Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems

Janez Brest, *Member, IEEE*, Sašo Greiner, Borko Bošković, Marjan Mernik, *Member, IEEE*, and Viljem Žumer, *Member, IEEE*

***Abstract*—We describe an efficient technique for adapting control parameter settings associated with differential evolution (DE). The DE algorithm has been used in many practical cases and has demonstrated good convergence properties. It has only a few control parameters, which are kept fixed throughout the entire evolutionary process. However, it is not an easy task to properly set control parameters in DE. We present an algorithm—a new version of the DE algorithm—for obtaining self-adaptive control parameter settings that show good performance on numerical benchmark problems. The results show that our algorithm with self-adaptive control parameter settings is better than, or at least comparable to, the standard DE algorithm and evolutionary algorithms from literature when considering the quality of the solutions obtained.**

***Index Terms*—Adaptive parameter control, differential evolution (DE), evolutionary optimization.**

I. Introduction

DIFFERENTIAL evolution (DE) is a simple yet powerful evolutionary algorithm (EA) for global optimization introduced by Price and Storn [1]. The DE algorithm has gradually become more popular and has been used in many practical cases, mainly because it has demonstrated good convergence properties and is principally easy to understand [2].

EAs [3] are a broad class of stochastic optimization algorithms inspired by biology and, in particular, by those biological processes that allow populations of organizms to adapt to their surrounding environments: genetic inheritance and survival of the fittest. EAs have a prominent advantage over other types of numerical methods. They only require information about the objective function itself, which can be either explicit or implicit. Other accessory properties such as differentiability or continuity are not necessary. As such, they are more flexible in dealing with a wide spectrum of problems.

When using an EA, it is also necessary to specify how candidate solutions will be changed to generate new solutions [4]. EA may have parameters, for instance, the probability of mutation, the tournament size of selection, or the population size. The values of these parameters greatly determine the quality of the solution obtained and the efficiency of the search [5]–[7]. Starting with a number of guessed solutions, the multipoint algorithm updates one or more solutions in a synergistic manner in the hope of steering the population toward the optimum [8], [9].

Choosing suitable parameter values is, frequently, a problem-dependent task and requires previous experience of the user. Despite its crucial importance, there is no consistent methodology for determining the control parameters of an EA, which are, most of the time, arbitrarily set within some predefined ranges [4].

In their early stage, EAs did not usually include control parameters as a part of the evolving object but considered them as external fixed parameters. Later, it was realized that in order to achieve optimal convergence, these parameters should be altered in the evolution process itself [5], [7].

The control parameters were adjusted over time by using heuristic rules, which take into account information about the progress achieved. However, heuristic rules, which might be optimal for one optimization problem, might be inefficient or even fail to guarantee convergence for another problem. A logical step in the development of EAs was to include control parameters into the evolving objects and allow them to evolve along with the main parameters [3], [10], [11].

Globally, we distinguish two major forms of setting parameter values: parameter *tuning* and parameter *control*. The former means the commonly practiced approach that tries to find good values for the parameters before running the algorithm, then tuning the algorithm using these values, which remain fixed during the run. The latter means that values for the parameters are changed during the run. According to Eiben *et al.* [5], [7], the change can be categorized into three classes.

1) *Deterministic parameter control* takes place when the value of a parameter is altered by some deterministic rule.
2) *Adaptive parameter control* is used to place when there is some form of feedback from the search that is used to determine the direction and/or the magnitude of the change to the parameter.
3) *Self-adaptive parameter control* is the idea that "evolution of the evolution" can be used to implement the self-adaptation of parameters. Here, the parameters to be adapted are encoded into the chromosome (individuals) and undergo the actions of genetic operators. The better values of these encoded parameters lead to better individuals which, in turn, are more likely to survive and produce offspring and, hence, propagate these better parameter values.

Manuscript received June 14, 2005; revised September 19, 2005 and November 9, 2005. This work was supported in part by the Slovenian Research Agency under Programme P2-0041, Computer Systems, Methodologies, and Intelligent Services.

The authors are with the Computer Architecture and Languages Laboratory, Institute of Computer Science, Faculty of Electrical Engineering and Computer Science, University of Maribor, SI-2000 Maribor, Slovenia (e-mail: janez.brest@uni-mb.si; saso.greiner@uni-mb.si; borko.boskovic@uni-mb.si; marjan.mernik@uni-mb.si; zumer@uni-mb.si).

Digital Object Identifier 10.1109/TEVC.2006.872133

Hence, it is seemingly natural to use an EA, not only for finding solutions to a problem but also for tuning the (same) algorithm to the particular problem. Technically speaking, we are trying to modify the values of parameters during the run of the algorithm by taking the actual search progress into account. As discussed in [5] and [7], there are two ways to do this. The first way is to use some heuristic rule which takes feedback from the current state of the search and modifies the parameter values accordingly (adaptive parameter control), such as the credit assignment process presented by [12]. A second way is to incorporate parameters into the chromosomes, thereby making them subject to evolution (self-adaptive parameter control) [13].

The proof of convergence of EAs with self-adaptation is difficult because control parameters are changed randomly and the selection does not affect their evolution directly [14], [15].

Since DE is a particular instance of EA, it is interesting to investigate how self-adaptivity can be applied to it. Until now, no research work on self-adaptivity in DE has been reported. First, we define a type of optimization problem.

In this paper, we will only concern ourselves with those optimization methods that use an objective function. In most cases, the objective function defines the optimization problem as a minimization task. To this end, the following investigation is further restricted to the minimization of problems. When the objective function is nonlinear and nondifferentiable, direct search approaches are the methods of choice [1]. In optimizing a function, an optimization algorithm aims to find $\mathbf{x}_{\min}$ such that $\forall \mathbf{x}$, $f(\mathbf{x}_{\min}) \leq f(\mathbf{x})$, where f does not need to be continuous but must be bounded. This paper only considers unconstrained function optimization.

DE is a floating point encoding an EA for global optimization over continuous spaces [2], [16], [17]. 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. DE has three parameters: amplification factor of the difference vector F, crossover control parameter CR, and population size NP. DE is also particularly easy to work with, having only a few control parameters, which are kept fixed throughout the entire optimization process [2], [16], [18]. Since the interaction of control parameters with the DE's performance is complex in practice, a DE user should select the initial parameter settings for the problem at hand from previous experiences or from literature. Then, the trial-and-error method has to be used for fine tuning the control parameters further. In practice, the optimization run has to be performed multiple times with different settings. In some cases, the time for finding these parameters is unacceptably long.

