A Gradient Projection based Hybrid Constrained Optimization Methodology using Genetic Algorithms*

Amit Saha, Rituparna Datta* and Kalyanmoy Deb

KanGAL Report Number 2010006

July 30, 2010

Abstract

Genetic Algorithms (GAs) are a highly successful population based approach to solve global optimization problems. They have carved out a niche for themselves in solving optimization problems of varying difficulty levels involving single and multiple objectives. Most real-world optimization problems involve equality and / or inequality constraints and hence posed as constrained optimization problems. The most common approach to solve such problems using GAs is the method of penalty functions, which however suffers from the drawback of appropriate selection of penalty parameters for their optimal functioning.

Given the nature of the problems at hand, we have used an adaptive mutation based Real-Coded GA (RGA), which uses a popular penalty parameter-less approach to handle constraints and search the feasible region effectively for the global best solution, and at the same time use an adaptive mutation strategy to maintain diversity in the population to enable creation of new solutions. We have coupled our RGA with ideas from the *gradient projection method* to specifically handle equality constraints. We have found our simple procedure working quite well in most of the test problems provided as part of the competition on Single-objective Constrained Real Parameter Optimization in CEC 2010 and hence simplicity remains the hallmark of our study here.

1 Introduction

Most real-world optimization problems involves many constraints that must be satisfied by the solution of the optimization problem. A constrained optimization problem is usually formulated as a nonlinear programming (NLP) problem of the following type:

$$\begin{array}{rl} \text{Minimize} & f(\vec{x}), \\ \text{Subject to} & g_j(\vec{x}) \geq 0, j = 1, .., J, \\ & h_k(\vec{x}) = 0, k = 1, .., K, \\ & x_i^l \leq x_i \leq x_i^u, i = 1, .., n. \end{array}$$

In the above NLP problem, there are n variables, J greater-than-equal-to type constraints, and K equality constraints. The function $f(\vec{x})$ is the objective function, $g_j(\vec{x})$ is the $j-$th inequality constraint and $h_k(\vec{x})$ is the $k-$th equality constraint. The i-th variable varies in the range $[x_i^l, x_i^u]$.

While solving a set of generic problems, it is always better to use a simple yet efficient strategy which will work adequately in most given problems. As the no-free lunch theory [12] suggests, no one algorithm can be found to work

*A. Saha, R. Datta and K. Deb are with the Kanpur Genetic Algorithms Laboratory (KanGAL), Department of Mechanical Engineering, Indian Institute of Technology Kanpur, PIN 208016, India (Email: {asaha,rdatta,deb}@iitk.ac.in)
*corresponding author

most efficiently in all problems. Thus, it is natural to expect that a simple strategy will not work well in some problems, particularly if the set of problems to be solved is large. In this study, we have decided to use a simple strategy and hopefully the loss of its performance in some problems will get out-weighed by their ability to solve a large number of problems.

The most common procedure to handle constraints in a optimization problem is the penalty function based approach [7, 1, 10, 11]. In the penalty function method a penalty term corresponding to the constraint violation is added to the objective function. GA based techniques to solve constraint optimization problems also adopt the same strategy. The main drawback in this approach is the performance dependency of the search procedure on the selection of the penalty parameters. To save us from the additional burden of being able to select the optimal values of the penalty parameters, we have used Deb's penalty parameter less approach for constraint handling [2]. Maintenance of population diversity in constrained problems is an important consideration since feasible solutions may soon occupy most of the search space and new diverse solutions will not be formed, thus decreasing the exploring ability of the algorithm. Rather on relying on a fixed mutation probability as is usually done is not a very good idea. We have used an *adaptive mutation* operator, to maintain diversity in the search space. [2]

This paper is devoted to the competition on 18 single objective constrained real-parameter optimization problems, differing from each other in their unimodality or multi-modality, equality and/or inequality constraints, number of constraints and feasibility region. [8] The results suggest that our approach is able to find good feasible solutions in most of the problems.

In the following section, we describe the algorithm and the proposed procedure. In Section 3 and 4, we present the parameter setting and simulation results in tabular and graphical forms, respectively, and the paper is concluded in section 5.

2 Description of the Algorithm

2.1 Real-coded Genetic Algorithms

GAs are generic search procedures which use principles inspired by natural evolution to evolve solutions to a problems. The basic idea is to maintain a *population* of solutions (each solution is called a *chromosome*) and evolve them over time through a process of controlled exploration and exploitation. Each solution is assigned a *fitness* value which is used to *select* the solutions (*parents*) which shall recombine to form new solutions *children*. The fitness also determines the survival of a solution in this *simulated evolution*. The new solutions are created using the genetic operators- *crossover* and *mutation*. Together with the *selection* operator, these GA operators enable the algorithm to effectively explore the entire search space, looking for the best solution(s).

