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

Self-Adaptive Differential Evolution Algorithm in Constrained Real-Parameter Optimization

Janez Brest, *Member, IEEE*, Viljem Žumer, *Member, IEEE*, and Mirjam Sepesy Maučec

***Abstract*— Differential Evolution (DE) has been shown to be a powerful evolutionary algorithm for global optimization in many real problems. Self-adaptation has been found to be high beneficial for adjusting control parameters during evolutionary process, especially when done without any user interaction. In this paper we investigate a self-adaptive differential evolution algorithm where more DE strategies are used and control parameters F and CR are self-adapted. The performance of the self-adaptive differential evolution algorithm is evaluated on the set of 24 benchmark functions provided for the CEC2006 special session on constrained real parameter optimization.**

I. Introduction

The general nonlinear programming problem an optimization algorithm is concerned with is to find $\vec{x}$ so as to optimize $f(\vec{x}); \vec{x} = \{x_1, x_2, ..., x_D\}$. D is the dimensionality of the function. Domains of the variables are defined by their lower and upper bounds: $x_{j,low}, x_{j,upp};\ j \in \{1, ..., D\}$. The feasible region is defined by a set of m additional constraints ($m \geq 0$):

$$g_i(\vec{x}) \leq 0, \text{ for } i = 1, ..., q, \text{ and}$$

$$h_j(\vec{x}) = 0, \text{ for } j = q+1, ..., m$$

Differential Evolution (DE) is a floating-point encoding evolutionary algorithm for global optimization over continuous spaces [20], [18], [11], [12], [10], [22]. Although the DE algorithm has been shown to be a simple yet powerful evolutionary algorithm for optimizing continuous functions, users are still faced with the problem of preliminary testing and hand-tuning of the evolutionary parameters prior to commencing the actual optimization process [22]. As a solution, self-adaptation has been found to be highly beneficial in automatically and dynamically adjusting evolutionary parameters such as crossover rates and mutation rates. Self-adaptation allows an evolution strategy to adapt itself to any general class of problems by reconfiguring itself accordingly, and to do this without any user interaction [2], [3], [6]. In literature, self-adaptation is usually applied to the F and CR control parameters. The efficiency and robustness of the DE algorithm are much more sensitive to the setting of F and CR control parameters than to the setting of the third DE parameter, that is NP.

Teo in [22] proposed an attempt at self-adapting the population size parameter in addition to self-adapting crossover and mutation rates.

Janez Brest, Viljem Žumer, and Mirjam Sepesy Maučec are with the Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova ul. 17, 2000 Maribor, Slovenia, (email:{janez.brest, zumer, mirjam.sepesy}@uni-mb.si).

The main objective of this paper is a performance evaluation of our self-adaptive jDE-2 [4] algorithm, which uses self-adapting control parameter mechanism on the control parameters F and CR. The performance of the algorithm is evaluated on the set of 24 benchmark functions provided for the CEC2006 special session on real parameter optimization [9].

The article is structured as follows. Section II gives an overview of work dealt with the DE. Section III shortly summarize the differential evolution. Section IV describes constrain-handling differential evolution. In Section V differential evolution jDE-2 algorithm, which use self-adaptive adjusting control parameters, is described. We conclude this section by proposing some improvements of our self-adapting jDE algorithm. In Section VI experimental results our self-adaptive jDE-2 algorithm on CEC 2006 benchmark functions are presented. Detailed performance analysis of the algorithm is given. Section VII concludes the paper with some final remarks.

II. Work Related with the Differential Evolution

The DE [20], [19] algorithm was proposed by Storn and Price, and since then the DE algorithm has been used in many practical cases. The original DE was modified, and many new versions proposed. Liu and Lampinen [11] reported that the effectiveness, efficiency and robustness of the DE algorithm are sensitive to the settings of the control parameters. The best settings for the control parameters depends on the function and requirements for consumption time and accuracy. Quite different conclusions were reported about the rules for choosing the control parameters of DE. In [15] it is stated that the control parameters of DE are not difficult to choose. On the other hand, Gämperle et al. [7] reported that choosing the proper control parameters for DE is more difficult than expected.

Ali and Törn in [1] proposed new versions of the DE algorithm, and also suggested some modifications to the classical DE in order to improve its efficiency and robustness. They introduced an auxiliary population of NP individuals alongside the original population (noted in [1], a notation using sets is used – population set-based methods). Next they proposed a rule for calculating the control parameter F automatically.

Sun et al. [21] proposed a combination of the DE algorithm and the estimation of distribution algorithm (EDA), which tries to guide its search towards a promising area by sampling new solutions from a probability model. Based

0-7803-9487-9/06/$20.00/©2006 IEEE

on experimental results it has been demonstrated that the DE/EDA algorithm outperforms both the DE and the EDA algorithms.

Qin and Suganthan in [17] proposed Self-adaptive Differential Evolution algorithm (SaDE), where the choice of learning strategy and the two control parameters F and CR are not required to be pre-defined. During evolution, the suitable learning strategy and parameter settings are gradually self-adapted according to the learning experience.

J. Brest et al. [4] compared version of a self-adaptive jDE algorithm with others adaptive and self-adaptive algorithms: FADE [12], DESAP [22], SaDE [17], and jDE [5]. The reported results show, that the jDE algorithm performs better than FADE and DESAP algorithms and self-adaptive jDE-2 algorithm gives comparable results on benchmark functions as the SaDE algorithm, which uses local search procedure, additionally. The comparison of jDE and jDE-2 algorithms was shown, the jDE-2 algorithm, which uses two DE strategies, performed better than jDE algorithm, which uses only one strategy. Comparative study exposed the jDE-2 as prospective algorithm for a global optimization.