In our paper, the parameter control technique is based on the self-adaptation of two parameters (F and CR), associated with the evolutionary process. The main goal here is to produce a flexible DE, in terms of control parameters F and CR.

This paper introduces a novel approach to the self-adapting control parameter of DE. It gives some comparisons against several adaptive and nonadaptive methods for a set of test functions. The paper is organized as follows. Related work is described in Section II. The DE is briefly presented in Section III. Some suggested choices for the fixed settings of the control parameters from literature are collected in Section IV. In Section V, the proposed new version of the DE algorithm with self-adapted control parameters is described in detail. Benchmark functions are presented in Section VI. Experiments are then presented in Section VII. A comparison of the self-adaptive DE and DE algorithms with other EP algorithms is made, followed by an experiment on the parameter settings for the DE algorithm. Then experiments with F and CR values by the adaptive DE are presented, and finally a comparison of self-adaptive DE algorithm with fuzzy adaptive differential evolution algorithm is shown. In conclusion, some remarks are given in Section VIII.

II. Related Work

This section reviews papers that already compare DE with other instances of EAs, such as particle swarm optimization and genetic algorithms, as well as papers that compare a different extension of DE with the original DE. After that, we concentrate on papers that deal with parameter control in DE. In the end, we mention papers on EA that use similar benchmark functions as presented in this paper.

DE was proposed by Price and Storn [1], [18]. It is a very simple and straightforward strategy.

Vesterstroem *et al.* [19] compared the DE algorithm with particle swarm optimization (PSO) and EAs on numerical benchmark problems. DE outperformed PSO and EAs in terms of the solution's quality on most benchmark problems. The benchmark functions in [19] are similar to benchmark functions used in our paper.

Ali and Törn in [9] proposed new versions of the DE algorithm and also suggested some modifications to classical DE to improve its efficiency and robustness. They introduced an auxiliary population of NP individuals alongside the original population (noted in [9], a notation using sets is used—population set-based methods). Next, they proposed a rule for calculating the control parameter F automatically (see Section IV).

Sun *et al.* [20] proposed a combination of DE algorithms and the estimation of distribution algorithm (EDA), which tries to guide its search toward a promising area by sampling new solutions from a probability model. Based on experimental results, it has been demonstrated that the DE/EDA algorithm outperforms the DE algorithm and the EDA.

There are quite different conclusions about the rules for choosing the control parameters of DE. In [21], it is stated that the control parameters of DE are not difficult to choose. On the other hand, Gämperle *et al.* [22] reported that choosing the proper control parameters for DE is more difficult than expected.

Liu and Lampinen [2] reported that effectiveness, efficiency, and robustness of the DE algorithm are sensitive to the settings of the control parameters. The best settings for the control parameters can be different for different functions and the same function with different requirements for consumption time and accuracy.

However, there still exists a lack of knowledge on how to find reasonably good values for the control parameters of DE for a given function [16]. Liu and Lampinen [16] proposed a new version of DE, where the mutation control parameter and the crossover control parameter are adaptive. It is called the

fuzzy adaptive differential evolution (FADE) algorithm. It dynamically controls DE parameters F and/or CR. The FADE algorithm, especially when adapting F and CR, converges much faster than the traditional DE, particularly when the dimensionality of the problem is high or the problem concerned is complicated [16].

In this paper, we compare our version of a self-adaptive DE with the classical DE algorithm and with the FADE algorithm. A performance comparison is also made with EP algorithms, described in the following.

In [23], a "fast EP" (FEP) is proposed which uses a Cauchy, instead of Gaussian, mutation as the primary search operator. In [24], a further generalization of FEP is described by using mutation based on the Lévy probability distribution. With Lévy probability distribution, one can extend and generalize FEP because the Cauchy probability distribution is a special case of the Lévy probability distribution. The large variation at a single mutation enables Lévy mutation to discover a wider region of the search space globally [24]. The Lévy-mutated variables cover a wider range than those mutated by Gaussian distributions. Large variations of the mutated offspring can help to escape from local optima.

Finally, we give two more references which have dealt with function optimizations evaluated on some similar benchmark test functions. Tu *et al.* [25] suggest the use of the stochastic genetic algorithm (StGA), where the stochastic coding strategy is employed. The search space is explored region by region. Regions are dynamically created using a stochastic method. In each region, a number of children are produced through random sampling, and the best child is chosen to represent the region. The variance values are decreased if at least one of five generated children results in improved fitness; otherwise, the variance values are increased. The StGA codes each chromosome as a representative of a stochastic region described by a multivariate Gaussian distribution rather than a single candidate solution, as in the conventional GA. The paper [26] presents a technique for adapting control parameter settings associated with genetic operators using fuzzy logic controllers and coevolution.

III. DE Algorithm

There are several variants of DE [1], [18]. In this paper, we use the DE scheme which can be classified using notation [1], [18] as *DE/rand/1/bin* strategy. This strategy is the most often used in practice [1], [2], [20], [22] and can be described as follows.

A set of D optimization parameters is called an individual. It is represented by a D-dimensional parameter vector. A population consists of NP parameter vectors $\mathbf{x}_{i,G}$, $i = 1, 2, \ldots, NP$. G denotes one generation. We have one population for each generation.

NP is the number of members in a population. It is not changed during the minimization process. The initial population is chosen randomly with uniform distribution.

According to Storn and Price [1], [18], we have three operations: mutation, crossover, and selection.

The crucial idea behind DE is a scheme for generating trial parameter vectors. Mutation and crossover are used to generate new vectors (trial vectors), and selection then determines which of the vectors will survive into the next generation.

A. Mutation

For each target vector $\mathbf{x}_{i,G}$, a mutant vector $\mathbf{v}$ is generated according to

$$\mathbf{v}_{i,G+1} = \mathbf{x}_{r_1,G} + F\left(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}\right), \quad r_1 \neq r_2 \neq r_3 \neq i$$

with randomly chosen indexes $r_1, r_2, r_3 \in [1, NP]$. Note that indexes have to be different from each other and from the running index i so that NP must be at least four. F is a real number ($F \in [0, 2]$) that controls the amplification of the difference vector $(\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G})$.

If a component of a mutant vector goes off the box S, then this component is set to bound value. The same "solution" is used by classic DE too.

