Decision Science Letters 4 (2015) 125–136

Contents lists available at GrowingScience

Decision Science Letters

homepage: www.GrowingScience.com/dsl

A hybrid of genetic algorithm and Fletcher-Reeves for bound constrained optimization problems

Asoke Kumar Bhunia[a], Pintu Pal[b*] and Samiran Chattopadhyay[c]

[a]*Department of Mathematics, The University of Burdwan,Burdwan-713104, India*
[b]*Department of Computer Application, Asansol Engineering College, Asansol - 713305, West Bengal, India*
[c]*Department of Information Technology, Jadavpur University, Kolkata -700 032, India*

C H R O N I C L E

Article history:
Received July 10, 2014
Received in revised format:
January 4, 2015
Accepted January 5, 2015
Available online
January 5 2015

Keywords:
Hybrid Algorithm
Fletcher Reeves method
Genetic Algorithm
Bound Constrained Optimization problem
Global-optima

A B S T R A C T

In this paper a hybrid algorithm for solving bound constrained optimization problems having continuously differentiable objective functions using Fletcher Reeves method and advanced Genetic Algorithm (GA) have been proposed. In this approach, GA with advanced operators has been applied for computing the step length in the feasible direction in each iteration of Fletcher Reeves 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 and a new method, called population based Fletcher Reeves method, has been proposed to find the global or nearer to global optimum. Finally to study the performance of the proposed method, several multi-dimensional standard test functions having continuous partial derivatives have been solved. The results have been compared with the same of recently developed hybrid algorithm with respect to different comparative factors.

1. Introduction

Due to the globalization of market economy and competitive market situations, optimization is an active research area during the last few decades. In every sector of our real life situation, there are more decision making problems which are more and more complex. Most of these problems are multimodal and nonlinear in nature. The domains of these problems are non-convex. So, it is very challenging to find the global solution of these problems. Since these problems are non-linear, non-convex and multimodal, to solve those, powerful optimization techniques are to be applied. In this connection, one may apply the faster search techniques like, traditional gradient based iterative methods, which require the derivative information of the objective function. However, these methods have some difficulties, like, these methods (*i*) are dependent on the initial guess, (ii) converge to the nearest local optima from the initial guess. On the other hand, to solve the same problems, an efficient well known heuristic method, say Genetic Algorithm (GA) may be applied. But, this algorithm does not reach to the peak

* Corresponding author. Tel: +91-3213-254112 / +91-9477751512
E-mail address: p5_pal@yahoo.co.in (P. Pal)

doi: 10.5267/j.dsl.2015.1.003

always, due to premature convergence as well as the randomness of the algorithm. So, to overcome the difficulties of gradient based methods as well as genetic algorithm, researchers are motivated to develop an efficient algorithm combining both the methods. This type of algorithm is known as hybrid algorithm.

At the end of twentieth century, to increase the efficiency of any algorithm, researchers are motivated to develop hybrid algorithms combining one or more algorithms. Chelouah and Siarry (2000) developed a continuous GA designed for the global optimization of multimodal functions. Later, Yiu et al. (2004) proposed a hybrid decent method for solving global optimization problems. Pal et al. (2005) introduced an application of real coded GA for mixed integer non-linear programming in the area of inventory problems. Yao et al. (2006) provided an overview of some recent advances in evolutionary computation with a wide range of topics in optimization, learning and design using evolutionary approaches and techniques. Deep and Thakur (2007) investigated a new crossover operator for real coded GA in the area of optimization problems. Karaboga and Basturk (2008) introduced artificial bee colony (ABC) algorithm as an optimization algorithm for solving different multi-dimensional optimization problems. Luo et al. (2008) developed a hybrid approach for solving systems of nonlinear equations by including the global search capabilities of chaos optimization and the high local convergence rate of quasi-Newton method. Bhunia et al. (2011) proposed a hybrid real coded GA for an inventory model of two warehouse system with some real conditions. Recently, Rao and Patel (2012) introduced an elitist teaching-learning-based optimization algorithm for solving complex constrained optimization problems. Teaching-Learning-based optimization (TLBO) is one of the recently proposed population based algorithms which simulates the teaching-learning process of the class room. Ibrahim et al. (2014) proposed a new hybrid method (known as the BFGS-CG method) by combining the search direction between conjugate gradient methods and quasi-Newton methods.

