An Improved Self-adaptive Differential Evolution Algorithm in Single Objective Constrained Real-Parameter Optimization

Janez Brest, *Member IEEE*, Borko Bošković, *Student Member IEEE*, Viljem Žumer, *Member IEEE*,

***Abstract*— In this paper we describe an improved version of self-adaptive differential evolution algorithm. Our algorithm uses more strategies, ageing mechanism to reinitialize an individual which stagnates in local optima, an ϵ level controlling of constraint violation. The performance of the proposed algorithm is evaluated on the set of 18 scalable benchmark functions provided for the CEC 2010 competition and special session on single objective constrained real-parameter optimization, when the dimension of decision variables is set to 10 and 30, respectively. The obtained results show that our algorithm is suitable for solving constrained optimization problems.**

I. Introduction

THE optimization problems with constraints arise in many real world problems. An optimization algorithm needs to find $\vec{x}$ which minimize (maximize) function $f(\vec{x})$ where $\vec{x} = (x_1, x_2, ..., x_D)$, and D denotes the dimensionality of the function. Domains of the variables are defined by their lower and upper bounds: $x_{j,low}, x_{j,upp};\ j = 1, 2, ..., D$. The feasible region is defined by a set of m additional constraints ($m \geq 0$):

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

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

Differential Evolution (DE) [25] is a floating-point encoding evolutionary algorithm for global optimization over continuous spaces. DE has some good properties, efficiency and robustness, and it is simple yet powerful evolutionary algorithm. The original DE algorithm has three control parameters, which are being fixed during the optimization process. The best settings for the control parameters depend on the function and requirements for consumption time and accuracy.

Many times, users are still faced with the problem of preliminary testing and hand-tuning of the evolutionary parameters prior to commencing the actual optimization process [28]. Recently, there were many attempts to make the parameterless DE algorithm or to apply some self-adaptive or other techniques that reduce the number of parameters. In literature, a self-adaptation is usually applied to the F and CR control parameters [20], [21], [4], [29], and rarely to the NP parameter [28], [18].

In this paper we present a performance evaluation of our self-adaptive *jDEsoco* algorithm, which uses self-adapting control parameter mechanism on the control parameters F and CR, ϵ level for handling constraint violation, ageing mechanism and more DE strategies.

The performance of the algorithm is evaluated on the set of benchmark functions provided for the CEC 2010 special session on single objective constrained real-parameter optimization [11].

The article is structured as follows. Section II gives an overview of work dealt with the DE. Section III-A gives background for this work. We shortly summarize the differential evolution, describe constraint handling, and outline self-adaptive jDE algorithm. Section IV presents our self-adaptive DE algorithm (*jDEsoco*), which is based on the ϵ-jDE algorithm for constrained optimization. In Section V experimental results of our algorithm on CEC 2010 benchmark functions are presented. Section VI concludes the paper with some final remarks.

II. Work Related with the Differential Evolution

The DE [25], [24] algorithm was proposed by Storn and Price, and since then the DE algorithm has been used in different engineering areas. The original DE was modified, and many new versions were proposed.

Qin and Suganthan in [21] 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.

An empirical comparison of some Differential Evolution variants to solve global optimization problems is presented by E. Mezura Montes et al. [15]. Differential evolution and its modified versions were widely used for constrained optimization [23], [13], [9], [27], [1], [6], [16]. In [14] a comparison of four bio-inspired algorithms with a similar constraint-handling technique (Deb's feasibility rules) on a set of 24 benchmark functions is presented.

J. Brest et al. [4] proposed a self-adaptive jDE algorithm. The jDE algorithm was adopted to work for constrained optimization problems [6], [2]. It was used in algorithms for unconstrained optimization [4], [3], [5]. An another version of the jDE algorithm with population size reduction was used for large scale global optimization [7]. Yet another version developed based on the jDE algorithm was used for dynamic optimization problems [8].

During the last few years several methods for handling constraints were proposed [17], [10], [12]. Takahama and Sakai in [27] pointed out that, for problems with equality

Janez Brest, Borko Bošković, Viljem Žumer are with the Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova ul. 17, 2000 Maribor, Slovenia, (email:janez.brest@uni-mb.si).

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

constraints, the ϵ level should be controlled properly to obtain high quality solutions. The ϵ constrained method is used together with the differential evolution in [26], and also in our previous work [2] where ϵ-jDE algorithm was presented.

III. Background

A. The Differential Evolution

DE creates new candidate solutions by combining the parent individual and several other individuals of the same population. A candidate individual replaces the parent only if it has better fitness value.

The population of the original DE algorithm [24], [25], [22] contains NP D-dimensional vectors:

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

where 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 uniform 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.

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

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. $\vec{x}_{best,G}$ denotes the best vector in generation G.

2) 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.

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

$$\text{for } i = 1, 2, ..., NP \text{ and } j = 1, 2, ..., D.$$

CR denotes 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 [25], [24], [22], [19] are as follows: they are reflected into bounds, set on bounds, or used as they are (set out of bounds).