B. Crossover

The target vector is mixed with the mutated vector, using the following scheme, to yield the trial vector

$$\mathbf{u}_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Di,G+1})$$

where

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & \text{if } r(j) \leq CR \text{ or } j = rn(i) \\ x_{ji,G}, & \text{if } r(j) > CR \text{ and } j \neq rn(i) \end{cases}$$

for $j = 1, 2, \ldots, D$. $r(j) \in [0, 1]$ is the jth evaluation of a uniform random generator number. CR is the crossover constant $\in [0, 1]$, which has to be determined by the user. $rn(i) \in (1, 2, \ldots, D)$ is a randomly chosen index which ensures that $\mathbf{u}_{i,G+1}$ gets at least one element from $\mathbf{v}_{i,G+1}$. Otherwise, no new parent vector would be produced and the population would not alter.

C. Selection

A greedy selection scheme is used

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G+1}, & \text{if } f(\mathbf{u}_{i,G+1}) < f(\mathbf{x}_{i,G}) \text{for} \\ & \quad \text{minimization problems} \\ \mathbf{x}_{i,G}, & \text{otherwise} \end{cases}$$

for $j = 1, 2, \ldots, D$. If, and only if, the trial vector $\mathbf{u}_{i,G+1}$ yields a better cost function value than $\mathbf{x}_{i,G}$, then $\mathbf{x}_{i,G+1}$ is set to $\mathbf{u}_{i,G+1}$; otherwise, the old value $\mathbf{x}_{i,G}$ is retained.

IV. Control Parameter Settings for DE Algorithm

According to Storn *et al.* [1], [18], DE is much more sensitive to the choice of F than it is to the choice of CR.

The suggested choices by Storn in [1] and [16] are:

1) $F \in [0.5, 1]$;
2) $CR \in [0.8, 1]$;
3) $NP = 10 \cdot D$.

Recall that D is the dimensionality of the problem.

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

Fig. 1. Self-adapting: encoding aspect.

Liu and Lampinen in [16] used control parameters set to $F = 0.9$, $CR = 0.9$. The values were chosen based on discussions in [21].

Ali and Törn in [9] empirically obtained an optimal value for CR. They used $CR = 0.5$. F was calculated according to the following scheme:

$$F = \begin{cases} \max\left(l_{\min}, 1 - \left|\frac{f_{\max}}{f_{\min}}\right|\right), & \text{if } \left|\frac{f_{\max}}{f_{\min}}\right| < 1 \\ \max\left(l_{\min}, 1 - \left|\frac{f_{\min}}{f_{\max}}\right|\right), & \text{otherwise} \end{cases}$$

ensuring that $F \in [l_{\min}, 1]$. $f_{\max}$ and $f_{\min}$ are the maximum and minimum values of vectors $\mathbf{x}_{i,G}$, respectively. $l_{\min}$ is the lower bound for F. In [9], $l_{\min} = 0.4$ is used.

In our paper, we use a self-adaptive control mechanism to change the control parameters F and CR during the run. The third control parameter NP is not changed during the run.

V. Self-Adapting Parameters—New Version of DE Algorithm

Choosing suitable control parameter values is, frequently, a problem-dependent task. The trial-and-error method used for tuning the control parameters requires multiple optimization runs. In this section, we propose a self-adaptive approach for control parameters. Each individual in the population is extended with parameter values. In Fig. 1, the control parameters that will be adjusted by means of evolution are F and CR. Both of them are applied at the individual level. 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.

The solution (Fig. 1) is represented by a D-dimensional vector $\mathbf{x}_{i,G}$, $i = 1, 2, \ldots, NP$. New control parameters or factors $F_{i,G+1}$ and $CR_{i,G+1}$ are calculated as

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

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

and they produce factors F and CR in a new parent vector. $rand_j$, $j \in \{1, 2, 3, 4\}$ are uniform random values $\in [0, 1]$. τ_1 and τ_2 represent probabilities to adjust factors F and CR, respectively. In our experiments, we set $\tau_1 = \tau_2 = 0.1$. Because $F_l = 0.1$ and $F_u = 0.9$, the new F takes a value form [0.1,1.0] in a random manner. The new CR takes a value from [0,1]. $F_{i,G+1}$ 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 $\mathbf{x}_{i,G+1}$.

We have made a decision about the range for F, which is determined by values F_l and F_u, based on the suggested values by other authors and based on the experimental results. In the literature, F is rarely greater than one. If control parameter $F = 0$, the new trial vector is generated using crossover but no mutation; therefore, we propose $F_l = 0.1$.

The classic DE has three control parameters that need to be adjusted by the user. It seems that our self-adaptive DE has even more parameters, but please note that we have used fixed values for F_l, F_u, τ_1, and τ_2 for all benchmark functions in our self-adaptive DE algorithm. The user does not need to adjust those (additional) parameters.

Suitable control parameters are different for different function problems. Which are the best values of control parameters and how could we get them? Are there any universal directions on how to get *good* initial values for control parameters? In our method, the algorithm can change control parameters with some probabilities (τ_1 and τ_2) and after that, better control parameters are used in the next generations.

We have made additional experiments with some combinations with τ_1 and τ_2 using values: 0.05, 0.1, 0.2, and 0.3, and we did not notice any significant difference in results. Therefore, we peaked at $\tau_1 = \tau_2 = 0.1$, and those values were used in this paper.

The main contribution of our approach is that user does not need to guess the *good* values for F and CR, which are problem dependent. The rules for self-adapting control parameters F and CR are quite simple; therefore, the new version of the DE algorithm does not increase the time complexity, in comparison to the original DE algorithm.

VI. Benchmark Functions

Twenty-one benchmark functions from [23] were used to test the performance of our DE algorithm to assure a fair comparison. If the number of test problems were smaller, it would be very difficult to make a general conclusion. Using a test set which is too small also has the potential risk that the algorithm is biased (optimized) toward the chosen set of problems. Such bias might not be useful for other problems of interest. The benchmark functions are given in Table I. D denotes the dimensionality of the test problem, S denotes the ranges of the variables, and $f_{\min}$ is a function value of the global optimum. A more detailed description of each function is given in [23] and [24], where the functions were divided into three classes: functions with no local minima, many local minima, and a few local minima.

Functions $f_1 - f_{13}$ are high-dimensional problems. Functions $f_1 - f_5$ are unimodal. Function f_6 is the step function which has one minimum and is discontinuous. Function f_7 is a noisy quadratic function. Functions $f_8 - f_{13}$ are multimodal functions where the number of local minima increases exponentially with the problem dimension [23], [27]. Functions $f_{14} - f_{21}$ are low-dimensional functions which have only a few local minima [23], [27].