In the present days, most of the real life problems, i.e., the problems of business, engineering, medicine, etc. are more and more complex. Most of these problems are multimodal and nonlinear and the domains of the problems are non-convex. So it is very challenging to find the global optimality of these problems. Since these problems are very complex, some powerful optimization algorithms are needed. One such technique is gradient based iterative method which needs the derivative information of a function and thus it is a faster search technique. Among all the gradient based methods, the steepest descent method of Cauchy (1947) is most widely known optimization scheme for the bound constrained optimization problems having continuously differentiable objective functions. This method converges in a zig-zag way. So to overcome this difficulty, a more improved method called Conjugate Gradient (Fletcher-Reeves) method is used in this paper.

In the proposed method the step length of the Fletcher-Reeves method in each iteration is evaluated by GA. This idea is used to a set of initial points instead of a single point to overcome the convergency of the problem to a local optimum to form a new method called population based Fletcher Reeves method (PBFRM). Finally, to study the performance of the proposed method, a set of test functions having continuously partial differentiable has been solved. The results have been compared with the same of recently developed hybrid method.

2. Fletcher-Reeves Method

The well-known steepest descent method due to Cauchy is one of the oldest and most widely known simplest methods for solving unconstrained minimization problem defined as

$\min\ f(x)$

where $f : R^n \to R$ is continuously partial differentiable function.

This is an iterative method which generates a sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \cdots$ belonging to the domain of definition of $f(x)$ for which $f\left(x^{(k)}\right) \to f^*$ as $k \to \infty$ where f^* is the minimum value of $f(x)$. However, in this method, the search direction $d^{(k)} = -\nabla f\left(x^{(k)}\right)$ give rise to the sequence of iterates $\left\{x^{(k)}\right\}$ which converges to the local minimizer x^* in a zig-zag way i.e., very slowly. So, there is a need to generate a new search directions $d^{(k)}$ which will make the sequence of iterates $\left\{x^{(k)}\right\}$ to converge rapidly to x^*. For this purpose, an improved method with better search direction (better in the sense that this direction produces the sequence of iterates to converge faster to x^*.) was developed by Fletcher and Reeves. This method is known as Fletcher and Reeves method. The different steps of this method are given in Algorithm-1.

***Algorithm*-1**

Step-1 : Start with an initial point $x^{(0)}$. Set $k = 1$.

Step-2 : Find the search direction $d^{(k-1)} = -\nabla f\left(x^{(k-1)}\right)$.

Step-3 : Determine the optimal step length $\lambda^{(k-1)}$ by minimizing

$$\phi(\lambda) = f\left(x^{(k-1)} + \lambda^{(k-1)}\right).$$

Step-4 : Compute $x^{(k)} = x^{(k-1)} + \lambda^{(k-1)} d^{(k-1)}$.

Step-5 : Compute $d^{(k)} = -g^{(k)} + \beta^{(k-1)} d^{(k-1)}$

where $\beta^{(k-1)} = \dfrac{\left\langle g^{(k)}, g^{(k)} \right\rangle}{\left\langle g^{(k-1)}, g^{(k-1)} \right\rangle}$ and $g^{(k)} = \nabla f\left(x^{(k)}\right)$.

[$\langle ... \rangle$ denotes the inner (scalar) product.]

Step-6 : Compute the optimal step length $\lambda^{(k)}$ in the direction $d^{(k)}$ and find the new point

$$x^{(k+1)} = x^{(k)} + \lambda^{(k)} d^{(k)}.$$

Step-7 : Test the optimality of the point $x^{(k+1)}$. If $x^{(k+1)}$ is optimum, stop the process. Otherwise, set $k = k+1$ and go to step-5.

To test the optimality of the point $x^{(k+1)}$, any one condition of the following can be used to terminate the iterative process in Algorithm – 1.

