Large Scale Global Optimization using Self-adaptive Differential Evolution Algorithm

Janez Brest, *Member, IEEE*, Aleš Zamuda, *Student Member IEEE*, Iztok Fister, Mirjam Sepesy Maučec

***Abstract*— In this paper we present self-adaptive differential evolution algorithm** jDElsgo **on large scale global optimization. The experimental results obtained by our algorithm on benchmark functions provided for the CEC 2010 competition and special session on Large Scale Global Optimization are presented. The experiments were performed on 20 benchmark functions with high dimension $D = 1000$. Obtained results show that our algorithm performs highly competitive in comparison with the DECC-G*, DECC-G and MLCC algorithms.**

I. Introduction

THE general problem of a global optimization algorithm is to find variables of vector $\vec{x} = \{x_1, x_2, ..., x_D\}$, where D denotes the dimensionality of a function, such that objective function $f(\vec{x})$ is optimized. Domains of the variables are defined by their lower and upper bounds $x_{j,low}, x_{j,upp}$ for $j = 1, 2, ..., D$.

Differential Evolution (DE) [22] belongs to a group of evolutionary algorithms. The DE uses floating-point encoding and has some good properties, efficiency and robustness, and it is simple yet powerful algorithm for global optimization over continuous spaces. The original DE algorithm has three control parameters, which are being fixed during the optimization process. The best settings of the control parameters depend on the function and requirements for consumption time and accuracy. If the size of problems is increased then much more time is needed to perform preliminary testing and hand-tuning of the evolutionary parameters prior to commencing the actual optimization process. In literature various adaptation mechanisms are proposed to overcome the hand-tuning problems of the DE parameters [18], [19], [3], [30], [13].

In this paper a performance evaluation of our self-adaptive jDElsgo algorithm is analyzed. The performance of the algorithm is evaluated on the set of benchmark functions provided for the CEC 2010 special session on large-scale global optimization [23].

The article is structured as follows. In Section II an overview of related work is presented. Sections III and IV give background for this work. In these sections an overview of DE and description of the self-adaptive *jDEdynNP-F* algorithm are given. Section V presents a description of our self-adaptive jDElsgo algorithm which is used in these experiments. In Section VI experimental results of the jDElsgo algorithm on CEC 2010 benchmark functions are presented. Section VII concludes the paper with some final remarks.

Janez Brest, Aleš Zamuda, Iztok Fister, 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@uni-mb.si).

II. Related Work

Differential Evolution (DE) algorithm belongs to Evolutionary Algorithms (EAs). The DE [22] algorithm was proposed by Storn and Price, and since then the DE algorithm has been used in different areas. The original DE was modified, and many new versions were proposed in [17], [30], [9], [20]. Neri and Tirronen [15] give a survey of recent advances in differential evolution.

Qin and Suganthan in [19], [18] 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. [3] proposed a self-adaptive jDE algorithm. The self-adaptive mechanism was used in algorithms [3], [2], [4] for unconstrained optimization. An another version of the jDE algorithm with population size reduction was used for large scale global optimization on CEC 2008 [6]. Yet another developed version that is based on the jDE algorithm was used for dynamic optimization problems [7]. The jDE algorithm was also adopted to work for constrained optimization problems [5], [1].

Historically, scaling EAs to large size problems have attracted much interest, including both theoretical and practical studies [24]. To solve high-dimensional problems [12], [24] cooperative coevolution [16], [21], [29] is used. Liu et al. [11] used FEP (fast evolutionary programming) with cooperative coevolution (FEPCC) to speedup convergence rates on the large-scale problems, Bergh and Engelbrecht [25] used a Cooperative Approach to Particle Swarm Optimisation (PSO). Yang, Tang and Yao recently used differential evolution with cooperative coevolution (DECC) [28]. Gao and Wang [10] used a memetic DE algorithm for high-dimensional problem optimization.

Recently, Yang, Tang, and Yao proposed DECC-G [26] algorithm, and MLCC [27] algorithm. The both algorithms use cooperative coevolution for solving large-scale optimization problems. In this work we will compare experimental results of our algorithm with the preliminary results of DECC-G, DECC-G* [1], and MLCC algorithms presented at `http://nical.ustc.edu.cn/cec10ss.php`.

[1]DECC-G*: The same as DECC-G, except that the grouping structure was used as prior knownledge. The parameter group size was set to s=50, and the adaptive weighting strategy of DECC-G was not used.

III. The Differential Evolution

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.

The population of the original DE algorithm [22] consists of 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. For each vector during one generation, DE employs the mutation and crossover operations to produce a trial vector $\vec{u}_i^{(G)}$. Then a selection operation is used to choose vectors for the next generation $(G+1)$.

A mutant vector $\vec{v}_i^{(G)}$ is created by 'rand/1' strategy as follows:

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

where F denotes a mutation scale factor within the range $[0, 2]$, usually less than 1. Indexes r_1, r_2, and r_3 represent random integers within set $\{1, NP\}$ and $r_1 \neq r_2 \neq r_3 \neq i$.