III. The Differential Evolution Algorithm

DE creates new candidate solutions by combining the parent individual and several other individuals of the same population. A candidate replaces the parent only if it has better fitness value. DE has three parameters: amplification factor of the difference vector – F, crossover control parameter – CR and population size – NP.

Original DE algorithm keeps all three control parameters fixed during the optimization process. However, there still exists a lack of knowledge of how to find reasonably good values for the control parameters of DE for a given function [12].

The population of the original DE algorithm [19], [20], [18] contains NP D-dimensional vectors:

$$\vec{x}_{i,G} = \{x_{i,1,G}, x_{i,2,G}, ..., x_{i,D,G}\},\ i = 1, 2, ..., NP$$

G denotes the generation. During one generation for each vector, DE employs the mutation and crossover operations to produce a trial vector:

$$\vec{u}_{i,G} = \{u_{i,1,G}, u_{i,2,G}, ..., u_{i,D,G}\},\ i = 1, 2, ..., NP$$

Then a selection operation is used to choose vectors for the next generation $(G + 1)$.

The initial population is selected uniformly randomly between the lower $(x_{j,low})$ and upper $(x_{j,upp})$ bounds defined for each variable x_j. These bounds are specified by the user according to the nature of the problem. After initialization, DE performs several vector transforms (operations), in a process called evolution.

A. Mutation operation

Mutation for each population vector creates a mutant vector:

$$\vec{x}_{i,G} \Rightarrow \vec{v}_{i,G} = \{v_{i,1,G}, v_{i,2,G}, ..., v_{i,D,G}\},\ i = 1, 2, ..., NP$$

Mutant vector can be created by using one of the mutation strategies. The most useful strategies are:

- "rand/1": $\vec{v}_{i,G} = \vec{x}_{r_1,G} + F \cdot (\vec{x}_{r_2,G} - \vec{x}_{r_3,G})$
- "best/1": $\vec{v}_{i,G} = \vec{x}_{best,G} + F \cdot (\vec{x}_{r_1,G} - \vec{x}_{r_2,G})$
- "current to best/1": $\vec{v}_{i,G} = \vec{x}_{i,G} + F \cdot (\vec{x}_{best,G} - \vec{x}_{i,G}) + F \cdot (\vec{x}_{r_1,G} - \vec{x}_{r_2,G})$
- "best/2": $\vec{v}_{i,G} = \vec{x}_{best,G} + F \cdot (\vec{x}_{r_1,G} - \vec{x}_{r_2,G}) + F \cdot (\vec{x}_{r_3,G} - \vec{x}_{r_4,G})$
- "rand/2": $\vec{v}_{i,G} = \vec{x}_{r_1,G} + F \cdot (\vec{x}_{r_2,G} - \vec{x}_{r_3,G}) + F \cdot (\vec{x}_{r_4,G} - \vec{x}_{r_5,G})$

where the indexes r_1, r_2, r_3, r_4, r_5 represent the random and mutually different integers generated within range $[1, NP]$ and also different from index i. F is a mutation scale factor within the range $[0, 2]$, usually less than 1. $\vec{x}_{best,G}$ is the best vector in generation G.

B. Crossover operation

After mutation, a "binary" crossover operation forms the final trial vector, according to the i-th population vector and its corresponding mutant vector.

$$u_{i,j,G} = \begin{cases} v_{i,j,G} & \text{if } rand(0,1) \leq CR \text{ or } j = j_{rand}, \\ x_{i,j,G} & \text{otherwise} \end{cases}$$
$$i = 1, 2, ..., NP \text{ and } j = 1, 2, ..., D$$

CR is a crossover parameter or factor within the range $[0, 1)$ and presents the probability of creating parameters for trial vector from a mutant vector. Index j_{rand} is a randomly chosen integer within the range $[1, NP]$. It is responsible for the trial vector containing at least one parameter from the mutant vector.

If the control parameters from the trial vector are out of bounds, the proposed solutions in literature [20], [19], [18], [16] are: they are reflected into bounds, set on bounds or used as they are (out of bounds).

C. Selection operation

The selection operation selects according to the fitness value of the population vector and its corresponding trial vector, which vector will survive to be a member of the next generation. For example, if we have a minimization problem, we will use the following selection rule:

$$\vec{x}_{i,G+1} = \begin{cases} \vec{u}_{i,G} & \text{if } f(\vec{u}_{i,G}) < f(\vec{x}_{i,G}), \\ \vec{x}_{i,G} & \text{otherwise.} \end{cases}$$

IV. Constraint-handling

During the last few years several methods were proposed for handling constraints by genetic algorithms for parameter optimization problems. These methods were grouped by Michalewicz et al. [14], [8] into four categories:

- *methods based on preserving feasibility of solutions.* The idea behind the method is based on specialized

operators which transform feasible individuals into feasible individuals. The method assumes linear constrains only and a feasible starting point on feasible initial population.

- *methods based on penalty functions.* Many evolutionary algorithms incorporate a constraint-handling method based on the concept of exterior penalty functions which penalize infeasible solutions. These methods differ in important details, how the penalty function is designed and applied to infeasible solutions.
- *methods, which make a clear distinction between feasible and infeasible solutions.* There are few methods which emphasize the distinction between feasible and infeasible solutions in the search space. One of those method distinguishes between feasible and infeasible individuals: for any feasible individual $\vec{x}$ and any infeasible individually $\vec{y}$: $f(\vec{x}) < f(\vec{y})$, i.e. any feasible solution is better than any infeasible one.
- *other hybrid methods.* These methods combine evolutionary computation techniques with deterministic procedures for numerical optimization problems.

