2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Self-adaptive Differential Evolution Algorithm for Constrained Real-Parameter Optimization

V. L. Huang, A. K. Qin, *Member, IEEE* and P. N. Suganthan, *Senior Member, IEEE*

***Abstract*—In this paper, we propose an extension of Self-adaptive Differential Evolution algorithm (SaDE) to solve optimization problems with constraints. In comparison with the original SaDE algorithm, the replacement criterion was modified for handling constraints. The performance of the proposed method is reported on the set of 24 benchmark problems provided by CEC2006 special session on constrained real parameter optimization.**

I. Introduction

MANY optimization problems in science and engineering have a number of constraints. Evolutionary algorithms have been successful in a wide range of applications. However, evolutionary algorithms naturally perform unconstrained search. Therefore, when used for solving constrained optimization problems, they require additional mechanisms to handle constraints in their fitness function. In the literature, several constraints handling techniques have been suggested for solving constrained optimization by using evolutionary algorithms.

Michalewicz and Schoenauer [3] grouped the methods for handling constraints by evolutionary algorithm 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 most common approach to deal with constraints is the method based on penalty functions, which penalize infeasible solutions. However, penalty functions have, in general, several limitations. They require a careful tuning to determine the most appropriate penalty factors. Also, they tend to behave ill when trying to solve a problem in which the optimum is at the boundary between the feasible and the infeasible regions or when the feasible region is disjoint. For some difficult problems in which it is extremely difficult to locate a feasible solution due to inappropriate representation scheme, researchers designed special representations and operators to preserve the feasibility of solutions at all the time. Recently there are a few methods which emphasize the distinction between feasible and infeasible solutions in the search space, such as behavioral memory method, superiority of feasible solutions to infeasible solutions, and repairing infeasible solutions. Also researchers developed hybrid methods which combine evolutionary algorithm with another technique (normally a numerical optimization approach) to handle constraints.

The Self-adaptive Differential Evolution algorithm (SaDE) was introduced in [1], in which the choice of learning strategy and the two control parameters F and CR are not required to be pre-specified. During evolution, the suitable learning strategy and parameter settings are gradually self-adapted according to the learning experience. In [1], SaDE was tested on a set of benchmark functions without constraints. In this work, we generalize SaDE to handle problems with constraints and investigate the performance on 24 constrained problems.

The authors are with School of Electrical and Electronic Engineering, Nanyang Technological University, 50 Nanyang Ave., 639798 Singapore (email: huangling@pmail.ntu.edu.sg, qinkai@pmail.ntu.edu.sg, epnsugan@ntu.edu.sg)

II. Differential Evolution Algorithm

Differential evolution (DE) algorithm, proposed by Storn and Price [4], is a simple but powerful population-based stochastic search technique for solving global optimization problems. The original DE algorithm is described in detail as follows: Let $\mathbf{S} \subset \Re^n$ be the n-dimensional search space of the problem under consideration. The DE evolves a population of NP n-dimensional individual vectors, i.e. solution candidates, $\mathbf{X}_i = (x_{i1}, \ldots, x_{in}) \in \mathbf{S}$, $i = 1, \ldots, NP$, from one generation to the next. The initial population should ideally cover the entire parameter space by randomly distributing each parameter of an individual vector with uniform distribution between the prescribed upper and lower parameter bounds x_j^u and x_j^l.

At each generation G, DE employs mutation and crossover operations to produce a trial vector $\mathbf{U}_{i,G}$ for each individual vector $\mathbf{X}_{i,G}$, also called target vector, in the current population.

A. Mutation operation

For each target vector $\mathbf{X}_{i,G}$ at generation G, an associated mutated vector $\mathbf{V}_{i,G} = \{v_{1i,G}, v_{2i,G}, \ldots, v_{ni,G}\}$ can usually be generated by using one of the following 5 strategies as shown in the online codes available at http://www.icsi.berkeley.edu/~storn/code.html:

0-7803-9487-9/06/$20.00/©2006 IEEE

"DE/rand/1": $\mathbf{V}_{i,G} = \mathbf{X}_{r_1,G} + F \cdot \left(\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G} \right)$

"DE/best/1": $\mathbf{V}_{i,G} = \mathbf{X}_{best,G} + F \cdot \left(\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G} \right)$

"DE/current to best/1":

$\mathbf{V}_{i,G} = \mathbf{X}_{i,G} + F \cdot \left(\mathbf{X}_{best,G} - \mathbf{X}_{i,G} \right) + F \cdot \left(\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G} \right)$

"DE/best/2":

$\mathbf{V}_{i,G} = \mathbf{X}_{best,G} + F \cdot \left(\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G} \right) + F \cdot \left(\mathbf{X}_{r_3,G} - \mathbf{X}_{r_4,G} \right)$

"DE/rand/2": $\mathbf{V}_{i,G} = \mathbf{X}_{r_1,G} + F \cdot \left(\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G} \right) + F \cdot \left(\mathbf{X}_{r_4,G} - \mathbf{X}_{r_5,G} \right)$

where indices r_1, r_2, r_3, r_4, r_5 are random and mutually different integers generated in the range $[1,\ NP]$, which should also be different from the current trial vector's index i. F is a factor in $(0,\ 1+)$ for scaling differential vectors and $\mathbf{X}_{best,G}$ is the individual vector with best fitness value in the population at generation G.