In this work, we use the following solution for volatile boundary constraints (when $F \le 1$):

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

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

3) Selection operation: The selection operator selects individual to survive according to the fitness value between the population vector and its corresponding trial vector. However, the fittest vector will survive and become the member of the next generation. For example, if we have a minimization problem, we will use the following selection rule:

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

B. Constraints Handling

A measure to estimate the constraint violation is often useful when handling constraints. A solution $\vec{x}$ is regarded as *feasible* if the following inequalities are satisfied:

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

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

where equality constraints are transformed into inequalities. In CEC 2010 [11] special section ϵ is set to 0.0001. Mean violations $\overline{v}$ is defined as follows:

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

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

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

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.

C. The Self-adaptive jDE Algorithm

Let us describe jDE algorithm [4]. It uses self-adapting mechanism on the control parameters F and CR, and the "*rand/1/bin*" strategy.

Each individual in population is extended with parameter values F and CR (see Figure 1). 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.

Fig. 1. Self-adaptation of i-th individual.

New control parameters $F_{i,G+1}$ and $CR_{i,G+1}$ are calculated before the mutation is performed as follows:

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

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

where $rand_j, j = 1, ..., 4$ are uniform random values within the range $[0, 1]$. τ_1 and τ_2 represent probabilities to adjust control parameters F and CR, respectively. Parameters τ_1, τ_2, F_l, F_u were fixed to 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 $[0, 1]$.

Note that $F_{i,G+1}$ 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 $\vec{x}_{i,G+1}$.

IV. Improved Self-adaptive DE Algorithm

In this section we present an improved version of jDE algorithm (named *jDEsoco*) for solving constrained optimization problems. The *jDEsoco* algorithm uses ϵ level for controlling constraint violations (it was used in ϵ-jDE algorithm), ageing mechanism, and more DE strategies. It is described more detailed in the rest of this section.

A. The ϵ-jDE Algorithm

Recently, we proposed *ϵ-jDE* algorithm [2] that uses a self-adaptive mechanism on the control parameters F and CR, more strategies during mutation operation, the ϵ-constrained method, and population size reduction during the evolutionary process.

1) Controlling the ϵ Level: Let us explain present ϵ level controlling in our *ϵ-jDE* algorithm [2]. In the proposed method ϵ level constraint violation precedes the objective function. A method of controlling the ϵ level is defined according to equations (1)–(4). The ϵ level is updated until the number of generations G reaches the control generation G_c. After the number of generations exceeds G_c, the ϵ level is set to zero to obtain solutions with minimum constraint violation as follows:

$$\epsilon_0 = \epsilon, \tag{1}$$

$$v_0 = \overline{v}(\vec{x}_\theta), \tag{2}$$

$$\overline{v}_G = \begin{cases} \alpha_1 v_{G-1}, & \alpha_2 \overline{v}(\vec{x}_\beta) < v_{G-1},\ \epsilon_{G-1} < \epsilon, \\ & 0 < G < G_c, \\ v_{G-1}, & \text{otherwise,} \end{cases} \tag{3}$$

$$\epsilon_G = \begin{cases} \max\{v_G(1 - \frac{G}{G_c})^{c_p}, \epsilon\}, & 0 < G < G_c, \\ 0, & G \geq G_c, \end{cases} \tag{4}$$

where $\vec{x}_\theta$ is the top θ-th individual and $\theta = 0.3NP$. Note that $\epsilon(0) = 0$ when mean violation $\overline{v}(\vec{x}_\theta)$ is calculated. Similarly, $\vec{x}_\beta$ is the top β-th individual and $\beta = 0.7NP$. The parameter c_p controls the speed of constraints' reducing relaxation, while parameters $\alpha_1 < 1$ and $\alpha_2 > 1$ are used to adaptively control v_G value, which also controls the speed of the constraints' reducing relaxation. Parameters α_1 and α_2 can only decrease the v_G value by a small amount when top β individuals have mean violation $\overline{v}(\vec{x}_\beta)$ multiplied by α_1 less than v_G. Using this adaptation, the ϵ level could reach zero before $G \geq G_c$.

In this study we use $c_p = 5$, $\alpha_1 = 0.8$, $\alpha_2 = 2.0$, and G_c=0.35G_{max}.

B. Constraints Handling and Cost Function

In this work we divide the population into two parts and our algorithm uses two rules. For individuals that belong to first part, our algorithm distinguishes between feasible and infeasible solutions, and any feasible solution is better than any infeasible one.

Our algorithm compares individuals that belong to second part only based on the function $f(\vec{x})$, i.e. it does not distinguish between feasible and infeasible solutions.

The size of first part is $0.6NP$.

C. The Ageing Mechanism

We have employed ageing at individual level. An individual which stagnates in local minimum should be re-initialized in some way. We used the aging approach for an individual, say $\vec{x}_i$, as presented in Algorithm 1.