No special extensions of the DE algorithm are necessary to make it suitable for handling constraints [23]. Most constraint problems can be handled by the penalty method. A measure of the constraint violation is often useful when handling constraints. A solution $\vec{x}$ is regarded as *feasible* if

$$g_i(\vec{x}) \leq 0, \text{ for } i = 1, ..., q, \text{ and}$$
$$|h_j(\vec{x})| - \epsilon \leq 0, \text{ for } j = q+1, ..., m,$$

where equality constraints are transformed into inequalities. In CEC2006 [9] special section ϵ is set to 0.0001. Mean violations $\overline{v}$ is defined:

$$\overline{v} = \frac{(\sum_{i=1}^{q} G_i(\vec{x}) + \sum_{j=q+1}^{m} H_j(\vec{x}))}{m}, \text{ where}$$

$$G_i(\vec{x}) = \begin{cases} g_i(\vec{x}) & \text{if } g_i(\vec{x}) > 0 \\ 0 & \text{if } g_i(\vec{x}) \leq 0 \end{cases}$$

$$H_j(\vec{x}) = \begin{cases} |h_j(\vec{x})| & \text{if } |h_j(\vec{x})| - \epsilon > 0 \\ 0 & \text{if } |h_j(\vec{x})| - \epsilon \leq 0 \end{cases}$$

The sum of all constraint violations is zero for feasible solutions and positive when at least one constraint is violated. An obvious application of the constraint violation is to use it to guide the search towards feasible areas of the search space. There was quite a lot of work on such ideas and other constraint techniques in the EA-community during the 1990's. A summary of these techniques can be found in Michalewicz's and Fogel's book [13], which also contains information on many other stochastic techniques.

V. The Self-adaptive Differential Evolution Algorithms

Let us first describe self-adaptive jDE algorithm [5], [4]. It uses self-adapting mechanism on the control parameters F and CR.

In [5] the self-adapting control parameter mechanism of "*rand/1/bin*" strategy was used. This strategy is the most often used in practice [20], [21], [7], [11].

In [5] a self-adaptive control mechanism was used to change the control parameters F and CR during the run. The third control parameter NP was not changed during the run. Each individual in population was extended with parameter values. The control parameters that were adjusted by means of evolution are F and CR (see Figure 1). Both of them were applied at individual levels. The better values of these (encoded) control parameters lead to better individuals which, in turn, are more likely to survive and produce offspring and, hence, propagate these better parameter values.

$\vec{x}_{1,G}$	$F_{1,G}$	$CR_{1,G}$
$\vec{x}_{2,G}$	$F_{2,G}$	$CR_{2,G}$
...	...	...
$\vec{x}_{NP,G}$	$F_{NP,G}$	$CR_{NP,G}$

Fig. 1. Self-adapting: encoding aspect.

New control parameters $F_{i,G+1}$ and $CR_{i,G+1}$ were calculated as follows:

$$F_{i,G+1} = \begin{cases} F_l + rand_1 * F_u & \text{if } rand_2 < \tau_1, \\ F_{i,G} & \text{otherwise,} \end{cases}$$

$$CR_{i,G+1} = \begin{cases} rand_3 & \text{if } rand_4 < \tau_2, \\ CR_{i,G} & \text{otherwise.} \end{cases}$$

and they produce control parameters F and CR in a new parent vector. $rand_j, j \in \{1, 2, 3, 4\}$ are uniform random values within the range $[0, 1]$. τ_1 and τ_2 represent probabilities to adjust control parameters F and CR, respectively. τ_1, τ_2, F_l, F_u were taken fixed values $0.1, 0.1, 0.1, 0.9$, respectively. The new F takes a value from $[0.1, 1.0]$, and the new CR from $[0, 1]$ in a random manner. $F_{i,G+1}$ and $CR_{i,G+1}$ are obtained before the mutation is performed. So they influence the mutation, crossover and selection operations of the new vector $\vec{x}_{i,G+1}$.

$\vec{x}_{1,G}$	$F^1_{1,G}$	$CR^1_{1,G}$	$F^2_{1,G}$	$CR^2_{1,G}$
$\vec{x}_{2,G}$	$F^1_{2,G}$	$CR^1_{2,G}$	$F^2_{2,G}$	$CR^2_{2,G}$
...	...	...	...	...
$\vec{x}_{NP,G}$	$F^1_{NP,G}$	$CR^1_{NP,G}$	$F^2_{NP,G}$	$CR^2_{NP,G}$

Fig. 2. Self-adapting: encoding aspect of two strategies.

Some ideas, how to improve jDE algorithm, are reported in [4]. In the rest of this section we will outline them.

To keep solution of bound-constrained problems feasible, trial parameters that volatile boundary constraints are set to

