Evaluation of Novel Adaptive Evolutionary Programming on Four Constraint Handling Techniques

R. Mallipeddi and P. N. Suganthan, *Senior Member, IEEE*

***Abstract*— This paper presents empirical studies carried out to evaluate the performance of different constraint handling methods on Constrained Real-Parameter Optimization using a novel adaptive Evolutionary Programming (EP). 25 runs have been conducted for each of the 13 test problems considered. Our experimental results show that no single Constraint Handling method can be the best for *all* problems i.e, each Constraint Handling method is suitable only for a subset of problems. We also show that the novel adaptive EP proposed in this paper has improved performance over the Classical EP (CEP).**

I. Introduction

Most real-world optimization problems involve constraints, the presence of which reduces the feasible region and complicates the optimization process. EP can be considered as a population-based variant of generate-and-test algorithms [1]. Although different versions of evolutionary programming (EP) (classical EP (CEP) proposed by Fogel [2], fast EP (FEP) proposed by X.Yao [3]) exist, we consider the CEP in our study. The original CEP algorithm is modified by developing a novel self adaptation procedure for the adaptation of the parameter η.

In the Classical EP [3] the η parameter is varied randomly independent of the problem and independent of EPs performance so far on the problem. Since constants τ and τ' values are used, the η value can become too large or too small leading to stagnation. Thus a lower bound is used for the η value and the optimal setting of lower bound is problem dependant. Thus a self adaptation of the lower bound of η is proposed in [4]. In the self-adaptation procedure used in this paper we adapt the η parameter based on the η values of the successful offspring that replace the parents in the next generation.

When dealing with constrained optimization problems, individuals that satisfy all the constraints are called feasible individuals while individuals that do not satisfy at least one of the constraints are called infeasible individuals. One of the major issues of constraint optimization is how to deal with the infeasible individuals throughout the search process. One way to handle infeasible individuals is to completely disregard them and continue the search process with feasible individuals only. But this has a drawback because GAs are probabilistic search methods and some of the information contained in the infeasible individuals could be unutilised. If the feasible search space is multimodal and discontinuous, then the GA can also be trapped in local minima. Therefore, different techniques have been developed to exploit the information in infeasible individuals. Michalewicz and Schoenauer [5] grouped the methods for handling constraints by evolutionary algorithms into four categories: i) preserving feasibility of solutions, ii) penalty functions, iii) make a separation between feasible and infeasible solutions, and iv) other hybrid methods. The constraint handling methods considered in this paper are:

- Superiority of Feasible solutions (SoF) [6]
- Self Adaptive Penalty (SAP) [7]
- ϵ constraint (EC)[8]
- Stochastic Ranking (SR)[9]

The remainder of this paper is organized as follows. Section II presents the modified EP algorithm. Section III reviews the related works of handling constrained optimization problems using GAs. Section IV presents the test problems used for the study. Section V discusses the results obtained for the benchmark functions. Finally, Section VI presents some concluding remarks and relevant observations.

II. The Algorithm

Different versions of EP use different mutation strategies (Gaussian, Cauchy and Levy's). In [10], it was proposed that Cauchy mutation is very good at searching in a large neighborhood while Gaussian mutation is better at searching in a small local neighborhood. In this paper we include both Gaussian and Cauchy mutations to facilitate the local and global search simultaneously throughout the entire search process. The number of population members that produce offspring through a particular mutation strategy depends on the success of the offsprings produced by that mutation strategy in the previous generations.

In CEP the η value is initialized with a constant (η=3) and the η is updated based on the τ and τ' values. In our proposed initialization of η values is done based on the search range of the problem and the way we update the η values is based on the EP's success in the previous generations. The values are updated based on the η values of the success population known as $sneta$. The $sneta$ is measured over a learning period of 5 recent past generations.

A global minimization problem can be formulated as a pair (S, f),where $S \subseteq R^n$ is a bounded set on R^n and $f :\mapsto R$ is an n-dimensional real-valued function. The problem is to find a point $\mathbf{x_{min}} \in \mathbf{S}$ such that $f(\mathbf{x_{min}})$ is a global minimum on S. More specially,it is required to find an $\mathbf{x_{min}} \in \mathbf{S}$ such

Manuscript received December 14, 2007. This work was supported by the A*Star (Agency for Science, Technology and Research) under the grant # 052 101 0020.

The authors are with Nanyang Technological University, School of Electrical and Electronic Engineering, Singapore, 639798 (phone: 65-67905404; fax: 65-67933318; e-mails: Mallipeddi@ntu.edu.sg , epnsugan@ntu.edu.sg).

978-1-4244-1823-7/08/$25.00©2008 IEEE

that

$$\forall x \in S : f(\mathbf{x_{min}}) \leq \mathbf{f}(\mathbf{x})$$

Here f does not need to be continuous but it must be bounded.

To compare CEP with our proposed algorithm, we modify CEP by using both mutation strategies. The mixed Gaussian and Cauchy EP is shown below for comparision:

1) Generate the initial population of μ individuals and set $k = 1$. Each individual is taken as a pair of real-valued vectors, $(x_i, \eta_{G,i}, \eta_{C,i}), \forall i \in \{1, ..., \mu\}$. where x_i's are objective variables and $\eta_{G,i}$'s are standard deviations for Gaussian mutations (also known as strategy parameters in self-adaptive evolutionary algorithms) and $\eta_{C,i}$'s are standard deviation for Cauchy mutations.

$$\eta_{G,i}(j) = 3$$

$$\eta_{C,i}(j) = 3$$

Let the learning period be lp and the probability p_g be 0.5. p_g is the ratio of offsprings produced by Gaussian mutation to that of that total offsprings.

2) Evaluate the fitness score for each individual $(x_i, \eta_{G,i}, \eta_{C,i}), \forall i \in \{1, ..., \mu\}$, of the population based on the objective function, $f(x_i)$ and $G_i(x)$

3) When $k > lp$ update p_g.