1: $ageFactor = 30 * D$; {D is dimension}
2: $probAge = 0.05$; {5 %}
3: **if** $age > ageFactor$ **and** $rand() < probAge$ **then**
4: re-initialize individual $\vec{x}_i$
5: **end if**

Algorithm 1: Aging of the i-th individual

D. DE Strategies

Fig. 2. Encoding aspect of two strategies.

TABLE I

FUNCTION VALUES ACHIEVED WHEN FES= 2×10^4, FES= 1×10^5, FES= 2×10^5 FOR 10D PROBLEMS C01–C06.

FES		C01	C02	C03	C04	C05	C06
$\mathbf{2 \times 10^4}$	Best	-0.7468925 (0)	0.5832421 (0)	9.3091684e-05 (0)	-0.5229745 (4)	-482.8116159 (2)	-373.2061820 (2)
	Median	-0.7391804 (0)	2.6543062 (1)	9.7461646 (1)	-1.3887673 (4)	-460.4315889 (2)	-564.0149713 (2)
	Worst	-0.6752158 (0)	3.5381246 (1)	172.8060433 (1)	-1.4605776 (4)	-385.6906053 (2)	-303.1469294 (2)
	c	0, 0, 0	0, 0, 1	0, 1, 1	4, 4, 4	0, 1, 2	1, 2, 2
	$\overline{v}$	0.0000	0.0002	0.0188	23.5108	0.1373	1.3428
	Mean	-7.3473e-01	1.7702	1.6990e+01	-1.2975	-3.5680e+02	-2.9604e+02
	Std	1.5831e-02	1.3264	3.3164e+01	2.4656e-01	2.0564e+02	3.0071e+02
$\mathbf{1 \times 10^5}$	Best	-0.7473102 (0)	-0.9260471 (0)	0 (0)	-9.9999999e-06 (0)	-483.6106247 (0)	-578.6623422 (0)
	Median	-0.7405571 (0)	0.5832421 (0)	8.8755525 (0)	-9.9999999e-06 (0)	-483.6106247 (0)	-577.1758986 (0)
	Worst	-0.6782502 (0)	0.9905283 (1)	8.8755538 (0)	-9.9230054e-06 (0)	-0.6432138 (1)	-495.1183588 (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	-7.3835e-01	4.8340e-01	7.8105	-9.9969e-06	-3.0206e+02	-5.7408e+02
	Std	1.6006e-02	1.1034	2.9437	1.5399e-08	3.0250e+02	1.6461e+01
$\mathbf{2 \times 10^5}$	Best	-0.7473103 (0)	-1.1176240 (0)	0 (0)	-9.9999999e-06 (0)	-483.6106247 (0)	-578.6623660 (0)
	Median	-0.7405572 (0)	0.4970970 (0)	8.8755523 (0)	-9.9999999e-06 (0)	-483.6106247 (0)	-577.1759095 (0)
	Worst	-0.6782507 (0)	0.5735050 (1)	8.8755524 (0)	-9.9999999e-06 (0)	-0.6432138 (1)	-495.1183588 (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	-7.3836e-01	5.6359e-01	7.8105	-1.0000e-05	-3.0217e+02	-5.7408e+02
	Std	1.6006e-02	1.1044	2.9437	9.4831e-16	3.0256e+02	1.6461e+01

TABLE II

FUNCTION VALUES ACHIEVED WHEN FES= 2×10^4, FES= 1×10^5, FES= 2×10^5 FOR 10D PROBLEMS C07–C12.

FES		C07	C08	C09	C10	C11	C12
$\mathbf{2 \times 10^4}$	Best	0.0871921 (0)	0.0001106 (0)	9.6478355e-06 (0)	6.9338047e-07 (0)	-0.1538870 (1)	-124.8691803 (1)
	Median	2.8812390 (0)	4.8946589 (0)	19.2972991 (1)	50.4545275 (1)	-1.5141397 (1)	-30.3698920 (1)
	Worst	19.2715240 (0)	108.4851765 (0)	1732.5702677 (1)	185.7632140 (1)	-9.8220781 (1)	-7.6686510 (1)
	c	0, 0, 0	0, 0, 0	0, 0, 1	0, 0, 1	1, 1, 1	1, 1, 1
	$\overline{v}$	0.0000	0.0000	0.0027	0.0028	95833439.1417	19.0026
	Mean	4.1778	1.5289e+01	8.2347e+05	2.2309e+06	-3.4954	-8.6459e+01
	Std	4.5773	2.8648e+01	3.6289e+06	1.1033e+07	6.1131	9.7739e+01
$\mathbf{1 \times 10^5}$	Best	0 (0)	0 (0)	0 (0)	0 (0)	0.0201933 (1)	-0.1992457 (0)
	Median	2.1867224e-27 (0)	5.3800313e-27 (0)	1.9109366e-26 (0)	41.7263056 (0)	-0.0159706 (1)	-0.1992447 (0)
	Worst	0.0497082 (0)	50.8450403 (0)	4.4081527 (0)	41.7293247 (0)	1.5586648 (1)	-554.3465763 (1)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	1, 1, 1	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0000	1.3618	0.0000
	Mean	1.9883e-03	3.7421	5.2898e-01	3.1713e+01	5.2810e-02	-2.6778e+01
	Std	9.9416e-03	1.0330e+01	1.4620	1.8188e+01	3.7105e-01	1.1210e+02
$\mathbf{2 \times 10^5}$	Best	0 (0)	0 (0)	0 (0)	0 (0)	-0.0015227 (0)	-0.1992457 (0)
	Median	0 (0)	3.8180867e-28 (0)	0 (0)	41.7259350 (0)	-0.0015227 (0)	-0.1992457 (0)
	Worst	6.4120192e-26 (0)	50.8450399 (0)	4.4081527 (0)	41.7270003 (0)	-0.0873415 (1)	-554.3465666 (1)
	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	6.4192e-27	3.7421	5.2898e-01	3.1712e+01	-8.2555e-03	-2.2365e+01
	Std	1.3515e-26	1.0330e+01	1.4620	1.8188e+01	2.3807e-02	1.1083e+02