TABLE I

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

FES		g01	g02	g03	g04	g05	g06
$\mathbf{5 \times 10^3}$	Best	2.0505 (0)	0.4070 (0)	0.7406 (0)	20.9966 (0)	15.1226 (7)	60.7180 (0)
	Median	5.1047 (0)	0.5030 (0)	0.9976 (0)	63.3410 (0)	201.3292 (8)	251.4440 (0)
	Worst	7.2914 (0)	0.5411 (0)	0.9932 (0)	161.2217 (0)	976.1121 (9)	831.1240 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	2, 3, 3	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0000	1.0834	0.0000
	Mean	5.3034	4.9680e-01	9.4358e-01	6.3542e+01	3.0997e+02	2.8087e+02
	Std	1.0670	3.0675e-02	7.4900e-02	2.8478e+01	3.3878e+02	1.7917e+02
$\mathbf{5 \times 10^4}$	Best	4.7410e-05 (0)	0.0262 (0)	0.4738 (0)	6.5150e-07 (0)	9.0323 (0)	2.3646e-11 (0)
	Median	0.0001 (0)	0.0834 (0)	0.7231 (0)	4.6348e-06 (0)	155.0791 (0)	2.3101e-10 (0)
	Worst	0.0005 (0)	0.1560 (0)	0.9155 (0)	1.4717e-05 (0)	5.4202 (2)	1.0268e-09 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	1.2610e-04	8.5485e-02	7.1313e-01	5.2805e-06	1.8287e+02	2.7077e-10
	Std	9.3563e-05	2.8747e-02	1.3929e-01	3.0574e-06	1.3900e+02	2.0464e-10
$\mathbf{5 \times 10^5}$	Best	0 (0)	7.7521e-11 (0)	0.0890 (0)	0 (0)	0 (0)	1.1823e-11 (0)
	Median	0 (0)	3.3051e-09 (0)	0.3481 (0)	0 (0)	0 (0)	1.1823e-11 (0)
	Worst	0 (0)	0.0110 (0)	0.5117 (0)	0 (0)	9.0697 (0)	1.1823e-11 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	0.0000	8.8089e-04	3.3808e-01	0.0000	1.0756	1.1823e-11
	Std	0.0000	3.0488e-03	1.0140e-01	0.0000	2.4193	0.0000

TABLE II

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

FES		g07	g08	g09	g10	g11	g12
$\mathbf{5 \times 10^3}$	Best	47.0147 (0)	6.5370e-08 (0)	13.3137 (0)	2711.5904 (0)	0.0022 (0)	2.5178e-05 (0)
	Median	92.1652 (0)	2.5660e-06 (0)	43.1004 (0)	7322.9075 (0)	0.1346 (0)	0.0002 (0)
	Worst	170.4361 (0)	1.7141e-05 (0)	67.6963 (0)	13759.8095 (0)	0.2501 (0)	0.0081 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	9.2682e+01	4.8370e-06	4.1988e+01	7.8258e+03	1.3221e-01	1.0618e-03
	Std	3.4874e+01	5.1240e-06	1.2917e+01	2.7236e+03	8.8471e-02	2.2902e-03
$\mathbf{5 \times 10^4}$	Best	0.0454 (0)	4.1633e-17 (0)	9.7979e-05 (0)	6.6643 (0)	0 (0)	0 (0)
	Median	0.0810 (0)	5.5511e-17 (0)	0.0002 (0)	14.5927 (0)	2.6639e-11 (0)	0 (0)
	Worst	0.1195 (0)	5.5511e-17 (0)	0.0004 (0)	79.4762 (0)	0.1004 (0)	0 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	8.0109e-02	4.8850e-17	2.8265e-04	1.7685e+01	4.1403e-03	0.0000
	Std	2.0492e-02	7.0763e-18	9.2069e-05	1.3857e+01	2.0074e-02	0.0000
$\mathbf{5 \times 10^5}$	Best	-1.8829e-13 (0)	4.1633e-17 (0)	1.1368e-13 (0)	-1.8189e-12 (0)	0 (0)	0 (0)
	Median	-1.8829e-13 (0)	4.1633e-17 (0)	2.2737e-13 (0)	-9.0949e-13 (0)	0 (0)	0 (0)
	Worst	-1.8474e-13 (0)	4.1633e-17 (0)	2.2737e-13 (0)	4.5274e-07 (0)	0.0022 (0)	0 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	-1.8701e-13	4.1633e-17	2.0918e-13	2.1288e-08	9.1079e-05	0.0000
	Std	1.7405e-15	1.2580e-32	4.2538e-14	9.1279e-08	4.5540e-04	0.0000

bound values by jDE algorithm [5]. Rönkönen, Kukkonen and Price [18] suggest the solution that volatile boundary constraints should be reflected back from the bound by the amount of violation:

$$u_{i,j,G} = \begin{cases} 2 \cdot x_{j,low} - u_{j,i,g} & \text{if } u_{j,i,g} < x_{j,low}, \\ 2 \cdot x_{j,upp} - u_{j,i,g} & \text{if } u_{j,i,g} > x_{j,upp}. \end{cases}$$

The jDE-2 [4] algorithm uses both solutions for volatile boundary constraints with equal probability in a random manner:

$$t = rand(0,1), \quad p_0 = 0.5,$$

$$u_{i,j,G} = \begin{cases} x_{j,low} & \text{if } (t \le p_0) \wedge (u_{j,i,g} < x_{j,low}), \\ x_{j,upp} & \text{if } (t \le p_0) \wedge (u_{j,i,g} > x_{j,upp}), \\ 2 \cdot x_{j,low} - u_{j,i,g} & \text{if } (t > p_0) \wedge (u_{j,i,g} < x_{j,low}), \\ 2 \cdot x_{j,upp} - u_{j,i,g} & \text{if } (t > p_0) \wedge (u_{j,i,g} > x_{j,upp}). \end{cases}$$