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

$$\left| \frac{f\left(x^{(k+1)}\right) - f\left(x^{(k)}\right)}{f\left(x^{(k)}\right)} \right| \le \varepsilon_1$$

(ii) When the magnitude of all the partial derivatives (components of the gradient) of *f* are small, *i.e.*, $\left| \dfrac{\partial f}{\partial x_i} \right| \le \varepsilon_2$, $i = 1,2,3,\ldots,n$

(iii) When the norm of gradient of *f* is very small, *i.e.*, $\|\nabla f\| \le \varepsilon_3$

(iv) When the change in the design vector in two consecutive iterations is small, *i.e.*, $\left|x^{(k+1)} - x^{(k)}\right| \le \varepsilon_4$,

where ε_i ($i=1, 2, 3,4$) are very small positive numbers. In Fletcher Reeves method, the main task is to find the optimal step length for getting the next 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 minimize $\phi(\lambda) = \min f\left(x^{(k-1)} - \lambda \nabla f\left(x^{(k-1)}\right)\right)$ where $x^{(k-1)}$ is the (k-1)-th approximation. A necessary and sufficient condition for λ to be optimal is that $\phi'(\lambda)=0$ which is a non-linear equation in λ and can be solved by any method, like, Newton-Raphson, Regular Falsi, Fixed point iteration method, etc. The main disadvantage for using these methods is to find the location of roots of the non-linear equation of Fletcher Reeves method. To overcome this difficulty, we shall use genetic algorithm (GA) for finding the best found step length (By GA, the obtained value of λ is either optimal or nearer to the optimal value. We shall call this value as best found value as we cannot prove the optimality property analytically). The different steps for finding the best found value of λ in each iteration of Fletcher Reeves method are given in the following algorithm:

***Algorithm*-2**

Step-1: Initialize the parameters of GA,
Step-2: Set $t = 0$ (t represents the current generation),
Step-3: Initialize $P(t)$ [$P(t)$ represents the current generation],
Step-4: Evaluate the fitness value of each individual of $P(t)$,
Step-5: Find the best individual from $P(t)$,
Step-6: If the termination condition is not satisfied, go to Step-7; otherwise, go to Step-11,
Step-7: Recombine $P(t)$ to produce offspring $C(t)$ using genetic operators (crossover and mutation),
Step-8: Evaluate the fitness value of each individual of $P(t)$,
Step-9: Select $P(t+1)$ from $P(t)$ and $C(t)$,
Step-10: Go to Step-6,
Step-11: Print the result,
Step-12: Stop.

3. Genetic Algorithm

To solve an optimization problem through GA (Holland, 1975; Goldberg, 1989; Michalawicz, 1996; Mitchell, 1996; Gen et al., 2000; Sakawa, 2002), it is very important to design an appropriate chromosome representation of solution. There are different types of representations among which binary and real coding representations are popular. In binary coding representation each variable is represented as binary substrings with desired precision. In this case the string length of an individual will be large and GA would perform poorly. In real coding representation each chromosome vector is encoded as a vector of floating point number of same length as the solution vector. This type of representation is easy to handle and is capable of representing quite large domains (or case of unknown domains). In this representation a vector (x_1, x_2,…, x_n) is used as an individual (chromosome) to represent a solution of the optimization problem. The next step is to initialize the chromosomes which will take part in the artificial genetic operations like natural genetics. In this way population size of chromosomes are produced in which each element is initially selected randomly within the desired domain. Among many processes for selection of a random number, here we have used the uniform distribution.

3.1 Evaluation

After initialization of population from any generation of GA, we have to find out the fitness value of each chromosome. This fitness value of each chromosome is obtained in different ways by considering different fitness function. Generally the objective function is taken as the fitness function.

3.2 Selection

The main objective of selection operator is to select good solutions for the next generation of GA replacing the bad solutions based on the well know evolutionary principle "survival of the fittest" by Charles Darwin keeping the population size as constant. Some commonly used selection procedures are roulette wheel selection, truncation selection, universal stochastic sampling selection, steady state selection, ranking selection (linear or exponential), tournament selection etc. In this work, we have used the well known exponential ranking selection.

3.3 Crossover

In each generation, crossover produces more improved offspring by combining the features of the parents. In this operation, expected N (the integral part of $p\_cross \times p\_size$, p_cross being the probability of crossover) number of chromosomes will take part. In our work, we have used whole arithmetical crossover. In this crossover, two different linear combinations of two parent chromosomes (vectors) are considered to produce the offspring. In any generation, if the parent chromosomes S_v and S_w are selected randomly from the population for crossover operation, then the produced offspring will be as follows:

$S'_w = a \times S_v + (1-a)S_w$ and $S'_v = a * S_w + (1-a)S_v$ where a is a proper fraction which can be chosen randomly.