A key consideration before using a GA to solve a problem is the representation of the solution in a population of solutions. The binary coded GA (BGA), uses binary encoding to represent a solution in the population, i.e a solution is simply a bit string. Though they are simple to use and implement, BGA suffers from limitations which make their usage impractical to real world search and optimization problems [5].

A very useful form of representation is the real-coded GAs (RGAs). In this representation, the solutions are represented as they are, in the population and hence eliminates the need to encode the solutions, as required in a BGA. This also makes the RGA extremely suitable for a large class of search and optimization tasks. A large number of crossover and mutation strategies have been devised for RGA. Herrera et. al [6] provides a extensive overview of RGAs. In our work we have used the RGA with the *Simulated Binary Crossover* (SBX) as the crossover operator and a variation of the *polynomial mutation* operator [4].

2.2 Adaptive polynomial mutation

We start with a certain mutation probability p_m, and a polynomial mutation distribution index η_m [4]. The property of η_m is such that by varying its value, we can vary the perturbance in the mutated solution from the original solution. If the value of η_m is large, a small perturbation in the value of a variable is achieved. The effect of p_m is to control the number of variables to be mutated.

In the initial stages, the population diversity is well maintained. Hence, we do not want a high degree of mutation, but we want that the mutated solutions are created not too near the original solutions. Hence we would like a small value of p_m and a small value of η_m. As the GA progresses, the population becomes less and less diverse. Thus, we want introduce a high mutation probability p_m and less perturbance (high η_m) in the mutated solutions. We use the following rules to achieve the above adaptation:

$$\begin{aligned} p_m &= \frac{1}{n} + \frac{t}{t_{max}}(1 - \frac{1}{n}), \quad (1) \\ \eta_m &= 80 + t. \quad (2) \end{aligned}$$

where t is the current generation, t_{max} is the maximum number of generations, n is the dimensionality of the problem. Initial values of $p_m = 1/n$ and $\eta_m = 80$ are chosen according to the standard fixed mutation strategy applications of a real-coded GA [4].

2.3 Parameter-less approach to constraint handling in GAs

Various constraint handling methods in GAs have been described elsewhere and they most commonly fall into one of the five categories [4, 9]:

1. Methods based on preserving feasibility of solutions
2. Methods based on penalty functions
3. Methods biasing feasible over infeasible solutions
4. Methods based on decoders
5. Hybrid methods

Among these, the methods based on penalty functions is very commonly used because of its simple idea: *penalize the infeasible solution* [10]. However, as has been shown elsewhere, this method suffers from the difficulty of being able to choose the right penalty parameter [2]. In our work, we have used Deb's penalty parameter-less approach which eliminates the need to define any penalty parameter, by making use of intuitive ideas in the *tournament selection* procedure. A feasible solution is always chosen in favor of a infeasible solution. When two infeasible solutions are compared, the one with lesser *constraint violation* is selected in favor of the other. When two feasible solutions are compared, the one with a better objective function value is chosen. Since no infeasible solution is compared with any feasible solution numerically, there is no need of any penalty parameter. Although this is a simple strategy and has been found to work on many problems, there may exist some typical problems where a more sophisticated strategy to differentiate an infeasible solution from a feasible solution may be needed. But for solving a generic set of test problems, we use a generic yet efficient constraint handling strategy. Besides the constraint handling strategy, we also adopt an adaptive mutation operator which mutates more variables but with a lesser extent as the iterations progress.

Steps of the proposed procedure are described below:

1. Population is initialized randomly.
2. RGA is applied on the above population.
3. As the population evolves, the mutation operator is adaptive to ensure maintenance of diversity.
4. Invoke *Gradient projection* as a repair operator to introduce feasible solutions in problems with equality constraints (described later).
5. The maximum number of solution evaluations allowed is used as the termination criteria.

2.4 Handling of equality constraints by Gradient projection method

In our experiments we found that problems containing *equality constraints* were specially difficult to solve using the above procedure. We observed that due to the extremely small feasible region (for example in problems $C02, C03, C04$), the number of feasible solutions were very less and hence the RGA resulted in an unsuccessful search. To increase the number of feasible solutions in the population greedily, we borrowed ideas from the *gradient projection method* (GP) [3], which is a method to solve NLP problems having equality constraints only. Without going into the details of the method, we shall just describe the idea that we have borrowed from the method for our work.

One intermediate step in this method is to project a solution iteratively to the intersecting surface of the active constraints, thus creating a feasible point. This is the idea that we have borrowed. This step can be represented as follows:

$$\vec{X}^{t+1} = \vec{X}^{t} - A^T(AA^T)^{-1}H$$

where $\vec{X}^{t+1}$ and $\vec{X}^{t}$ are the next and the current solutions respectively. A is the *grad* of the constraints, (the gradients have been calculated numerically) and H is a column matrix containing the constraint values at $\vec{X}$ respectively. The iteration continues till $\max(H) \geq \epsilon$. The value of epsilon has a direct effect on the quality of the solutions obtained and in our experiments we have used a value of 0.0001 in most of the problems and have relaxed it to 0.001 in some of the problems, which (according to us) were slightly more difficult.After we have $\vec{X}^{t+1}$, we replace the current solution, $\vec{X}$ by $\vec{X}^{t+1}$. Hence, a feasible solution get introduced into the population after a GP operation.