Strategy "rand/1/bin" is used in jDE algorithm and control

TABLE III

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

FES		g13	g14	g15	g16	g17	g18
$\mathbf{5 \times 10^3}$	Best	0.9279 (5)	5.7028 (6)	1.2410 (2)	0.0800 (0)	133.6529 (11)	0.6759 (0)
	Median	0.9472 (6)	4.2986 (6)	0.3196 (4)	0.1233 (0)	103.6969 (12)	0.8548 (4)
	Worst	0.6896 (5)	-10.0665 (6)	0.2469 (4)	0.2275 (0)	77.6164 (12)	1.1177 (13)
	c	0, 3, 3	0, 3, 3	0, 2, 2	0, 0, 0	4, 4, 4	0, 2, 2
	$\overline{v}$	0.1420	0.1655	0.0431	0.0000	2.5360	0.0236
	Mean	1.0686	2.7371	2.2858	1.3856e-01	1.1166e+02	8.3581e-01
	Std	9.4822e-01	5.4970	2.1405	4.3618e-02	7.3462e+01	1.8863e-01
$\mathbf{5 \times 10^4}$	Best	0.8635 (0)	0.2289 (0)	0.0001 (0)	7.6055e-07 (0)	30.9149 (0)	0.0020 (0)
	Median	0.9450 (2)	1.0538 (0)	0.8793 (0)	1.6831e-06 (0)	330.3851 (1)	0.0043 (0)
	Worst	0.9458 (4)	2.1280 (0)	4.3730 (0)	3.6573e-06 (0)	27.2941 (4)	0.0189 (0)
	c	0, 0, 2	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 1	0, 0, 0
	$\overline{v}$	0.0006	0.0000	0.0000	0.0000	0.0002	0.0000
	Mean	9.3799e-01	1.1351	1.2074	1.8758e-06	1.0019e+02	5.2779e-03
	Std	6.8257e-02	5.1352e-01	1.2471	6.9733e-07	8.4818e+01	3.2909e-03
$\mathbf{5 \times 10^5}$	Best	0.2970 (0)	1.4210e-14 (0)	0 (0)	5.1070e-15 (0)	1.2732e-11 (0)	3.3306e-16 (0)
	Median	0.6800 (0)	2.1316e-14 (0)	0 (0)	5.1070e-15 (0)	10.4896 (0)	4.4408e-16 (0)
	Worst	0.9449 (0)	2.1316e-14 (0)	0.1100 (0)	5.1070e-15 (0)	86.7535 (0)	4.4408e-16 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Mean	6.9148e-01	1.8758e-14	4.4040e-03	5.1070e-15	3.9906e+01	4.3077e-16
	Std	2.2376e-01	3.4809e-15	2.2020e-02	0.0000	3.8319e+01	3.6822e-17

TABLE IV

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

FES		g19	g20	g21	g22	g23	g24
$\mathbf{5 \times 10^3}$	Best	197.9016 (0)	3.1215 (38)	396.3713 (8)	8137.0973 (57)	400.0551 (0)	0.0006 (0)
	Median	294.2522 (0)	5.1358 (40)	431.0758 (7)	8756.7611 (55)	-249.9385 (10)	0.0066 (0)
	Worst	537.2860 (0)	8.1603 (42)	317.5561 (9)	19763.5690 (51)	-411.0136 (16)	0.0142 (0)
	c	0, 0, 0	2, 18, 20	0, 3, 4	17, 19, 19	0, 5, 5	0, 0, 0
	$\overline{v}$	0.0000	2.0017	0.2702	2429365.3684	0.3974	0.0000
	Mean	3.1332e+02	4.3551	4.7242e+02	1.3564e+04	4.4412e+01	6.6979e-03
	Std	7.0448e+01	1.3059	1.5794e+02	5.0437e+03	3.4592e+02	3.3507e-03
$\mathbf{5 \times 10^4}$	Best	0.6191 (0)	0.0795 (21)	0.0498 (0)	6300.6825 (58)	223.7870 (0)	5.5067e-14 (0)
	Median	1.1097 (0)	0.0792 (24)	0.0823 (0)	14782.0213 (57)	475.3514 (0)	5.5067e-14 (0)
	Worst	2.6947 (0)	0.0491 (26)	582.6489 (0)	5627.0472 (57)	422.4294 (1)	6.0396e-14 (0)
	c	0, 0, 0	0, 4, 20	0, 0, 0	19, 19, 19	0, 0, 0	0, 0, 0
	$\overline{v}$	0.0000	0.0203	0.0000	30001.6677	0.0000	0.0000
	Mean	1.2044	7.9528e-02	2.8678e+01	1.0550e+04	4.4242e+02	5.5778e-14
	Std	4.4409e-01	1.7970e-02	1.1840e+02	4.4761e+03	1.0704e+02	1.4950e-15
$\mathbf{5 \times 10^5}$	Best	2.8421e-14 (0)	0.0506 (0)	-1.7053e-13 (0)	9151.6956 (8)	0 (0)	5.5067e-14 (0)
	Median	4.2632e-14 (0)	0.1082 (2)	-2.8421e-14 (0)	8033.6537 (8)	2.2737e-13 (0)	5.5067e-14 (0)
	Worst	5.4711e-13 (0)	0.1068 (5)	130.9783 (0)	19337.2039 (16)	300.0085 (0)	5.5067e-14 (0)
	c	0, 0, 0	0, 1, 1	0, 0, 0	2, 3, 3	0, 0, 0	0, 0, 0
	$\overline{v}$	0.0000	0.0072	0.0000	3.5107	0.0000	0.0000
	Mean	7.5033e-14	1.0591e-01	1.0478e+01	9.6221e+03	1.2102e+01	5.5067e-14
	Std	1.0531e-13	1.1510e-02	3.6266e+01	5.1748e+03	5.9983e+01	1.9323e-29

parameters F and CR are encoded in each individual. In [17] authors proposed self-adapting SaDE algorithm which uses two of five original DE's strategies to be applied to individuals in the current population.

Figure 2 shows a new solution how the control parameters of two original DE's strategies are encoded in each individual. Each strategy uses its own control parameters. The solution to apply more strategies into our algorithm is straight-forward.

In experiments in this paper, the proposed jDE-2 algorithm uses three strategies "rand/1/bin", "current to best/1/bin" and "rand2/bin", and the first pair of self-adaptive control parameters F and CR belongs to the "rand/1/bin" strategy and the second pair belongs to "current to best/1/bin" strategy, etc. The population size NP was set to 200.

