High-Dimensional Real-Parameter Optimization using Self-Adaptive Differential Evolution Algorithm with Population Size Reduction

Janez Brest, *Member, IEEE*, Aleš Zamuda, *Student Member, IEEE*, Borko Bošković, *Student Member, IEEE*, Mirjam Sepesy Maučec, and Viljem Žumer, *Member, IEEE*

***Abstract*— In this paper we investigate a Self-Adaptive Differential Evolution algorithm (*jDEdynNP-F*) where F and CR control parameters are self-adapted and a population size reduction method is used. Additionally the proposed *jDEdynNP-F* algorithm uses a mechanism for sign changing of F control parameter with some probability based on the fitness values of randomly chosen vectors, which are multiplied by the F control parameter (scaling factor) in the mutation operation of DE algorithm. The performance of the *jDEdynNP-F* algorithm is evaluated on the set of 7 benchmark functions provided for the CEC'2008 special session on high-dimensional real-parameter optimization.**

I. Introduction

THE general problem tackled using an optimization algorithm 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: $x_{j,low}, x_{j,upp}$; $j \in \{1, ..., D\}$.

Differential Evolution (DE) is a floating-point encoding evolutionary algorithm for global optimization over continuous spaces [30], [23], [15]. 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 [32]. As a solution, self-adaptation [5], [6], [13] has been found to be highly beneficial in automatically and dynamically adjusting evolutionary parameters such as crossover rates and mutation rates, and to do this without any user interaction.

The main objective of this paper is a performance evaluation of our self-adaptive differential evolution algorithm, named *jDEdynNP-F* which uses a self-adapting mechanism on the control parameters F and CR [9], [11], a method for gradually reducing population size [10] during the optimization process, and a new mechanism for exchanging the sign of control parameter F. The performance of the algorithm is evaluated on the set of benchmark functions provided for the CEC'2008 special session on high-dimensional real-parameter optimization [31].

The article is structured as follows. Section II gives an overview of work dealing with DE. Section III shortly summarizes the differential evolution. In Section IV the differential evolution *jDEdynNP-F* algorithm is described. In Section V experimental results of our self-adaptive *jDEdynNP-F* algorithm on CEC 2008 benchmark functions are presented and detailed performance analysis of the algorithm is given. Section VI concludes the paper.

Janez Brest, Aleš Zamuda, Borko Bošković, Mirjam Sepesy Maučec, and 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*

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

II. Related Work

Differential Evolution (DE) algorithm belongs to Evolutionary Algorithms (EAs). Historically, scaling EAs to large size problems have attracted much interest, including both theoretical and practical studies [31]. The DE algorithm was proposed by Storn and Price [30], [29], and since then the DE algorithm has been modified and widely used in many researches, practical applications, etc. [23], [7], [1], [17], [4], [12], [21], [25], [26].

The TDE algorithm [14] used the new trigonometric mutation operation. Teo [32] proposed a DE algorithm with a dynamic population sizing strategy based on self-adaptation, while F and CR control parameters are also self-adapted. Qin and Suganthan [24] proposed Self-adaptive Differential Evolution algorithm (SaDE), where control parameters F and CR are gradually self-adapted using statistical learning strategy. Brest et al. [8] compared some versions of adaptive and self-adaptive algorithms: FADE [18], DESAP [32], SaDE [24], and jDE [9].

Ali [2] proposed a study of the mutation operation in DE algorithm. He used an auxiliary population set in his works [2], [3].

To solve high-dimensional problems [20], [31] cooperative coevolution [22], [28] is used. Liu et al. [19] used FEP (fast evolutionary programming) with cooperative coevolution (FEPCC) to speedup convergence rates on the large-scale problems, Bergh and Engelbrecht [33] used a Cooperative Approach to Particle Swarm Optimisation (PSO), Yang, Tang and Yao recently used differential evolution with cooperative coevolution (DECC) [34]. Gao and Wang [16] used a memetic DE algorithm for high-dimensional problem 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

978-1-4244-1823-7/08/$25.00©2008 IEEE

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 problem [18].

The population of the original DE algorithm [29], [30], [27] 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 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 transformations (operations) in a process called evolution.

A. Mutation operation

Mutation for each population vector creates a mutant vector:

$$\vec{v}_i{}^{(G)} = \{v_{i,1}^{(G)}, v_{i,2}^{(G)}, ..., v_{i,D}^{(G)}\},\ i = 1, 2, ..., NP.$$

Mutant vector can be created using one of the mutation strategies. In literature one of the most useful mutation strategy is [23], [15]:

- "rand/1": $\vec{v}_i{}^{(G)} = \vec{x}_{r_1}^{(G)} + F \cdot (\vec{x}_{r_2}{}^{(G)} - \vec{x}_{r_3}{}^{(G)})$,

where the indexes r_1, r_2, r_3 represent the random and mutually different integers generated within range $[1, NP]$ and also different from index i. Mutation scale factor F is within the range $[0, 2]$, usually less than 1.