3.4 Mutation

Like other genetic operations, mutation operation takes part to maintain genetic diversity into the population. These variations are small. It performs with low probability (mutation probability or mutation rate). Normally, after the crossover, the offspring are mutated. The probability of mutating a variable is inversely proportional to the number of variables. Thus the mutation rate is independent of the size of the population. Some methods of mutation are uniform mutation, non-uniform mutation, boundary mutation etc. In our work we have used non-uniform mutation.
If the gene (element) V_{ik} of chromosome V_i is selected for this operation and if the domain of V_{ik} is an interval $[l_k^0, l_k^1]$ then the reduced value of V_{ik} is given by

$$V'_{ik} = V_{ik} + \Delta(t, l_k^1 - V_{ik}), \text{ if a random digit is 0}$$
$$= V_{ik} - \Delta(t, V_{ik} - l_k^0), \text{ if a random digit is 1}$$

where $k \in \{1, 2, \ldots, n\}$ and $\Delta(t, y)$ returns a value in the range $[0, y]$. Here, we have taken $\Delta(t, y) = yr\left(1 - \frac{t}{m\_gen}\right)^b$, where r is a random value in [0, 1], m_gen and t represent maximum generation number and the current generation respectively and b is called the non-uniform mutation parameter which is constant.

4. Population Based Fletcher-Reeves Method (PBFRM)

The solution obtained from genetic algorithm based Fletcher Reeves method may or may not always converge to the global optimum solution. The reason behind this is that the method is sensitive to the initial approximation. To overcome this difficulty, we have proposed a new method (we call this method as population based Fletcher Reeves method) by extending the idea of single-point approximation search to a multi-point approximation search. The multiple approximations produce a

series of paths among which at least one converges to the global optimum. In this method, all the chromosomes will be improved by Fletcher Reeves method whereas the step length will be computed by genetic algorithm. The different steps in this method are given in the following algorithm:

***Algorithm*-3**

Step-1: Set $k = 0$,

Step-2: Create an initial population $x^{(k)}$, by generating each component/ gene of each individual/ chromosome $x_i^{(k)} \left(i = 1, 2, \cdots, p\_size\right)$ randomly from the search space (in case of unconstrained optimization problems, a large space is considered as search space). [p_size denotes the population size],

Step-3: Compute the function values $f\left(x_i^{(k)}\right)$ for all i,

Step-4: Find the best value of f from all $f\left(x_i^{(k)}\right)$ along with $x_i^{(k)}$ and store it in $f_{\text{old}}^{(k)}$ and $x_{\text{old}}^{(k)}$ respectively,

Step-5: Increase the value of *k* by unity i.e., $k = k + 1$,

Step-6: Set $i = 1$,

Step-7: Find the search direction $d_i^{(k-1)} = -\nabla f\left(x_i^{(k-1)}\right)$,

Step-8: Find the best found value of step length λ using Algorithm-3 and store this value in $\lambda_i^{(k-1)}$,

Step-9: Compute $x_i^{(k)} = x_i^{(k-1)} + \lambda_i^{(k-1)} d_i^{(k-1)}$,

Step-10: Compute $d_i^{(k)} = -g_i^{(k)} + \beta_i^{(k-1)} d_i^{(k-1)}$

where $\beta_i^{(k-1)} = \dfrac{\left\langle g_i^{(k)}, g_i^{(k)} \right\rangle}{\left\langle g_i^{(k-1)}, g_i^{(k-1)} \right\rangle}$ and $g_i^{(k)} = \nabla f\left(x_i^{(k)}\right)$,

Step-11: Compute the best found value of step length $\lambda_i^{(k)}$ in the direction $d_i^{(k)}$ by using Algorithm-3 and store this value in $\lambda_i^{(k)}$,

Step-12: Improve the solution $x_i^{(k+1)} = x_i^{(k)} + \lambda_i^{(k)} d_i^{(k)}$,

Step-13: Compute $f\left(x_i^{(k+1)}\right)$,

Step-14: Increase the value of *i* by unity i.e., $i = i + 1$,

Step-15: If $i =< p\_size$, go to step-7,

Step-16: Find the best value of f from all $f\left(x_i^{(k)}\right)$ along with $x_i^{(k)}$ and store it in $f_{\text{new}}^{(k)}$ and $x_{\text{new}}^{(k)}$ respectively,

Step-17: If the termination criterion is satisfied, go to step-19. Otherwise, go to step-18,

Step-18: If $f_{\text{new}}^{(k)} < f_{\text{old}}^{(k)}$, assign $f_{\text{old}}^{(k)} = f_{\text{new}}^{(k)}$ and $x_{\text{old}}^{(k)} = x_{\text{new}}^{(k)}$ and then go to step-5,

Step-19: Print the result and stop the process.