Yao *et al.* [23] described the benchmark functions and convergence rates of algorithms, as follows. For unimodal functions, the convergence rates of FEP and classical EP (CEP) algorithms are more interesting than the final results of optimization, as there are other methods which are specifically designed to optimize unimodal functions. For multimodal functions, the final

TABLE I
BENCHMARK FUNCTIONS

Test function	D	S	f_{min}
$f_1(x) = \sum_{i=1}^{D} x_i^2$	30	$[-100, 100]^D$	0
$f_2(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	30	$[-10, 10]^D$	0
$f_3(x) = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$	30	$[-100, 100]^D$	0
$f_4(x) = \max_i \{\lvert x_i \rvert, 1 \le i \le D\}$	30	$[-100, 100]^D$	0
$f_5(x) = \sum_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	$[-30, 30]^D$	0
$f_6(x) = \sum_{i=1}^{D} (\lfloor x_i + 0.5 \rfloor)^2$	30	$[-100, 100]^D$	0
$f_7(x) = \sum_{i=1}^{D} i x_i^4 + random[0, 1)$	30	$[-1.28, 1.28]^D$	0
$f_8(x) = \sum_{i=1}^{D} -x_i \sin(\sqrt{\lvert x_i \rvert})$	30	$[-500, 500]^D$	-12569.5
$f_9(x) = \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i) + 10]$	30	$[-5.12, 5.12]^D$	0
$f_{10}(x) = -20 \exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D} \cos 2\pi x_i\right) + 20 + e$	30	$[-32, 32]^D$	0
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600, 600]^D$	0
$f_{12}(x) = \frac{\pi}{D}\{10\sin^2(\pi y_i) + \sum_{i=1}^{D-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})]$ $+(y_D - 1)^2\} + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$, $y_i = 1 + \frac{1}{4}(x_i + 1)$, $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \le x_i \le a, \\ k(-x_i - a)^m, & x_i < -a. \end{cases}$	30	$[-50, 50]^D$	0
$f_{13}(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{D-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})]$ $+(x_D - 1)^2[1 + \sin^2(2\pi x_D)]\} + \sum_{i=1}^{D} u(x_i, 5, 100, 4)$	30	$[-50, 50]^D$	0
$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$	2	$[-65.536, 65.536]^D$	0.998004
$f_{15}(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	4	$[-5, 5]^D$	0.0003075
$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^D$	-1.0316285
$f_{17}(x) = (x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos x_1 + 10$	2	$[-5, 10]x[0, 15]$	0.398
$f_{18}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $\times[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	2	$[-2, 2]^D$	3
$f_{19}(x) = -\sum_{i=1}^{5}[(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^D$	-10.1532
$f_{20}(x) = -\sum_{i=1}^{7}[(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^D$	-10.4029
$f_{21}(x) = -\sum_{i=1}^{10}[(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^D$	-10.5364

results are much more important since they reflect the algorithm's ability to escape from poor local optima and locate a good near-global optimum.

VII. EXPERIMENTAL RESULTS

We applied self-adaptive DE and (original) DE to a set of benchmark optimization problems.

The initial population was generated uniformly at random in the range, as specified in Table I.

Throughout this paper, we have used $F = 0.5$ and $CR = 0.9$ for the (original) DE algorithm. Our decision for using those values is based on proposed values from literature [1], [9], [16], [19].

A. Comparison of Self-Adaptive DE and DE Algorithms With FEP and CEP Algorithms

In the experiment, we set the parameters as in [23] for fair performance comparison. The following parameters were used in our experiment:

1) population size 100;
2) maximum number of generations: 1500 for f_1, f_6, f_{10}, f_{12}, and f_{13}, 2000 for f_2 and f_{11}, 3000 for f_7, 4000 for f_{15}, 5000 for f_3, f_4, and f_9, 9000 for f_8, 20 000 for f_3, and 100 for $f_{14}, f_{16}, \ldots f_{21}$.

Therefore, in our experiment, self-adaptive DE and DE used the same population size as in [23] and the same stopping criteria (i.e., equal number of function evaluations).

The average results of 50 independent runs are summarized in Table II. Results for the FEP and CEP algorithms are taken from [23, Tables II–IV].

The comparison shows that self-adaptive DE gives better results on benchmark functions than FEP and CEP. Self-adaptive DE algorithm performs better than DE, while DE does not always perform better than FEP and CEP.

When Compared With IFEP: Yao *et al.* in [23] proposed an improved FEP (IFEP) based on mixing (rather than switching) different mutation operators. IFEP generates two candidate offspring from each parent, one by Cauchy mutation and one by

TABLE II
EXPERIMENTAL RESULTS, AVERAGED OVER 50 INDEPENDENT RUNS, OF SELF-ADAPTIVE DE, DE, FEP, AND CEP ALGORITHMS. "MEAN BEST" INDICATES AVERAGE OF MINIMUM VALUES OBTAINED AND "STD DEV" STANDS FOR STANDARD DEVIATION. t-TEST TESTS SELF-ADAPTIVE DE AGAINST OTHER ALGORITHMS, RESPECTIVELY