B. Crossover operation

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

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

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

CR is a crossover parameter or factor within the range $[0, 1)$ and presents the probability of creating parameters for trial vector $\vec{u}$ from a mutant vector $\vec{v}$. Index j_{rand} is a randomly chosen integer value within the range $[1, NP]$. It is responsible that the trial vector contains at least one parameter from the mutant vector. Here we describe binary crossover operation. The other one is exponential but it is rarely used in practice.

If the control parameter values from the trial vector $\vec{u}$ are out of bounds, the proposed solutions in literature [30], [29], [27], [23] are: they are set on bounds, used as they are (out of bounds), or are reflected into bounds (when $F \le 1$):

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

C. Selection operation

The third operation that of the DE algorithm is selection. It selects according to the fitness value $(f(\vec{x}))$ of the population vector $\vec{x}$ and its corresponding trial vector $\vec{u}$, which vector will survive to be a member of the next generation $(G + 1)$. In case of a minimization problem, the following selection operation is used:

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

IV. Our algorithm

The jDE algorithm [9] uses a self-adapting mechanism on control parameters F and CR. Let us first describe the self-adaptive jDE algorithm, then we will present the dynamic population size method, and finally we will describe our algorithm, which is used for experiments in this paper.

A. The Self-adaptive Differential Evolution Algorithm

In [9] the self-adapting control parameter mechanism of "*rand/1/bin*" strategy was used. The self-adaptive control mechanism was used to change the control parameters F and CR during the run. Each individual in population was extended with two values of the control parameters $F_i^{(G)}$ and $CR_i^{(G)}$ that were adjusted by means of evolution. The better values of these (encoded) control parameters lead to better individuals which, in turn, were more likely to survive in the selection process and produce offspring and, hence, propagate these better parameter values. The third control parameter, population size (NP), was not changed during the evolutionary process in [9].

The self-adaptive control parameters $F_i^{(G+1)}$ and $CR_i^{(G+1)}$ are calculated as follows:

$$F_i^{(G+1)} = \begin{cases} F_l + rand_1 \cdot F_u & \text{if } rand_2 < \tau_1, \\ F_i^{(G)} & \text{otherwise,} \end{cases}$$

$$CR_i^{(G+1)} = \begin{cases} rand_3 & \text{if } rand_4 < \tau_2, \\ CR_i^{(G)} & \text{otherwise.} \end{cases}$$

They produce control parameters F and CR in a new parent vector. The quantities $rand_j, j \in \{1, 2, 3, 4\}$ represent uniform random values within the range $[0, 1]$. τ_1 and τ_2 are 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. The control parameter values $F_i^{(G+1)}$ and $CR_i^{(G+1)}$ are obtained before the mutation operation is performed. This means they influence the mutation, crossover and selection operations of the new vector $\vec{x}_i^{G+1}$.

TABLE I

ERROR VALUES ACHIEVED FOR PROBLEMS 1-7, WITH $D = 100$

FES		1	2	3	4	5	6	7
	1^{st} (Best)	1.1234e+05	1.1331e+02	2.5674e+10	1.2128e+03	8.7685e+02	1.9808e+01	-9.0034e+02
	7^{th}	1.2617e+05	1.1897e+02	4.2406e+10	1.2536e+03	1.0551e+03	2.0023e+01	-8.8232e+02
	13^{th} (Median)	1.3541e+05	1.2082e+02	4.4896e+10	1.2913e+03	1.1237e+03	2.0143e+01	-8.6512e+02
$\mathbf{5.00e+3}$	19^{th}	1.4661e+05	1.2282e+02	4.9900e+10	1.3195e+03	1.1929e+03	2.0251e+01	-8.5241e+02
	25^{th} (Worst)	1.6494e+05	1.2850e+02	6.5184e+10	1.3765e+03	1.2697e+03	2.0429e+01	-8.3771e+02
	Mean	1.3630e+05	1.2122e+02	4.5628e+10	1.2889e+03	1.1163e+03	2.0131e+01	-8.6670e+02
	Std	1.3923e+04	3.7367	8.5890e+09	4.5604e+01	9.2438e+01	1.6217e-01	1.7922e+01
	1^{st} (Best)	2.4892	3.2628e+01	9.3574e+03	2.8500e+02	7.9203e-01	4.0365e-01	-1.2017e+03
	7^{th}	2.8334	3.4888e+01	2.2053e+04	3.2962e+02	8.5990e-01	5.1325e-01	-1.1724e+03
	13^{th} (Median)	3.1589	3.5834e+01	3.0970e+04	3.4869e+02	9.1115e-01	5.8703e-01	-1.1506e+03
$\mathbf{5.00e+4}$	19^{th}	4.2026	3.8442e+01	4.4655e+04	3.5664e+02	9.4162e-01	6.6028e-01	-1.1330e+03
	25^{th} (Worst)	7.1115	4.4748e+01	7.7325e+04	4.0255e+02	1.0214	7.8441e-01	-1.1122e+03
	Mean	3.7501	3.6921e+01	3.4041e+04	3.4258e+02	9.0875e-01	5.8728e-01	-1.1515e+03
	Std	1.2110	3.2655	1.7516e+04	2.4548e+01	6.5617e-02	1.0515e-01	2.4061e+01
	1^{st} (Best)	5.6843e-14	1.2996e-01	4.0958e+01	0.0000	2.8421e-14	5.6843e-14	-1.4881e+03
	7^{th}	5.6843e-14	2.5310e-01	7.9455e+01	5.6843e-14	2.8421e-14	5.6843e-14	-1.4815e+03
	13^{th} (Median)	5.6843e-14	4.2633e-01	1.1336e+02	5.6843e-14	2.8421e-14	5.6843e-14	-1.4761e+03
$\mathbf{5.00e+5}$	19^{th}	5.6843e-14	4.9249e-01	1.2445e+02	5.6843e-14	2.8421e-14	5.6843e-14	-1.4716e+03
	25^{th} (Worst)	5.6843e-14	1.0264	2.4621e+02	5.6843e-14	2.8421e-14	5.6843e-14	-1.4663e+03
	Mean	5.6843e-14	4.2875e-01	1.1158e+02	5.4570e-14	2.8422e-14	5.6843e-14	-1.4768e+03
	Std	0.0000	2.2701e-01	4.4760e+01	1.1369e-14	0.0000	0.0000	6.2274