Then a crossover operation forms the trial vector $\vec{u}_i^{(G)}$:

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

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

CR denotes a crossover parameter within the range $[0, 1)$ and presents the probability of creating components for trial vector from a mutant vector. Index $j_{rand} \in \{1, NP\}$ is a randomly chosen integer that is responsible for the trial vector containing at least one component from the mutant vector.

The selection operation selects individuals according to their fitness values between the population vector and its corresponding trial vector. This better vector will survive and become a member of the next generation. For example, in a minimization problem the following selection rule is applied:

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

IV. The *jDEdynNP-F* Algorithm

For solving high-dimensional real-parameter optimization problems on CEC 2008 we used *jDEdynNP-F* algorithm [6]. In this section we give an overview of the *jDEdynNP-F* algorithm. In this algorithm F and CR control parameters are self-adapted (as in *jDE* algorithm), and two mechanisms for population size reduction and F sign changing are used.

A. The Self-adaptive jDE Algorithm

Our jDE algorithm [3] uses self-adapting mechanism of two control parameters and "*rand/1/bin*" strategy. Therefore, each individual in population is extended with control parameter values F and CR as follows: $\vec{x}_i^{(G)} = (x_{i,1}^{(G)}, x_{i,2}^{(G)}, ..., x_{i,D}^{(G)}, F_i^{(G)}, CR_i^{(G)})$. 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 in the next generation.

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 $rand_j$, for $j \in \{1, 2, 3, 4\}$ are uniform random values within the range $[0, 1]$. Parameters τ_1 and τ_2 represent probabilities to adjust control parameters F and CR, respectively. Parameters τ_1, τ_2, F_l, F_u are 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 $[0, 1]$.

Note that $F_i^{(G+1)}$ and $CR_i^{(G+1)}$ are obtained before the mutation is performed. So they influence on the mutation, crossover, and selection operations of the new vector $\vec{x}_i^{(G+1)}$.

B. Population Size Reduction

In the *jDEdynNP-F* algorithm population size NP decreases during the evolutionary process. If we assume that we have to optimize a minimization problem then one step of the population size reduction is calculated as follows [4], [6]:

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

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

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

Only a few populations are exposed to reduction. The generation, whose population is to be reduced, is denoted 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 $\vec{x}_{\frac{NP}{2}+i}^{(G)}$ are compared with regard to their fitness values and the better one is placed (as a survivor) at position i in the first half of the current population. The first part of current population is assumed as the population of next generation. In the proposed reduction scheme the new population size is equal to half the last population size.

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. The gen_p denotes the number of generations with population size NP_p.

Special care is needed to preserve the minimal requirements of the original DE algorithm population size. Actually, the 'rand/1/bin' strategy requires at least four population members (since indexes i, r_1, r_2, and r_3 must be mutually different).

```
 1: {$pop$ ... population}
 2: {$imin$ ... index of currently best individual}
 3: {$\vec{x}_i$ ... i-th individual of population}
 4: {$MAX\_FEs$ ... maximum number of function evaluations}
 5: Initialization() {Generate uniformly distributed random population}
 6: for ($it = 0$; $it < MAX\_FEs$; $it++$) do
 7:   $i = it$ mod $NP$ {mod ... modulo operation}
 8:
 9:   {Try to improve currently best individual two times per generation. Perform these steps only for two selected individuals (when
      $i = 0$ or $i = 1$), and when iteration exceeded half of MAX_FEs.}
10:   if (($it > MAX\_FEs/2$) and ($i < 2$)) then
11:     $\vec{x}_i = InitRand()$ {Initialize randomly individual $\vec{x}_i$ (without function evaluation)}
12:
13:     {*** Perform one iteration of jDE using strategy to improve currently best individual***}
14:     $F_l = 0.01$ {Lower bound of $F$ control parameter}
15:     $F_u = 0.09$ {Upper bound of $F$ control parameter}
16:     $\vec{u}$ =jDEoneIter($imin$, $pop$) {Perform one iteration of the jDE.}
17:   else
18:     if i mod 2 == 0 then
19:       {Otherwise perform one iteration of the jDE with bin crossover using values $F_l = 0.1; F_u = 0.9$; on individual $\vec{x}_i$}
20:       $\vec{u}$ =jDEoneIter($i$, $pop$)
21:     else
22:       {Otherwise perform one iteration of the jDE with exp crossover using values $F_l = 0.4; F_u = 0.6$; and $CR$ from interval
          $[0.6, 1.0]$ on individual $\vec{x}_i$}
23:       $\vec{u}$ =jDEoneIter($i$, $pop$)
24:     end if
25:   end if
26:   $f(\vec{u})$ {Function evaluation}
27:   ... {Selection, etc.}
28: end for
```

Algorithm 1: jDElsgo algorithm

The good properties of the population size reduction are:

- it follows the inspiration of the original DE selection operation and
- it does not require any additional operations, e.g. sorting of all individuals based on their fitness value.