The jDE-2 algorithm replaces k worst individuals at every l-th generation with parameter values distributed uniformly

TABLE V
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	46559	50354	56968	5.0386e+04	1.9651e+03	100%	100%	50386
g02	101201	138102	173964	1.2349e+05	4.0592e+04	100%	92%	145899
g03						100%	0%	
g04	38288	40958	42880	4.0728e+04	1.2670e+03	100%	100%	40728
g05	133340	328023	482304	2.0662e+05	1.7274e+05	100%	68%	446839
g06	26830	29844	31299	2.9488e+04	1.2772e+03	100%	100%	29488
g07	114899	126637	141847	1.2774e+05	6.4762e+03	100%	100%	127744
g08	1567	3564	4485	3.2364e+03	7.4535e+02	100%	100%	3236
g09	49118	55515	58230	5.4919e+04	2.2551e+03	100%	100%	54919
g10	139095	144247	165498	1.4615e+05	6.6675e+03	100%	100%	146150
g11	17834	36343	432169	4.9700e+04	8.0409e+04	100%	96%	53928
g12	1820	6684	9693	6.3556e+03	2.1567e+03	100%	100%	6356
g13						100%	0%	
g14	88954	98135	107951	9.7845e+04	5.0348e+03	100%	100%	97845
g15	51321	261549	432766	2.2246e+05	1.1398e+05	100%	96%	241383
g16	28230	31753	34182	3.1695e+04	1.3132e+03	100%	100%	31695
g17	449306	449306	449306	1.7971e+04	8.9861e+04	100%	4%	11232650
g18	91049	101076	142674	1.0446e+05	1.2062e+04	100%	100%	104462
g19	170950	197319	234038	1.9985e+05	1.5735e+04	100%	100%	199850
g20						4%	0%	
g21	96552	113883	147030	1.0708e+05	3.4592e+04	100%	92%	126507
g22						0%	0%	
g23	205404	319611	495721	3.0255e+05	1.2447e+05	100%	92%	357452
g24	7587	10354	11550	1.0196e+04	9.2836e+02	100%	100%	10196

randomly between lower and upper bounds without evaluating those k individuals. In this paper we set $l = 1000$ and $k = 70$.

The jDE-2 algorithm emphasizes constrains as follows. It compare two solutions, say i and j, during selection operation (see section III-C):

$$\vec{x}_{i,G+1} = \begin{cases} \vec{x}_{j,G} & \text{if } (\overline{v}_{i,G} > \overline{v}_{j,G}), \\ \vec{x}_{j,G} & \text{else if } (\overline{v}_{j,G} = 0) \wedge (f(\vec{x}_{i,G}) > f(\vec{x}_{j,G})), \\ \vec{x}_{i,G} & \text{otherwise.} \end{cases}$$

The algorithm distinguishes between feasible ($\overline{v} = 0$) and infeasible individuals: any feasible solution is better than any infeasible one.

VI. EXPERIMENTAL RESULTS

The jDE-2 algorithm was tested on 24 CEC2006 special session benchmark functions. The obtained results are presented in Tables I– IV. For 22 functions the jDE-2 algorithm was successfully found feasible solution. For $g20$ function one feasible solution was found, and for $g22$ function any feasible solution was not found.

Table V shows 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 performed by jDE-2 algorithm. Our algorithm has found at least one successful run for 20 benchmark functions. The success rate is more than 90% for 17 benchmark functions. The overall success rate for all 24 benchmark functions was 77%. Our algorithm did not achieve the fixed accuracy for $g03$, $g13$, $g20$, and $g22$ functions. It can be noticed that our algorithms has difficulties to solve problems with (many) equality constraints.

Convergence graphs are presented in Figures 3–6. Algorithm complexity is presented in Table VI.

PC Configure:
System: GNU/Linux CPU: 2.4 GHz RAM: 1 GB
Language: C/C++ Algorithm: jDE-2 – self-adaptive DE

TABLE VI
COMPUTATIONAL COMPLEXITY

$T1$	$T2$	$(T2 - T1)/T1$
0.39 s	1.30 s	2.33

VII. CONCLUSIONS

In this paper the performance of the self-adaptive differential evolution jDE-2 algorithm was evaluated on the set of 24 benchmark functions provided by CEC2006 special session on constrained real parameter optimization.

The best settings for the control parameters highly depends on the benchmark function. A self-adaptive control mechanism is used by jDE-2 algorithm to change the (DE strategy) control parameters F and CR during the run.

The results of this paper give evidence that the jDE-2 algorithm with the self-adaptive F and CR control parameters on