Some ideas, how to improve the *jDE* algorithm, are reported in [8].

TABLE IV

TYPICAL RUN, WHEN $maxnfeval$ =500,000 AND $pmax = 4$ ARE ASSUMED

p	1	2	3	4
NP_p	200	100	50	25
gen_p	650	1250	2500	5000
$NP_p \times gen_p$	125,000	125,000	125,000	125,000

B. Dynamic Population Size

The self-adaptive *jDE* [9] revised in Section IV-A is not concerned with the third control parameter NP. This section revises an improved and more robust version of the self-adaptive DE algorithm, dealing with population size NP, called a dynNP-DE algorithm [10]. In [10] we have extended our self-adaptive *jDE*[9] algorithm using dynamic population size.

In the *jDEdynNP-F* algorithm population size decreases during the evolutionary process. It could also be possible to increase the population size when certain criteria is reached (e.g. population variance drops to a certain predefined lower bound). But in our case, we never increase population size.

One step of the population size reduction is as follows [10] (in case of minimization):

$$\vec{x}_i^{(G)} = \begin{cases} \vec{x}^{(G)}_{\frac{NP}{2}+i} & \text{if } f(\vec{x}^{(G)}_{\frac{NP}{2}+i}) < f(\vec{x}_i^{(G)}) \text{ and } G = G_R, \\ \vec{x}_i^{(G)} & \text{otherwise,} \end{cases} \tag{1}$$

$$NP^{(G+1)} = \begin{cases} \frac{NP^{(G)}}{2} & \text{if } G = G_R, \\ NP^{(G)} & \text{otherwise.} \end{cases} \tag{2}$$

$$i = 1, 2, ..., \frac{NP}{2}.$$

Only a few populations are exposed to reduction. We denote the generation, whose population is to be reduced, as G_R. One vector (individual) from the first half ($\vec{x}_i^{(G)}$) of the current population and corresponding individual from the second half ($\vec{x}^{(G)}_{\frac{NP}{2}+i}$) are compared based on their fitness values and the better one is placed (as a survivor) in the first half at position i of the current population. The first part of current population is assumed to be the population which is to be the parent population in the next generation. In the proposed reduction scheme the new population size is equal to half the previous population size.

In this paper we give some remarks about implementation. The reduction of population size is performed as depicted in Fig. 1.

We used `swap` function. Actually, it was unnecessary to swap those individuals (vectors) and their fitness values