C. Control Parameter F sign changing

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. 1.

```
$prob = 0.75$;
if ($rand(0,1) < prob$ && $f(\vec{x}_{r_2}) > f(\vec{x}_{r_3})$)
    $F = -F$;   // sign change
```

Fig. 1. The control parameter F changes sign

V. The proposed Algorithm

In this section our new algorithm jDElsgo for solving large-scale global optimization is presented. The jDElsgo algorithm is extended version of the *jDEdynNP-F*. The new proposed algorithm uses:

- self-adaptive F and CR control parameters as described in Section IV-A,
- population size reduction mechanism as presented in Section IV-B, and
- control parameter F sign change mechanism as described in Section IV-C.

The self-adaptation mechanism proposed in [3] has been widely used in many practical cases [8] and has shown good results. The population size reduction mechanism has a good performance in terms of robustness [15].

The main structure of jDElsgo is given in Alg. 1. If we compare the *jDEdynNP-F* with the jDElsgo algorithm, the later has added the new features implemented by lines 8–16 in Alg. 1. These features act on two individuals ($i < 2$) in each generation as follows:

- in order to add some randomness in population, individuals $\vec{x}_0$ and $\vec{x}_1$ are perturbed randomly (like in the initialization phase), since we use a small population size during the evolutionary process,
- F takes values from a narrower interval $[0.01, 0.1]$ (smaller values for $F_l = 0.01$ and $F_u = 0.09$),
- try to improve currently best individual. Our algorithm does not perform function evaluations for $\vec{x}_0$ and $\vec{x}_1$. In contrary, it tries to improve currently best individual.

Note, the jDElsgo can be seen as an algorithm that uses some kind of *scale factor local search* [14] when it improves currently best individual in a generation.

The main part of the jDElsgo algorithm is similar to the *jDEdynNP-F* algorithm (lines 17–25 in Alg. 1) with some changes in parameter values that are required prior to commencing the optimization process. The jDElsgo algorithm uses also 'exp' crossover variant.

TABLE I

PROPERTIES OF THE BENCHMARK FUNCTIONS

Function	Modality	Shifted	Separability	Scalability	x_i
F_1 : Shifted Elliptic Function	Unimodal	Shifted	Separable	Scalable	[−100, 100]
F_2 : Shifted Rastrigin's Function	Multimodal	Shifted	Separable	Scalable	[−5, 5]
F_3 : Shifted Ackley's Function	Multimodal	Shifted	Separable	Scalable	[−32, 32]
F_4 : Single-group Shifted and m-rotated Elliptic Function	Unimodal	Shifted	Single-group m-rotated	Single-group m-nonseparable	[−100, 100]
F_5 : Single-group Shifted and m-rotated Rastrigin's Function	Multimodal	Shifted	Single-group m-rotated	Single-group m-nonseparable	[−5, 5]
F_6 : Single-group Shifted and m-rotated Ackley's Function	Multimodal	Shifted	Single-group m-rotated	Single-group m-nonseparable	[−32, 32]
F_7 : Single-group Shifted m-dimensional Rosenbrock's Function	Unimodal	Shifted	-	Single-group m-nonseparable	[−100, 100]
F_8 : Single-group Shifted m-dimensional Rosenbrock's Function	Multimodal	Shifted	-	Single-group m-nonseparable	[−100, 100]
F_9 : $\frac{D}{2m}$-group Shifted and m-rotated Elliptic Function	Unimodal	Shifted	$\frac{D}{2m}$-group m-rotated	$\frac{D}{2m}$-group m-nonseparable	[−100, 100]
F_{10} : $\frac{D}{2m}$-group Shifted and m-rotated Rastrigin's Function	Multimodal	Shifted	$\frac{D}{2m}$-group m-rotated	$\frac{D}{2m}$-group m-nonseparable	[−5, 5]
F_{11} : $\frac{D}{2m}$-group Shifted and m-rotated Ackley's Function	Multimodal	Shifted	$\frac{D}{2m}$-group m-rotated	$\frac{D}{2m}$-group m-nonseparable	[−32, 32]
F_{12} : $\frac{D}{2m}$-group Shifted m-dimensional Schwefel's Problem 1.2	Unimodal	Shifted	-	$\frac{D}{2m}$-group m-nonseparable	[−100, 100]
F_{13} : $\frac{D}{2m}$-group Shifted m-dimensional Rosenbrock's Function	Multimodal	Shifted	-	$\frac{D}{2m}$-group m-nonseparable	[−100, 100]
F_{14} : $\frac{D}{m}$-group Shifted and m-rotated Elliptic Function	Unimodal	Shifted	$\frac{D}{m}$-group m-rotated	$\frac{D}{m}$-group m-nonseparable	[−100, 100]
F_{15} : $\frac{D}{m}$-group Shifted and m-rotated Rastrigin's Function	Multimodal	Shifted	$\frac{D}{m}$-group m-rotated	$\frac{D}{m}$-group m-nonseparable	[−5, 5]
F_{16} : $\frac{D}{m}$-group Shifted and m-rotated Ackley's Function	Multimodal	Shifted	$\frac{D}{m}$-group m-rotated	$\frac{D}{m}$-group m-nonseparable	[−32, 32]
F_{17} : $\frac{D}{m}$-group Shifted m-dimensional Schwefel's Problem 1.2	Unimodal	Shifted	-	$\frac{D}{m}$-group m-nonseparable	[−100, 100]
F_{18} : $\frac{D}{m}$-group Shifted m-dimensional Rosenbrock's Function	Multimodal	Shifted	-	$\frac{D}{m}$-group m-nonseparable	[−100, 100]
F_{19} : Shifted Schwefel's Problem 1.2	Unimodal	Shifted	-	Fully-nonseparable	[−100, 100]
F_{20} : Shifted Rosenbrock's Function	Multimodal	Shifted	-	Fully-nonseparable	[−100, 100]