B. Crossover operation

After the mutation phase, the "binominal" crossover operation is applied to each pair of the generated mutant vector $\mathbf{V}_{i,G}$ and its corresponding target vector $\mathbf{X}_{i,G}$ to generate a trial vector: $\mathbf{U}_{i,G} = \left(u_{1i,G}, u_{2i,G},\ \ldots,\ u_{ni,G} \right)$.

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{if } \left(rand_j[0,\ 1] \le CR \right) \text{ or } \left(j = j_{rand} \right) \\ x_{j,i,G}, & \text{otherwise} \end{cases}$$

$$j = 1,\ 2,\ \ldots,n$$

where CR is a user-specified crossover constant in the range $[0,1)$ and j_{rand} is a randomly chosen integer in the range $[1,\ n]$ to ensure that the trial vector $\mathbf{U}_{i,G}$ will differ from its corresponding target vector $\mathbf{X}_{i,G}$ by at least one parameter.

C. Selection operation

If the values of some parameters of a newly generated trial vector exceed the corresponding upper and lower bounds, we randomly and uniformly reinitialize it within the search range. Then the fitness values of all trial vectors are evaluated. After that, a selection operation is performed. The fitness value of each trial vector $f\left(\mathbf{U}_{i,G}\right)$ is compared to that of its corresponding target vector $f\left(\mathbf{X}_{i,G}\right)$ in the current population. If the trial vector has smaller or equal fitness value (for minimization problem) than the corresponding target vector, the trial vector will replace the target vector and enter the population of the next generation. Otherwise, the target vector will remain in the population for the next generation. The operation is expressed as follows:

$$\mathbf{X}_{i,G+1} = \begin{cases} \mathbf{U}_{i,G}, & \text{if } f(\mathbf{U}_{i,G}) \le f(\mathbf{X}_{i,G}) \\ \mathbf{X}_{i,G} & \text{otherwise} \end{cases}$$

The above 3 steps are repeated generation after generation until some specific stopping criteria are satisfied.

III. SaDE Algorithm

To achieve good performance on a specific problem by using the original DE algorithm, we need to try all available (usually 5) learning strategies in the mutation phase and fine-tune the corresponding critical control parameters CR, F and NP. The performance of the original DE algorithm is highly dependent on the strategies and parameter settings. Although we may find the most suitable strategy and the corresponding control parameters for a specific problem, it may require a huge amount of computation time. Also, during different evolution stages, different strategies and different parameter settings with different global and local search capabilities might be preferred. Therefore, we developed SaDE algorithm that can automatically adapt the learning strategies and the parameters settings during evolution. The main ideas of the SaDE algorithm are summarized below.

A. Strategy Adaptation

SaDE probabilistically selects one out of several available learning strategies for each individual in the current population. Hence, we should have several candidate learning strategies available to be chosen and also we need to develop a procedure to determine the probability of applying each learning strategy. In the preliminary SaDE version [1], only two candidate strategies are employed, i.e. "rand/1/bin" and "current to best/2/bin". Our recent work suggests that incorporating more strategies can further improve the performance of the SaDE. Here, we use 4 strategies instead of the original two to enhance the SaDE.

DE/rand/1: $\mathbf{V}_{i,G} = \mathbf{X}_{r_1,G} + F \cdot \left(\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G} \right)$

DE/current to best/2:

$\mathbf{V}_{i,G} = \mathbf{X}_{i,G} + F \cdot \left(\mathbf{X}_{best,G} - \mathbf{X}_{i,G} \right) + F \cdot \left(\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G} \right) + F \cdot \left(\mathbf{X}_{r_3,G} - \mathbf{X}_{r_4,G} \right)$

DE/rand/2:

$\mathbf{V}_{i,G} = \mathbf{X}_{r_1,G} + F \cdot \left(\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G} \right) + F \cdot \left(\mathbf{X}_{r_4,G} - \mathbf{X}_{r_5,G} \right)$

DE/current-to-rand/1:

$U_{i,G} = \mathbf{X}_{r_1,G} + K \cdot \left(\mathbf{X}_{r_3,G} - \mathbf{X}_{i,G} \right) + F \cdot \left(\mathbf{X}_{r_1,G} - \mathbf{X}_{r_2,G} \right)$

In strategy "DE/current-to-rand/1", K is the coefficient of combination in $[-0.5,1.5]$.