Our algorithm uses two DE strategies: "rand/1/bin" and "best/1/bin". The first one is used with probability 0.9 and the second one with probability 0.1. Figure 2 shows how the control parameters of two DE's strategies are encoded in each individual. Each strategy uses its own control parameters. The first pair of self-adaptive control parameters F and CR belongs to the "rand/1/bin" strategy and the second pair belongs to "best/1/bin" strategy. The solution to apply more strategies into our algorithm is straight-forward.

V. EXPERIMENTAL RESULTS

Our algorithm was tested on CEC 2010 competition and special session benchmark functions. There are 18 scalable benchmark functions. For each function 25 runs are

TABLE III

FUNCTION VALUES ACHIEVED WHEN FES= 2×10^4, FES= 1×10^5, FES= 2×10^5 FOR 10D PROBLEMS C13–C18.

FES		C13	C14	C15	C16	C17	C18
$\mathbf{2 \times 10^4}$	Best	-67.9727786 (0)	1905017636.25639 (0)	4334375355.10851 (0)	1.0093354 (0)	364.7105108 (0)	2709.7126152 (0)
	Median	-66.8789736 (0)	56105341415.3025 (0)	5751015632961.47 (0)	0.9662981 (2)	424.7978256 (1)	10483.7769622 (2)
	Worst	-59.7393891 (0)	4172175940420.96 (0)	258479038630832 (1)	0.9530591 (2)	24.1015801 (1)	10951.2389597 (2)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 2	0, 0, 1	0, 0, 2
	$\overline{v}$	0.0000	0.0000	0.0000	0.0011	0.0003	0.0002
	Mean	-6.5978e+01	3.4041e+11	3.6735e+13	9.5556e-01	3.1450e+02	9.2011e+03
	Std	2.3656	8.8330e+11	6.4562e+13	1.6400e-01	1.9462e+02	5.3339e+03
$\mathbf{1 \times 10^5}$	Best	-68.4293614 (0)	0 (0)	1702.5964076 (0)	0.0001545 (0)	0.1330508 (0)	12.3580646 (0)
	Median	-68.4293047 (0)	1.2904817e-25 (0)	1170608020.90008 (0)	0.4476178 (0)	52.5796188 (0)	269.3970346 (0)
	Worst	-65.5784660 (0)	10.8455682 (0)	1811707625336.46 (0)	1.0869632 (0)	342.7806836 (0)	8342.3689004 (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	-6.8290e+01	9.1221e-01	2.6030e+11	5.0256e-01	9.5000e+01	9.4653e+02
	Std	5.6774e-01	2.4538	5.4047e+11	3.4361e-01	1.0012e+02	1.7902e+03
$\mathbf{2 \times 10^5}$	Best	-68.4293648 (0)	0 (0)	2.0257948e-26 (0)	0 (0)	0.0302187 (0)	0.1100901 (0)
	Median	-68.4293632 (0)	4.8877821e-27 (0)	302530212.313899 (0)	0.2808222 (0)	46.8450211 (0)	123.0249839 (0)
	Worst	-65.5784661 (0)	10.8455658 (0)	19317505327.3236 (0)	1.0450375 (0)	330.0330126 (0)	2988.5346337 (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	-6.8315e+01	9.1221e-01	1.2452e+09	4.1102e-01	8.8958e+01	4.0500e+02
	Std	5.7018e-01	2.4538	3.8127e+09	3.8359e-01	9.9131e+01	7.3762e+02

TABLE IV

FUNCTION VALUES ACHIEVED WHEN FES= 6×10^4, FES= 3×10^5, FES= 6×10^5 FOR 30D PROBLEMS C01–C06.