5. Numerical Illustration

To test the performance of the proposed hybrid algorithm, numerical experiments have been carried out considering 13 test problems (bound constrained optimization problems) given in Appendix-1. For solving these test problems, the algorithm for population based Fletcher Reeves method has been coded

in C programming and implemented on a Pentium IV, 2.66 GHz with 1 GB RAM PC in LINUX environment. Each test problem has been solved for different values of *n* (the number of decision variables) and in each case, 30 independent runs have been performed and the corresponding results have been collected. For the statistical analysis of the results, the following characteristics have been computed and shown in Tables 1-13.

(i) best and worst function values for 30 runs.
(ii) mean and standard deviation (S.D.) of function values in 30 runs.

It is to be noted that the population size (p_size) of GA for calculating the step length in each iteration of Fletcher Reeves method is considered as 30 in each problem whereas the maximum number of generation (m_gen) of GA and also the population size $(popsize)$ of the proposed method are different for different values of n, the number of variables of each problem. These are mentioned in Table 1 -13. In all cases, the probability of crossover (*p_cross*) and mutation (*p_mute*) are taken as 0.9 and 0.15 respectively. To compare the performance of our proposed method (PBFRM) with the method HX-NUM-GA of Deep & Thakur (2007), we have done a comparative analysis with respect to different characteristics, such as average number of function evaluation, average execution time, mean and standard deviation of the objective function value. The results have been shown in Table 14. In order to compare the quality of the solution found among these two methods, a list of various characteristics like mean, standard deviation of the best objective function value, average number of function evaluation and average execution time corresponding to each test problem as shown in Appendix-1 has been given. It is observed from this Table that for each test problem, in our proposed method, the average number of function evaluations of PBFRM is significantly lesser than the same of HX-NUM-GA method. It is also found that out of 13 test problems, PBFRM takes lesser execution time in 9 test cases and in one test case it is almost equal. Again, out of 13 test problems it is observed that in most of the cases our proposed method shows that the best and worst objective function values are same with global objective function values which have been shown in Appendix-1. However, for test problem 11, the best and worst objective function values are very close to global objective function value.

Table 1
Computational Results of Problem -1

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	0.0	0.0	0.0	0.00002	8723	6.33	10	20
5	0.0	0.0	0.0	0.0	10300	10.76	10	20
10	0.0	0.0	0.0	0.0	4070	9.01	10	20
50	0.0	0.0	0.0	0.0	9613	200.17	200	50
100	0.0	0.0	0.0	0.0	7853	847.05	200	50

Table 2
Computational Results of Problem -2

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	-0.2	0.0	-0.2	-0.2	24	0.01	10	20
5	-0.5	0.0	-0.5	-0.5	25	0.02	10	20
10	-1	0.0	-1	-1	26	0.04	10	20
50	-5	0.0	-5	-5	25	0.43	10	20
100	-10	0.0	-10	-10	24	0.93	10	20
1000	-100	0.0	-100	-100	38	3.42	10	20
2000	-200	0.0	-200	-200	33	4.99	10	20
3000	-300	0.0	-300	-300	34	7.22	10	20
4000	-400	0.0	-400	-400	35	9.40	10	20
5000	-500	0.0	-500	-500	32	10.74	10	20
10000	-1000	0.0	-1000	-1000	35	21.97	10	20

Table 3
Computational Results of Problem -3

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	-1.0	0.0	-1.0	-1.0	30	0.01	10	20
5	-1.0	0.0	-1.0	-1.0	32	0.01	10	20
10	-1.0	0.0	-1.0	-1.0	30	0.03	10	20
50	-1.0	0.0	-1.0	-1.0	86	2.21	10	20
100	-1.0	0.0	-1.0	-1.0	79	4.88	10	20
500	-1.0	0.0	-1.0	-1.0	283	86.34	100	20
1000	-1.0	0.0	-1.0	-1.0	500	324.69	200	20