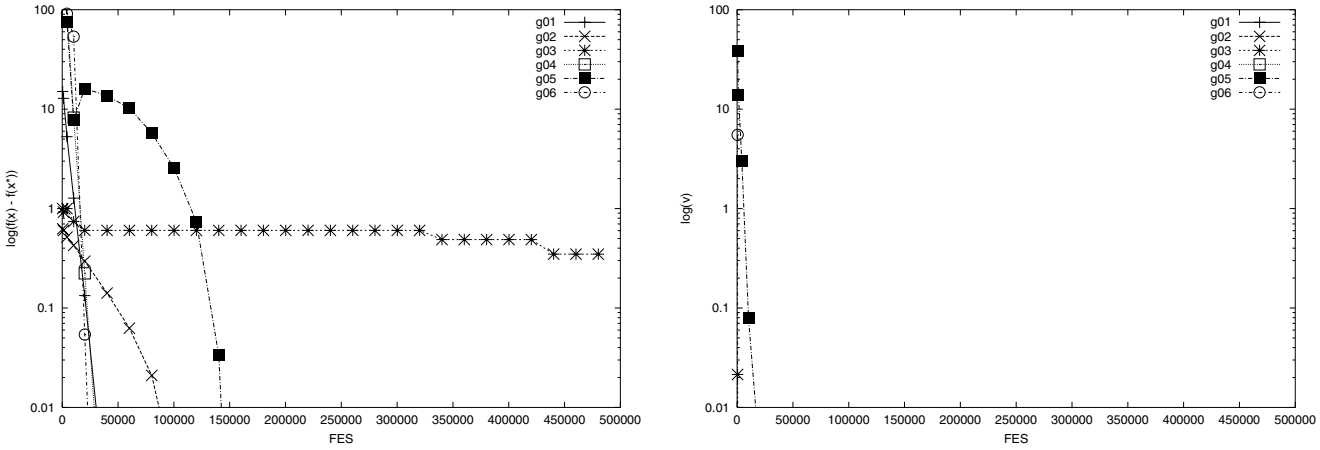

Fig. 3. Convergence Graph for Problems 1-6

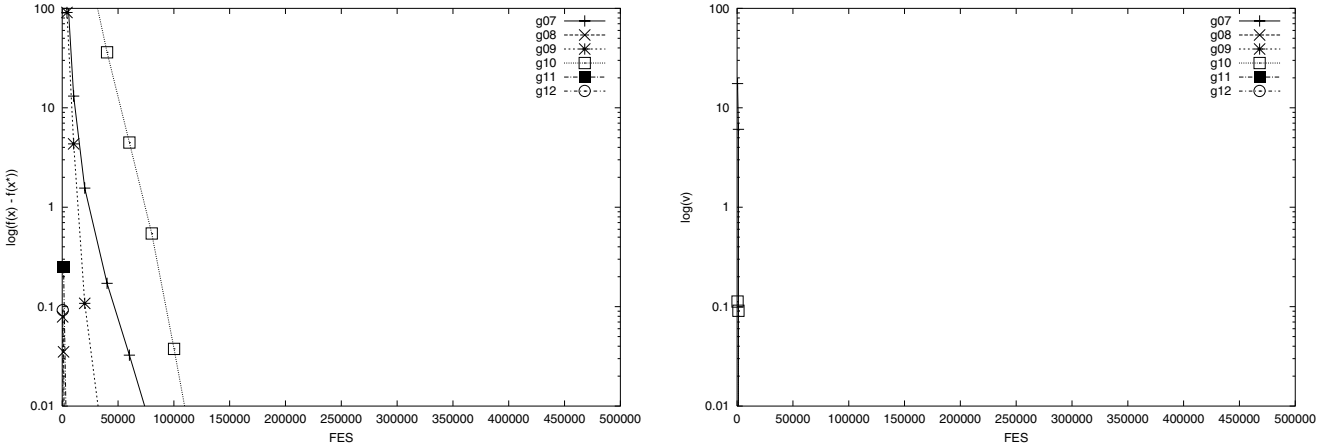

Fig. 4. Convergence Graph for Problems 7-12

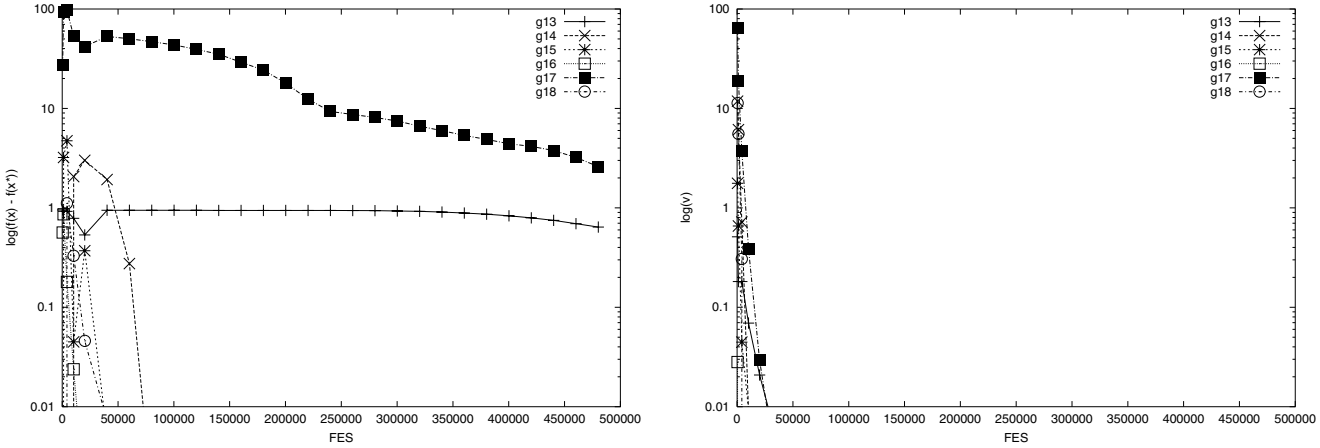

Fig. 5. Convergence Graph for Problems 13-18

three different DE strategies: ”rand/1/bin”, ”rand/2/bin” and ”current to best/1/bin” strategies is competitive algorithm for non-linear, non-separable constrained global optimization.

ACKNOWLEDGMENT

This work was supported in part by the Slovenian Research Agency under programs P2-0041 – Computer Systems, Methodologies, and Intelligent Services, and P2-0069 – Advanced Methods of Interaction in Telecommunications.