4) Each parent $(x_i, \eta_{G,i}, \eta_{C,i}), i = 1, ..., \eta$, creates a single offspring $(x_i', \eta_{G,i}', \eta_{C,i}')$ by:

$IF(rand \leq p_g)$

$$\eta_{G,i}'(j) = \eta_{G,i}(j) * exp(\tau' N(0,1) + \tau N_j(0,1))$$

$$\eta_{C,i}'(j) = \eta_{C,i}(j)$$

$$x_i'(j) = x_i(j) + \eta_{G,i}'(j) * Gaussian(0,1),$$

$ELSE$

$$\eta_{C,i}'(j) = \eta_{C,i}(j) * exp(\tau' N(0,1) + \tau N_j(0,1))$$

$$\eta_{G,i}'(j) = \eta_{G,i}(j)$$

$$x_i'(j) = x_i(j) + \eta_{C,i}'(j) * Cauchy(0,1),$$

END

where $x_i(j), x_i'(j), \eta_i(j)$ and $\eta_i'(j)$ denote the j th component of the vectors x_i, x_i', η_i and η_i',respectively. $Gaussian(0,1)$ and $Cauchy(0,1)$ denotes a normally distributed one-dimensional Gaussian and Cauchy random numbers with zero mean and standard deviation one. The factors τ and τ' have commonly set to $(\sqrt{2\sqrt{(n)}})^{-1}$ and $(\sqrt{2n})^{-1}$.

5) Calculate the fitness of each offspring $(x_i', \eta_{i,G}', \eta_{i,C}'), \forall i \in \{1, ..., \mu\}$.

6) Conduct pairwise comparison over the union of parents $(x_i, \eta_{G,i}, \eta_{C,i})$ and offsprings $(x_i', \eta_{i,G}', \eta_{i,C}'), \forall i \in \{1, ..., \mu\}$. For each individual, opponents are chosen uniformly at random from all the parents and offsprings. For each comparison, if the individual's fitness is smaller than the opponent's, it receives a "win".

7) Stop if the stopping criterion is satisfied; otherwise, $k = k + 1$ and go to Step 3.

The Adaptive mixed Gaussian and Cauchy EP proposed in this paper is as follows:

1) Generate the initial population of μ individuals and set $k = 1$. Each individual is taken as a pair of real-valued vectors, $(x_i, \eta_{G,i}, \eta_{C,i}), \forall i \in \{1, ..., \mu\}$. where x_i's are objective variables and $\eta_{G,i}$'s are standard deviations for Gaussian mutations (also known as strategy parameters in self-adaptive evolutionary algorithms) and $\eta_{C,i}$'s are standard deviation for Cauchy mutations. Instead of giving a constant value for all the dimensions $(j = 1, ..., n)$,η is initialized scaled to the dimension as :

$$\eta_{G,i}(j) = [0.3 + 0.2 * rand] * (X_{max}(j) - X_{min}(j))$$

$$\eta_{C,i}(j) = [0.3 + 0.2 * rand] * (X_{max}(j) - X_{min}(j))$$

Let the learning period be lp and the probability p_g be 0.5. p_g is the ratio of offsprings produced by Gaussian mutation to that of that total offsprings.

2) Evaluate the fitness score for each individual $(x_i, \eta_{G,i}, \eta_{C,i}), \forall i \in \{1, ..., \mu\}$, of the population based on the objective function, $f(x_i)$ and $G_i(x)$

3) When $k > lp$ update $sneta_G$, $sneta_C$ and p_g.

4) Each parent $(x_i, \eta_{G,i}, \eta_{C,i}), i = 1, ..., \eta$, creates a single offspring $(x_i', \eta_{G,i}', \eta_{C,i}')$ by:

$IF(rand \leq p_g)$

$$\eta_{G,i}'(j) = 0.3 * sneta_G(j) * [2 * rand - 1] + sneta_G(j)$$

$$\eta_{C,i}'(j) = sneta_{C,i}(j)$$

$$x_i'(j) = x_i(j) + \eta_{G,i}'(j) * Gaussian(0,1),$$

$ELSE$

$$\eta_{C,i}'(j) = 0.3 * sneta_C(j) * [2 * rand - 1] + sneta_C(j)$$

$$\eta_{G,i}'(j) = sneta_{G,i}(j)$$

$$x_i'(j) = x_i(j) + \eta_{C,i}'(j) * Cauchy(0,1),$$

END

where $x_i(j), x_i'(j), \eta_i(j)$ and $\eta_i'(j)$ denote the j th component of the vectors x_i, x_i', η_i and η_i',respectively. $Gaussian(0,1)$ and $Cauchy(0,1)$ denotes a normally distributed one-dimensional Gaussian and Cauchy random numbers with zero mean and standard deviation one.

5) Calculate the fitness of each offspring $(x_i', \eta_{i,G}', \eta_{i,C}'), \forall i \in \{1, ..., \mu\}$.

6) Conduct pairwise comparison over the union of parents $(x_i, \eta_{G,i}, \eta_{C,i})$ and offsprings $(x_i', \eta_{i,G}', \eta_{i,C}'), \forall i \in \{1, ..., \mu\}$. For each individual, opponents are chosen uniformly at random from all the parents and offsprings. For each comparison, if the individual's fitness is smaller than the opponent's, it receives a "win".

7) Stop if the stopping criterion is satisfied; otherwise, $k = k + 1$ and go to Step 3.