The *jDEdynNP-F* algorithm in CEC 2008 used $NP_{init} = D$ and $pmax = 6$ when $D = 1000$, while the jDElsgo algorithm in this research uses $NP_{init} = 400$ and $pmax = 5$.

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.

TABLE II

TYPICAL RUN, WHEN Max_FEs =3,000,000 AND $pmax = 4$ ARE ASSUMED

p	1	2	3	4
NP_p	100	50	25	12
$NP_p \times gen_p$	750,000	750,000	750,000	750,000

In this paper, the population size NP was set to 100, number of reduction was set $pmax = 4$, and an equal number of evaluations $\frac{Max\_FEs}{pmax}$ for each population size. Table II illustrates the case, when we assume: $Max\_FEs = 3,000,000$, $D = 1000$, and $pmax = 4$.

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

VI. EXPERIMENTS AND RESULTS

A. Benchmark Functions

The jDElsgo algorithm was tested on 20 CEC 2010 special session benchmark functions [23]. The benchmark functions are scalable. The dimension of functions was $D = 1000$ and 25 runs of algorithm were needed for each function. The optimal solution results are known for all benchmark functions.

Properties of these functions are presented in Table I. Note, separability of a function presents a measure of difficulty to solve it. However, a function $f(\vec{x})$ is separable if its parameters x_i are independent. In general, separable problems are considered as easiest, while the fully-nonseparable are usually the most difficult problems to solve [23]. Degree of separability can be controlled by randomly dividing of the object variables into several groups each of which contains a definite numbers of variables. Although some of used functions were separable in their original form, applying techniques as Salomon's random coordinate rotation technique, make them non-separable. In addition, the global optimum can be shifted.

In our research variables grouping based on separability was not used.

B. Experimental Results

In the experiments parameters were set as follows:

- F was self-adaptive, initially set to 0.5,
- CR was self-adaptive, initially set to 0.9,
- NP was adaptive, initial value $NP_{init} = 100$,
- $pmax$ was fixed during the optimization, $pmax = 4$.

One can observe that $pmax$ values imply that population size is 12 after $pmax - 1$ population size reductions.

The obtained results (error values $f(\vec{x}) - f(\vec{x}^*)$) are presented in Table III. Table IV show preliminary result of cooperative coevolution algorithms from literature along with results of jDElsgo algorithm when $FEs = 3.0e + 6$. If we make the comparison with regard to the mean value then our jDElsgo algorithm performs 11, 18, 16 times better than the DECC-G*, DECC-G, and MLCC, respectively. The results indicate that our algorithm has almost similar performance