F	#Gen.	Self-Adaptive DE	DE	FEP	CEP
		Mean Best (Std Dev)	Mean Best (Std Dev)	Mean Best (Std Dev)	Mean Best (Std Dev)
f_1	1500	1.1×10^{-28} (1.0×10^{-28})	8.2×10^{-14} (5.9×10^{-14})†	5.7×10^{-4} (1.3×10^{-4})†	2.2×10^{-4} (5.9×10^{-4})†
f_2	2000	1.0×10^{-23} (9.7×10^{-24})	1.5×10^{-9} (9.9×10^{-10})†	8.1×10^{-3} (7.7×10^{-4})†	2.6×10^{-3} (1.7×10^{-4})†
f_3	5000	3.1×10^{-14} (5.9×10^{-14})	6.8×10^{-11} (7.4×10^{-11})†	1.6×10^{-2} (1.4×10^{-2})†	5.0×10^{-2} (6.6×10^{-2})†
f_4	5000	0 (0)	0 (0)	0.3 (0.5)†	2.0 (1.2)†
f_5	20000	0 (0)	0 (0)	5.06 (5.87)†	6.17 (13.61)†
f_6	1500	0 (0)	0 (0)	0 (0)	577.76 (1125.76)†
f_7	3000	3.15×10^{-3} (7.5×10^{-4})	4.63×10^{-3} (1.2×10^{-3})	7.6×10^{-3} (2.6×10^{-3})†	1.8×10^{-3} (6.4×10^{-3})
f_8	9000	-12569.5 (7.0×10^{-12})	-11080.1 (574.7)†	-12554.5 (52.6)†	-7917.1 (634.5)†
f_9	5000	0 (0)	69.2 (38.8)†	4.6×10^{-2} (1.2×10^{-2})†	89.0 (23.1)†
f_{10}	1500	7.7×10^{-15} (1.4×10^{-15})	9.7×10^{-8} (4.2×10^{-8})†	1.8×10^{-2} (2.1×10^{-3})†	9.2 (2.8)†
f_{11}	2000	0 (0)	0 (0)	1.6×10^{-2} (2.2×10^{-2})†	8.6×10^{-2} (0.12)†
f_{12}	1500	6.6×10^{-30} (7.9×10^{-30})	7.9×10^{-15} (8.0×10^{-15})†	9.2×10^{-6} (3.6×10^{-6})†	1.76 (2.4)†
f_{13}	1500	5.0×10^{-29} (3.9×10^{-29})	5.1×10^{-14} (4.8×10^{-14})†	1.6×10^{-4} (7.3×10^{-5})†	1.4 (3.7)†
f_{14}	100	0.998004 (2.6×10^{-16})	0.998004 (3.3×10^{-16})	1.22 (0.56)†	1.66 (1.19)†
f_{15}	4000	4.0×10^{-4} (2.7×10^{-4})	4.5×10^{-4} (3.3×10^{-4})	5.0×10^{-4} (3.2×10^{-4})	4.7×10^{-4} (3.0×10^{-4})
f_{16}	100	-1.03163 (9.7×10^{-12})	-1.03163 (3.1×10^{-13})	-1.03 (4.9×10^{-7})	-1.03 (4.9×10^{-7})
f_{17}	100	0.397887 (2.3×10^{-8})	0.397887 (9.9×10^{-9})	0.398 (1.5×10^{-7})	0.398 (1.5×10^{-7})
f_{18}	100	3 (1.7×10^{-15})	3 (2.0×10^{-15})	3.02 (0.11)	3.0 (0)
f_{19}	100	-10.1532 (2.2×10^{-6})	-10.1532 (2.5×10^{-6})	-5.52 (1.59)	-6.86 (2.67)
f_{20}	100	-10.4029 (4.9×10^{-7})	-10.4029 (3.9×10^{-7})	-5.52 (2.12)	-8.27 (2.95)
f_{21}	100	-10.5364 (5.8×10^{-6})	-10.5364 (1.9×10^{-7})	-6.57 (3.14)	-9.10 (2.92)

† The t value of 49 degree of freedom is significant at a 0.05 level of significance by two-tailed t-test.

Gaussian mutation. The better one is then chosen as the offspring. IFEP has improved FEP's performance significantly.

If we compared self-adaptive DE with IFEP taken from [23, Table X], it is clear that self-adaptive DE is certainly better than IFEP, too.

Many test functions take their minimum in the middlepoint of S. Three additional experiments for high-dimensional problems ($f_1 - f_{13}$) were performed to make sure that our algorithm performs well, too, if S was not symmetrical about the point where the objective function takes its minimum: 1) middle point is shifted; 2) lower bound was set to zero; and 3) upper bound was set to zero. Albeit no systematical experiments have been carried out, it can be observed, according to preliminary results, that our approach is not significantly influenced when function does not take its minimum in the middlepoint of S.

B. Comparison of Self-Adaptive DE and DE Algorithms With Adaptive LEP and Best Lévy Algorithms

In the experiment, we used the same function set and the parameters as in [24]. The following parameters were used in our experiments:

1) population size 100;
2) maximum number of generations: 1500 for $f_1, f_3, f_5, f_8, f_9, \ldots, f_{13}$, 30 for f_{16} and f_{18}, and 100 for f_{19}, f_{20}, and f_{21}.

Table III summarizes the average results of 50 independent runs. A comparison with results from [24] is made. It is clear that no algorithm performs superiorly better than others, but on average self-adaptive DE performs better than the other algorithms.

For the unimodal functions f_1 and f_5, both self-adaptive DE and DE are better than adaptive LEP and Best Lévy. For function f_3, adaptive LEP performs better than self-adaptive DE. The t-test shows a statistically significant difference (please note, in Table II, self-adaptive DE gives good results when number of generations is 5000). Adaptive LEP and self-adaptive DE outperform DE and Best Lévy.

For the multimodal functions with many local minima, i.e., $f_8 - f_{13}$, it is clear that the best results are obtained by self-adaptive DE. Interestingly, DE is worse than adaptive LEP and Best Lévy for functions f_8 and f_9 and better for functions $f_{10} - f_{13}$.

For the functions f_{16} and f_{18} with only a few local minima, the dimension of the functions is also small. In this case, it is hard to judge the performances of individual algorithms. All algorithms were able to find optimal solutions for these two functions.

For functions $f_{19} - f_{21}$, there is no superior algorithm either. For f_{19}, self-adaptive DE and DE are better than adaptive LEP and Best Lévy. There are similar algorithm performances for functions f_{20} and f_{21}, except adaptive LEP, which performed slightly worse for function f_{20}.

Fig. 2 shows average best fitness curves for the self-adaptive DE algorithm with over 50 independent runs for selected benchmark functions f_1, f_8, f_9, f_{11}.