Since here we have four candidate strategies instead of two strategies in [1], assuming that the probability of applying the four different strategies to each individual in the current population is p_i, $i = 1,2,3,4$. The initial probabilities are set to be equal to 0.25, i.e., $p_1 = p_2 = p_3 = p_4 = 0.25$. Therefore, each strategy has equal probability to be applied

to every individual in the initial population. According to the probability, we apply Roulette Wheel selection to select the strategy for each individual in the current population. After evaluation of all newly generated trial vectors, the number of trial vectors successfully entering the next generation while generated by each strategy is recorded as ns_i, $i=1,2,3,4$ respectively, and the numbers of trial vectors discarded while generated by each strategy is recorded as nf_i, $i=1,2,3,4$. ns_i and nf_i are accumulated within a specified number of generations (20 in our experiments), called the "learning period". Then, the probability of p_i is updated as:

$$p_i = \frac{ns_i}{ns_i + nf_i}$$

The above expression represents the percentage of the success rate of trial vectors generated by each strategy during the learning period. Therefore, the probabilities of applying those four strategies are updated every generation, after the learning period. We only accumulate the value of ns_i and nf_i in recent 20 generations to avoid the possible side-effect accumulated in the far previous learning stage. This adaptation procedure can gradually evolve the most suitable learning strategy at different stages during the evolution for the problem under consideration.

B. Parameter Adaptation

In the original DE, the 3 control parameters CR, F and NP are closely related to the problem under consideration. Here, we keep NP as a user-specified value as in the original DE, so as to deal with problems with different dimensionalities. Between the two parameters CR and F, CR is much more sensitive to the problem's property and complexity such as the multi-modality, while F is more related to the convergence speed. Here, we allow F to take different random values in the range $(0, 2]$ with normal distributions of mean 0.5 and standard deviation 0.3 for different individuals in the current population. This scheme can keep both local (with small F values) and global (with large F values) search ability to generate the potential good mutant vector throughout the evolution process. For the control parameter K in strategy "DE/current-to-rand/1", experiments show that it is always successful to optimize a function using a normally distributed random value for K. So here we set $K = F$ to reduce one more tuning parameter. The control parameter CR plays an essential role in the original DE algorithm. The proper choice of CR may lead to good performance under several learning strategies while a wrong choice may result in performance deterioration under any learning strategy. Also, the good CR parameter value usually falls within a small range, in which the algorithm can perform consistently well on a complex problem. Therefore, we consider accumulating the previous learning experience within a certain generational interval so as to dynamically adapt the value of CR to a suitable range. We assume that CR is normally distributed in a range with mean CRm and standard deviation 0.1. Initially, CRm is set at 0.5 and different CR values conforming this normal distribution are generated for each individual in the current population. These CR values for all individuals remain for 5 generations and then a new set of CR values is generated under the same normal distribution. During every generation, the CR values associated with trial vectors successfully entering the next generation are recorded. After a specified number of generations (20 in our experiments), CR has been changed for several times (20/5=4 times in our experiments) under the same normal distribution with center CRm and standard deviation 0.1, and we recalculate the mean of normal distribution of CR according to all the recorded CR values corresponding to successful trial vectors during this period. With this new normal distribution's mean and the standard devidation 0.1, we repeat the above procedure. As a result, the proper CR value range for the current problem can be learned to suit the particular problem. Note that we will reset the record of the successful CR values to zero once we recalculate the normal distribution's mean to avoid the possible inappropriate long-term accumulation effects.

We introduce the above learning strategy adaptation schemes into the original DE algorithm and develop a Self-adaptive Differential Evolution algorithm (SaDE) algorithm. The SaDE does not require the choice of a certain learning strategy and the setting of specific values to critical control parameters CR and F. The learning strategy and control parameter CR, which are highly dependent on the problem's characteristic and complexity, are self-adapted by using the previous learning experience. Therefore, the SaDE algorithm can demonstrate consistently good performance on problems with different properties, such as unimodal and multimodal problems. The influence on the performance of SaDE by the number of generations during which previous learning information is collected is not significant.

C. Local search

To speed up the convergence of the SaDE algorithm, we apply a local search procedure once every 500 generations, on 5% of individuals including the best individual found so far and the randomly selected individuals out of the best 50% of the individuals in the current population. Here, we employ the Sequential Quadratic Programming (SQP) method as the local search method.

IV. HANDLING CONSTRAINTS

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:

Minimize: $f(\mathbf{x})$, $\mathbf{x}=(x_1,x_2,\ldots,x_n)$ and $\mathbf{x}\in\mathcal{S}$

Subject to:

$$g_i(\mathbf{x})\le 0,\ i=1,\ldots,q$$
$$h_j(\mathbf{x})=0,\ j=q+1,\ldots,m$$

$\mathcal{S}$ is the whole search space. q is the number of inequality constraints. The number of equality constraints is m-q. For convenience, the equality constraints are always transformed into the inequality form, and then we can combine all the constraints as

$$G_i(\mathbf{x})=\begin{cases}\max\{g_i(\mathbf{x}),0\} & i=1,\ldots q\\ |h_i(\mathbf{x})| & i=q+1,\ldots,m\end{cases}$$

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 constraints $G_i(\mathbf{x})$.