TABLE II
ERROR VALUES ACHIEVED FOR PROBLEMS 1-7, WITH $D = 500$

FES		1	2	3	4	5	6	7
$\mathbf{2.50e+4}$	1^{st} (Best)	1.6025e+06	1.6207e+02	9.5642e+11	8.8073e+03	1.2569e+04	2.1246e+01	-3.6288e+03
	7^{th}	1.7576e+06	1.6431e+02	1.2659e+12	9.3208e+03	1.4659e+04	2.1272e+01	-3.5721e+03
	13^{th} (Median)	1.7977e+06	1.6539e+02	1.4190e+12	9.4219e+03	1.5154e+04	2.1284e+01	-3.5465e+03
	19^{th}	1.8262e+06	1.6711e+02	1.4752e+12	9.4810e+03	1.5397e+04	2.1292e+01	-3.5335e+03
	25^{th} (Worst)	1.8615e+06	1.6838e+02	1.5871e+12	9.6161e+03	1.5924e+04	2.1318e+01	-3.4764e+03
	Mean	1.7846e+06	1.6553e+02	1.3649e+12	9.3746e+03	1.5003e+04	2.1283e+01	-3.5490e+03
	Std	5.9938e+04	1.6684	1.6757e+11	1.8424e+02	7.1697e+02	1.6561e-02	3.3857e+01
$\mathbf{2.50e+5}$	1^{st} (Best)	2.7745e+04	1.2597e+02	2.8236e+09	4.4326e+03	2.5075e+02	1.1679e+01	-4.4533e+03
	7^{th}	3.0486e+04	1.2767e+02	3.5108e+09	4.5887e+03	2.7978e+02	1.2144e+01	-4.3708e+03
	13^{th} (Median)	3.1897e+04	1.2826e+02	3.9378e+09	4.6260e+03	2.9699e+02	1.2328e+01	-4.3434e+03
	19^{th}	3.3575e+04	1.2934e+02	4.2065e+09	4.6538e+03	3.1644e+02	1.2406e+01	-4.3227e+03
	25^{th} (Worst)	3.5231e+04	1.3084e+02	4.9603e+09	4.6932e+03	3.4005e+02	1.2889e+01	-4.2756e+03
	Mean	3.1741e+04	1.2853e+02	3.8791e+09	4.6030e+03	2.9932e+02	1.2268e+01	-4.3496e+03
	Std	2.0179e+03	1.2379	5.5968e+08	7.2762e+01	2.3789e+01	2.7653e-01	4.2065e+01
$\mathbf{2.50e+6}$	1^{st} (Best)	5.6843e-14	5.9865	5.1621e+02	1.1368e-13	2.8421e-14	1.1368e-13	-6.9395e+03
	7^{th}	5.6843e-14	7.2485	6.2577e+02	2.2737e-13	2.8421e-14	1.1368e-13	-6.8977e+03
	13^{th} (Median)	1.1368e-13	8.0337	6.4456e+02	4.5474e-13	2.8421e-14	1.1368e-13	-6.8876e+03
	19^{th}	1.1368e-13	9.2700	6.9820e+02	7.9580e-13	5.6843e-14	1.4210e-13	-6.8526e+03
	25^{th} (Worst)	1.1368e-13	1.3115e+01	8.7290e+02	1.6598e-11	5.6843e-14	7.1054e-13	-6.8410e+03
	Mean	9.3223e-14	8.4648	6.6115e+02	1.4688e-12	4.2064e-14	1.4893e-13	-6.8816e+03
	Std	2.7847e-14	1.7360	8.2652e+01	3.4839e-12	1.4492e-14	1.1812e-13	2.9375e+01

```
// individuals' fitness values are already stored in array named cost
for (i=0; i < NP/2; i++) {
    if (cost[i] < cost[NP/2+i]) {    // comparison of two individuals
        swap(x[i], x[NP/2+i]);       // swap individuals
        swap(cost[i], cost[NP/2+i])  // and swap their fitness values
    }
}
NP = NP/2;    // new population size
```

Fig. 1. The population size reduction using C/C++ language

$f(\vec{x}_i)$ and $f(\vec{x}_{\frac{NP}{2}+i})$, which are stored in array `cost` at indexes i and $\frac{NP}{2}+i$, respectively.[1] It could be argued that the assignment should be used rather than `swap`. It is true that when using the assignment instead of swap, the algorithm is slightly faster. We decided to use swap, because we did not want to change the original DE program code written in C programming language for calculating population mean and variance. Another reason for leaving swap in our program is that the proposed population size reduction is executed very rarely during the whole optimization process and does not considerably influence the speed of the algorithm.

How many population size reductions should be performed and when? There are many possibilities, but perhaps one of the simplest was presented in [10] and here we shortly revise it. The stopping criteria in our study and also in this paper has a predefined number of evaluations, denoted as $maxnfeval$. Let $pmax$ be the number of different population sizes used in the evolutionary process. Then $pmax - 1$ reductions need to be performed. $NP_1 = NP_{init}$ is the initial population size and NP_p $(p = 1, 2, ..., pmax)$ is the population size after $p - 1$ reductions. gen_p denotes the number of generations with population size NP_p. In our experiments we used an equal number of evaluations $\frac{maxnfeval}{pmax}$ for each population size. Table IV illustrates the case, when we assume: $maxnfeval = 500,000$ (for test problem with dimension $D = 100$) and $pmax = 4$. The number of generations gen_p with the population size NP_p is calculated as:

$$gen_p = \left\lfloor \frac{maxnfeval}{pmax \cdot NP_p} \right\rfloor + r_p,$$

where $r_p \geq 0$ is a small integer value greater than 0 when $maxnfeval$ is not divisible by $pmax$. In this case, the algorithm needs to perform additional r_p iterations to reach a predefined number of iterations in one run. Usually, the value of r_p is 0. The reduction is performed in the following generations:

$$G_R \in \{gen_1, gen_1 + gen_2, ..., \sum_{p=1}^{pmax-1} gen_p\}.$$

We have put special attention to satisfy the restriction about violating the predefined maximum number of evaluations

[1] In C/C++ index of array starts at 0.