FES		C01	C02	C03	C04	C05	C06
$\mathbf{6 \times 10^4}$	Best	-0.8150998 (0)	1.3223959 (0)	29.1035614 (1)	-0.7858901 (4)	8.0807010 (2)	-495.2683938 (2)
	Median	-0.7981644 (0)	1.7810502 (0)	23.6969263 (1)	-1.2854148 (4)	-56.0783684 (2)	-494.8584840 (2)
	Worst	-0.7176261 (0)	3.2617359 (1)	173.7888321 (1)	-1.4713261 (4)	-87.2234402 (2)	-60.4526236 (2)
	c	0, 0, 0	0, 0, 0	0, 1, 1	4, 4, 4	0, 2, 2	0, 2, 2
	$\overline{v}$	0.0000	0.0000	0.1078	80.3123	0.1837	0.1099
	Mean	-7.9351e-01	3.3406	4.8281e+01	-1.2094	-3.4576e+01	-4.4115e+02
	Std	2.0258e-02	1.2769	3.6145e+01	1.8590e-01	1.7600e+02	1.3400e+02
$\mathbf{3 \times 10^5}$	Best	-0.8218814 (0)	0.7835756 (0)	2.0376818e-18 (0)	0.0004562 (0)	-19.9503700 (0)	-530.6370096 (0)
	Median	-0.8179393 (0)	1.5939691 (0)	30.8354889 (0)	0.0013189 (0)	97.5787730 (0)	-520.8834991 (0)
	Worst	-0.7229028 (0)	3.9787879 (1)	180.5699314 (1)	0.0166396 (0)	34.5718961 (1)	-521.4205144 (1)
	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	-8.0987e-01	1.8649	6.8094e+01	2.0914e-03	1.2705e+02	-4.7271e+02
	Std	2.1696e-02	1.0189	6.2353e+01	3.1216e-03	1.4694e+02	1.2737e+02
$\mathbf{6 \times 10^5}$	Best	-0.8218841 (0)	0.6792295 (0)	9.9936848e-22 (0)	8.0490978e-05 (0)	-19.9503700 (0)	-530.6377271 (0)
	Median	-0.8179473 (0)	1.2122034 (0)	28.6734661 (0)	0.0003091 (0)	87.6076496 (0)	-522.1069634 (0)
	Worst	-0.7707666 (0)	3.9787879 (0)	187.6674137 (0)	0.0009947 (0)	-5.9921895 (2)	-521.4205144 (1)
	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	-8.1238e-01	1.5603	6.1447e+01	3.5187e-04	1.0822e+02	-4.7284e+02
	Std	1.3187e-02	8.4705e-01	5.7577e+01	2.3948e-04	1.5203e+02	1.2743e+02

performed. Maximum function evaluations (Max_FES) is 200000 for 10D and 600000 for 30D, respectively.

In experiments in this paper, the population size NP was set to 25.

The obtained results are presented in Tables I–VI. For each function, we present the following: best, median, worst result, mean value and standard deviation for the 25 runs. We indicate the number of violated constraints (including the number of violations by more than 1, 0.01, and 0.0001) and the mean violation $\overline{v}$ at the median solution.

Our algorithm has successfully found feasible solutions for all functions and all dimensions with high rate. The feasible rates are shown in Table VII. Moreover, it has found feasible solutions when considering median solutions for all functions and all dimensions (indicators in row c are zeros in Tables I–

TABLE V

FUNCTION VALUES ACHIEVED WHEN FES= 6×10^4, FES= 3×10^5, FES= 6×10^5 FOR 30D PROBLEMS C07–C12.

FES		C07	C08	C09	C10	C11	C12
$\mathbf{6 \times 10^4}$	Best	12.5043316 (0)	20.5988914 (0)	7313878.1846808 (1)	21.6302152 (0)	-0.2355759 (1)	-68.6236151 (1)
	Median	20.7869722 (0)	25.9040863 (0)	6653594811780.43 (1)	254.1883566 (1)	-1.2405235 (1)	-154.7540169 (1)
	Worst	89.8492665 (0)	1225.6762535 (0)	52434271105.7283 (1)	520.5235365 (1)	-14.5780300 (1)	-1055.8611088 (1)
	c	0, 0, 0	0, 0, 0	0, 0, 1	0, 0, 1	1, 1, 1	1, 1, 1
	$\overline{v}$	0.0000	0.0000	0.0083	0.0004	2406139818.0931	501657.9978
	Mean	4.1209e+01	1.4347e+02	2.6847e+11	1.1069e+04	-3.3566	-4.3185e+02
	Std	3.0754e+01	2.4919e+02	1.3303e+12	4.3732e+04	3.6283	4.7416e+02
$\mathbf{3 \times 10^5}$	Best	1.7043344e-24 (0)	2.2289191e-21 (0)	3.0673184e-25 (0)	7.8803400e-12 (0)	0.0057009 (1)	-0.1992608 (0)
	Median	1.4405621e-19 (0)	2.1107277e-15 (0)	4.2475086e-17 (0)	31.3119658 (0)	0.0530064 (1)	-0.1992508 (0)
	Worst	2.5592577e-15 (0)	1220.7890506 (0)	66.8406582 (0)	38.0653708 (0)	0.0195651 (1)	-0.1990736 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	1, 1, 1	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0000	12.2467	0.0000
	Mean	2.4521e-16	8.2750e+01	6.8509	2.9611e+01	3.8181e-02	-1.9923e-01
	Std	7.0120e-16	2.4389e+02	1.8214e+01	7.3589	1.5717e-02	4.4046e-05
$\mathbf{6 \times 10^5}$	Best	4.2197747e-26 (0)	7.2310934e-26 (0)	2.7729234e-25 (0)	1.0862047e-25 (0)	-0.0003920 (0)	-0.1992634 (0)
	Median	5.8710603e-25 (0)	7.8435445e-24 (0)	3.6461577e-24 (0)	31.3092214 (0)	-0.0003848 (0)	-0.1992625 (0)
	Worst	1.6394712e-22 (0)	1220.7836239 (0)	40.6067288 (0)	33.1279564 (0)	0.0186710 (1)	-0.1991485 (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	8.7396e-24	8.2585e+01	2.4743	2.9386e+01	1.1667e-03	-1.9925e-01
	Std	3.2529e-23	2.4395e+02	8.7782	7.1786	5.2690e-03	2.3453e-05