Table 4
Computational Results of Problem -4

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	0.0	0.0	0.0	0.0	36933	27.97	10	20
5	0.00002	0.00003	0.0	0.00011	75447	79.56	200	20

Table 5
Computational Results of Problem -5

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	0.0	0.0	0.0	0.0	80	0.03	10	20
10	0.0	0.0	0.0	0.0	410	3.50	100	100
50	0.0	0.0	0.0	0.0	1360	104.70	200	100
100	0.0	0.0	0.0	0.0	1400	264.06	300	100

Table 6
Computational Results of Problem -6

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	-3.5	0.0	-3.5	-3.5	29	0.02	10	10
5	-3.5	0.0	-3.5	-3.5	23	0.02	10	10
10	-3.5	0.0	-3.5	-3.5	38	0.07	10	10
50	-3.5	0.0	-3.5	-3.5	212	2.69	100	20
100	-3.5	0.0	-3.5	-3.5	227	6.47	100	20
500	-3.5	0.0	-3.5	-3.5	440	124.5	200	50

Table 7
Computational Results of Problem -7

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	0.0	0.0	0.0	0.0	74	0.02	10	20
5	0.0	0.0	0.0	0.0	87	0.03	10	20
10	0.0	0.0	0.0	0.0	92	0.08	10	20
50	0.0	0.0	0.0	0.0	100	1.08	10	20
500	0.0	0.0	0.0	0.0	116	3.36	10	20
1000	0.0	0.0	0.0	0.0	120	4.16	10	20
2000	0.0	0.0	0.0	0.0	120	5.59	10	20
3000	0.0	0.0	0.0	0.0	121	7.05	10	20
4000	0.0	0.0	0.0	0.0	130	9	10	20
5000	0.0	0.0	0.0	0.0	130	10.6	10	20
10000	0.0	0.0	0.0	0.0	130	18.49	10	20

Table 8
Computational Results of Problem -8

n	Mean	S.D.	Best	Worst	Function	CPU Time	*popsize*	*m_gen*
2	0.0	0.0	0.0	0.0	78	0.04	10	20
5	0.0	0.0	0.0	0.0	288	0.18	10	20

Table 9
Computational Results of Problem -9

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	0.0	0.0	0.0	0.0	71	0.04	10	20
5	0.0	0.0	0.0	0.0	90	0.06	10	20
10	0.0	0.0	0.0	0.0	101	0.11	10	20
50	0.0	0.0	0.0	0.0	130	1.44	10	20
100	0.0	0.0	0.0	0.0	140	3.55	10	20
1000	0.0	0.0	0.0	0.0	180	6.44	10	20
2000	0.0	0.0	0.0	0.0	190	9.08	10	20
3000	0.0	0.0	0.0	0.0	190	11.3	10	20
4000	0.0	0.0	0.0	0.0	200	14.25	10	20
5000	0.0	0.0	0.0	0.0	200	16.61	10	20
10000	0.0	0.0	0.0	0.0	186	26.65	10	20

Table 10
Computational Results of Problem -10

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	0.0	0.0	0.0	0.0	68	0.02	10	20
5	0.0	0.0	0.0	0.0	81	0.03	10	20
10	0.0	0.0	0.0	0.0	120	0.11	10	20
20	0.0	0.0	0.0	0.0	185	0.2	10	20
30	0.0	0.0	0.0	0.0	312	1.48	10	20

Table 11
Computational Results of Problem -11

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	0.00347	0.00127	0.00022	0.005	17795	11.96	10	20

Table 12
Computational Results of Problem -12

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
10	-45.7785	0	-45.7785	-45.7785	24	0.22	10	20

Table 13
Computational Results of Problem -13

n	Mean	S.D.	Best Value	Worst Value	Function Evaluation	CPU Time (in s)	*popsize*	*m_gen*
2	0.0	0.0	0.0	0.0	430	0.25	10	20
5	0.0	0.0	0.0	0.0	799	0.80	10	20
10	0.0	0.0	0.0	0.0	1198	3.21	10	20
50	0.0	0.0	0.0	0.0	1189	25.42	10	20
100	0.0	0.0	0.0	0.0	1681	85.36	10	20
500	0.0	0.0	0.0	0.0	3843	400.65	10	20
1000	0.0	0.0	0.0	0.0	5376	921.00	10	20