TABLE III

Experimental results obtained by the jDElsgo algorithm

FEs		F_1	F_2	F_3	F_4	F_5	F_6	F_7
1.20e+05	Best	2.78e+09	1.06e+04	1.81e+01	8.06e+13	2.98e+08	3.36e+06	2.89e+10
	Median	3.72e+09	1.09e+04	1.88e+01	1.43e+14	3.38e+08	4.24e+06	5.40e+10
	Worst	4.89e+09	1.13e+04	1.97e+01	2.30e+14	3.75e+08	4.84e+06	7.23e+10
	Mean	3.70e+09	1.09e+04	1.87e+01	1.40e+14	3.39e+08	4.26e+06	5.39e+10
	Std	5.11e+08	1.75e+02	4.46e-01	3.69e+13	1.82e+07	3.81e+05	1.07e+10
6.00e+05	Best	7.04e+04	3.67e+03	9.70e-01	7.89e+12	1.42e+08	2.20e+01	3.36e+09
	Median	8.71e+04	3.93e+03	1.18e+00	1.29e+13	1.87e+08	3.24e+01	6.41e+09
	Worst	1.23e+05	4.20e+03	1.58e+00	2.67e+13	2.30e+08	3.14e+02	1.10e+10
	Mean	8.99e+04	3.95e+03	1.22e+00	1.39e+13	1.88e+08	5.07e+01	6.43e+09
	Std	1.39e+04	1.32e+02	1.38e-01	4.60e+12	2.31e+07	5.81e+01	2.12e+09
3.00e+06	Best	4.78e-20	1.09e-11	1.63e-12	3.09e+10	7.42e+07	7.14e-09	2.69e-05
	Median	6.63e-20	4.69e-11	2.35e-12	8.28e+10	9.82e+07	7.22e-09	1.04e-04
	Worst	2.24e-19	1.12e+00	2.24e-11	1.34e+11	1.24e+08	2.10e-07	3.19e-01
	Mean	8.86e-20	1.25e-01	3.81e-12	8.06e+10	9.72e+07	1.70e-08	1.31e-02
	Std	4.51e-20	3.45e-01	5.02e-12	3.08e+10	1.44e+07	4.03e-08	6.38e-02
FEs		F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
1.20e+05	Best	1.04e+09	1.43e+10	1.31e+04	2.02e+02	2.76e+06	2.23e+09	1.95e+10
	Median	2.29e+09	1.59e+10	1.43e+04	2.20e+02	3.18e+06	3.70e+09	2.32e+10
	Worst	5.42e+09	2.07e+10	1.51e+04	2.26e+02	3.65e+06	5.44e+09	2.76e+10
	Mean	2.39e+09	1.64e+10	1.43e+04	2.19e+02	3.15e+06	3.76e+09	2.32e+10
	Std	9.13e+08	1.73e+09	4.38e+02	5.92e+00	2.19e+05	1.04e+09	2.03e+09
6.00e+05	Best	3.57e+07	1.45e+09	7.66e+03	7.73e+01	8.61e+05	2.90e+04	3.61e+09
	Median	4.65e+07	1.64e+09	8.69e+03	1.14e+02	9.35e+05	4.95e+04	4.11e+09
	Worst	1.39e+08	1.82e+09	9.49e+03	1.49e+02	9.88e+05	9.02e+04	4.72e+09
	Mean	6.82e+07	1.66e+09	8.67e+03	1.17e+02	9.39e+05	5.32e+04	4.10e+09
	Std	3.53e+07	8.29e+07	3.99e+02	1.87e+01	2.96e+04	1.70e+04	2.89e+08
3.00e+06	Best	3.40e-03	2.36e+07	2.10e+03	1.27e+00	8.32e+03	4.79e+02	1.28e+08
	Median	1.25e+06	3.04e+07	2.66e+03	1.95e+01	1.18e+04	6.91e+02	1.72e+08
	Worst	8.08e+06	4.22e+07	3.30e+03	5.81e+01	1.71e+04	1.02e+03	2.02e+08
	Mean	3.15e+06	3.11e+07	2.64e+03	2.20e+01	1.21e+04	7.11e+02	1.69e+08
	Std	3.27e+06	5.00e+06	3.19e+02	1.53e+01	2.04e+03	1.37e+02	2.08e+07
FEs		F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}	
1.20e+05	Best	1.44e+04	4.09e+02	4.28e+06	5.25e+10	2.07e+07	5.80e+10	
	Median	1.55e+04	4.17e+02	4.79e+06	6.37e+10	2.88e+07	8.14e+10	
	Worst	1.59e+04	4.24e+02	5.71e+06	8.76e+10	3.56e+07	1.11e+11	
	Mean	1.54e+04	4.17e+02	4.85e+06	6.60e+10	2.85e+07	7.99e+10	
	Std	3.33e+02	3.28e+00	3.53e+05	9.47e+09	3.38e+06	1.25e+10	
6.00e+05	Best	1.10e+04	2.67e+02	1.81e+06	6.95e+05	4.16e+06	5.98e+05	
	Median	1.22e+04	2.97e+02	1.97e+06	1.04e+06	5.48e+06	1.05e+06	
	Worst	1.26e+04	3.53e+02	2.10e+06	1.39e+06	1.22e+07	1.66e+06	
	Mean	1.20e+04	2.99e+02	1.95e+06	1.03e+06	6.09e+06	1.01e+06	
	Std	5.30e+02	1.91e+01	6.54e+04	2.08e+05	1.65e+06	2.48e+05	
3.00e+06	Best	5.20e+03	7.30e+01	7.75e+04	1.31e+03	2.39e+05	1.24e+03	
	Median	5.78e+03	1.46e+02	1.00e+05	1.88e+03	2.77e+05	1.55e+03	
	Worst	6.84e+03	2.00e+02	1.28e+05	2.57e+03	3.21e+05	1.83e+03	
	Mean	5.84e+03	1.44e+02	1.02e+05	1.85e+03	2.74e+05	1.53e+03	
	Std	4.48e+02	3.43e+01	1.26e+04	3.18e+02	2.12e+04	1.32e+02	

as the DECC-G* algorithm and outperformed the DECC-G and MLCC algorithms.

Convergence graphs of the jDElsgo algorithm for eight selected benchmark functions are presented in Fig. 2.