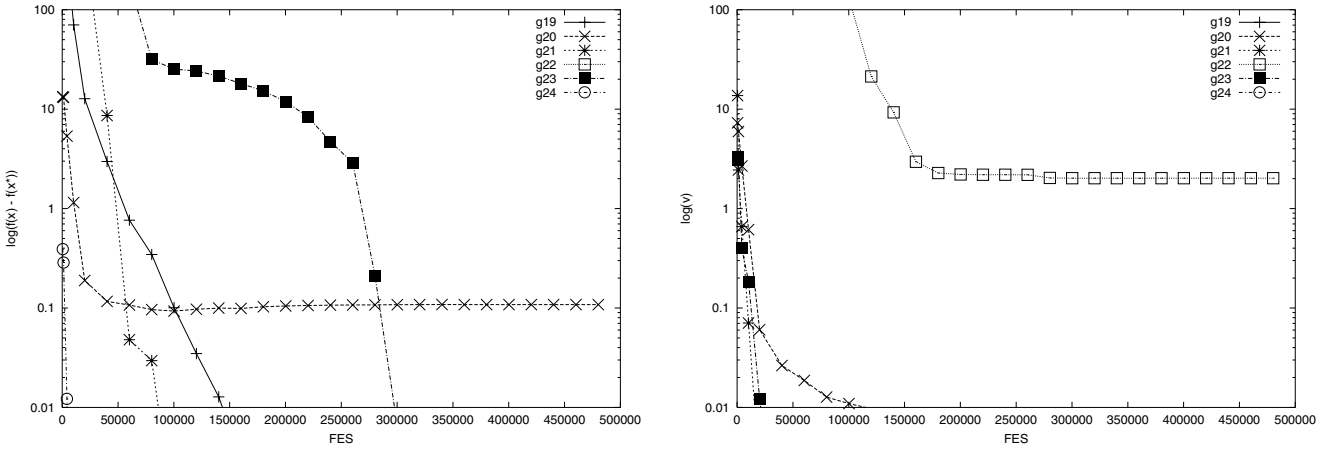

Fig. 6. Convergence Graph for Problems 19-24

The authors would also like to acknowledge the efforts of the organizers of this session.

REFERENCES

[1] M. M. Ali and A. Törn. Population Set-Based Global Optimization Algorithms: Some Modifications and Numerical Studies. *Computers & Operations Research*, 31(10):1703–1725, 2004.

[2] T. Bäck. Adaptive Business Intelligence Based on Evolution Strategies: Some Application Examples of Self-Adaptive Software. *Information Sciences*, 148:113–121, 2002.

[3] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[4] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M. Sepesy Maučec. Performance Comparison of Self-Adaptive and Adaptive Differential Evolution Algorithms. Technical Report #2006-1-LABRAJ, University of Maribor, Faculty of Electrical Engineering and Computer Science, Slovenia, 2006. http://marcel.uni-mb.si/janez/brest-TR1.html.

[5] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on Evolutionary Computation*. Accepted.

[6] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Natural Computing. Springer-Verlag, Berlin, 2003.

[7] R. Gämperle, S. D. Müller, and P. Koumoutsakos. A Parameter Study for Differential Evolution. In *WSEAS NNA-FSFS-EC 2002*, Interlaken, Switzerland, February 11-15 2002. WSEAS.

[8] S. Koziel and Z. Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[9] J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, N. Suganthan, C. A. C. Coello, and K. Deb. Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization. Technical Report Report #2006005, Nanyang Technological University, Singapore and et al., Dec, 2005. http://www.ntu.edu.sg/home/EPNSugan

[10] J. Liu and J. Lampinen. Adaptive Parameter Control of Differential Evolution. In *Proceedings of the 8th International Conference on Soft Computing (MENDEL 2002)*, pages 19–26, 2002.

[11] J. Liu and J. Lampinen. On Setting the Control Parameter of the Differential Evolution Method. In *Proceedings of the 8th International Conference on Soft Computing (MENDEL 2002)*, pages 11–18, 2002.

[12] J. Liu and J. Lampinen. A Fuzzy Adaptive Differential Evolution Algorithm. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9(6):448–462, 2005.

[13] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, Berlin, 2000.

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

[15] K. Price and R. Storn. Differential Evolution: A Simple Evolution Strategy for Fast Optimization. *Dr. Dobb's Journal of Software Tools*, 22(4):18–24, April 1997.

[16] K. V. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution, A Practical Approach to Global Optimization*. Springer, 2005.

[17] A. K. Qin and P. N. Suganthan. Self-adaptive Differential Evolution Algorithm for Numerical Optimization. In *The 2005 IEEE Congress on Evolutionary Computation CEC2005*, volume 2, pages 1785–1791. IEEE Press, Sept. 2005. DOI: 10.1109/CEC.2005.1554904.

[18] J. Rönkkönen, S. Kukkonen, and K. V. Price. Real-Parameter Optimization with Differential Evolution. In *The 2005 IEEE Congress on Evolutionary Computation CEC2005*, volume 1, pages 506 – 513. IEEE Press, Sept. 2005.

[19] R. Storn and K. Price. Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, Berkeley, CA, 1995.

[20] R. Storn and K. Price. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[21] J. Sun, Q. Zhang, and E. P. K. Tsang. DE/EDA: A New Evolutionary Algorithm for Global Optimization. *Information Sciences*, 169:249–262, 2004.

[22] J. Teo. Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 2005. DOI: 10.1007/s00500-005-0537-1.

[23] R. K. Ursem and P. Vadstrup. Parameter identification of induction motors using differential evolution. In Ruhul Sarker, Robert Reynolds, Hussein Abbass, Kay Chen Tan, Bob McKay, Daryl Essam, and Tom Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 790–796, Canberra, 8-12 December 2003. IEEE Press.