TABLE III
ERROR VALUES ACHIEVED FOR PROBLEMS 1-7, WITH $D = 1000$

FES		1	2	3	4	5	6	7
$\mathbf{5.00e+4}$	1^{st} (Best)	3.6422e+06	1.7336e+02	2.9692e+12	1.9389e+04	3.3277e+04	2.1370e+01	-6.9313e+03
	7^{th}	3.9761e+06	1.7624e+02	3.3363e+12	2.0232e+04	3.6510e+04	2.1393e+01	-6.7238e+03
	13^{th} (Median)	4.2183e+06	1.7675e+02	3.5533e+12	2.0501e+04	3.7364e+04	2.1406e+01	-6.6888e+03
	19^{th}	4.3016e+06	1.7766e+02	3.7664e+12	2.0707e+04	3.8373e+04	2.1414e+01	-6.6697e+03
	25^{th} (Worst)	4.3882e+06	1.7836e+02	4.2294e+12	2.0869e+04	3.9860e+04	2.1419e+01	-6.6156e+03
	Mean	4.1416e+06	1.7679e+02	3.5746e+12	2.0416e+04	3.7412e+04	2.1402e+01	-6.7076e+03
	Std	2.1952e+05	1.1312	3.3083e+11	3.5816e+02	1.5201e+03	1.4733e-02	6.7899e+01
$\mathbf{5.00e+5}$	1^{st} (Best)	2.6946e+05	1.5195e+02	5.8021e+10	1.0851e+04	2.1474e+03	1.7527e+01	-8.0670e+03
	7^{th}	2.8567e+05	1.5394e+02	6.5893e+10	1.1139e+04	2.4524e+03	1.7967e+01	-7.8929e+03
	13^{th} (Median)	2.9371e+05	1.5441e+02	7.2415e+10	1.1179e+04	2.5762e+03	1.8080e+01	-7.8557e+03
	19^{th}	3.0123e+05	1.5492e+02	7.5357e+10	1.1253e+04	2.6307e+03	1.8203e+01	-7.8103e+03
	25^{th} (Worst)	3.1673e+05	1.5663e+02	8.1070e+10	1.1335e+04	2.7789e+03	1.8370e+01	-7.7480e+03
	Mean	2.9215e+05	1.5443e+02	7.0955e+10	1.1174e+04	2.5314e+03	1.8077e+01	-7.8568e+03
	Std	1.2171e+04	9.7975e-01	5.9926e+09	1.1400e+02	1.6569e+02	1.8138e-01	6.8574e+01
$\mathbf{5.00e+6}$	1^{st} (Best)	1.1368e-13	1.6031e+01	1.1359e+03	9.2726e-06	2.8421e-14	1.3073e-12	-1.3593e+04
	7^{th}	1.1368e-13	1.7592e+01	1.1907e+03	1.8623e-05	2.8421e-14	4.0074e-12	-1.3524e+04
	13^{th} (Median)	1.1368e-13	1.9318e+01	1.2893e+03	4.2234e-05	2.8421e-14	6.8496e-12	-1.3480e+04
	19^{th}	1.1368e-13	2.2206e+01	1.3930e+03	1.5211e-04	5.6843e-14	1.0857e-11	-1.3455e+04
	25^{th} (Worst)	1.1368e-13	2.3161e+01	1.5714e+03	1.4753e-03	5.6843e-14	1.1962e-10	-1.3424e+04
	Mean	1.1369e-13	1.9529e+01	1.3136e+03	2.1668e-04	3.9790e-14	1.4687e-11	-1.3491e+04
	Std	0.0000	2.2525	1.3635e+02	4.0563e-04	1.4211e-14	2.4310e-11	4.6038e+01

proposed at this special session. In our experiments we try to use such $pmax$ values which imply $r_p = 0$, and in code we have put an additional stopping condition to check if $nfeval$ reaches the predefined number of evaluations $maxnfeval$.

The revised dynamic population size reduction is executed $pmax - 1$ times per optimization run. The rules for population size reduction are quite simple and, therefore, in comparison to the original DE algorithm, our version of the DE algorithm does not considerably increase the time complexity.

The characteristics of the proposed population size reduction are:

- it follows the inspiration of the original DE selection operation,
- does not require many additional operations, e.g. sorting of all individuals based on their fitness values, and
- can be implemented efficiently (one population size reduction requires $\frac{NP}{2}$ comparisons of fitness values and $\frac{NP}{4}$ swapping of individuals, on average).

Special attention is needed to preserve minimal requirements about the population size of the original DE algorithm, which is at least 4 for 'rand/1/bin' strategy (indexes i, r_1, r_2 and r_3 must be mutually different (see Formula (III-A)).

C. Our Algorithm

We named our algorithm *jDEdynNP-F* and used it in experiments that follow. The *jDEdynNP-F* algorithm uses self-adaptive control parameters F and CR (Section IV-B) and dynamic population size (Section IV-A). Additionally, the *jDEdynNP-F* algorithm uses a mechanism that changes the sign of the control parameter F with some probability ($prob = 0.75$) when $f(\vec{x}_{r_2}) > f(\vec{x}_{r_3})$ during the mutation operation as presented in Fig. 2.

```
// individuals' fitness values are stored in array named cost
prob = 0.75;
if (rand < prob && cost[r2] > cost[r3])
    F = -F;   // sign change
```

Fig. 2. The control parameter F changes sign

Our algorithm optimizes a function as a black box, and in terms of optimizing high dimensional problems, no cooperative coevolution with any divide-and-conquer strategy is used.