III. CONSTRAINT HANDLING

In real world applications, most optimization problems have complex constraints. A constrained optimization problem is usually written as a nonlinear programming problem of the following form [6]:

$$\text{Minimize:} f(\mathbf{x}), \mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}) \text{ and } \mathbf{X} \in \mathbf{S}$$
$$\text{Subject to:}$$
$$g_i(\mathbf{x}) \le \mathbf{0}, \mathbf{i} = \mathbf{1}, ..., \mathbf{q}$$
$$h_j(\mathbf{x}) = \mathbf{0}, \mathbf{j} = \mathbf{q} + \mathbf{1}, ..., \mathbf{m}$$

S is the whole search space. q is the number of inequality constraints. The number of equality constraints is $m - q$. The inequality constraints that satisfy $g_i(X) = 0$ at the global optimum solution are called active constraints. All equality constraints are active constraints. For convenience, the equality constraints are always transformed into the inequality form, and then we can combine all the constraints as

$$G_i(\mathbf{x}) = \begin{cases} max\{g_i(\mathbf{x}), \mathbf{0}\} \ \mathbf{i} = \mathbf{1}, ...\mathbf{q} \\ \\ \mid h_i(\mathbf{x}) - \delta \mid \ \ \mathbf{i} = \mathbf{q} + \mathbf{1}, ..., \mathbf{m} \end{cases}$$

Therefore, the objective of our algorithm is to minimize the fitness function $f(\mathbf{x})$, at the same time the optimum solutions obtained must satisfy all the inequality constraints $G_i(\mathbf{x})$. The value of $\delta = 1e^{-4}$ is considered.

A. *Superiority of Feasible Solutions*

The superiority of feasible solutions proposed by Deb [11] is considered. Deb's selection criterion has no parameter to fine tune. During the selection procedure, the offspring is compared to that of its corresponding parent in the current population considering both the fitness value and constraints. The offspring will replace the parent and enter the population of the next generation if any of the following condition is true.

1) The offspring is feasible and the parent is not.
2) The offspring and the parent are both feasible and the offspring has smaller or equal fitness value (for minimization problem) than the corresponding parent.
3) The offspring and the parent are both infeasible, but offspring has a smaller overall constraint violation.

In methods based on preference of feasible solutions over infeasible ones, feasible ones are always considered better than infeasible ones. Infeasible solutions will be compared based on their constraint violation, while feasible solutions will be compared based on their objective function value only. Hence feasible solutions dominate the infeasible ones. The main drawback of the algorithm is that once there are many feasible individuals in the population, the infeasible individuals will be less used in the search process, and the algorithm will not be able to explore the whole search space. This may lead the algorithm to get struck in a local optimum.

B. *Self adaptive Penalty*

The simplest and the earliest method of involving infeasible individuals in the search process is by using penalty methods. Penalty functions are by far the simplest and the most commonly used methods for handling constraints using GAs. In death penalty function methods, individuals that violate any one of the constraints are completely rejected. No information is extracted from those infeasible individuals. On the other hand, some penalty function approaches convert a constrained optimization problem into an unconstraint one by adding penalty values to individuals violating the constraints. The way the penalties are added determines the type of penalty function. If the penalties added do not depend on the current generation number and remain constant during the entire evolutionary process, then the penalty function is called static penalty function. In static penalty function methods the penalties are the weighted sum of the constraint violations. If, alternatively, the current generation number is involved in determining the penalties then the method is called dynamic penalty function method.

Although penalty functions are very simple and easy to implement they often require several parameters to be chosen heuristically by users. These parameters are problem-dependant and need prior knowledge if the degree of constraint violation present in a problem. Therefore, tuning the parameters leads to unnecessary computation for simple problems. Although dynamic penalty functions work better than static penalty functions, they require even more parameters to be tuned. In general, the problems associated with static penalty functions are also present with dynamic penalties: if a bad penalty is chosen, the EA may converge to either non-optimal feasible solutions (if the penalty is too high) or to infeasible solutions (if the penalty is too low) [12].

In this paper, the adaptive penalty strategy proposed in [7] is considered to solve constrained optimization problems. Two types of penalties are added to each infeasible individual to identify the best infeasible individuals in the current population. The amount of each of the two penalties added is controlled by the number of feasible individuals currently present in the population. If there are few feasible individuals available, a higher amount of penalty is added to infeasible individuals with a higher amount of constraint violation. On the other hand, if there are sufficient number of feasible individuals present, then infeasible individuals small objective function values will have small penalties added to their fitness value. These two penalties will allow the algorithm to switch between finding more feasible solutions and searching for the optimum solution at anytime during the search process. This algorithm needs no parameter tuning.

C. *ϵ constraint*

The difficulties that occur while solving constrained optimization problems using evolutionary algorithms are:

1) The ability to solve multi-modal problems that have many local solutions is insufficient. Sometimes they

get trapped to a local solution and cannot search for an optimal solution.

2) Obtained solutions in problems with equality constraints are inadequate. Generally constrained optimization solve problems with equality constraints by converting equality constraints into relaxed inequality constraints. As a result, the feasibility of the obtained solutions is inadequate. Also, it is difficult to solve problems with many equality constraints.
3) The stability and efficiency of searches is low. Sometimes the algorithm cannot overcome the effect of randomness in the search process of some problems. Thus the stability of the search becomes low and incurs high computational costs.

In our work, the algorithm proposed in [8] is used to overcome the above problems. In [8], a simple way of controlling the ϵ is defined as follows. The ϵ level is updated until the number of generation k becomes the control generation T_c. After the number of generations exceeds T_c, the level is set to 0 to obtain solutions with no constraint violation.

$$\epsilon(0) = \nu(x_\theta)$$
$$\epsilon(k) = \begin{cases} \epsilon(0)(1-\frac{k}{T_c})^{cp}, 0 < k < T_c, \\ 0, \qquad k \geq T_c \end{cases}$$

where x_θ is the top $\theta - th$ individual and $\theta = 0.2N$. $cp \in [2, 10]$. In this paper $cp = 5$ is considered.

D. Stochastic Ranking

In [9], Runarsson and Yao introduced a stochastic ranking method to achieve a balance between objective and penalty functions stochastically. A probability factor P_f is used to determine whether the objective function value or the constraint violation value determines the rank of each individual. In this paper, the value of $P_f = 0.475$ is selected.