As described, we use the GP method as a *repair* operator in a GA (henceforth referred to as a *GA-GP*) so as to increase the number of feasible solutions in the population. In the early stages of the GA, the population is mainly composed of random individuals, with a possibly very high constraint violation. Our experiments suggest that the GP approach could repair even the worst of individuals, it takes a huge number of iterations of the (and hence gradient calculations) to do so, as expected. This prevented us from reaching the global optima within the maximum number of function evaluations allowed for the CEC 2010 competion problems [8]. To avoid exceeding the maximum number of function evaluations before closing in on the optima, we use our adaptive mutation operator in the early stages of the GA and start applying our GP operator at a later stage.

3 Parameters Setting

3.1 Test Suite

The performance of our algorithm is tested on a set of 18 benchmark problems [8].

3.2 PC Configuration

- System: Debian Linux 5.0 (2.6.26 Kernel)
- CPU: P-IV 3.0 GHz (Dual core)
- RAM: 2 GB
- Language: ANSI-C
- Compiler Used: GCC version-4.3.2

3.3 Parameters Setting

- Population size (N) = 150.

- Number of Generations = $\frac{\text{FES}}{populationsize}$
  (where FES is the number of function evaluations permitted) or, exhaustion of FES, whichever is earlier
- Probability of crossover $p_c = 0.9$
- Distribution index for SBX crossover $\eta_c = 10$
- Initial probability of mutation $p_m = \frac{1}{n}$, where n is the number of variables in the problem
- Initial distribution index for polynomial mutation $\eta_m = 100$

4 Simulation Results

In this section, we discuss the simulation results.

4.1 Function values achieved

The obtained results are presented in Tables 1, 2, 3, 4, 5 and 6. Best, median, worst, mean and standard deviation of 25 runs for each test problem are done for different number of function evaluations (FES).

To demonstrate the convergence of our algorithm, Figures 1 to 4 show the variation of best objective function value during a generation with the generation counter for some of the problems. We have considered the results after the maximum allowed function evaluations (FES) for the problems. In all of these cases, the objective function steadily reduces to a value, which we have reported in the tables discussed above.

4.2 Feasibility Rate

A feasible run is one during which at least one feasible solution is found. The feasibility rate for each of the 18 problems $C01$ to $C18$ is defined as: (# of feasible runs)$/25$. The results for each problem are shown in Table 8. The results are based on the runs for the maximum allowed function evaluation for 10D as well as 30D.

4.3 Algorithm Complexity

Table 7 shows the complexity of the procedure. $T1 = (\sum_{i=1}^{m} t1_i)/m$, where $t1_i$ is the computing time for 10,000 function evaluation for problem i and m is the total number of test problems. Here $m = 18$.
$T2 = (\sum_{i=1}^{m} t2_i)/m$, where $t2_i$ is the computing time for the algorithm with 10,000 function evaluation for problem i.

5 Conclusions

In this paper, we have presented a hybrid approach to solve constrained optimization problems, which seems to be quite successful in solving single objective constrained real parameter optimization problems. The performance of the algorithm is tested on 18 test problems. Constraints are handled using a penalty parameter-less approach suggested elsewhere [2]. This operator compares two solutions (feasible or infeasible) and chooses the better feasible solution in terms of objective function value, better infeasible solution in terms of constraint violation value, and feasible solution when compared with an infeasible solution.

As already mentioned, we faced difficulties getting feasible solutions in problems having equality constraints and hence used ideas from a classical optimization method to help our search. Our results suggest that our approach performs highly for most of the problems, with particularly bad results for problems C03, C08, C09, C14, C15. We wish to emphasize that in our experiments, we could achieve near-optimal (according to us) solutions for these and other problems, when we allowed the GA to evolve over more generations, which however exceeded the maximum allowed function evaluations by a huge number. We hope to rectify this limitation of in a future work.

Simplicity is the hallmark of our approach in solving the test problems. The succesful coupling of the GP with a GA remains the contribution of our paper. A self-adaptive approach can be developed which would be able to use the GP in a *switch on/off* fashion, as a solution repair operator (which now is hard-coded) when the solution quality doesn't improve. It may not be necessary to apply the GP to every individual in the population, like we are doing now. The gradient calculation in the GP operator is an expensive operation and a suitable gradient approximation method may be tried for the same. Either of these will drastically reduce the number of function evaluations required for a successfull search. Further experiments with the GP approach is required to ensure its claim as a definitive and robust approach to handle equality constraints in GA and other Evolutionary Algorithms.