Among various constraints handling methods mentioned in the introduction, some methods based on superiority of feasible solutions, such as the approach proposed by Deb [5], has demonstrated promising performance, as indicated in [6][7] which deal with constraints using DE. Besides this, Deb's selection criterion [5] has no parameter to fine-tune, which is also the motivation of our SaDE too - no fine-tuning of parameters as much possible. Hence, we incorporate this constraints handling technique as follows:

During the selection procedure, the trial vector is compared to that of its corresponding target vector in the current population considering both the fitness value and constraints. The trial vector will replace the target vector and enter the population of the next generation if any of the following conditions is true.

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

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

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

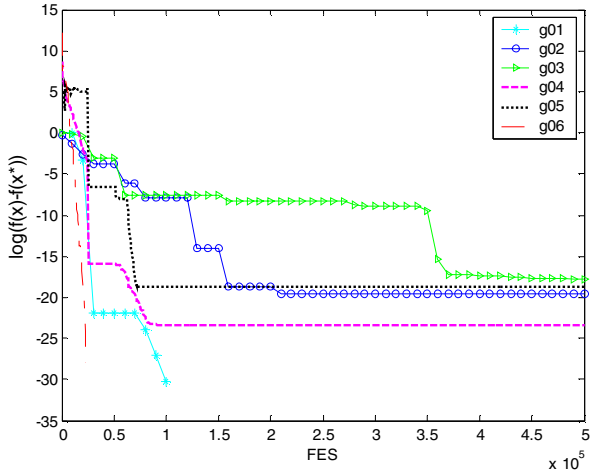
where $w_i=\dfrac{1}{G\max_i}$ is a weighted parameter, $G\max_i$ is the maximum violation of the constraints $G_i(\mathbf{x})$ obtained so far. Here, we set w_i as $\dfrac{1}{G\max_i}$ which varies during the evolution in order to accurately normalize the constraints of the problem, thus the overall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

V. Experimental Results

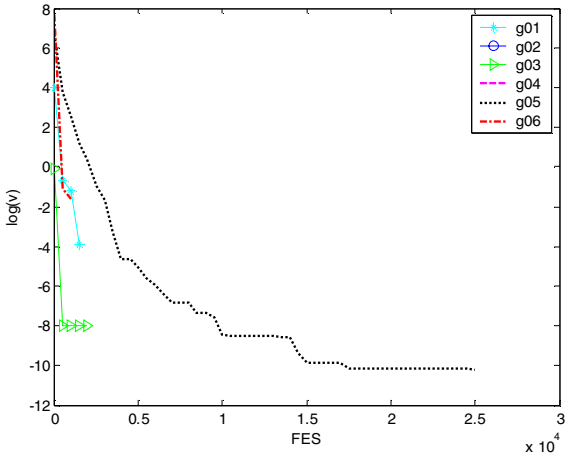
We evaluate the performance of the SaDE algorithm on 24 benchmark functions with constraints [2], which include linear, nonlinear, quadratic, cubic, polynomial constraints. The population size is set at 50.

For each function, the SaDE algorithm runs 25 times. We use the fitness value of best known solutions ($f(\mathbf{x}^*)$) newly updated in [2]. The error values achieved when FES=5e+3, FES=5e+4, FES=5e+5 for the 24 test functions are listed in Tables I-IV. We record the FES needed in each run for finding a solution satisfying the successful condition [2] in Table V. The success rate, feasible rate, and success performance are also listed.

The convergence maps of SaDE on functions 1-6, functions 7-12, functions 13-18, and functions 19-24 are plotted in Figures 1-4 respectively.