The overall constraint violation is a weighted mean value of all the constraints,which is expressed as following,

$$\nu(x) = \frac{\sum_{i=1}^{m} w_i(G_i(\mathbf{x}))}{\sum_{i=1}^{m} w_i}$$

where $w_i = \frac{1}{G_{max_i}}$ is a weighted parameter, G_{max_i} is the maximum violation of the constraint $G_i(\mathbf{x})$ obtained so far. Here, we set w_i as $\frac{1}{G_{max_i}}$ which varies during the evolution in order to accurately normalize the constraints of the problem. Thus the overall constraint violation can represent all constraints more equally

IV. Test Functions

The first 13 test functions of CEC 2006 Special Session on Constrained Real-Parameter Optimization are considered [13]. The details of the test functions are provided in Table I.

Table I:Details of Test Functions

Prob	n	Type of function	ρ	LI	NI	LE	NE	a
$g01$	13	quadratic	0.0111%	9	0	0	0	6
$g02$	20	nonlinear	99.9971%	0	2	0	0	1
$g03$	10	polynomial	0.0000%	0	0	0	1	1
$g04$	5	quadratic	52.1230%	0	6	0	0	2
$g05$	4	cubic	0.0000%	2	0	0	3	3
$g06$	2	cubic	0.0066%	0	2	0	0	2
$g07$	10	quadratic	0.0003%	3	5	0	0	6
$g08$	2	nonlinear	0.8560%	0	2	0	0	0
$g09$	7	polynomial	0.5121%	0	4	0	0	2
$g10$	8	linear	0.0010%	3	3	0	0	6
$g11$	2	quadratic	0.0000%	0	0	0	01	1
$g12$	3	quadratic	4.7713%	0	1	0	0	0
$g13$	5	nonlinear	0.0000%	0	0	0	3	3

g01

Minimize :
$$f(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$
Subject to:
$$g_1(\vec{x}) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0$$
$$g_2(\vec{x}) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0$$
$$g_3(\vec{x}) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0$$
$$g_4(\vec{x}) = -8x_1 + x_{10} \leq 0$$
$$g_5(\vec{x}) = -8x_2 + x_{11} \leq 0$$
$$g_6(\vec{x}) = -8x_3 + x_{12} \leq 0$$
$$g_7(\vec{x}) = -2x_4 - x_5 + x_{10} \leq 0$$
$$g_8(\vec{x}) = -2x_6 - x_7 + x_{11} \leq 0$$
$$g_9(\vec{x}) = -2x_8 - x_9 + x_{12} \leq 0$$

where the bounds are $0 \leq x_1 \leq 1 (i = 1, ..., 9), 0 \leq x_i \leq 100 (i = 10, 11, 12)$ and $0 \leq x_{13} \leq 1$. The global minimum is at $\vec{x}^* = (1,1,1,1,1,1,1,1,1,3,3,3,1)$ where six constraints are active $(g_1, g_2, g_3, g_7, g_8$ and $g_9)$ and $f(\vec{x}^*) = -15$.

g02

Minimize :
$$f(\vec{x}) = -\left|\frac{\sum_{i=1}^{n}\cos^4(x_i) - 2\prod_{i=1}^{n}\cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}}\right|$$
Subject to:
$$g_1(\vec{x}) = 0.75 - \prod_{i=1}^{n} x_i \leq 0$$
$$g_2(\vec{x}) = \sum_{i=1}^{n} x_i - 7.5n \leq 0$$

where $n = 20$ and $0 \leq x_i \leq 10 (i = 1, ..., n$. The optimum is located at $\vec{x}^* =$ (3.16246061572185, 3.12833142812967, 3.09479212988791, 3.06145059523469, 3.02792915885555, 2.99382606701730, 2.95866871765285, 2.92184227312450, 0.49482511456933, 1.4883571105490, 0.48231642711865, 0.47664475092742, 0.44826762241853, 0.44424700958760, 0.44038285956317) where $f(\vec{x}^*) = -0.80361910412559$.

g03

Minimize :
$$f(\vec{x}) = -(\sqrt{n})^n \prod_{i=1}^{n} x_i$$
Subject to:
$$h_1(\vec{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

where $n = 10$ and $0 \leq x_i \leq 1 (i = 1, ..., n)$. The optimum solution is $\vec{x}^* =$ (0.31624357647283069, 0.316243577414338339, 0.316243578012345927, 0.316243575664017895, 0.3162243578205526066, 0.31624357738855069, 0.316243575472949512, 0.316243577164883938, 0.316243578155920302, 0.316243576147374916) where $f(\vec{x}^*) = -1.00050010001000$.

g04

$$\text{Minimize :}$$
$$f(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 0.40792.141$$
Subject to:
$$g_1(\vec{x}) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0$$
$$g_2(\vec{x}) = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0$$
$$g_3(\vec{x}) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0$$
$$g_4(\vec{x}) = -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0$$
$$g_5(\vec{x}) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0$$
$$g_6(\vec{x}) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0$$

where $78 \leq x_1 102$, $33 \leq x_2 \leq 45$, and $27 \leq x_4 \leq 45 (i = 3, 4, 5)$. The optimum solution is $\vec{x}^* = (78, 33, 29.9952560256815985, 45, 36.7758129057882073)$ where $f(\vec{x}^*) = -3.066553867178332e + 004$. Two constraints are active (g_1 and g_6).

g05

$$\text{Minimize :}$$
$$f(\vec{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$$
Subject to:
$$g_1(\vec{x}) = -x_4 + x_3 - 0.55 \leq 0$$
$$g_2(\vec{x}) = -x_3 + x_4 - 0.55 \leq 0$$
$$h_3(\vec{x}) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$
$$h_4(\vec{x}) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$
$$h_5(\vec{x}) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0$$

where $0 \leq x_1 1200$, $0 \leq x_2 \leq 1200$, $-0.55 \leq x_3 \leq 0.55$ and $-0.55 \leq x_4 \leq 0.55$. The optimum solution is $\vec{x}^* =$ (679.945148297028709, 1026.06697600004691, 0.118876369094410433,−0.39623348521517826) where $f(\vec{x}^*) = 5126.4967140071$.