The source code will be made available from our website (http://www.iitk.ac.in/kangal/) at a later date.

6 Acknowledgement

The study is funded by Department of Science and Technology, Government of India under SERC-Engineering Sciences scheme (No. SR/S3/MERC/091/2009).

References

[1] K. Deb. Optimal design of a welded beam structure via genetic algorithms. *AIAA Journal*, 29(11):2013–2015, 1991.

[2] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2-4):311–338, 2000.

[3] K. Deb. *Optimization for engineering design: Algorithms and examples*. PHI, 1998.

[4] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001.

[5] D. Goldberg. *Genetic Algorithms in Search and Optimization*. Addison-wesley, 1989.

[6] F. Herrera, M. Lozano, and J. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[7] A. Homaifar, S. H.-V. Lai, and X. Qi. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–254, 1994.

[8] R. Mallipeddi and P. Suganthan. R. Mallipeddi, P. N. Suganthan, "Problem Definitions and Evaluation Criteria for the CEC 2010 Competition on Constrained Real-Parameter Optimization", Technical Report, Nanyang Technological University, Singapore
2010.

[9] Z. Michalewicz. Genetic algorithms, numerical optimization, and constraints. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 151–158. Citeseer, 1995.

[10] Z. Michalewicz and C. Janikow. Handling constraints in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 151–157. Morgan Kaufmann Publishers, 1991.

[11] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation Journal*, 4(1):1–32, 1996.

[12] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Table 1: Function Values achieved when FES = 2 $\times 10^4$, FES = 1 $\times 10^5$, FES = 2 $\times 10^5$ for 10D Problems C01-C06.

FES		C01	C02	C03	C04	C05	C06
$\mathbf{2 \times 10^4}$	Best	-0.7462	-1.5715	2.4850e10	0.5071	20.3412	-553.7438
	Median	-0.7280	-0.0965	2.5493e10	12.7567	43.9763	-248.6010
	Worst	-0.6686	1.8055	1.1570e15	32.2091	84.889	-66.8885
	c	0,0,0	0,0,0	1,0,0	4,0,0	0,2,0	0,2,0
	v	0	0	573	10.3640	0.055	0.055
	Mean	-0.7215	-0.0162	2.5493e12	11.4361	51.618	118.16
	Std	0.0235	0.8232	2.3647e7	7.3616	29.3460	288.15
$\mathbf{1 \times 10^5}$	Best	-0.7472	-2.0276	1.4152e8	0.0002	20.3412	-553.7438
	Median	-0.7289	-1.1903	1.6123e9	1.6072	40.3649	-364.9186
	Worst	-0.6521	-0.2652	6.4076e11	4.7730	82.1236	-66.8885
	c	0,0,0	0,0,0	1,0,0	2,2,0	0,2,0	0,1,1
	v	0	0	0.1341	3.2334	0.0055	0.0025
	Mean	-0.7212	-0.1185	2.5493e12	9.9056	51.618	-118.16
	Std	0.0267	0.5284	2.3647e7	8.7894	29.3460	288.15
$\mathbf{2 \times 10^5}$	Best	-0.7472	-2.2347	1.4152e8	0.0000	43.9763	-553.7438
	Median	-0.7291	-1.5203	1.6123e9	0.0000	104.2504	-364.9186
	Worst	-0.6650	-0.5258	6.4076e11	0.0155	132.3323	-66.8885
	c	0,0,0	0,0,0	1,0,0	0,0,0	0,2,0	0,1,1
	v	0	0	0.1341	0	0.044	0.0025
	Mean	-0.7212	1.4816	2.5493e12	0.0012	51.618	-118.16
	Std	0.0262	0.4935	2.3647e7	0.0042	29.3460	-118.16