V. EXPERIMENTAL RESULTS

The *jDEdynNP-F* algorithm was tested on 7 CEC'2008 special session benchmark functions [31]. Each function required calculations for three different dimensions ($D = 100$, $D = 500$, and $D = 1000$). 25 runs of algorithm were needed for each function.

Parameter settings used in the experiments were:

- F was self-adaptive, initially set to 0.5,
- CR was self-adaptive, initially set to 0.9,

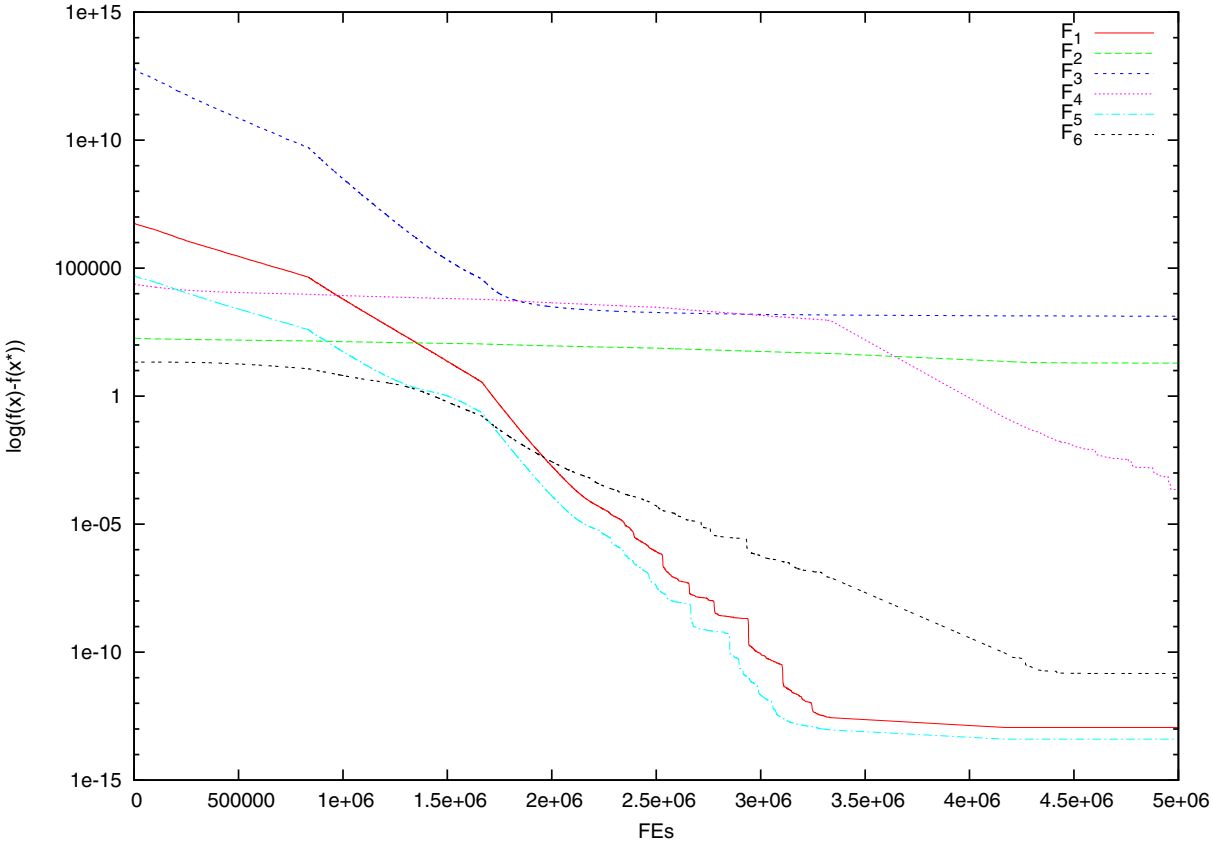

Fig. 3. Convergence graphs for problems 1-6

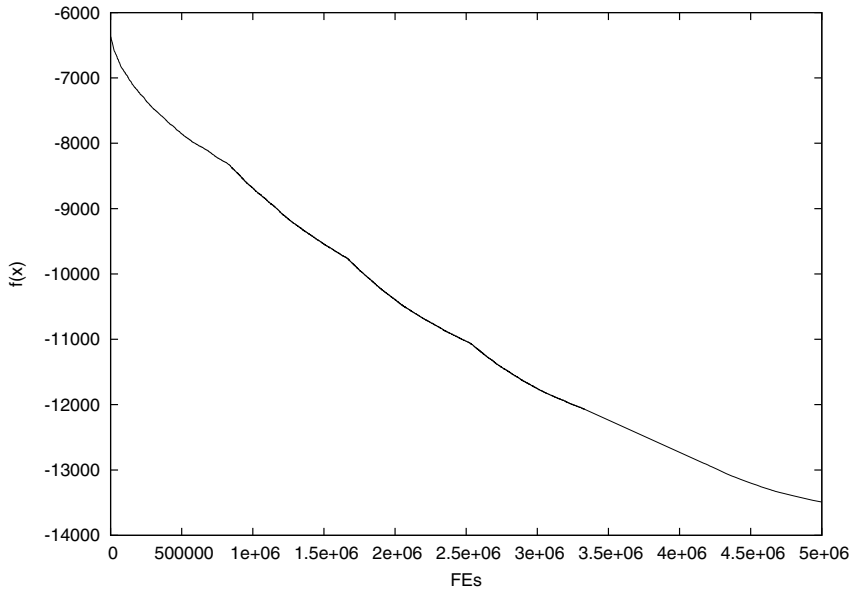

Fig. 4. Convergence graph for problem 7

TABLE V
ERROR VALUES ACHIEVED FOR PROBLEMS 1-7, WITH $D = 1000$ BY *jDEdynNP-F* ALGORITHM WITHOUT USING THE SIGN CHANGE OF F CONTROL PARAMETER.

FES		1	2	3	4	5	6	7
$\mathbf{5.00e+6}$	1^{st} (Best)	1.1368e-13	1.7350e+01	1.1868e+03	6.5144e-04	2.8421e-14	3.4415e-10	-1.3578e+04
	7^{th}	1.1368e-13	1.9321e+01	1.3966e+03	7.5542e-03	5.6843e-14	1.1594e-09	-1.3496e+04
	13^{th} (Median)	1.1368e-13	2.1510e+01	1.4806e+03	1.5775e-02	5.6843e-14	3.8918e-09	-1.3468e+04
	19^{th}	1.1368e-13	2.3027e+01	1.5933e+03	2.0297e-02	5.6843e-14	1.6011e-08	-1.3458e+04
	25^{th} (Worst)	1.7053e-13	2.5730e+01	1.8471e+03	3.6480e-02	5.6843e-14	5.1012e-07	-1.3418e+04
	Mean	1.2278e-13	2.1199e+01	1.5010e+03	1.4679e-02	5.3433e-14	4.9366e-08	-1.3476e+04
	Std	2.1269e-14	2.5289	1.4782e+02	9.0288e-03	9.4264e-15	1.3254e-07	3.7444e+01

- NP was adaptive, initial value $NP_{init} = D$,
- $pmax$ was fixed during the optimization: $pmax = 3$ when $D = 100$, $pmax = 5$ when $D = 500$, and $pmax = 6$ when $D = 1000$.

One can observe, that $pmax$ values imply that population sizes after $pmax - 1$, NP_{pmax}, are greater than 20 for all three dimensions ($D = 100$, $D = 500$, and $D = 1000$).

The obtained results (error values $f(\vec{x}) - f(\vec{x}^*)$) are presented in Tables I– III.

The optimal solution results are known for benchmark functions F_1–F_6, while for function F_7 optimal solution value was not given. Values for F_7 $f(\vec{x})$ are presented in Tables I–III.