g06

$$\text{Minimize :}$$
$$f(\vec{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$
Subject to:
$$g_1(\vec{x}) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$$
$$g_2(\vec{x}) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0$$

where $13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$. The optimum solution is $\vec{x}^* = (14.09500000000000064, 0.8429607892154795668)$ where $f(\vec{x}^*) = -6961.81387558015$.Both constraints are active.

g07

$$\text{Minimize :}$$
$$f(\vec{x}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_10 - 7)^2 + 45.$$
Subject to:
$$g_1(\vec{x}) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0$$
$$g_2(\vec{x}) = 10x_1 - 8x_2 - 17x_7 = 2x_8 \leq 0$$
$$g_3(\vec{x}) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0$$
$$g_4(\vec{x}) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0$$
$$g_5(\vec{x}) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0$$
$$g_6(\vec{x}) = x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0$$
$$g_7(\vec{x}) = 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0$$
$$g_8(\vec{x}) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0$$

where $-10 \leq x_i \leq 10 (i = 1, ..., 10)$.The optimum solution is $\vec{x}^* =$ (2.17199634142692, 2.3636830416034,8.77392573913157, 5.09598443745173, 0.990654756560493, 1.43057392853463, 1.32164415364306, 9.82872576524495, 8.2800915887356, 8.3759266477347) where $f(\vec{x}^*) =$ 24.30620906818(The recorded results may suffer from rounding errors which may cause slight infeasibility sometimes in the best given solutions). Six constraints are active(g_1, g_2, g_3, g_4, g_5 and g_6).

g08

$$\text{Minimize :}$$
$$f(\vec{x}) = -\frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$
Subject to:
$$g_1(\vec{x}) = x_1^2 - x_2 + 1 \leq 0$$
$$g_2(\vec{x}) = 1 - x_1 + (x_2 - 4)^2 \leq 0$$

where $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. The optimum is located at $\vec{x}^* = (1.22797135260752599, 4.24537336612274885)$ where $f(\vec{x}^*) = -0.0958250414180359$.

g09

$$\text{Minimize :}$$
$$f(\vec{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$
Subject to:
$$g_1(\vec{x}) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0$$
$$g_2(\vec{x}) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0$$
$$g_3(\vec{x}) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0$$
$$g_4(\vec{x}) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x + 6 - 11x_7 \leq 0$$

where $-10 \leq x_i \leq 10$ for $(i = 1, ..., 7)$. The optimum solution is $\vec{x}^* =$ (2.33049935147405174, 1.95137236847114592, −0.477541399510615805, 4.36572624923625874, −0.624486959100388983, 1.03813099410962173, 1.5942266780671519) where $f(\vec{x}^*) =$ 680.630057374402. Two constraints are active (g_1 and g_4).

g10

Minimize :
$f(\vec{x}) = x_1 + x_2 + x_3$
Subject to:
$g_1(\vec{x}) = -1 + 0.0025(x_4 + x_6) \leq 0$
$g_2(\vec{x}) = -1 + 0.0025(x_5 + x_7 - x_4) \leq 0$
$g_3(\vec{x}) = -1 + 0.01(x_8 - x_5) \leq 0$
$g_4(\vec{x}) = -x_1x_6 + 833.33252x_4 + 100x_1 - 83333.333 \leq 0$
$g_5(\vec{x}) = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 \leq 0$
$g_6(\vec{x}) = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 \leq 0$

where $100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000 (i = 2, 3)$. The optimum solution is $\vec{x}^* = (579.306685017979589, 1359.97067807935605, 5109.97065743133317, 182.01769963061534, 295.601173702746792, 217.982300369384632, 286.41652592786852, 395.601173702746735)$, where $f(\vec{x}^*) = 7049.24802052867$. All constraints are active (g_1, g_2 and g_4).

g11

Minimize :
$f(\vec{x}) = x_1^2 + (x_2 - 1)^2$
Subject to:
$h(\vec{x}) = x_2 - x_1^2 = 0$

where $-1 \leq x_1 \leq 1$ and $-1 \leq x_2 \leq 1$. The optimum solution is $\vec{x}^* = (-0.707036070037170616, 0.500000004333606807)$ where $f(\vec{x}^*) = 0.7499$.

g12

Minimize :
$f(\vec{x}) = -(100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2)/100$
Subject to:
$g(\vec{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0$

where $0 \leq x_i \leq 10 (i = 1, 2, 3)$ and $p, q, r = 1, 2, ..., 9$.The feasible region of the search space consists of 9^3 disjoint spheres. A point (x_1, x_2, x_3) is feasible if and only if there exist p, q, r such that the above inequality holds. The optimum is located at $\vec{x}^* = (5, 5, 5)$ where $f(\vec{x}^*) = -1$. The solution lies within the feasible region.

g13

Minimize :
$f(\vec{x}) = e^{x_1x_2x_3x_4x_5}$
Subject to:
$h_1(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0$
$h_2(\vec{x}) = x_2x_3 - 5x_4x_5 = 0$
$h_3(\vec{x}) = x_1^3 + x_2^3 + 1 = 0$

where $-2.3 \leq x_i \leq 2.3 (i = 1, 2)$ and $-3.2 \leq x_i \leq 3.2 (i = 3, 4, 5)$. The optimum solution is $\vec{x}^* = (-1.71714224003, 1.59572124049468, 1.8272502406271, -0.763659881912867, -0.76365986736498)$ where $f(\vec{x}^*) = 0.053941514041898$. The solution lies within the feasible region.

V. Experimental Results

13 benchmark functions for CEC 2006 Special Session on Constrained Real-Parameter Optimization are taken for the study. The details of the functions used for the study are provided in Table I. n is the number of decision variables. $\rho = | F | / | S |$ is the estimated ratio between the feasible region and the search space, LI is the number of linear inequality constraints, NI is the number of nonlinear inequality constraints, LE is the number of linear equality constraints and NE is the number of nonlinear equality constraints. a is the number of active constraints at $\mathbf{x}$. A run during which at least one feasible solution is found in Max_FEs is considered as feasible run. A run during which the algorithm finds a feasible solution satisfying $f(\vec{\mathbf{x}}) - f(\vec{\mathbf{x}}^*) \leq 0.01$ is considered as a successful run.