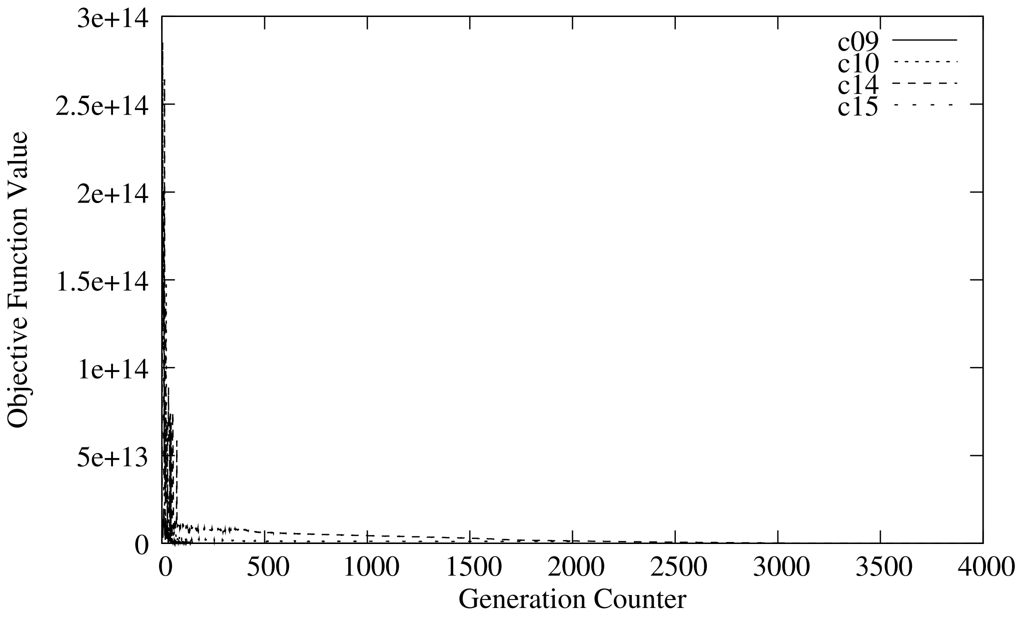

Figure 1: Convergence graph for 10D problems - C09,C10,C14,C15

Table 2: Function Values achieved when FES = 2 $\times 10^4$, FES = 1 $\times 10^5$, FES = 2 $\times 10^5$ for 10D Problems C07-C12.

FES		C07	C08	C09	C10	C11	C12
$\mathbf{2 \times 10^4}$	Best	19.4353	5.8496e5	6.0398e7	7.9453e8	-4.0799	-333.7201
	Median	69.3404	3.8256e6	2.8186e8	8.1816e9	0.0855	-150.2771
	Worst	129.3569	7.6837e6	4.7713e9	9.6874e10	1.1281	-0.0440
	c	0,0,0	0,0,0	0,0,0	0,0,0	1,0,0	1,0,0
	v	0	0	0	0	76.806	501.714
	Mean	81.371	8.9891e5	2.7182e8	1.0399e9	0.8741	-14.922
	Std	46.277	2.87e5	3.1212e2	5.8391e4	2.6878	167.47
$\mathbf{1 \times 10^5}$	Best	7.8184	1.3523	6.0398e6	6.1238e8	-4.1990	-333.7201
	Median	9.1042	43.1368	4.5726e7	7.6239e8	-0.0086	-150.2271
	Worst	15.1409	130.2569	5.6925e7	9.8225e9	0.9432	-0.0440
	c	0,0,0	0,0,0	0,0,0	0,0,0	1,0,0	1,0,0
	v	0	0	0	0	3.8420	501.714
	Mean	10.195	40.674	2.7182e8	1.0399e9	0.6580	-14.922
	Std	3.2708	54.514	3.1212e2	5.8391e4	2.7323	167.47
$\mathbf{2 \times 10^5}$	Best	6.9610	0.5762	6.0398e6	1.8238e8	-4.1958	-333.7201
	Median	7.9722	1.3484	4.5726e7	5.3639e8	-0.0071	-150.2271
	Worst	14.7666	29.9938	5.6925e7	9.7825e9	0.9039	-0.0440
	c	0,0,0	0,0,0	0,0,0	0,0,0	1,0,0	1,0,0
	v	0	0	0	0	1.5031	501.714
	Mean	6.2082	6.2082	2.7182e8	1.0399e9	0.7430	-14.922
	Std	9.2989	9.2989	3.1212e2	5.8391e4	2.6494	167.47