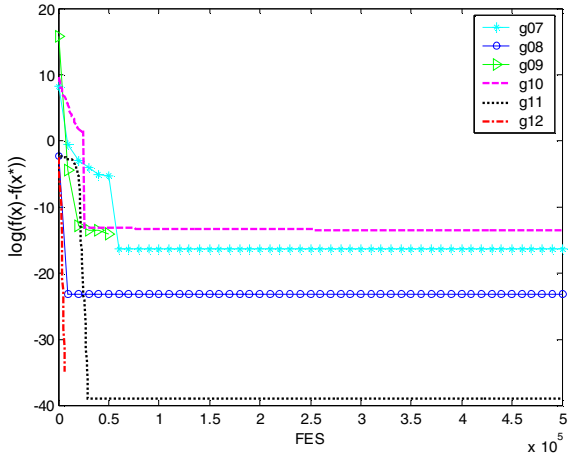


(1-a) log(f(x)-f(x*)) vs FES

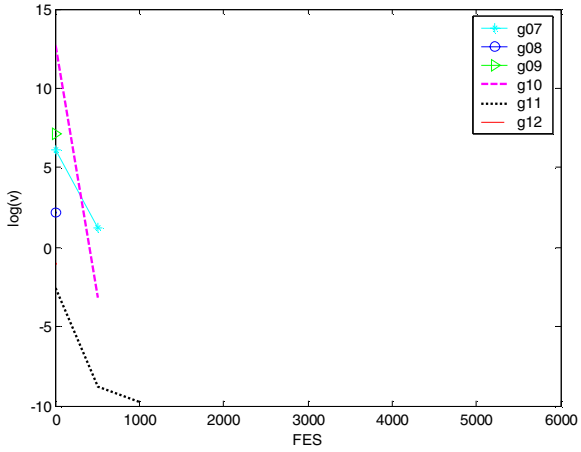


(1-b) log(v) vs FES

Figure 1: Convergence Graph for Function 1-6

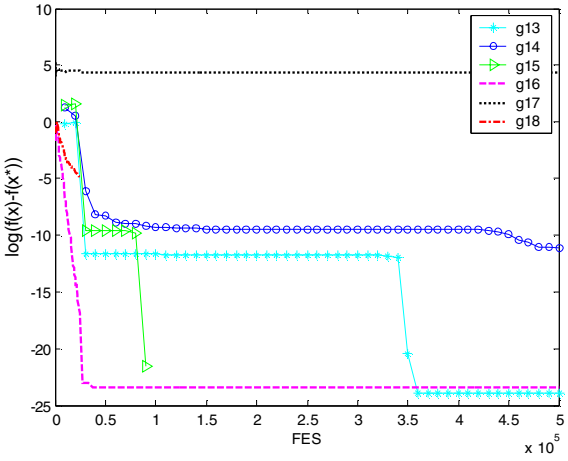


(2-a) log(f(x)-f(x*)) vs FES

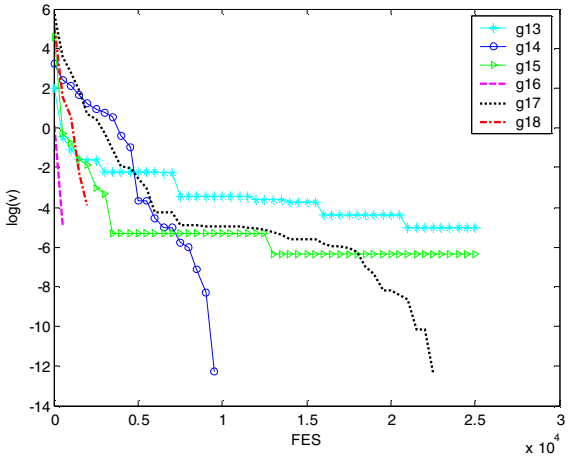


(2-b) log(v) vs FES

Figure 2 Convergence Graph for Function 7-12

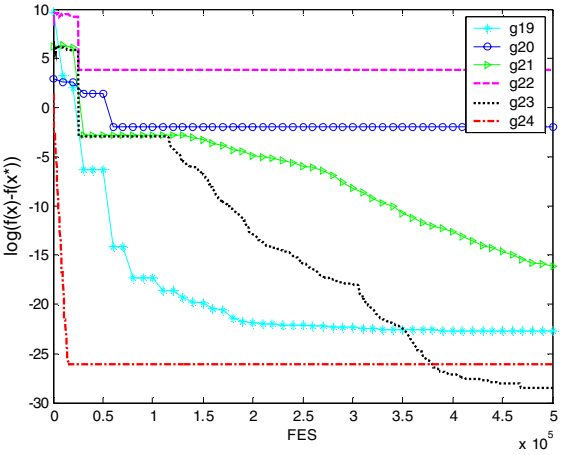


(3-a) log(f(x)-f(x*)) vs FES

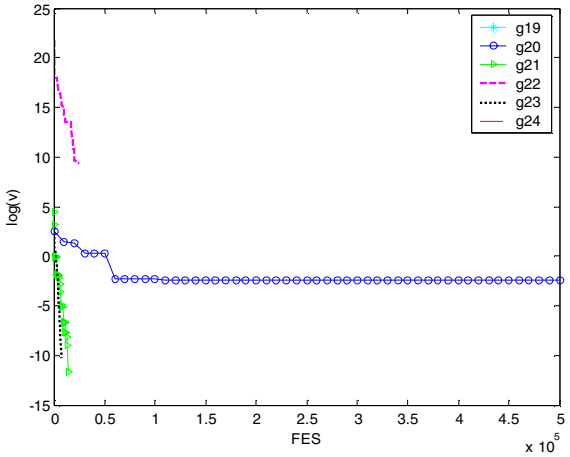


(3-b) log(v) vs FES

Figure 3 Convergence Graph for Function 13-18



(4-a) log(f(x)-f(x*)) vs FES



(4-b) log(v) vs FES

Figure 4 Convergence Graph for Function 19-24

From the results, we could observe that, for all problems, the SaDE algorithm could reach the newly updated best known solutions except problems 20 and 22.

As shown in Table V, the feasible rates of all problems are

100%, except problem 20 and 22. Problem 20 is highly constrained and no algorithm in the literature found feasible solutions. The successful rates are very encouraging, as most problems have 100%. Problems 2, 3, 10, 14, 18, 21 and 23 have 84%, 96%, 80%, 92%, 60% and 88% respectively. Problem 17 has 4%, with successfully finding better solution only once. Although the successful rate of problem 22 is 0, the result we obtained indeed is much better than previous best known solutions, and approximates the newly updated best known solutions.

We set MAX_FES as 5e+5, however from the experiment results we could find that SaDE actually achieved the best known solutions within 5e+4 for many problems.

