{(k)} \leftarrow x_{new}^{(k)}$*;*

end while

print $f_{new}^{(k)}$, $X_{new}^{(k)}$ *and k, the number of iterations at which the solution is obtained;*

end

5. Numerical Results and discussions

To demonstrate the performance of our proposed algorithm, numerical experiments have been carried out independently 15 times (15 trials) considering five standard test functions (f_1 to f_5) available in the existing literature. For solving these test functions, the algorithms for GASDM and PSDM have been coded in C programming and implemented on a Pentium IV, 2.66 GHZ with 512 MB RAM PC in LINUX environment. The results have been compared with the same of hybrid Taguchi Genetic Algorithm (Tsai *et al.*, 2004) and hybrid descent method (Yiu *et al.*, 2004) and displayed those in Table 2 and Table 3. In numerical experiments, GA parameters (like crossover rate, mutation rate and maximum generation numbers) of both GASDM and PSDM (for step length computation) are defined as follows:
crossover rate $[p_c(t)] \in [0.8, 0.9]$, mutation rate $[p_m(t)] \in [0.15,0.20]$, Maximum generation (m_gen) = 5

On the other hand, the population sizes for step length computation in both the methods are displayed in Table 1.

Table 1. Population size (p_size) in GASDM and PSDM

Method	Test functions				
	f_1	f_2	f_3	f_4	f_5
GASDM	10	20	10	9	17
PSDM	10	20	10	20	15

The number of initial approximations (population) in PSDM for test functions f_1 to f_4 is 20 and for f_5, it is 15.

The details of test functions along with discussions are given below:

f_1. $f(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$

This function has global minimum at $x^* = (0, 0, 0, ..., 0)$ with $f(x^*) = 0$. For both GASDM and PSDM methods with n = 1000, the initial approximation(s) has been taken randomly from $[-n, n]^n$. In GASDM, 80% of the solutions converge to the global or close to global optima and it takes on an average 70.83s performing 2461 function evaluations. However, in PSDM method, 100% success rate has been found. It takes on an average 91.4s to converge to global point, performing 61280 function evaluations. The time taken to converge to global point is found to be far better than that obtained from HDM.

f_2. $f(x) = \frac{\pi}{n} k \sin^2(\pi y_1) + \sum_{i=1}^{n-1}\{(y_i - a)^2(1 + k\sin^2(\pi y_{i+1})\} + (y_n - a)^2)$

$y_i = 1 + 0.25(x_i - 1), \quad -10 \le x_i \le 10, \quad i = 1, 2, ..., n$

where the constant k and a are fixed at 10 and 1 respectively. This function has several local minima, with only one global minimum at $x^* = (1, 1, ..., 1)$ with $f(x^*) = 0$ irrespective of the dimension of the problem. In GASDM method, the success rate of this function has been found to be poor in case of higher dimension i.e., for n = 100, n = 1000 and much better in lower dimension i.e., for n = 10. The PSDM method shows better performance giving 100% success for n = 10 as well as n = 100 but, success rate is little down by 20% for n = 1000. The time taken by each of these methods is comparatively lower than that of HDM except the same for n = 10 in case of PSDM.

f_3. $f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$, $-100 \le x_i \le 100$, $n = 30$

This function has so many local minima. Therefore, it is challenging to obtain the global solution. From Table 3, it is seen that the success rates in GASDM and PSDM methods are 87% and 100% respectively. In this case, the average number of function evaluations in GASDM is much lower than HTGA whereas the same is much higher in PSDM.

f_4. $f(x) = \sum_{i=1}^{n} x_i^2$, $-100 \le x_i \le 100$

This function has only one local minimum which is also global located at $x^* = (0, 0, ..., 0)$ with a function value $f(x^*) = 0$. In both the methods GASDM and PSDM with dimension n = 30, 100% success rates have been achieved. On the other hand, the average number of function evaluations in each of these methods is much lower than HTGA.

f_5. $f(x) = \sum_{i=1}^{n-1} \left[100\left(x_j^2 - x_{j+1}\right)^2 + \left(x_j - 1\right)^2 \right]$, $-5 \le x_i \le 10$

The above function has a global minimum at $x^* = (1, 1, 1, ..., 1)$ with $f(x^*) = 0$. In both GASDM and PSDM methods with dimension n = 100, 100% success has been found. These methods take on average 11.05*s* and 26.12 minutes CPU times with 36110 and 325878 function evaluations respectively to reach the global value. In HTGA, the average number of function evaluations is higher than that of GASPM but lower than PSDM.

Again, from Table 3, it is observed that the results of PSDM are either better or encouraging than that of GASDM, HTGA and HDM for all test functions. In HTGA, the average number of function evaluations is lower than the same in PSDM, for all test functions except the test functions f_4. However, the memory requirement for HTGA is higher than PSDM as HTGA is a combined method of GA and well known Taguchi method which creates a large array of orthogonal matrix for generating better chromosomes from randomly selected two chromosomes. As a result, HTGA will take larger CPU time than that of PSDM. So, in comparison of different methods in the context of different factors, it can be concluded that the multi-point approximations approach PSDM is better than other methods mentioned in this work.