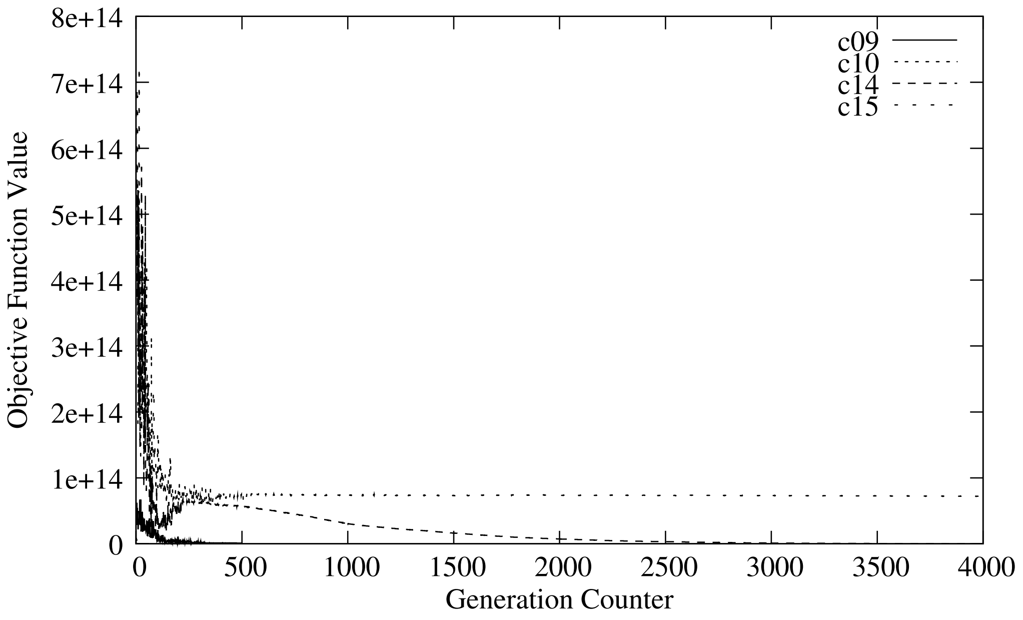

Figure 2: Convergence graph for 30D problems - C09,C10,C14,C15

Table 3: Function Values achieved when FES = 2 $\times 10^4$, FES = 1 $\times 10^5$, FES = 2 $\times 10^5$ for 10D Problems C13-C18.

FES		C13	C14	C15	C16	C17	C18
$\mathbf{2 \times 10^4}$	Best	-68.2238	7.1849e8	8.5452e8	0.1504	0.6973	0.2422
	Median	-66.7409	8.1130e9	9.1845e8	0.9393	2.8911	1.6693
	Worst	-61.6230	9.7086e10	11.4561e8	1.2214	6.4109	2.2058
	c	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0
	v	0	0,0,0	0	0	0	0
	Mean	-65.913	8.8713e8	8.9212e8	0.8298	3.8884	2.3670
	Std	1.9200	2.6712e4	4.1219e4	0.3712	3.4646	1.9309
$\mathbf{1 \times 10^5}$	Best	-68.4110	6.3720e4	8.5462e8	0.1504	0.1721	0.2422
	Median	-66.8946	8.0369e8	9.1245e8	0.9393	0.5229	1.6693
	Worst	-62.2600	9.1365e9	10.1365e8	1.2214	0.7504	2.2058
	c	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0
	v	0	0	0	0	0	0
	Mean	-66.059	7.1213e5	8.9212e8	0.8298	1.1230	2.3670
	Std	1.9748	4.4231e4	4.1219e4	0.37112	1.3894	1.9309
$\mathbf{2 \times 10^5}$	Best	-68.4320	2.0106e3	8.5462e8	0.1504	0.0780	0.2422
	Median	-67.5296	8.0369e8	9.1245e8	0.9393	0.2816	1.6693
	Worst	-61.6483	9.1365e9	10.1365e8	1.2214	0.6305	2.4572
	c	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0
	v	0	0	0	0	0	0
	Mean	-66.105	4.9312e4	8.9212e	0.8298	0.6654	2.3670
	Std	2.2288	3.4121e3	4.1219e4	0.3711	1.0316	1.9309

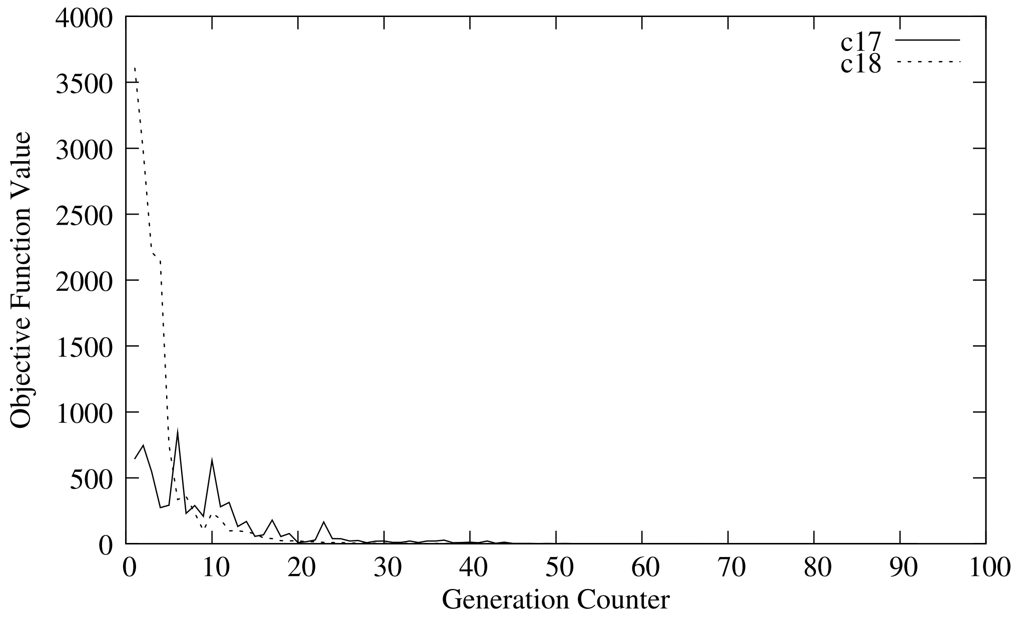

Figure 3: Convergence graph for 10D problems - C17,C18

Table 4: Function Values achieved when FES = 6 $\times 10^4$, FES = 3 $\times 10^5$, FES = 6 $\times 10^5$ for 30D Problems C01-C06.