We calculate the algorithm complexity according to [2] show in Table VI. We use Matlab 6.5 to implement the algorithm and the system configurations are:

Intel Pentium® 4 CPU 3.00 GHZ
2 GB of memory
Windows XP Professional Version 2002

TABLE VI: COMPUTATIONAL COMPLEXITY

T1	T2	(T2-T1)/T1
4.9182	8.3405	0.6958

VI. CONCLUSION

In this paper, we generalized the Self-adaptive Differential Evolution algorithm for handling optimization problem with multiple constraints, without introducing any additional parameters.

The performance of our approach was evaluated on the testbed for CEC2006 special session on constrained real parameter optimization. The SaDE algorithm demonstrated effectiveness and robustness.

REFERENCES

[1] A. K. Qin and P. N. Suganthan, "Self-adaptive Differential Evolution Algorithm for Numerical Optimization" In: *IEEE Congress on Evolutionary Computation (CEC 2005)* Edinburgh, Scotland, Sep 02-05, 2005.

[2] J. J. Liang, T. P. Runarsson, E. Mezura-montes, M. Clerc, P.N.Suganthan, C. A. C. Coello, and K.Deb, "Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Paremeter Optimization," Technical Report, 2005. http://www.ntu.edu.sg/home/EPNSugan

[3] Z. Michalewicz and M. Schoenauer, "Evolutionary Algorithms for Constrained Parameter Optimization Problems," *Evolutionary Computation*, 4(1):1–32, 1996.

[4] R. Storn and K. V. Price, "Differential evolution-A simple and Efficient Heuristic for Global Optimization over Continuous Spaces," Journal of Global Optimization 11:341-359. 1997.

[5] K. Deb. "An Efficient Constraint Handling Method for Genetic Algorithms", Computer Methods in Applied Mechanics and Engineering, 186(2/4):311–338, 2000.

[6] J. Lampinen. "A Constraint Handling Approach for the Differential Evolution Algorithm." In Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002), volume 2, pages 1468–1473, Piscataway, New Jersey, May 2002.

[7] R. Landa-Becerra and C. A. C. Coello. "Optimization with Constraints using a Cultured Differential Evolution Approach." In Proceedings of the Genetic and Evolutionary ComputationConference (GECCO'2005), volume 1, pages 27-34, New York, June 2005. Washington DC, USA, ACM Press.

TABLE I ERROR VALUES ACHIEVED WHEN FES= 5×10^3 , FES= 5×10^4, FES= 5×10^5 FOR PROBLEMS 1-6

FES \ Prob.		g01	g02	g03	g04	g05	g06
5×10^3	Best	2.9525e+000(0)	3.2958e-001(0)	3.3141e-001(0)	3.5714e+001(0)	2.2578e+001(3)	6.4548e+001(0)
	Median	4.2828e+000(0)	3.7275e-001(0)	8.4884e-001(0)	7.5864e+001(0)	2.7489e+002(3)	3.4622e+002(0)
	Worst	5.2143e+000(0)	4.3843e-001(0)	8.8551e-001(1)	1.1877e+002(0)	1.0948e+002(3)	1.7502e+003(0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 1, 3	0, 0, 0
	$\overline{v}$	0	0	0	0	4.7389e-003	0
	Mean	4.1779e+000	3.7920e-001	7.8931e-001	7.7970e+001	1.5908e+002	4.3473e+002
	Std	5.2257e-001	2.9111e-002	1.5394e-001	2.1465e+001	1.1344e+002	3.6649e+002
5×10^4	Best	2.8414e-010(0)	4.2504e-003(0)	5.9753e-005(0)	2.5043e-007(0)	1.0004e-011(0)	4.5475e-011(0)
	Median	2.9675e-010(0)	2.2353e-002(0)	9.9222e-004(0)	2.9835e-007(0)	5.2481e-004(0)	4.5475e-011(0)
	Worst	3.0000e-010(0)	3.8493e-002(0)	7.1664e-001(0)	3.3795e-007(0)	1.3955e-003(0)	4.5475e-011(0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	2.9488e-010	2.1825e-002	6.2425e-002	2.9795e-007	6.3977e-004	4.5475e-011
	Std	4.9526e-012	8.2730e-003	1.5421e-001	2.5148e-008	5.4286e-004	0
5×10^5	Best	0(0)	8.0719e-010(0)	1.3749e-010(0)	2.1667e-007(0)	0(0)	4.5475e-011(0)
	Median	0(0)	3.0800e-009(0)	1.7770e-008(0)	2.1667e-007(0)	0(0)	4.5475e-011(0)
	Worst	0(0)	1.8353e-002(0)	1.3389e-004(0)	2.1667e-007(0)	0(0)	4.5475e-011(0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	0	2.0560e-003	1.3532e-005	2.1667e-007	0	4.5475e-011
	Std	0	4.9786e-003	3.4743e-005	1.8550e-012	1.8190e-013	0