TABLE III
EXPERIMENTAL RESULTS, AVERAGED OVER 50 INDEPENDENT RUNS, OF SELF-ADAPTIVE DE, DE, ADAPTIVE LEP, AND BEST OF FOUR NONADAPTIVE LEP ALGORITHMS (BEST LÉVY). "MEAN BEST" INDICATES AVERAGE OF MINIMUM VALUES OBTAINED AND "STD DEV" STANDS FOR STANDARD DEVIATION. t-TEST TESTS SELF-ADAPTIVE DE AGAINST OTHER ALGORITHMS, RESPECTIVELY

F	#Gen.	Self-Adaptive DE Mean Best (Std Dev)	DE Mean Best (Std Dev)	Adaptive LEP Mean Best (Std Dev)	Best Lévy Mean Best (Std Dev)
f_1	1500	1.1×10^{-28} (1.0×10^{-28})	8.2×10^{-14} $(5.9 \times 10^{-14})^{\dagger}$	6.32×10^{-4} $(7.6 \times 10^{-5})^{\dagger}$	6.59×10^{-4} $(6.4 \times 10^{-5})^{\dagger}$
f_3	1500	0.090075 (0.080178)	1.630860 $(0.886153)^{\dagger}$	0.041850 $(0.059696)^{\ddagger}$	30.628906 $(22.113122)^{\dagger}$
f_5	1500	3.1×10^{-15} (8.3×10^{-15})	7.8×10^{-9} $(5.8 \times 10^{-09})^{\dagger}$	43.40 $(31.52)^{\dagger}$	57.75 $(41.60)^{\dagger}$
f_8	1500	-12569.5 (7.3×10^{-12})	-6304.4 $(643)^{\dagger}$	-11469.2 $(58.2)^{\dagger}$	-11898.9 $(52.2)^{\dagger}$
f_9	1500	1.5×10^{-15} (4.8×10^{-15})	173.405 $(13.841)^{\dagger}$	5.85 $(2.07)^{\dagger}$	12.50 $(2.29)^{\dagger}$
f_{10}	1500	7.7×10^{-15} (1.4×10^{-15})	9.7×10^{-8} $(4.2 \times 10^{-8})^{\dagger}$	1.9×10^{-2} $(1.0 \times 10^{-3})^{\dagger}$	3.1×10^{-2} $(2.0 \times 10^{-3})^{\dagger}$
f_{11}	1500	0 (0)	2.9×10^{-13} $(4.2 \times 10^{-13})^{\dagger}$	2.4×10^{-2} $(2.8 \times 10^{-2})^{\dagger}$	1.8×10^{-2} $(1.7 \times 10^{-2})^{\dagger}$
f_{12}	1500	6.6×10^{-30} (7.9×10^{-30})	7.9×10^{-15} $(8.0 \times 10^{-15})^{\dagger}$	6.0×10^{-6} $(1.0 \times 10^{-6})^{\dagger}$	3.0×10^{-5} $(4.0 \times 10^{-6})^{\dagger}$
f_{13}	1500	5.0×10^{-29} (3.9×10^{-29})	5.1×10^{-14} $(4.8 \times 10^{-14})^{\dagger}$	9.8×10^{-5} $(1.2 \times 10^{-5})^{\dagger}$	2.6×10^{-4} $(3.0 \times 10^{-5})^{\dagger}$
f_{16}	30	-1.0316 (5.7×10^{-5})	-1.0316 (3.2×10^{-5})	-1.031 $(0.0)^{\dagger}$	-1.031 $(0.0)^{\dagger}$
f_{18}	30	3.00006 (7.6×10^{-5})	3.00002 (2.0×10^{-5})	3.000 (0.000)	3.000 (0.000)
f_{19}	100	-10.1532 (2.2×10^{-6})	-10.1532 (2.5×10^{-6})	-9.54 $(1.69)^{\dagger}$	-9.95 (0.99)
f_{20}	100	-10.4029 (4.9×10^{-7})	-10.4029 (3.9×10^{-7})	-10.30 (0.74)	-10.40 (1.0×10^{-4})
f_{21}	100	-10.5364 (5.8×10^{-6})	-10.5364 (1.9×10^{-7})	-10.54 (4.9×10^{-5})	-10.54 (3.1×10^{-3})

$^{\dagger},^{\ddagger}$ The t value of 49 degree of freedom is significant at a 0.05 level of significance by two-tailed t-test. ‡ stands for negative t value, which means that corresponding algorithm is better than self-adaptive algorithm.

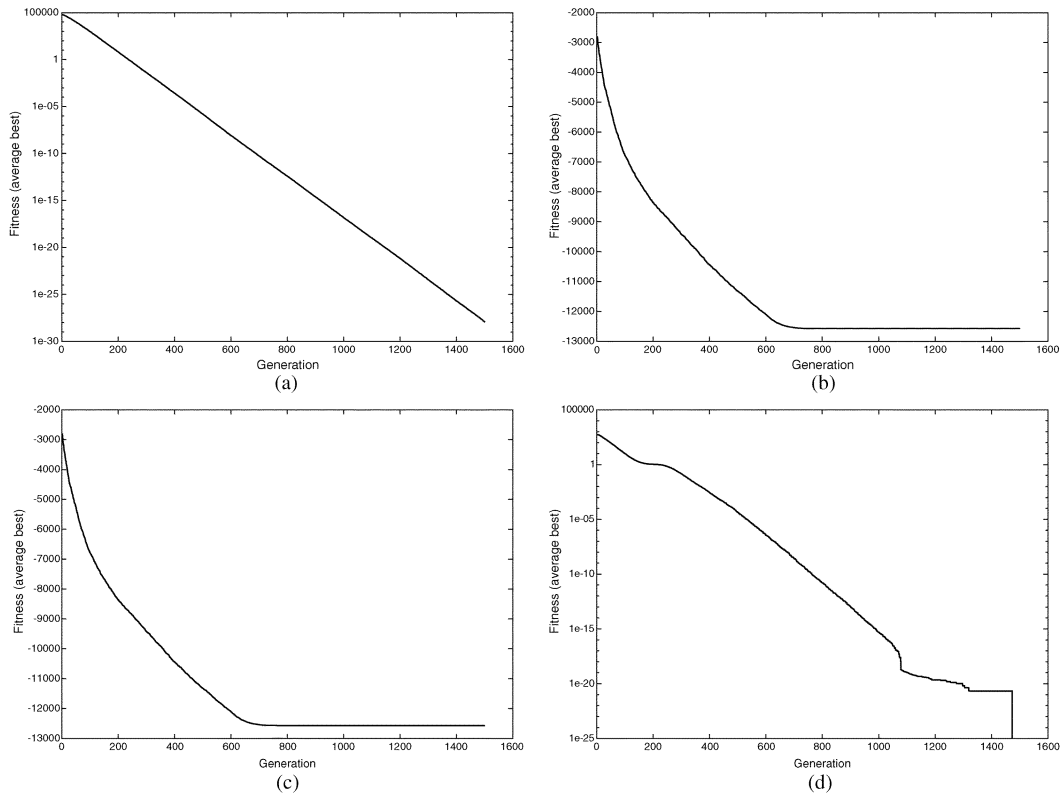


Fig. 2. Average best fitness curves of self-adaptive DE algorithm for selected benchmark functions. All results are means of 50 runs. (a) Test function f_1. (b) Test function f_8. (c) Test function f_9. (d) Test function f_{11}.