Table 14
Comparative Study among the methods HX-NUM-GA and PBFRM

Problem number	Methods	Average number of function evaluation	Average time of execution	Mean	Standard derivation
1	HX-NUM-GA	232901	4.134	1.28E-03	1.39E-03
	PBFRM	8723	6.33	0.0	0.0
2	HX-NUM-GA	304671	5.207	-2.98E+00	5.11E-02
	PBFRM	24	0.01	-0.2	0.0
3	HX-NUM-GA	60651	0.681	-9.99E-01	3.86
	PBFRM	30	0.01	-1.0	0.0
4	HX-NUM-GA	92131	2.172	1.43E-08	7.38E-08
	PBFRM	36933	27.97	0.0	0.0
5	HX-NUM-GA	426651	7.358	3.11E-13	5.01E-13
	PBFRM	80	0.03	0.0	0.0
6	HX-NUM-GA	142751	3.774	-3.41E+00	4.56E-01
	PBFRM	29	0.02	-3.50E+00	0.0
7	HX-NUM-GA	210791	5.060	2.86E-01	1.55E+00
	PBFRM	87	0.03	0.0	0.0
8	HX-NUM-GA	107581	1.165	3.64E-04	2.69E-04
	PBFRM	78	0.04	0.0	0.0
9	HX-NUM-GA	140451	1.521	4.67E-04	3.28E-04
	PBFRM	71	0.04	0.0	0.0
10	HX-NUM-GA	811	0.2	8.20E-05	2.91E-04
	PBFRM	68	0.2	0.0	0.0
11	HX-NUM-GA	415081	4.412	3.20E+00	6.25E+00
	PBFRM	17795	11.96	3.47E-03	1.27E-03
12	HX-NUM-GA	31461	0.608	-9.97E+05	1.86E+02
	PBFRM	24	0.22	-4.58E+01	0.0
13	HX-NUM-GA	354201	8.570	3.11E-02	8.67E-02
	PBFRM	430	0.25	0.0	0.0

6. Concluding Remarks

In the proposed method, the well known Fletcher Reeves method has been applied repeatedly in each chromosome of a population which is generated initially from the search domain of the problem. The proposed method is a multipoint approximation method. As a result, it requires more execution time and more function evaluations than single point approximation methods. Due to the random selection of initial approximations from the search space, the proposed method possesses the merits of global exploration and fast convergence. As the proposed method is gradient based, the method 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. The proposed method may be applied in solving some real life decision making problems in the areas of structural optimization, inventory control, robotics, circuit decision etc.

References

Baker, J. E. (1985, July). Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications* (pp. 101-111).

Bhunia, A., Pal, P., Chattopadhyay, S., & Medya, B. (2011). An inventory model of two-warehouse system with variable demand dependent on instantaneous displayed stock and marketing decisions via hybrid RCGA. *International Journal of Industrial Engineering Computations*, *2*(2), 351-368.

Chelouah, R., & Siarry, P. (2000). A continuous genetic algorithm designed for the global optimization of multimodal functions. *Journal of Heuristics*, *6*(2), 191-213.

Jana, R. K., & Biswal, M. P. (2004). Stochastic simulation-based genetic algorithm for chance constraint programming problems with continuous random variables. *International Journal of Computer Mathematics*, *81*(9), 1069-1076.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, *186*(2), 311-338.

Deb, K., Anand, A., & Joshi, D. (2002). A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary computation*, *10*(4), 371-395.

Deep, K., & Thakur, M. (2007). A new crossover operator for real coded genetic algorithms. *Applied Mathematics and Computation*, *188*(1), 895-911.

Gen, M. & Cheng, R. (2000). *Genetic Algorithms and Engineering Optimization*. John Wiley & Sons Inc.

Goldberg, D. E. (1989). *Genetic Algorithms: Search, Optimization and Machine Learning,* Addison Wesley.

Holland, J. H. (1975). *Adaptation of Natural and Artificial system*, University of Michigan Press, Ann Arbor.