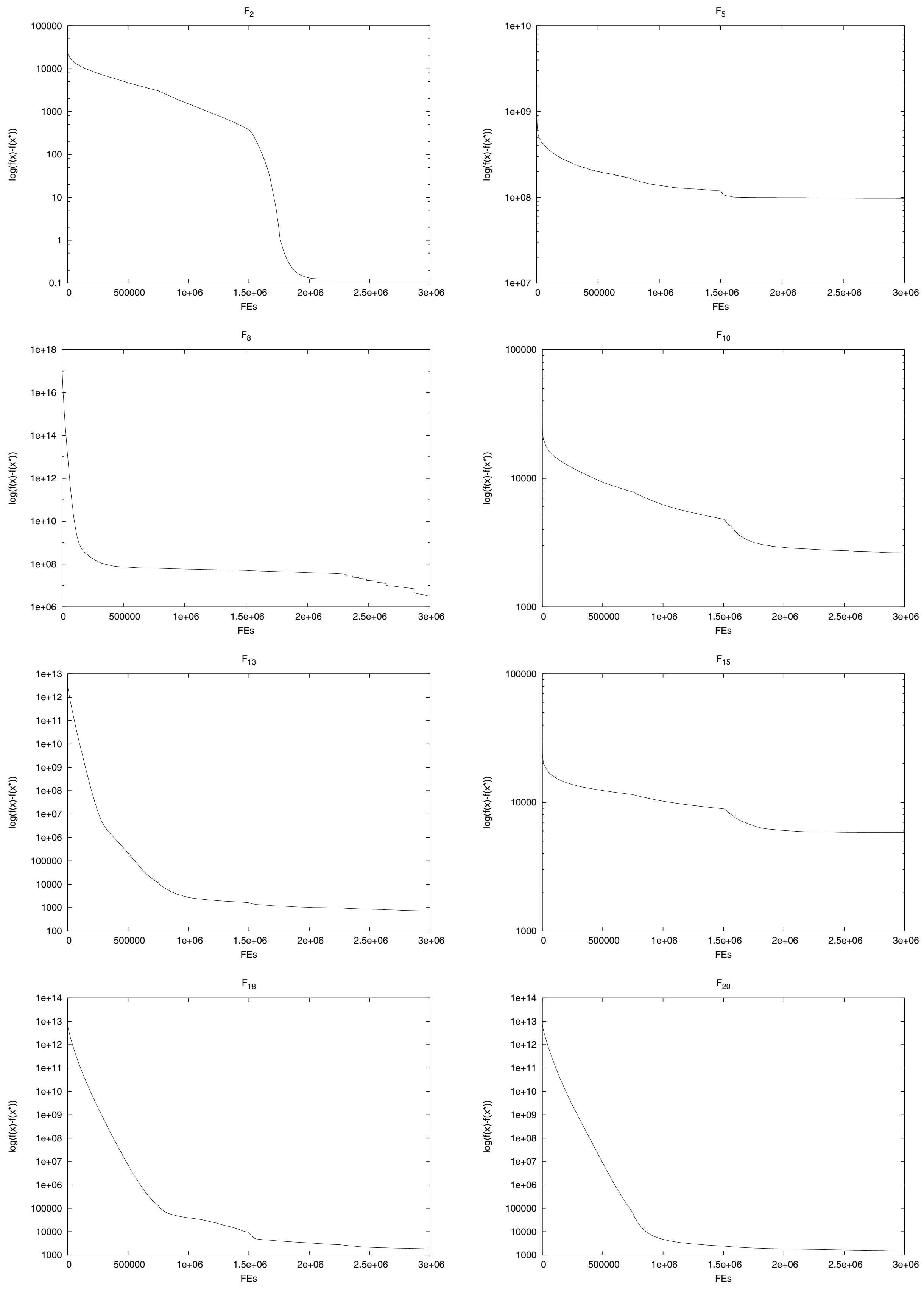

Fig. 2. Convergence Graph for Functions F_2, F_5, F_8, F_{10}, F_{13}, F_{15}, F_{18}, and F_{20}