FES		C01	C02	C03	C04	C05	C06
	Best	-0.8016	-0.07959	2.6807e12	1.9295	371.6287	-373.1025
	Median	-0.7696	0.841770	3.5586e13	18.1144	382.3989	-162.9108
$\mathbf{6 \times 10^4}$	Worst	-0.7154	1.782582	6.3661e13	37.2485	566.606	-16.9610
	c	0,0,0	0,0,0	1,0,0	4,0,0	0,2,0	0,0,2
	v	0	0	2584.4234	52.4613	0.2407	0.06
	Mean	-0.7644	0.7688	3.5687e12	19.837	387.91	-196.59
	Std	0.0261	0.4839	1.3714e15	11.627	158.44	93.993
	Best	-0.8016	-0.9861	2.6807e12	1.6512	329.8311	-373.1025
	Median	-0.7891	0.0012	3.5586e13	11.211	376.9897	-162.9108
$\mathbf{3 \times 10^5}$	Worst	-0.7231	0.1021	6.3661e13	48.8781	467.9871	-16.9610
	c	0,0,0	0,0,0	1,0,0	4,0,0	0,0,0	0,0,2
	v	0	0	2584.9811	1,3,0	0	0.06
	Mean	-0.7651	-0.0133	3.5687e1	19.7813	391.1213	-196.59
	Std	0.029	0.4778	1.3714e15	13.1811	89.1241	93.993
	Best	-0.8102	-1.1643	2.604e12	0.8394	124.6690	-373.1025
	Median	-0.7723	-0.4707	3.5586e13	9.5396	237.6667	-162.9108
$\mathbf{6 \times 10^5}$	Worst	-0.7071	0.02987	6.3661e13	36.9704	387.4809	-16.9610
	c	0,0,0	0,0,0	1,0,0	1,2,1	0,0,0	0,0,2
	v	0	0	1.0523	10.4896	0	0.06
	Mean	-0.7723	-0.3958	3.5687e12	17.148	187.1213	-196.59
	Std	0.032	0.5506	2.6807e12	13.632	78.9823	93.9338

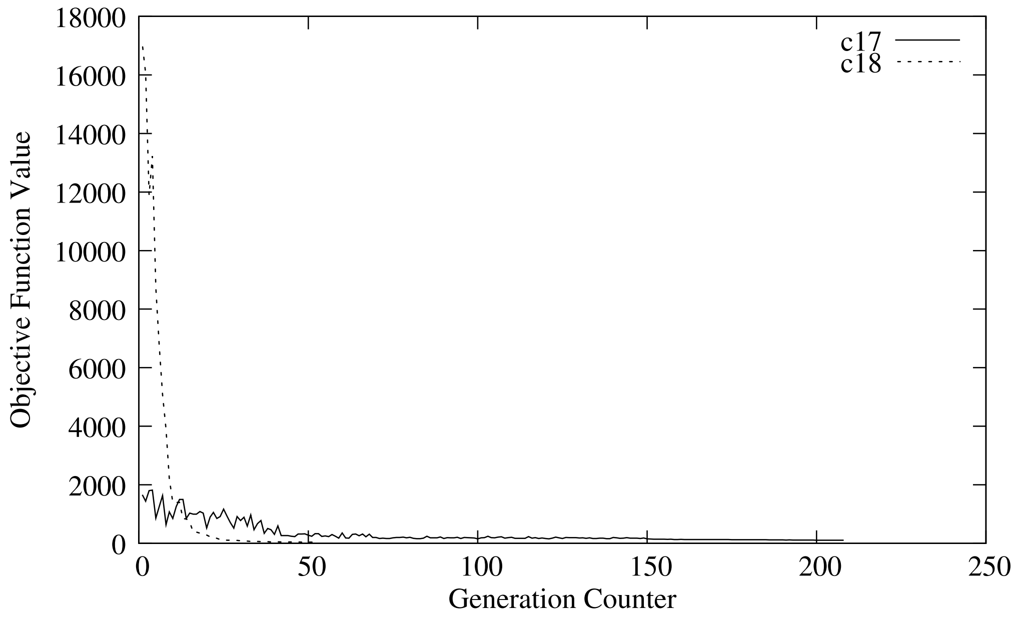

Figure 4: Convergence graph for 30D problems - C17,C18

Table 5: Function Values achieved when FES = 6 $\times 10^4$, FES = 3 $\times 10^5$, FES = 6 $\times 10^5$ for 30D Problems C07-C12.