The four constraint handling methods as discussed in the previous sections are applied to each of the test problem. Every problem is run 25 times and the best, worst, mean, median, standard deviation, the number of infeasible runs (I.R) and the number of success runs (S.R) in each case are presented in Table II and Table III. The success performance (SP) is calculated as:

$$SP = \frac{mean(FEsforS.R) * (TotalRuns}{No.ofS.R}$$

The SP is then normalized by the SP of the best algorithm (SPbest) to get the normalized success performance. Therefore small values of FEs and therefore large values in the empirical distribution graphs are preferable.

From Table II and Table III we observe that no single constraint handling method can solve all the constraint handling problems with consistency. In Table II and III the best results are represented in bold for comparison. We can observe that $g01$, $g03$,$g04$,$g08$ and $g10$ are better solved by Superiority of Feasible than by the other methods. Self Adaptive Penalty could solve $g02$, $g07$, $g09$, $g11$ and $g13$ better while ϵ Constraint could solve $g05$ and $g08$ better. By comparing Table II with Table III we can see the improvement in the performance of the EP algorithm due to adaptation of the parameter η based on the experience of previous generation. We can also observe that when the algorithm is changed the constraint handling method that performs better on a problem also changes, for example in problems g01 and g02.

In classical EP algorithm [3] the η value is adapted randomly based only on the dimension of the problem using the two constants ($\tau = (\sqrt(2\sqrt(n)))^{-1}$ and $\tau' = (\sqrt(2n))^{-1}$). This leads to the value of η being too small or too large. To overcome this the lower bound for the η value is used [4]. In self adaptive EP the updating of η value is done based on the η values of the success individuals that replace their parents in the next generation. The self adaptation used in our work prevents the value of η from becoming too large or too small. From the experimental observation it was found that the self adaptive EP performed better than the Classical EP. From Figs: 1-4, we can observe that the self-adaptation of η is overall superior to the previously used randomized adaptation scheme.

VI. Conclusions

In this paper we compared the performance various constraint handling methods on a test suite of 13 benchmark functions. From the results we observe that no single constraint handling method is consistent with its performance on

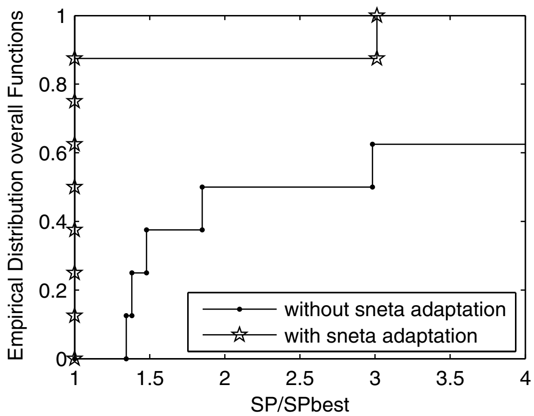

Fig. 1. Normalised Succcess Performance of SoF

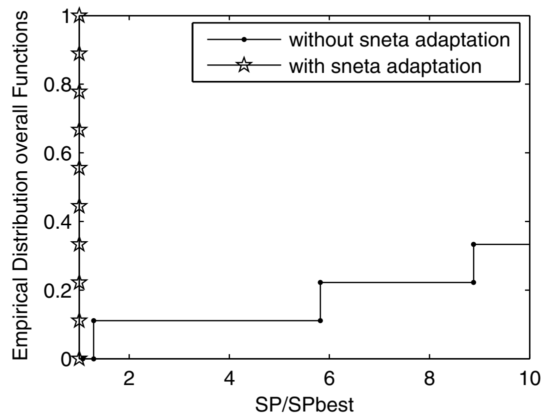

Fig. 2. Normalized Succcess Performance of SAP

all the functions. We also proposed a self adaptation of the parameter η.

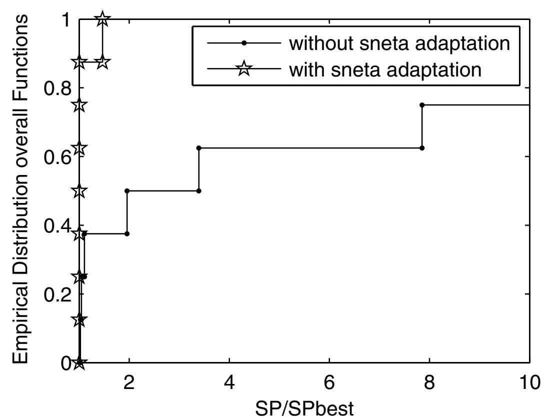

Fig. 3. Normalized Succcess Performance of EC

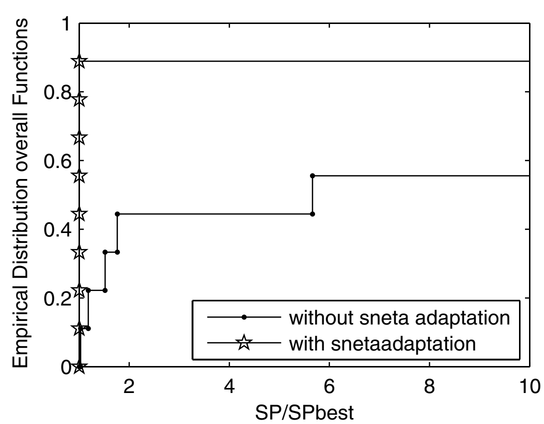

Fig. 4. Normalized Succcess Performance of SR

REFERENCES

[1] X Yao and Y. Liu, *Scaling Up Evolutionary Programming Algorithms*, Springer-Verlag, Berlin, 1998.

[2] D. B. Fogel, *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*, Ginn Press, Needham Heights, 1991.