TABLE IV

COMPARISON OF EXPERIMENTAL RESULTS WHEN $FEs = 3.0e+6$

		F_1	F_2	F_3	F_4	F_5	F_6	F_7
DECC-G	Best	1.63e-07	1.25e+03	1.20e+00	7.78e+12	1.50e+08	3.89e+06	4.26e+07
	Median	2.86e-07	1.31e+03	1.39e+00	1.51e+13	2.38e+08	4.80e+06	1.07e+08
	Worst	4.84e-07	1.40e+03	1.68e+00	2.65e+13	4.12e+08	7.73e+06	6.23e+08
	Mean	2.93e-07	1.31e+03	1.39e+00	1.70e+13	2.63e+08	4.96e+06	1.63e+08
	Std	8.62e-08	3.26e+01	9.73e-02	5.37e+12	8.44e+07	8.02e+05	1.37e+08
DECC-G*	Best	6.33e-12	4.21e+02	2.23e-08	9.76e+11	2.08e+08	5.07e-03	3.45e+06
	Median	8.97e-12	4.43e+02	3.30e-08	1.96e+12	2.49e+08	8.85e-03	1.04e+07
	Worst	1.31e-11	4.57e+02	4.16e-08	5.39e+12	2.72e+08	1.40e-02	2.28e+07
	Mean	8.81e-12	4.42e+02	3.30e-08	2.29e+12	2.45e+08	8.77e-03	1.10e+07
	Std	1.49e-12	9.94e+00	5.20e-09	9.97e+11	1.64e+07	2.46e-03	5.44e+06
MLCC	Best	0.00e+00	1.73e-11	1.28e-13	4.27e+12	2.15e+08	5.85e+06	4.16e+04
	Median	0.00e+00	6.43e-11	1.46e-13	1.03e+13	3.92e+08	1.95e+07	5.15e+05
	Worst	3.83e-26	1.09e+01	1.86e-11	1.62e+13	4.87e+08	1.98e+07	2.78e+06
	Mean	1.53e-27	5.57e-01	9.88e-13	9.61e+12	3.84e+08	1.62e+07	6.89e+05
	Std	7.66e-27	2.21e+00	3.70e-12	3.43e+12	6.93e+07	4.97e+06	7.37e+05
jDElsgo	Best	4.78e-20	1.09e-11	1.63e-12	3.09e+10	7.42e+07	7.14e-09	2.69e-05
	Median	6.63e-20	4.69e-11	2.35e-12	8.28e+10	9.82e+07	7.22e-09	1.04e-04
	Worst	2.24e-19	1.12e+00	2.24e-11	1.34e+11	1.24e+08	2.10e-07	3.19e-01
	Mean	8.86e-20	1.25e-01	3.81e-12	8.06e+10	9.72e+07	1.70e-08	1.31e-02
	Std	4.51e-20	3.45e-01	5.02e-12	3.08e+10	1.44e+07	4.03e-08	6.38e-02
		F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
DECC-G	Best	6.37e+06	2.66e+08	1.03e+04	2.06e+01	7.78e+04	1.78e+03	6.96e+08
	Median	6.70e+07	3.18e+08	1.07e+04	2.33e+01	8.87e+04	3.00e+03	8.07e+08
	Worst	9.22e+07	3.87e+08	1.17e+04	2.79e+01	1.07e+05	1.66e+04	9.06e+08
	Mean	6.44e+07	3.21e+08	1.06e+04	2.34e+01	8.93e+04	5.12e+03	8.08e+08
	Std	2.89e+07	3.38e+07	2.95e+02	1.78e+00	6.87e+03	3.95e+03	6.07e+07
DECC-G*	Best	2.79e+07	1.18e+07	2.33e+03	5.82e-08	6.16e+01	3.78e+02	2.46e+07
	Median	4.07e+07	1.41e+07	2.49e+03	7.52e-08	7.72e+01	5.40e+02	2.90e+07
	Worst	1.50e+08	1.77e+07	2.64e+03	8.79e-01	1.19e+02	7.55e+02	3.56e+07
	Mean	6.14e+07	1.41e+07	2.48e+03	3.52e-02	7.87e+01	5.50e+02	2.91e+07
	Std	3.24e+07	1.39e+06	7.63e+01	1.76e-01	1.41e+01	9.78e+01	2.91e+06
MLCC	Best	4.51e+04	8.96e+07	2.52e+03	1.96e+02	2.42e+04	1.01e+03	2.62e+08
	Median	4.67e+07	1.24e+08	3.16e+03	1.98e+02	3.47e+04	1.91e+03	3.16e+08
	Worst	9.06e+07	1.46e+08	5.90e+03	1.98e+02	4.25e+04	3.47e+03	3.77e+08
	Mean	4.38e+07	1.23e+08	3.43e+03	1.98e+02	3.49e+04	2.08e+03	3.16e+08
	Std	3.45e+07	1.33e+07	8.72e+02	6.98e-01	4.92e+03	7.27e+02	2.77e+07
jDElsgo	Best	1.48e+03	4.04e+07	1.35e+03	3.72e-06	5.70e+04	6.43e+02	1.66e+08
	Median	1.32e+06	4.93e+07	1.75e+03	1.05e+00	7.16e+04	9.19e+02	2.07e+08
	Worst	7.33e+06	5.87e+07	2.21e+03	2.55e+00	1.32e+05	1.20e+03	2.44e+08
	Mean	2.08e+06	4.98e+07	1.78e+03	1.01e+00	7.52e+04	9.24e+02	2.01e+08
	Std	2.16e+06	5.68e+06	2.38e+02	4.76e-01	1.56e+04	1.49e+02	1.74e+07
		F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}	
DECC-G	Best	1.09e+04	5.97e+01	2.50e+05	5.61e+03	1.02e+06	3.59e+03	
	Median	1.18e+04	7.51e+01	2.89e+05	2.30e+04	1.11e+06	3.98e+03	
	Worst	1.39e+04	9.24e+01	3.26e+05	4.71e+04	1.20e+06	5.32e+03	
	Mean	1.22e+04	7.66e+01	2.87e+05	2.46e+04	1.11e+06	4.06e+03	
	Std	8.97e+02	8.14e+00	1.98e+04	1.05e+04	5.15e+04	3.66e+02	
DECC-G*	Best	3.62e+03	7.04e-08	8.09e+01	8.37e+02	9.90e+05	2.83e+03	
	Median	3.88e+03	1.04e-07	1.03e+02	1.08e+03	1.15e+06	3.21e+03	
	Worst	4.25e+03	2.18e+00	1.33e+02	1.53e+03	1.23e+06	6.23e+03	
	Mean	3.88e+03	4.01e-01	1.03e+02	1.08e+03	1.14e+06	3.33e+03	
	Std	1.76e+02	6.59e-01	1.38e+01	1.61e+02	5.85e+04	6.63e+02	
MLCC	Best	5.30e+03	2.08e+02	1.38e+05	2.51e+03	1.21e+06	1.70e+03	
	Median	6.89e+03	3.95e+02	1.59e+05	4.17e+03	1.36e+06	2.04e+03	
	Worst	1.04e+04	3.97e+02	1.86e+05	1.62e+04	1.54e+06	2.34e+03	
	Mean	7.11e+03	3.76e+02	1.59e+05	7.09e+03	1.36e+06	2.05e+03	
	Std	1.34e+03	4.71e+01	1.43e+04	4.77e+03	7.35e+04	1.80e+02	
jDElsgo	Best	2.08e+03	1.38e+01	3.17e+05	1.83e+03	1.08e+06	1.76e+03	
	Median	3.10e+03	2.57e+01	3.55e+05	2.38e+03	1.20e+06	1.96e+03	
	Worst	3.61e+03	5.05e+01	4.24e+05	2.91e+03	1.38e+06	2.37e+03	
	Mean	3.05e+03	2.75e+01	3.60e+05	2.35e+03	1.19e+06	2.00e+03	
	Std	3.21e+02	1.12e+01	3.15e+04	3.07e+02	7.52e+04	1.67e+02	