For functions F_1 (*Shifted Sphere Function*), F_5 (*Shifted Griewank's Function*), and F_6 (*Shifted Ackley's Function*) the *jDEdynNP-F* algorithm successfully found mean solutions with function error values $f(\vec{x}) - f(\vec{x}^*) \leq 10^{-6}$ for all dimensions. Slightly worse results are obtained for function F_4 (*Shifted Rastrigin's Function*) for dimensions 500 and 1000. The worst results are obtained for functions F_2 (*Shifted Schwefel's Function*), and F_6 (*Shifted Rosenbrock's Function*).

Convergence graphs for benchmark functions F_1–F_6 when $D = 1000$ are presented in Fig. 3. Graph for function F_7 when $D = 1000$ is depicted in Fig. 4.

System: GNU/Linux x86_64 CPU: 2.4 GHz (2x Dual Core AMD Opteron). RAM: 8 GB.
Language: C/C++. Algorithm: *jDEdynNP-F* – Differential Evolution (DE), self-adaptive with population size reduction. Runs/problem: 25. Dimensions: 100, 500, and 1000. Runtime: 120 hours.

In order to present the amount of improvement achieved by the mechanism of control parameter F sign change, we performed an additional experiment with the *jDEdynNP-F* algorithm without the F sign changing mechanism (see Fig. 2). The obtained results are presented in Table V. If we compare results in Tables V and III we can see that *jDEdynNP-F* algorithm gives better results than the *jDEdynNP-F* algorithm without the F exchanging mechanism for all 7 benchmark functions when $D = 1000$.

VI. CONCLUSIONS

In this paper the performance of the *jDEdynNP-F* algorithm was evaluated on the set of benchmark functions provided by CEC'2008 special session on high-dimensional real-parameter optimization.

A self-adaptive control mechanism was used by the algorithm to change the control parameters (F and CR) during the optimization process. Additionally, the algorithm used a method for gradually reducing population size (NP), and a mechanism for changing the sign of F control parameter.