C. Discussion on Control Parameter Settings for DE Algorithm

In order to compare our self-adaptive version of DE algorithm with the DE algorithm, the *best* control parameter settings for DE may be needed. DE algorithm does not change control parameter values during optimization process.

For all benchmark function problems, the DE algorithm was performed with F and CR taken from [0.0, 0.95] by step 0.05. First, we set control parameters $F = 0.0$ and $CR = 0.0$ and kept them fixed during 30 independent runs. Then, we set $F = 0.05$ and $CR = 0.0$ for the next 30 runs, etc. The other

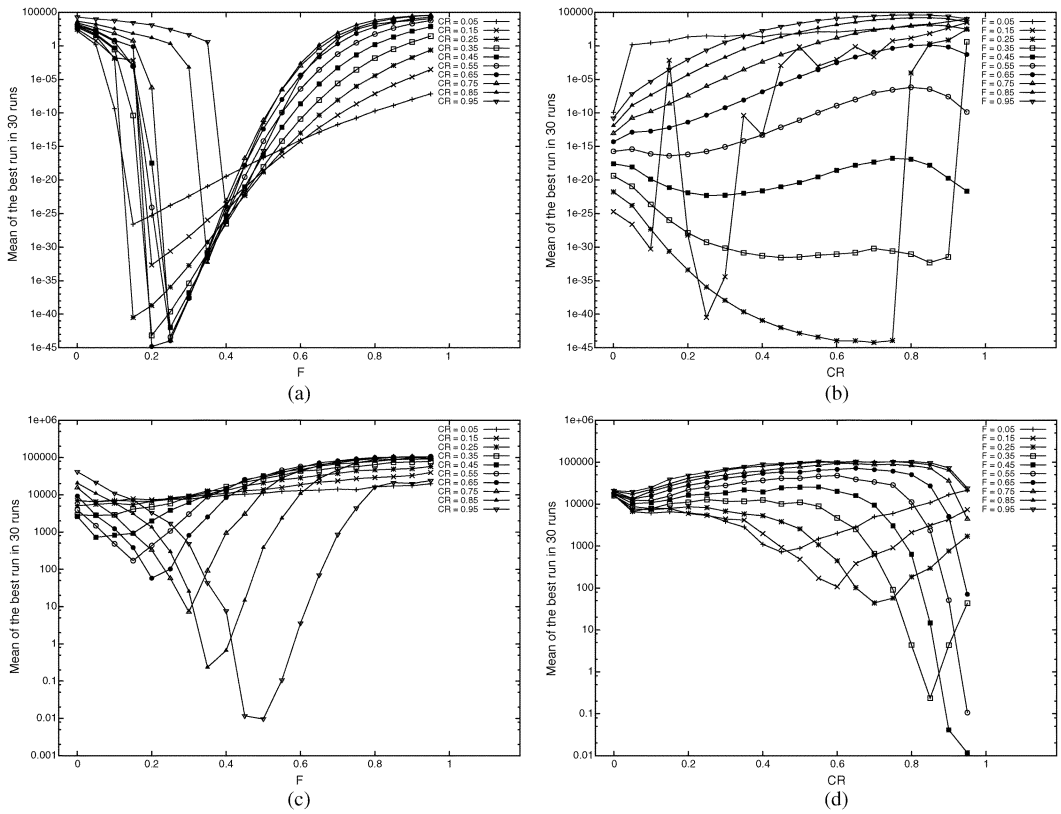


Fig. 3. Evolutionary processes of DE for functions f_1 and f_3. Results were averaged over 30 independent runs. (a) Test function f_1. (b) Test function f_1. (c) Test function f_3. (d) Test function f_3.

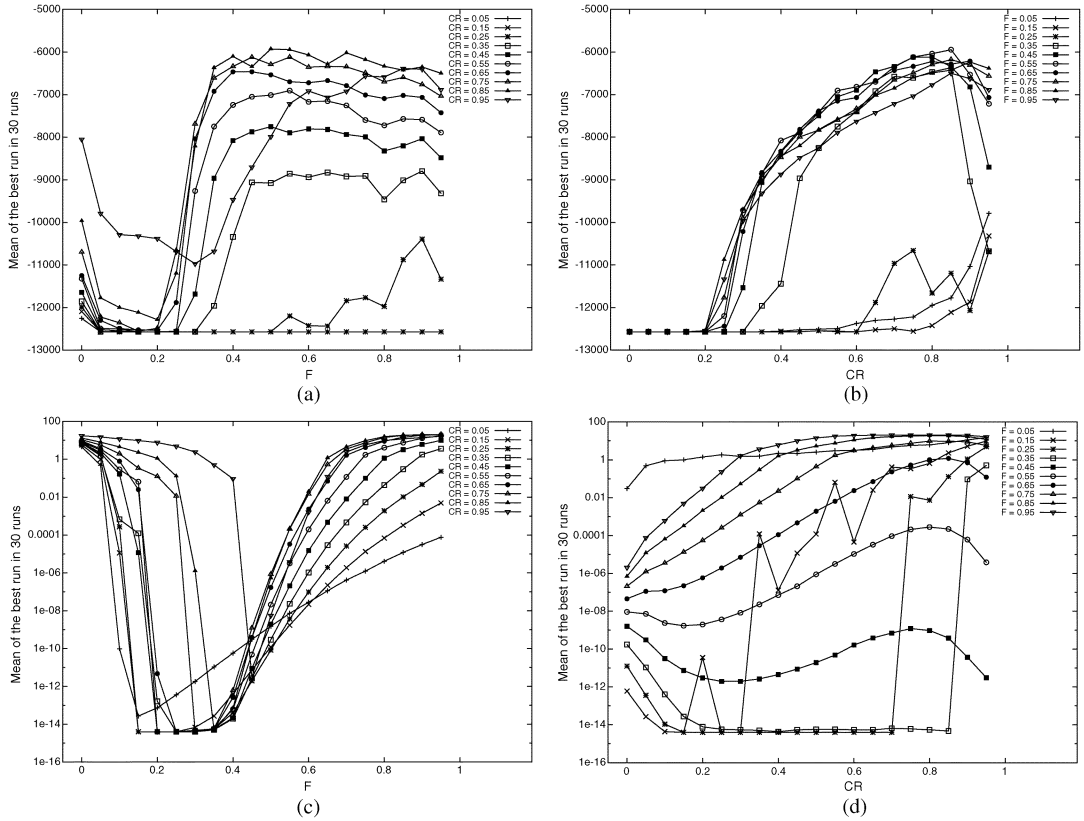


Fig. 4. Evolutionary processes of DE for functions f_8 and f_{10}. Results were averaged over 30 independent runs. (a) Test function f_8. (b) Test function f_8. (c) Test function f_{10}. (d) Test function f_{10}.