[3] X. Yao and Y. Liu, "Fast evolutionary programming," in *Evolutionary Programming V: Proc of the Fifth Annual Conference on Evolutionary Programming, The MIT Press*, 1996, pp. 451–460.

[4] K. H. Liang, X. Yao, and C. S. Newton, "Adapting self-adaptive parameters in evolutionary algorithms," *Applied Intelligence*, vol. 15, no. 3, pp. 171–180, November 2001.

[5] M. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996.

[6] V. L. Huang, A. K. Qin, and P. N. Suganthan, "Self-adaptive differential evolution algorithm for constrained real-parameter optimization," in *IEEE Congress on Evolutionary Computation, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada*, 2006, pp. 17–24.

[7] B Tessema and G. G. Yen, "A self adaptive penalty function based algorithm for constrained optimization," in *IEEE Congress on Evolutionary Computation, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada*, 2006, pp. 246–253.

[8] T. Takahama and S. Sakai, "Constrained optimization by the ϵ constrained differential evolution with gradient-based mutation and feasible elites," in *IEEE Congress on Evolutionary Computation, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada*, 2006, pp. 1–8.

[9] T. P. Runarsson and X. Yao, "Stochastic ranking for constraint evolutionary optimization," *IEEE Transactions on Evolutionary Computation*, vol. 4, pp. 344–354, 2000.

[10] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 82–102, 1999.

[11] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2/4, pp. 311–338, 2000.

[12] C. A. C. Coello, "Theoretical and numerical constraint handling techniques used with evolutionary algorithms: a survey of the state of the art," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, pp. 1245–1287, 2002.

[13] J. J. Liang, T. P. Runarsson, Efrén Mezura-Montes, Maurice Clerc, P. N. Suganthan, Carlos A. Coello Coello, and K. Deb, "Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization," Tech. Rep., Technical Report.

Table II:Performance without sneta adaptation

Prob.		SoF	SAP	EC	SR
$g01$	Worst	-10.217	-9.49	-11.724	-10.77
	Best	-13.252	-12.007	**-14.996**	-14.44
	Median	-11.711	-11.847	**-12.769**	-11.761
	Mean	-11.727	-11.115	**-13.149**	-11.937
	Std	0.859	1.4093	1.0017	0.9117
	I.R	0	22	0	0
	S.R	0	0	0	**1**
$g02$	Worst	-0.47533	-0.536	-0.4267	-0.5106
	Best	-0.75548	**-0.79056**	-0.76846	-0.76069
	Median	-0.57992	-0.64521	-0.6126	**-0.64603**
	Mean	-0.59801	**-0.66098**	-0.61259	-0.64158
	Std	0.085083	0.082853	0.095958	0.079112
	I.R	0	0	0	0
	S.R	0	2	0	0
$g03$	Worst	0	0	0	0
	Best	0	0	**-0.006734**	0
	Median	0	0	0	0
	Mean	0	0	**-0.0003893**	0
	Std	0	0	0.001424	0
	I.R	0	0	1	0
	S.R	0	0	0	0
$g04$	Worst	-30661	-30664	-30666	-30666
	Best	-30666	-30666	**-30666**	-30666
	Median	-30666	-30666	**-30666**	-30666
	Mean	-30665	-30665	**-30666**	-30666
	Std	0.83762	0.2184	1.60E-06	0.000663
	I.R	0	0	0	0
	S.R	24	24	**25**	**25**
$g05$	Worst	6112.2	-	6112.2	6112.2
	Best	5331.1	-	5246	**5177.8**
	Median	6092.4	-	**5791**	6102.3
	Mean	5987.6	-	**5727.7**	5909
	Std	218.8	-	255.43	297.73
	I.R	11	25	8	9
	S.R	0	0	0	0
$g06$	Worst	-6961.8	-1563.1	-6961.8	-6961.8
	Best	-6961.8	-6468.2	**-6961.8**	**-6961.8**
	Median	-6961.8	-5392.5	**-6961.8**	**-6961.8**
	Mean	-6961.8	-4836.9	**-6961.8**	**-6961.8**
	Std	3.71E-12	1478.9	3.71E-12	3.71E-12
	I.R	0	7	0	0
	S.R	**25**	0	**25**	**25**
$g07$	Worst	27.341	24.842	26.971	26.385
	Best	**24.317**	24.333	24.322	24.318
	Median	24.432	**24.409**	24.622	24.462
	Mean	24.717	**24.462**	24.652	24.769
	Std	0.727	0.1409	0.5342	0.5075
	I.Runs	0	0	0	0
	S.Runs	13	**18**	11	13
$g08$	Worst	-0.095825	0.020004	-0.095825	-0.095825
	Best	**-0.095825**	-0.090369	**-0.095825**	**-0.095825**
	Median	**-0.095825**	-0.001386	**-0.095825**	**-0.095825**
	Mean	**-0.095825**	-0.010506	**-0.095825**	**-0.095825**
	Std	4.01E-18	0.0033258	4.01E-18	4.01E-18
	I.Runs	0	17	0	0
	S.Runs	**25**	0	**25**	**25**
$g09$	Worst	680.79	680.73	680.88	681.12
	Best	680.63	**680.63**	680.63	**680.63**
	Median	680.65	**680.64**	680.66	**680.64**
	Mean	680.67	**680.66**	680.66	**680.67**
	Std	0.0463	0.02858	0.04892	0.10013
	I.R	0	0	0	0
	S.R	10	11	**14**	8
$g10$	Worst	10115	12827	14835	11348
	Best	7134.5	7244.5	7233.8	**7095.3**
	Median	**7734.5**	9952	8658.1	8070.9
	Mean	**7966.2**	9993.9	9238.6	8505.5
	Std	769.44	3067.5	1909.8	1354.7
	I.R	0	21	0	1
	S.R	0	0	0	0
$g11$	Worst	0.75581	0.75046	0.77468	0.75883
	Best	0.74994	**0.7499**	0.74997	0.7499
	Median	0.75043	**0.7499**	0.75062	0.75119
	Mean	0.75139	**0.74994**	0.75274	0.75169
	Std	0.001861	0.000135	0.005801	0.002264
	I.R	0	0	0	0
	S.R	**25**	**25**	23	**25**
$g12$	Worst	-1	-1	-1	-1
	Best	-1	-1	-1	-1
	Median	-1	-1	-1	-1
	Mean	-1	-1	-1	-1
	Std	0	0	0	0
	I.R	0	0	0	0
	S.R	25	25	25	25
$g13$	Worst	2.1838	0.9997	1.1759	3.4994
	Best	0.4475	**0.40458**	0.50132	0.84834
	Median	0.9967	**0.90089**	0.99532	0.9984
	Mean	0.9795	**0.8218**	0.9413	1.074
	Std	0.3292	0.20031	0.14324	0.50739
	I.R	0	0	0	0
	S.R	0	0	0	0