TABLE VI

FUNCTION VALUES ACHIEVED WHEN FES= 6×10^4, FES= 3×10^5, FES= 6×10^5 FOR 30D PROBLEMS C13–C18.

FES		C13	C14	C15	C16	C17	C18
$\mathbf{6 \times 10^4}$	Best	-65.1142093 (0)	1584256489421.88 (0)	3995811859.22798 (0)	0.9693954 (0)	508.5798429 (0)	12140.8122594 (0)
	Median	-61.8777493 (0)	9735145049639.63 (0)	41595709132.5943 (0)	1.0270297 (2)	1082.7562735 (1)	79622.6234728 (0)
	Worst	-59.4938936 (0)	32776484335256.8 (0)	23328204645602.2 (0)	0.9860408 (2)	1410.0535824 (1)	28801.9444267 (1)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 2	0, 0, 1	0, 0, 0
	$\overline{v}$	0.0000	0.0000	0.0000	0.0001	0.0003	0.0000
	Mean	-6.2281e+01	1.0746e+13	2.1610e+12	9.7747e-01	1.1351e+03	3.2557e+04
	Std	1.4488	8.1226e+12	5.2254e+12	1.3056e-01	4.2896e+02	1.5523e+04
$\mathbf{3 \times 10^5}$	Best	-67.6195235 (0)	3.7534950e-14 (0)	64604920.3715276 (0)	0.1789938 (0)	33.6693190 (0)	65.2930452 (0)
	Median	-66.4779470 (0)	5.9436438e-07 (0)	1165718687.10799 (0)	0.9743901 (0)	309.8412967 (0)	587.8479458 (0)
	Worst	-63.4889314 (0)	3.9884818 (0)	5247993389.99072 (0)	1.0656151 (0)	1273.2973876 (1)	5931.9538673 (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	-6.6341e+01	1.5960e-01	1.6687e+09	8.8118e-01	5.2065e+02	8.2488e+02
	Std	9.3834e-01	7.9768e-01	1.5124e+09	2.1645e-01	4.4229e+02	1.1330e+03
$\mathbf{6 \times 10^5}$	Best	-68.4293209 (0)	5.7101696e-26 (0)	9.6993452e-16 (0)	0.0812907 (0)	29.0137265 (0)	17.5655328 (0)
	Median	-67.5687202 (0)	5.4639661e-25 (0)	982933896.233957 (0)	0.7914845 (0)	279.2021451 (0)	260.3368217 (0)
	Worst	-66.2678951 (0)	3.9866238 (0)	5247993389.99072 (0)	1.0341961 (0)	1273.2973876 (1)	1264.8547992 (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	-6.7537e+01	1.5946e-01	1.5357e+09	7.3206e-01	5.0398e+02	3.0849e+02
	Std	5.0553e-01	7.9732e-01	1.6045e+09	2.9943e-01	4.4832e+02	3.0538e+02

VI).

Convergence graphs of selected functions (as required by [11]) are presented in Figures 3 and 4. The convergence plots show only feasible solutions of the best run.

Algorithm complexity is presented in Table VIII as required by [11].

PC Configure:
System: GNU/Linux CPU: 2.5 GHz RAM: 4 GB
Language: C/C++ Algorithm: self-adaptive DE (*jDEsoco*)

VI. CONCLUSIONS

In this paper the performance of the self-adaptive differential evolution algorithm (*jDEsoco*) was evaluated on the set

TABLE VII

FEASIBILITY RATE FOR 10D AND 30D.