TABLE II ERROR VALUES ACHIEVED WHEN FES= 5×10^3 , FES= 5×10^4, FES= 5×10^5 FOR PROBLEMS 7-12

FES \ Prob.		g07	g08	g09	g10	g11	g12
5×10^3	Best	3.8845e+000(0)	8.1964e-011(0)	4.1400e-001(0)	9.2363e+002(0)	3.3473e-004(0)	1.9651e-014(0)
	Median	6.7238e+000(0)	8.1964e-011(0)	7.0402e-001(0)	1.4735e+003(0)	9.6840e-002(0)	1.2552e-012(0)
	Worst	1.2048e+001(0)	8.1964e-011(0)	1.8411e+000(0)	2.0997e+003(0)	2.1645e-001(0)	1.2816e-010(0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	7.1258e+000	8.1964e-011	8.1345e-001	1.4603e+003	1.0756e-001	1.0822e-011
	Std	2.6450e+000	1.0599e-017	3.5365e-001	2.7973e+002	6.8547e-002	2.6763e-011
5×10^4	Best	6.8221e-008(0)	8.1964e-011(0)	3.7440e-007(0)	1.9910e-006(0)	0(0)	0(0)
	Median	2.7655e-003(0)	8.1964e-011(0)	7.1033e-007(0)	1.3888e-001(0)	0(0)	0(0)
	Worst	2.0290e-002(0)	8.1964e-011(0)	3.8943e-005(0)	3.4904e+000(0)	9.9985e-005(0)	0(0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	4.9297e-003	8.1964e-011	4.4545e-006	3.9758e-001	9.3997e-006	0
	Std	5.9290e-003	6.0492e-018	8.7853e-006	7.4290e-001	2.7549e-005	0
5×10^5	Best	6.8180e-008(0)	8.1964e-011(0)	3.7440e-007(0)	6.6393e-011(0)	0(0)	0(0)
	Median	1.4608e-007(0)	8.1964e-011(0)	3.7440e-007(0)	1.8120e-006(0)	0(0)	0(0)
	Worst	6.3431e-005(0)	8.1964e-011(0)	3.7440e-007(0)	7.8300e-006(0)	0(0)	0(0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	4.7432e-006	8.1964e-011	3.7440e-007	1.6907e-006	0	0
	Std	1.4993e-005	3.8426e-018	7.9851e-014	1.5244e-006	0	0

TABLE III ERROR VALUES ACHIEVED WHEN FES= 5×10^3 , FES= 5×10^4, FES= 5×10^5 FOR PROBLEMS 13-18

FES \ Prob.		g13	g14	g15	g16	g17	g18
5×10^3	Best	9.4326e-001(3)	4.3915e+000(3)	6.6347e-002(2)	2.8438e-002(0)	9.1761e+001(4)	1.2814e-001(0)
	Median	8.3030e-001(3)	5.4314e+000(3)	9.4042e-001(2)	5.7669e-002(0)	9.3311e+001(4)	2.6913e-001(0)
	Worst	5.2140e-001(3)	2.4726e+000(3)	1.6126e+000(2)	1.1087e-001(0)	9.5509e+001(4)	4.1448e-001(0)
	c	0, 2, 3	0, 3, 3	0, 1, 2	0, 0, 0	0, 3, 4	0, 0, 0
	$\overline{v}$	5.6738e-002	3.7325e-002	1.3524e-002	0	5.5918e-002	0
	Mean	1.1713e+000	4.9550e+000	1.4324e+000	5.9573e-002	1.0664e+002	2.7886e-001
	Std	1.2511e+000	1.7242e+000	1.5740e+000	1.9716e-002	6.1893e+001	7.7546e-002
5×10^4	Best	5.3102e-006(0)	1.5331e-005(0)	6.0822e-011(0)	6.5214e-011(0)	3.9816e+001(0)	2.7581e-011(0)
	Median	8.3300e-006(0)	1.4460e-004(0)	6.8150e-005(0)	6.5251e-011(0)	7.4058e+001(0)	1.0178e-010(0)
	Worst	3.8491e-001(0)	4.4106e-004(0)	1.4563e-004(0)	6.5455e-011(0)	7.4058e+001(0)	1.9163e-001(0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	1.0778e-001	1.6058e-004	6.3430e-005	6.5277e-011	7.2688e+001	1.5312e-002
	Std	1.7638e-001	1.0951e-004	6.0058e-005	6.7594e-014	6.8484e+000	5.2992e-002

5×10^5	Best	4.1898e-011(0)	2.9050e-006(0)	6.0822e-011(0)	6.5214e-011(0)	8.1858e-011(0)	1.5561e-011(0)
	Median	4.1898e-011(0)	1.4793e-005(0)	6.0822e-011(0)	6.5214e-011(0)	7.4058e+001(0)	1.5561e-011(0)
	Worst	1.0696e-006(0)	2.5233e-004(0)	6.0822e-011(0)	6.5214e-011(0)	7.4058e+001(0)	1.9104e-001(0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	8.0263e-008	4.6979e-005	6.0822e-011	6.5214e-011	6.9670e+001	1.5284e-002
	Std	2.7832e-007	6.4986e-005	0	0	1.6168e+001	5.2898e-002