Table III:Performance with sneta adaptation

Prob.		SoF	SAP	EC	SR
$g01$	Worst	-11.332	-10.649	-11.298	-11.121
	Best	**-15**	**-15**	**-15**	**-15**
	Median	-13.645	**-13.813**	-13.188	-13.809
	Mean	-13.582	-13.575	-13.329	**-13.712**
	Std	1.051	1.169	1.116	1.098
	I.R	0	9	0	0
	S.R	**3**	1	2	**3**
$g02$	Worst	-0.39986	-0.52133	-0.40775	-0.42319
	Best	-0.7926	**-0.80361**	-0.79255	-0.80360
	Median	-0.70322	**-0.75072**	-0.72981	-0.72228
	Mean	-0.68883	**-0.72887**	-0.69606	-0.70496
	Std	0.0922	0.0667	0.0963	0.0866
	I.R	0	0	0	0
	S.R	0	2	0	**3**
$g03$	Worst	-0.06988	-0.99738	-0.06336	-0.11869
	Best	**-1.0005**	-1.0004	-0.9106	-1.0005
	Median	-0.9147	**-0.9999**	-0.2990	-0.9342
	Mean	**-0.7662**	-0.9998	-3525	-0.7534
	Std	0.3007	0.0006	0.2539	0.3174
	I.R	0	0	0	0
	S.R	5	**25**	0	6
$g04$	Worst	**-30647**	-30606	-30602	-30634
	Best	**-30666**	-30666	-30666	-30666
	Median	**-30666**	-30666	-30666	-30666
	Mean	**-30664**	-30662	-30662	**-30664**
	Std	**4.6444**	12.998	13.100	6.2553
	I.R	0	0	0	0
	S.R	19	**22**	18	19
$g05$	Worst	6112.2	-	6102.5	6087.4
	Best	5127.5	-	**5126.5**	5126.7
	Median	5445.4	-	**5289.1**	5356.1
	Mean	5575.1	-	**5473.4**	5521
	Std	409.71	-	352.78	343.17
	I.R	0	25	0	0
	S.R	0	0	**1**	0
$g06$	Worst	-6961.8	-	-6961.8	-6961.8
	Best	-6961.8	-	-6961.8	-6961.8
	Median	-6961.8	-	-6961.8	-6961.8
	Mean	-6961.8	-	-6961.8	-6961.8
	Std	3.71E-12	-	3.71E-12	3.71E-12
	I.R	0	25	0	0
	S.R	25	0	25	25
$g07$	Worst	24.591	24.804	25.09	24.492
	Best	24.307	**24.306**	24.307	24.307
	Median	24.314	**24.310**	24.316	24.315
	Mean	24.361	**24.337**	24.453	24.345
	Std	0.0895	0.1001	0.2346	0.0567
	I.Runs	0	0	0	0
	S.Runs	13	18	11	13
$g08$	Worst	-0.095825	-0.029143	-0.095825	-0.095825
	Best	**-0.095825**	-0.095825	**-0.095825**	**-0.095825**
	Median	**-0.095825**	-0.095779	**-0.095825**	**-0.095825**
	Mean	**-0.095825**	-0.087818	**-0.095825**	**-0.095825**
	Std	**0**	0.2019	2.83E-18	4.91E-18
	I.Runs	0	0	0	0
	S.Runs	**25**	21	**25**	**25**
$g09$	Worst	680.84	680.81	680.91	680.84
	Best	680.63	**680.63**	680.63	680.63
	Median	680.65	**680.64**	680.67	680.64
	Mean	680.67	**680.66**	680.69	680.66
	Std	0.0466	0.0369	0.0612	0.0488
	I.R	0	0	0	0
	S.R	**9**	8	4	11
$g10$	Worst	7934	7080.5	8767.3	8621.4
	Best	**7051.2**	7080.5	7053.4	7052.8
	Median	7276.6	7080.5	7503.5	7271.9
	Mean	**7371.4**	7080.5	7601.8	7391.1
	Std	299.6	0	477.25	380.47
	I.R	0	24	0	0
	S.R	0	0	0	0
$g11$	Worst	0.9784	**0.7499**	0.9719	0.9755
	Best	0.7503	**0.7499**	0.7520	0.7555
	Median	0.9127	**0.7499**	0.7520	0.7555
	Mean	0.889	**0.7499**	0.8955	0.9037
	Std	0.0772	1.09E-07	0.0557	0.0716
	I.R	0	0	0	0
	S.R	3	25	0	0
$g12$	Worst	-1	-1	-1	-1
	Best	-1	-1	-1	-1
	Median	-1	-1	-1	-1
	Mean	-1	-1	-1	-1
	Std	0	9.10E-08	0	0
	I.R	0	0	0	0
	S.R	25	25	25	25
$g13$	Worst	0.9999	0.4389	0.9999	1.5985
	Best	0.9235	**0.0540**	0.4929	0.5712
	Median	0.9983	**0.0700**	0.9996	0.9949
	Mean	0.9908	**0.2172**	0.9644	0.9791
	Std	0.0169	0.1958	0.1081	0.1636
	I.R	0	13	0	0
	S.R	0	6	0	0