	10D	**30D**
C01	100%	100%
C02	88%	100%
C03	100%	100%
C04	100%	100%
C05	88%	88%
C06	100%	88%
C07	100%	100%
C08	100%	100%
C09	100%	100%
C10	100%	100%
C11	92%	88%
C12	96%	100%
C13	100%	100%
C14	100%	100%
C15	100%	100%
C16	100%	100%
C17	100%	92%
C18	100%	100%

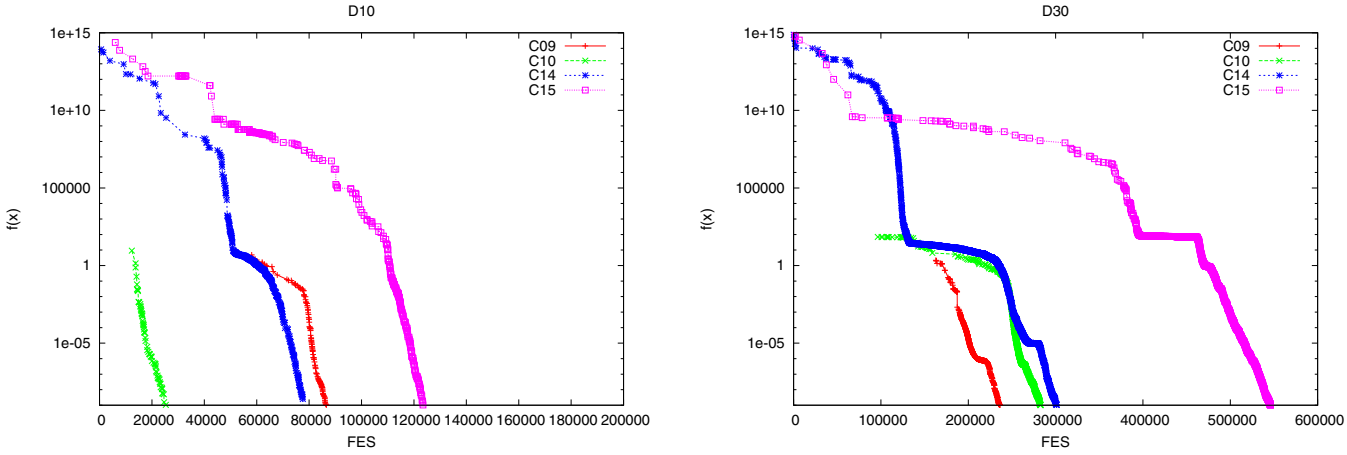

Fig. 3. Convergence Graph for Problems C09, C10, C14, C15.

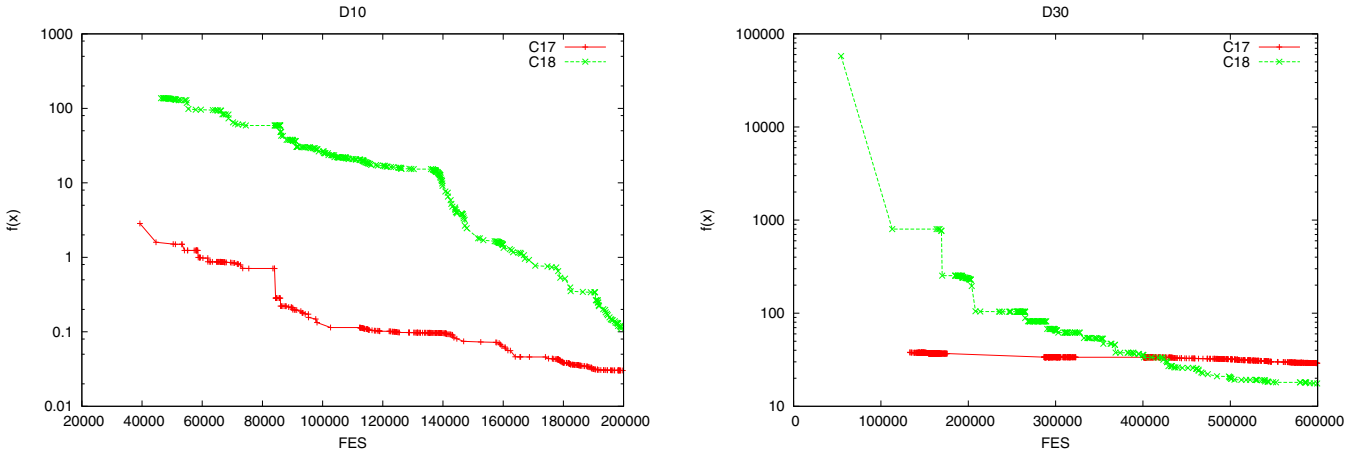

Fig. 4. Convergence Graph for Problems C17, C18.

TABLE VIII

COMPUTATIONAL COMPLEXITY

D	$T1$	$T2$	$(T2-T1)/T1$
10	0.0175 s	0.0252 s	0.44
30	0.0504 s	0.0625 s	0.24

of 18 scalable benchmark functions provided for CEC 2010 special session on single objective constrained real-parameter optimization. Our algorithm uses ϵ level controlling of constraint violation at the beginning of the evolutionary process, ageing mechanism for reinitialization an individual which stagnates in local optima, and two DE strategies.

Based on the obtained results on 10 and 30 dimension benchmark problems we can conclude that our algorithm is suitable for solving constrained optimization problems.