TABLE IV ERROR VALUES ACHIEVED WHEN FES= 5×10^3 , FES= 5×10^4, FES= 5×10^5 FOR PROBLEMS 19-24

FES \ Prob.		g19	g20	g21	g22	g23	g24
5×10^3	Best	9.9842e+001(0)	1.2960e+001(20)	4.6515e+002(5)	1.6363e+004(19)	6.1159e+002(4)	1.7199e-005(0)
	Median	1.5038e+002(0)	1.5603e+001(20)	3.9781e+002(5)	7.8714e+003(19)	2.8993e+002(4)	9.3936e-005(0)
	Worst	1.9265e+002(0)	1.3569e+001(20)	4.0739e+002(5)	1.3480e+004(19)	4.4529e+002(5)	1.7135e-004(0)
	c	0, 0, 0	2, 18, 20	0, 3, 5	14, 19, 19	0, 2, 4	0, 0, 0
	$\overline{v}$	0	5.6112e+000	6.3055e-002	1.7472e+007	6.0069e-003	0
	Mean	1.4462e+002	1.4265e+001	4.1002e+002	1.2034e+004	4.0600e+002	8.4004e-005
	Std	2.2065e+001	1.7690e+000	1.2050e+002	2.9316e+003	8.0627e+001	4.5396e-005
5×10^4	Best	5.8659e-007(0)	1.8020e-002(6)	4.6858e-002(0)	3.0199e+001(0)	7.6219e-003(0)	4.6372e-012(0)
	Median	8.5695e-004(0)	5.1702e-001(19)	6.1550e-002(0)	1.0372e+002(0)	5.5090e-002(0)	4.6372e-012(0)
	Worst	1.1303e-001(0)	4.5755e+000(19)	6.2397e-002(0)	1.1282e+004(16)	5.5144e-002(0)	4.6372e-012(0)
	c	0, 0, 0	1, 15, 19	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	1.5836e-001	0	0	0	0
	Mean	6.0660e-003	2.1193e+000	6.0340e-002	5.3477e+003	5.0237e-002	4.6372e-012
	Std	2.2429e-002	2.4755e+000	3.8090e-003	6.2993e+003	1.3633e-002	0
5×10^5	Best	5.4456e-011(0)	1.8020e-002(6)	0(0)	1.9483e+000(0)	0(0)	4.6372e-012(0)
	Median	1.3868e-010(0)	2.3757e-001(20)	2.5785e-008(0)	4.6907e+001(0)	3.9790e-013(0)	4.6372e-012(0)
	Worst	3.1141e-009(0)	5.3597e-001(20)	6.0712e-003(0)	1.2204e+002(0)	1.3788e-003(0)	4.6372e-012(0)
	c	0	0, 16, 20	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	8.8255e-002	0	0	0	0
	Mean	4.2052e-010	2.5564e-001	7.6753e-004	5.0203e+001	1.2061e-004	4.6372e-012
	Std	7.2976e-010	1.0638e-001	1.5058e-003	3.0415e+001	3.4116e-004	0

TABLE VI NUMBER OF FES TO ACHIEVE THE FIXED ACCURACY LEVEL ($(f(\vec{x}) - f(\vec{x}^*)) \leq 0.0001$), SUCCESS RATE, FEASIBLE RATE AND SUCCESS PERFORMANCE

Prob.	Best	Median	Worst	Mean	Std	Feasible Rate	Success Rate	Success Performance
g01	25115	25115	25115	25115	0	100%	100%	25115
g02	76915	128970	-	188990	142570	100%	84%	183850
g03	30000	261000	-	243520	135990	100%	96%	298960
g04	25107	25107	25113	25107	2	100%	100%	25107
g05	35500	65000	152000	74340	30484	100%	100%	73000
g06	12546	14404	18347	14394	1242	100%	100%	12546
g07	25195	101240	422860	143090	124800	100%	100%	27637
g08	782	1272	1775	1268	242	100%	100%	1323
g09	12960	16787	33166	18560	5170	100%	100%	21446
g10	26000	52000	153000	58760	33968	100%	100%	44167
g11	12643	25111	25120	23353	3482	100%	100%	25111
g12	463	1717	2576	1611	582	100%	100%	2576
g13	25161	25219	126080	42372	29775	100%	100%	25168
g14	32000	77000	-	154320	179040	100%	80%	45000
g15	25500	41500	97000	45240	18928	100%	100%	27000
g16	13144	14433	15797	14545	748	100%	100%	14948
g17	443000	-	-	497720	11400	100%	4%	12500000
g18	26000	26000	-	65400	130870	100%	92%	28261
g19	25531	51588	78048	48733	13552	100%	100%	52165
g20	-	-	-	-	-	0%	0%	-
g21	98500	320000	-	327660	166180	100%	60%	164170
g22	-	-	-	-	-	100%	0%	-
g23	82500	298500	-	294540	137090	100%	88%	129550
g24	4280	4843	5657	4847	360	100%	100%	4624