Ibrahim, M. A. H., Mamat, M., & Leong, W. J. (2014, March). The Hybrid BFGS-CG Method in Solving Unconstrained Optimization Problems. In *Abstract and Applied Analysis* (Vol. 2014). Hindawi Publishing Corporation.

Karaboga, D. & Basturk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, 8 (1), 687–697.

Luo, Y. Z., Tang, G. J., & Zhou, L. N. (2008). Hybrid approach for solving systems of nonlinear equations using chaos optimization and quasi-Newton method. *Applied Soft Computing***,** 8(2), 1068-1073.

Makinen, R.A.E., Periaux, J. & Toivanen, J. (1999). Multidisciplinary shape optimization in aerodynamics and electromagnetic using genetic algorithm, *International Journal for Numerical Methods in Fluids*, 30(2),149-159.

Miettinen, K., Mäkelä, M. M., & Toivanen, J. (2003). Numerical comparison of some penalty-based constraint handling techniques in genetic algorithms. *Journal of Global Optimization*, *27*(4), 427-446.

Michalawicz, Z. (1996). *Genetic Algorithms + Data structure= Evaluation Programs*. Springer Verlog, Berlin.

Mitchell, M. (1996). *Introduction to Genetic Algorithms*, PHI, New Delhi.

Pal, P., Das, B., Panda, A. & Bhunia, A. K. (2005). An application of real-coded genetic algorithm for mixed integer non-linear programming in an optimal two-warehouse inventory policy for deteriorating items with a linear trend in demand and a fixed planning horizon. *International Journal of Computer Mathematics*, 82(2), 167-175.

Rao, R. V. & Patel, V. (2012). An elitist teaching-learning-based optimization algorithm for solving complex constrained optimization problems. *International Journal of Industrial Engineering Computations*, 3, 535–560.

Sakawa, M. (2002). *Genetic Algorithms and fuzzy multiobjective optimization*. USA: Kluwer Academic Publishers.

Yao, X., & Xu, Y. (2006). Recent Advances in Evolutionary Computation. *Journal of Computer Science and Technology*, 21(1), 1-18.

Yiu, K. F. C., Liu, Y., & Teo, K. L. (2004). A hybrid descent method for global optimization. *Journal of Global Optimization*, *28*(2), 229-238.

Appendix 1
Numerical Examples

Test Functions	Search Region	Global Objective fn. Value
$f_1(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	[-30,30]	0
$f_2(x) = \sum_{i=1}^{n}x_i^2 - 0.1\sum_{i=1}^{n}\cos(5\pi x_i)$	[-1,1]	-0.1*n
$f_3(x) = -\exp\left(-0.5\sum_{i=1}^{n}x_i^2\right)$	[-1,1]	-1
$f_4(x) = 0.1\left(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right] + (x_n-1)^2\left[1 + \sin^2(2\pi x_n)\right]\right)$	[-5,5]	0
$f_5(x) = 10n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right]$	[-5.12,5.12]	0
$f_6(x) = -\left[2.5\prod_{i=1}^{n}\sin(x_i-30) + \prod_{i=1}^{n}\sin(5(x_i-30))\right]$	[0,π]	-3.5
$f_7(x) = \sum_{i=1}^{n}x_i^2 + \left(\sum_{i=1}^{n}\frac{i}{2}x_i\right)^2 + \left(\sum_{i=1}^{n}\frac{i}{2}x_i\right)^4$	[-5.12,5.12]	0
$f_8(x) = \sum_{i=1}^{n}x_i^2$	[-5.12,5.12]	0
$f_9(x) = \sum_{i=1}^{n}ix_i^2$	[-5.12,5.12]	0
$f_{10}(x) = \sum_{i=1}^{n}\left(x_i^4 + rand(0,1)\right)$	[-10,10]	0
$f_{11}(x) = \sum_{i=1}^{n}(x_i-1)^2$	[-n,n]	0
$f_{12}(x) = \sum_{i=1}^{n}\left[\left(\ln(x_i-2)\right)^2 + \left(\ln(10-x_i)\right)^2\right] - \left(\prod_{i=1}^{n}x_i\right)^{0.2}$	[2,10]	-45.77847
$f_{13}(x) = \frac{\pi}{n}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right] + (y_n-1)^2\right)$, $y_i = 1 + \frac{1}{4}(x_i+1)$	[-10,10]	0