The results of this paper give evidence that the *jDEdynNP-F* algorithm is a competitive algorithm for high-dimensional real-parameter optimization problems. One of future plans is to apply cooperative coevolution methods to *jDEdynNP-F* algorithm.

ACKNOWLEDGMENT

The authors would also like to acknowledge the efforts of the organizers of this session and availability of test function suite code.

REFERENCES

[1] Fawaz S. Al-Anzia and Ali Allahverdi. A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. *European Journal of Operational Research*, 182(1):80–94, 2007.
[2] M. M. Ali. Differential evolution with preferential crossover. *European Journal of Operational Research*, 181(3):1137–1147, 2007.
[3] 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.
[4] Andries P. Engelbrecht Ayed Salman and Mahamed G.H. Omran. Empirical analysis of self-adaptive differential evolution. *European Journal of Operational Research*, 183(2):758–804, 2007. DOI: 10.1016/j.ejor.2006.10.020.
[5] T. Bäck. Adaptive Business Intelligence Based on Evolution Strategies: Some Application Examples of Self-Adaptive Software. *Information Sciences*, 148:113–121, 2002.
[6] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.
[7] B. Bošković, S. Greiner, J. Brest, and V. Žumer. A Differential Evolution for the Tuning of a Chess Evaluation Function. In *The 2006 IEEE Congress on Evolutionary Computation CEC2006*, pages 6742–6747. IEEE Press, 2006.

[8] 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. DOI: 10.1007/s00500-006-0124-0.

[9] 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. DOI: 10.1109/TEVC.2006.872133.

[10] J. Brest and M. Sepesy Maučec. Population Size Reduction for the Differential Evolution Algorithm. *Applied Intelligence*. DOI: 10.1007/s10489-007-0091-x. Accepted.

[11] 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.

[12] Leandro dos Santos Coelho and Viviana Cocco Mariani. Improved differential evolution algorithms for handling economic dispatch optimization with generator constraints. *Energy Conversion and Management*, 48(5):1631–1639, 2007.

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

[14] Hui-Yuan Fan and Jouni Lampinen. A Trigonometric Mutation Operation to Differential Evolution. *Journal of Global Optimization*, 27(1):105–129, 2003.

[15] Vitaliy Feoktistov. *Differential Evolution: In Search of Solutions (Springer Optimization and Its Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[16] Yu Gao and Yong-Jun Wang. A Memetic Differential Evolutionary Algorithm for High Dimensional Functions' Optimization. *Third International Conference on Natural Computation (ICNC 2007)*, 4:188–192, 2007.

[17] Thiemo Krink, Sandra Paterlini, and Andrea Resti. Using differential evolution to improve the accuracy of bank rating systems. *Computational Statistics & Data Analysis*, 52(1):68–87, 2007.

[18] 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.

[19] Yong Liu, Xin Yao, Qiangfu Zhao, and Tetsuya Higuchi. Scaling Up Fast Evolutionary Programming with Cooperative Coevolution. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 1101–1108, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 2001. IEEE Press.

[20] Cara MacNish. Towards Unbiased Benchmarking of Evolutionary and Hybrid Algorithms for Real-valued Optimisation. *Connection Science*, 19(4):225–239, 2007.

[21] Amin Nobakhti and Hong Wang. A simple self-adaptive Differential Evolution algorithm with application on the ALSTOM gasifier. *Applied Soft Computing*, 8(1):350–370, 2008.

[22] Mitchell A. Potter and Kenneth De Jong. A Cooperative Coevolutionary Approach to Function Optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 249–257, Berlin, 1994. Springer.

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

[24] 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.

[25] Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M. A. Salama. Opposition-based differential evolution. *IEEE Transactions on Evolutionary Computation*, 12(1):64–79, 2008. DOI: 10.1109/TEVC.2007.894200.

[26] N. Noman and H. Iba. Accelerating Differential Evolution Using an Adaptive Local Search. *IEEE Transactions on Evolutionary Computation*, 12(1):107-125, 2008. DOI: 10.1109/TEVC.2007.895272.

[27] 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.

[28] D. Sofge, K. De Jong, and A. Schultz. A Blended Population Approach to Cooperative Coevolution for Decomposition of Complex Problems. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC2002)*, pages 413–418. IEEE, 2002.

[29] 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.

[30] 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.

[31] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang. Benchmark Functions for the CEC'2008 Special Session and Competition on High-Dimenasional Real-Parameter Optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007. http://nical.ustc.edu.cn/cec08ss.php.

[32] J. Teo. Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 10(8):673–686, 2006. DOI: 10.1007/s00500-005-0537-1.

[33] F. van den Bergh and A. P. Engelbrecht. A Cooperative Approach to Particle Swarm Optimisation. *IEEE Transactions on Evolutionary Computing*, 3:225–239, 2004.

[34] Zhenyu Yang, Ke Tang, and Xin Yao. Differential Evolution for High-Dimensional Function Optimization. In Dipti Srinivasan and Lipo Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages –, Singapore, 25-28 September 2007. IEEE Computational Intelligence Society, IEEE Press.