ACKNOWLEDGMENT

This work was supported in part by the Slovenian Research Agency under program P2-0041 – Computer Systems, Methodologies, and Intelligent Services.

REFERENCES

[1] Ricardo Landa Becerra and Carlos A. Coello Coello. Cultured differential evolution for constrained optimization. *Computer Methods in Applied Mechanics and Engineering*, 195(33-36):4303–4322, Jul 2006. DOI: 10.1016/j.cma.2005.09.006.

[2] J. Brest. Constrained Real-Parameter Optimization with ϵ-Self-Adaptive Differential Evolution. *Studies in Computational Intelligence, ISBN: 978-3-642-00618-0*, 198:73–93, 2009.

[3] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M. Sepesy Maučec. Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 11(7):617–629, 2007.

[4] 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*, 10(6):646–657, 2006.

[5] J. Brest and M. Sepesy Maučec. Population Size Reduction for the Differential Evolution Algorithm. *Applied Intelligence*, 29(3):228–247, 2008.

[6] J. Brest, V. Žumer, and M. Sepesy Maučec. Self-adaptive Differential Evolution Algorithm in Constrained Real-Parameter Optimization. In *The 2006 IEEE Congress on Evolutionary Computation CEC2006*, pages 919–926. IEEE Press, 2006.

[7] J. Brest, A. Zamuda, B. Bošković, M. S. Maučec, and V. Žumer. High-dimensional Real-parameter Optimization Using Self-adaptive Differential Evolution Algorithm with Population Size Reduction. In *2008 IEEE World Congress on Computational Intelligence*, pages 2032–2039. IEEE Press, 2008.

[8] J. Brest, A. Zamuda, B. Bošković, M. S. Maučec, and V. Žumer. Dynamic Optimization using Self-Adaptive Differential Evolution. In *IEEE Congress on Evolutionary Computation (CEC) 2009*, pages 415–422. IEEE Press, 2009.

[9] V. L. Huang, A. K. Qin, and P. N. Suganthan. Self-adaptive Differential Evolution Algorithm for Constrained Real-Parameter Optimization. In *The 2006 IEEE Congress on Evolutionary Computation CEC2006*, pages 17–24. IEEE Press, 2006.

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

[11] J. Mallipeddi and P. N. Suganthan. Problem Definitions and Evaluation Criteria for the CEC 2010 Competition and Special Session on Single Objective Constrained Real-Parameter Optimization. Technical report, Nanyang Technological University, Singapore, Nov, 2009. http://www.ntu.edu.sg/home/EPNSugan.

[12] R. Mallipeddi and P.N. Suganthan. Ensemble of Constraint Handling Techniques. *IEEE Transactions on Evolutionary Computation*.

[13] E. Mezura-Montes, J.Veláquez-Reyes, and C. A. Coello Coello. Modified Differential Evolution for Constrained Optimization. In *The 2006 IEEE Congress on Evolutionary Computation CEC2006*, pages 25–31. IEEE Press, 2006.

[14] Efrén Mezura-Montes and Blanca Cecilia López-Ramírez. Comparing Bio-Inspired Algorithms in Constrained Optimization Problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'2007)*, pages 662–66. IEEE Press, 2007.

[15] Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello. Promising infeasibility and multiple offspring incorporated to differential evolution for constrained optimization. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 225–232, New York, NY, USA, 2005. ACM.

[16] Efrén Mezura-Montes, Jesús Velázquez-Reyes, Carlos A. Coello Coello, and Lucía Mu noz Dávila. Multiple Trial Vectors in Differential Evolution for Engineering Design. *Engineering Optimization*, 39(5):567–589, Jul 2007.

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

[18] M. H. A. Hijazi N. S. Teng, J. Teo. Self-adaptive population sizing for a tune-free differential evolution. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 13(7):709–724, 2009.

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

[20] A. K. Qin, V. L. Huang, and P. N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, 2009.

[21] 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.

[22] 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.

[23] R. Storn. System Design by Constraint Adaptation and Differential Evolution. *Evolutionary Computation, IEEE Transactions on*, 3(1):22–34, Apr 1999.

[24] 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.

[25] 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.

[26] S. Sakai T. Takahama. Solving Difficult Constrained Optimization Problems by the ϵ Constrained Differential Evolution with Gradient-Based Mutation . *Studies in Computational Intelligence, ISBN: 978-3-642-00618-0*, 198:51–72, 2009.

[27] T. Takahama and S. Sakai. Constrained Optimization by the ϵ Constrained Differential Evolution with Gradient-BasedMutation Feasible Elites. In *The 2006 IEEE Congress on Evolutionary Computation CEC2006*, pages 17–24. IEEE Press, 2006.

[28] J. Teo. Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 10(8):673–686, 2006.

[29] Jingqiao Zhang and Arthur C. Sanderson. JADE: Adaptive Differential Evolution with Optional External Archive. *Evolutionary Computation, IEEE Transactions on*, 13(5):945–958, 2009.