FES		C07	C08	C09	C10	C11	C12
	Best	101.8074	1.8371e10	6.0398e8	7.9453e8	-1.3358	-491.4992
	Median	184.6778	3.2413e10	2.8186e8	8.1816e9	0.0205	-300.7299
$\mathbf{6 \times 10^4}$	Worst	824.4174	1.41e11	4.7713e9	9.6874e10	0.8710	0.3937
	c	0,0,0	0,0,0	0,0,0	0,0,0	1,0,0	1,0,0
	v	0	0	0	0	40.9431	2.3756
	Mean	255.06	2.878e10	2.7182e8	1.0399e9	-0.1566	-125.2102
	Std	173.89	4.2311e7	3.1212e4	5.8391e4	0.6505	153.70
	Best	89.781	1.8371e10	6.0398e8	7.9453e8	-1.4112	-491.4992
	Median	134.1221	3.2413e10	2.8186e8	8.1816e9	0.0031	-300.7299
$\mathbf{3 \times 10^5}$	Worst	521.4291	1.41e11	4.7713e9	9.7612e10	0.7132	0.3937
	c	0,0,0	0,0,0	0,0,0	0,0,0	1,0,0	1,0,0
	v	0	0	0	0	15.1213	2.3756
	Mean	178.1231	2.878e10	2.7182e8	1.03213e9	-0.2133	-125.2102
	Std	56.1221	4.2311e7	3.1212e4	5.912e4	0.6712	153.70
	Best	0.7935	1.8371e10	6.0398e8	6.9891e8	-1.4384	-491.4994
	Median	8.8104	3.2413e10	2.8186e8	7.12384e9	0.0002	-300.7299
$\mathbf{6 \times 10^5}$	Worst	77.5716	1.41e11	4.7713e9	8.9898e9	0.605	0.3937
	c	0,0,0	0,0,0	0,0,0	0,0,0	1,0,0	1,0,0
	v	0	0	0	0	26.295	2.3756
	Mean	36.470	2.878e10	2.7182e8	6.9898e8	-0.1681	-125.2102
	Std	22.701	4.2311e7	3.1212e4	6.1281e3	0.5718	153.70

Table 6: Function Values achieved when FES = 6 $\times 10^4$, FES = 3 $\times 10^5$, FES = 6 $\times 10^5$ for 30D Problems C13-C18.

FES		C13	C14	C15	C16	C17	C18
	Best	-66.3770	7.1831e8	8.6112e8	0.9380	61.7829	3948.1572
	Median	-63.5867	8.1312e9	9.8319e9	1.0522	98.5362	9840.5909
$\mathbf{6 \times 10^4}$	Worst	-61.1623	9.9892e9	9.9831e10	1.1114	127.7927	10103.1591
	c	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0
	v	0	0	0	0	0	0
	Mean	-63.481	8.7813e9	7.9893e9	1.0477	112.8771	8696.7
	Std	1.4800	2.1298e4	5.1297e4	0.0399	25.781	2209.3
	Best	-66.3770	7.1831e8	8.6132e8	0.9380	45.9898	2512.1431
	Median	-63.5867	8.1312e9	9.1233e9	1.0522	78.1231	3531.9782
$\mathbf{3 \times 10^5}$	Worst	-62.16872	9.9892e9	9.9831e10	1.1114	87.123	4122.9891
	c	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0
	v	0	0	0	0	0	0
	Mean	-63.2873	8.7813e9	7.9893e9	1.0477	67.2121	2987.1271
	Std	1.221	2.1298e4	5.1297e4	0.0399	24.1233	1931.1231
	Best	-66.3770	5.1831e6	8.6132e8	0.9380	29.8627	15.8264
	Median	-63.58674	8.1312e9	9.1233e9	1.0522	55.4727	49.6473
$\mathbf{6 \times 10^5}$	Worst	-62.1768	9.9892e9	9.9831e10	1.1114	83.3506	59.0756
	c	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0
	v	0	0	0	0		0
	Mean	-63.4121	8.7813e7	7.9893e9	1.0477	54.8781	44.038
	Std	1.2314	3.1298e3	5.1297e4	0.0399	19.7812	16.089

Table 7: Computational Complexity (Time in seconds)

$Dimension$	$T1$	$T2$	$(T2 - T1)/T1$
10	0.037	0.0678	0.8324
30	0.082	0.561	5.8414

Table 8: Feasibility Rate for Problems C01-C18.

Problem	$10D$	$30D$
C01	1	1
C02	1	1
C03	0	0
C04	1	0.80
C05	0.80	1
C06	0.80	0.60
C07	1	1
C08	1	1
C09	1	1
C10	1	1
C11	0.80	0.60
C12	0.80	0.80
C13	1	1
C14	1	1
C15	1	1
C16	1	1
C17	1	1
C18	1	1