PC Configure:
System: GNU/Linux CPU: 2.5 GHz RAM: 4 GB
Language: Java Algorithm: jDElsgo
Compiler: GNU Java Compiler (gcj)

VII. Conclusions

In this paper the performance of the jDElsgo algorithm was evaluated on the set of benchmark functions provided for CEC 2010 special session on large-scale global optimization.

The algorithm is based on the *jDEdynNP-F* algorithm, and both of them use a self-adaptive control mechanism to change the control parameters (F and CR) during the optimization process, gradually reducing population size, and using a mechanism for changing the sign of F control parameter. Additionally, the jDElsgo algorithm uses a smaller value for F control parameter to improve currently best individual after the half of maximal number of function evaluations is exceeded.

The results of experiments in this paper give evidence that the jDElsgo algorithm is a competitive algorithm for high-dimensional real-parameter optimization problems. One idea for future research it is to apply cooperative coevolution methods to jDElsgo algorithm.

Acknowledgment

The authors would also like to acknowledge the efforts of the organizers of this session and availability of source code of benchmark functions.

References

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] Andrea Caponio, Ferrante Neri, and Ville Tirronen. Super-fit control adaptation in memetic differential evolution frameworks. *Soft Comput.*, 13(8):811–831, 2009.

[9] S. Das, A. Abraham, U.K. Chakraborty, and A. Konar. Differential Evolution Using a Neighborhood-based Mutation Operator. *IEEE Transactions on Evolutionary Computation*, 13(3), 2009.

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

[11] 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.

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

[13] 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.

[14] Ferrante Neri and Ville Tirronen. Scale factor local search in differential evolution. *Memetic Comp.*, 1(2):153–171, 2009.

[15] Ferrante Neri and Ville Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, 33(1–2):61–106, 2010.

[16] 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.

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

[18] 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, APR 2009.

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

[20] S. Rahnamayan, H.R. Tizhoosh, and M.M.A. Salama. Opposition-Based Differential Evolution. *IEEE Transactions on Evolutionary Computation*, 12(1):64–79, 2008.

[21] 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 (CEC 2002)*, pages 413–418. IEEE, 2002.

[22] 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.

[23] K. Tang, Xiaodong Li, P. N. Suganthan, Z. Yang, and T. Weise. Benchmark Functions for the CEC 2010 Special Session and Competition on Large Scale Global Optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2009.

[24] 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.

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

[26] Z. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.

[27] Z. Yang, K. Tang, and X. Yao. Multilevel Cooperative Coevolution for Large Scale Optimization. In *Proc. IEEE World Congress on Computational Intelligence (WCCI 2008)*, pages 1663–1670. IEEE Press, 2008.

[28] 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 3523–3530, Singapore, 25-28 September 2007. IEEE Computational Intelligence Society, IEEE Press.

[29] A. Zamuda, J. Brest, B. Bošković, and V. Žumer. Large Scale Global Optimization Using Differential Evolution with Self-adaptation and Cooperative Co-evolution. In *2008 IEEE World Congress on Computational Intelligence*, pages 3719–3726. IEEE Press, 2008.

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