Table 2. Comparison of different methods

Test function (number of variables)	Average number of function evaluations (Frequency of occurrence of close to global solution [in %]) [Average CPU time]			
	PSDM	GASDM	HTGA	HDM
f_1(1000)	61280 (100) [91.4s]	7392 (80) [70.83]	- (-) [-]	- (-) [-]
f_2(10)	11336 (100) [3.18s]	5881 (73) [1.73s]	- (-) [-]	- (-) [1.57-2.25s]
f_2(100)	77507 (100) [54.94s]	8834 (33) [12.36s]	- (-) [-]	- (-) [183.4s-39.1s]
f_2(1000)	457040 (80) [36.25s]	28279 (47) [363.63s]	- (-) [-]	- (-) [4.5 – 8.8h]
f_3(30)	96831 (100) [44.8s]	9726 (87) [4.5s]	26469 (-) [-]	- (-) [-]
f_4(30)	115 (100) [0.01s]	42 (100) [0.001s]	20844 (-) [-]	- (-) [-]
f_5(100)	325878 (100) [26.12s]	36110 (100) [11.05s]	60737 (-) [-]	- (-) [-]

Table 3. Comparison of different methods

Test function (number of variables)	Best found solution (Worst found solution) {Average solution} [Standard Deviation]			
	PSDM	GASDM	HTGA	HDM
$f_1(1000)$	0 0 { 0 } [0]	0 (0.74×10^{-4}) $\{0.99 \times 10^{-5}\}$ $[0.26 \times 10^{-3}]$	- (-) { - } [-]	0 (-) { - } [-]
$f_2(10)$	0 0 { 0 } [0]	0 -0.9331 {0.1659} [0.3297]	0 (-) { - } [-]	0 (-) { - } [-]
$f_2(100)$	0 0 { 0 } [0]	0 -0.3755 {0.1206} [0.1500]	0 (-) { - } [-]	0 (-) { - } [-]
$f_2(1000)$	0 (0.31×10^{-3}) $\{0.16 \times 10^{-3}\}$ $[0.83 \times 10^{-4}]$	0 -0.0219 $\{0.42 \times 10^{-4}\}$ $[0.64 \times 10^{-4}]$	- (-) { - } [-]	0 (-) { - } [-]
$f_3(30)$	0 0 { 0 } [0]	0 (0.40×10^{-8}) $\{0.4 \times 10^{-9}\}$ $[0.11 \times 10^{-8}]$	0 (-) { - } [-]	0 (-) { - } [-]
$f_4(30)$	0 0 { 0 } [0]	0 0 { 0 } [0]	0 (-) { - } [-]	0 (-) { - } [-]
$f_5(100)$	0 0 { 0 } [0]	0 0 { 0 } [0]	0 (-) { - } [-]	0 (-) { - } [-]

6. Concluding remarks

In this paper, we have modified the existing steepest descent method (due to Cauchy, Armijo and others) by introducing an efficient heuristic method, called Genetic algorithm for finding the step length in each iteration. Then to overcome the difficulties faced in the steepest descent method, we have developed population based steepest descent method (PSDM) considering a set of points as initial approximations. As the method PSDM is a multipoint approximation method, it requires more time and more function evaluations than single point approximation methods. In the proposed PSDM, decent method is incorporated for each approximation. Due to the random selection of initial approximations from the search space, the proposed PSDM possesses the merits of global exploration, fast convergence and robustness and statistical soundness. The computational experiments show that the proposed PSDM can find the global or close to global optimal solutions and it takes lesser time and memory space than HTGA as well as HDM to solve the problem by computer. Due to this feature, this method is more efficient. However, for a particular type of problem with one global optimum point surrounded by large number of local optima, this method does not work properly. In most of the trials, it gets stuck to the local optima instead of global.

As both the methods are gradient based, the methods will be applicable only to those problems where the search space is continuous and the objective function is differentiable in R^n, n being the number of decision variables. Clearly, in solving some real life problems like; structural optimization, inventory control, numerical optimization, image processing, robotics, circuit design the proposed methods may be applied.

Acknowledgement

The authors would like to acknowledge the support under DRS Phase-III Programme provided by the UGC, India for conducting this research.

References

[1] Armijo, L. (1996), Minimization of functions having Lipschitz continuous first partial derivatives. Pacific Journal of Mathematics, 16, 1-3.

[2] Barzilai, J., Borwein, J. M. (1988). Two point step size gradient methods, IMA Journal of Numerical Analysis, 8, 141-148.

[3] Tsai, J.-T., Liu, T.-K., Chou, J.-H. (2004) Hybrid Taguchi - Genetic Algorithm for Global

Numerical Optimization, IEEE Transactions on evolutionary computation, 8(4) 365-377.

[4] Vrahatis, M. N., Androulakis, G. S., Lambrinos, J. N., Magoulas, G. D. (2000), A class of gradient unconstrained minimization algorithms with adaptive stepsize. Journal of Computational and Applied Mathematics, 114, 367-386.

[5] Yiu, K. F. C., Liu,Y., Teo, K. L. (2004), A Hybrid Descent Method for Global Optimization. Journal of Global Optimization, 28, 229-238.

[6] Bazaraa, M. S., Sherali, H. D., Shetty, C. M. (1979). Non linear programming, theory and algorithms, John Wiley and Sons.