(parameter) settings were the same as proposed in Section VII-B. The results were averaged over 30 independent runs. The selected function problems are depicted in Figs. 3 and 4.

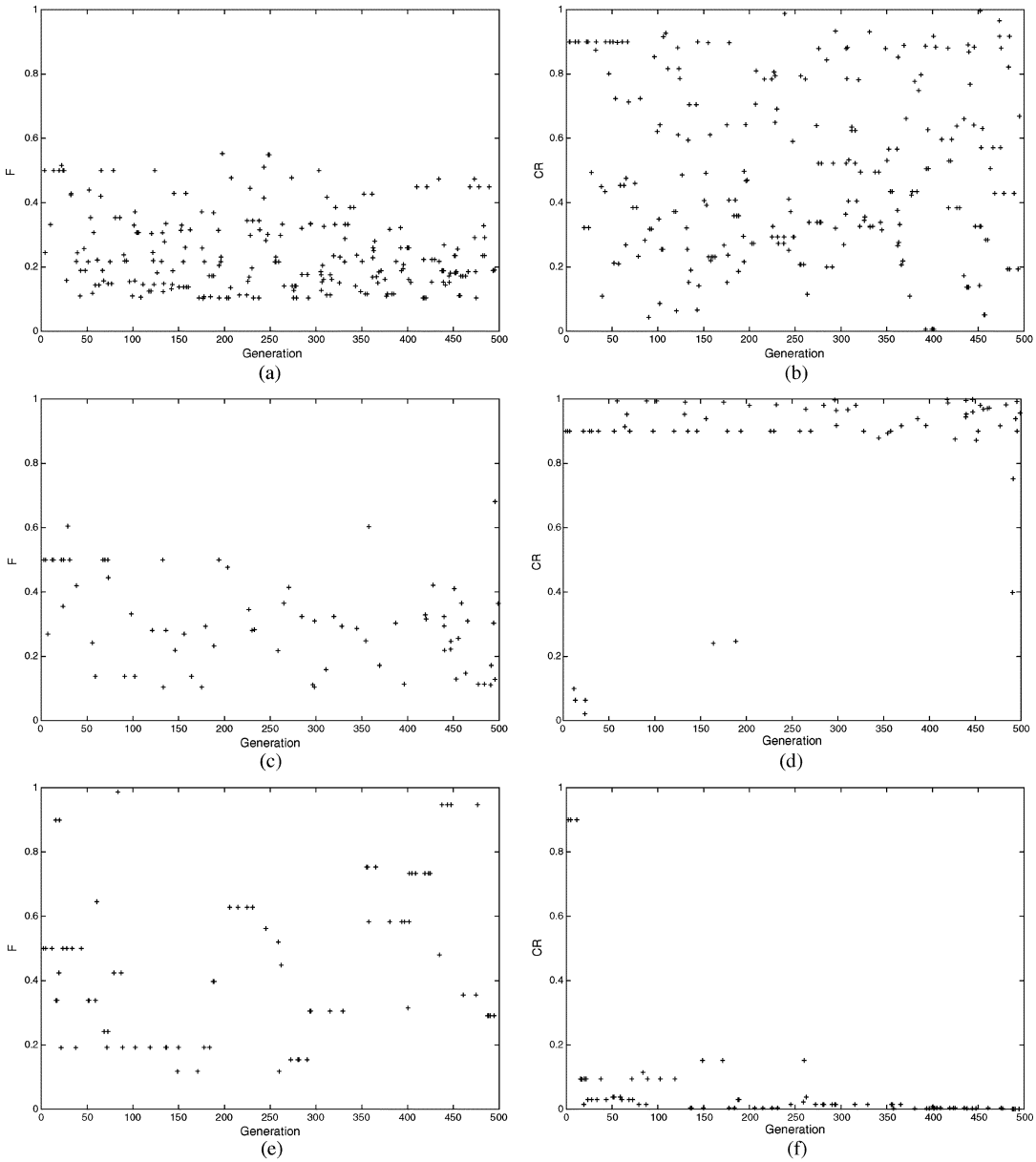


Fig. 5. CR and F values by self-adaptive DE for functions f_1, f_3, and f_8, respectively. Dot is plotted when best fitness value in generation is improved. (a) Test function f_1. (b) Test function f_1. (c) Test function f_3. (d) Test function f_3. (e) Test function f_8. (f) Test function f_8.

For function f_1, the *good* control values F and CR are from [0.15, 0.5] [Fig. 3(a)] and [0.4, 0.75] [Fig. 3(b)], respectively. The best averaged fitness value for function f_1 was obtained by $F = 0.2$ and $CR = 0.65$ (number of generation was 1500, and $NP = 100$). The best averaged fitness value for function f_3 was obtained by $F = 0.5$ and $CR = 0.95$ (for f_3 high values for CR give better results).

It is very interesting that apparently there are CR values that make F a *sensitive* parameter (where the mean best depends on the value of F), and there are CR values that make F a *robust* parameter (where the mean best does not depend on the value of F).

There are two disadvantages in DE. Parameter tuning requires multiple runs and it is usually not a feasible solution for problems which are very time consuming. The best control parameter settings of DE are problem dependent. The proposed self-adaptive DE overcomes those disadvantages, so there is no need for multiple runs to adjust control parameters, and self-adaptive DE is much more problem independent than DE.

D. F and CR Values for Self-Adaptive DE

In self-adaptive DE, F and CR values are being changed during evolutionary process. If we want to look into an evolutionary process, we should look at fitness curves. The most important is the best fitness curve.

For the selected functions f_1, f_3, f_8, f_9, and f_{11}, F and CR values are depicted in Figs. 5 and 6 only when the best fitness value in generation is improved. For example, most of the CR values for functions f_8 and f_9 are lower than 0.2, while for function f_3 they are greater than 0.8. If we know that $CR = 0.95$ is good for function f_3, we can use this "knowledge" in initialization by DE and also by our self-adaptive DE.

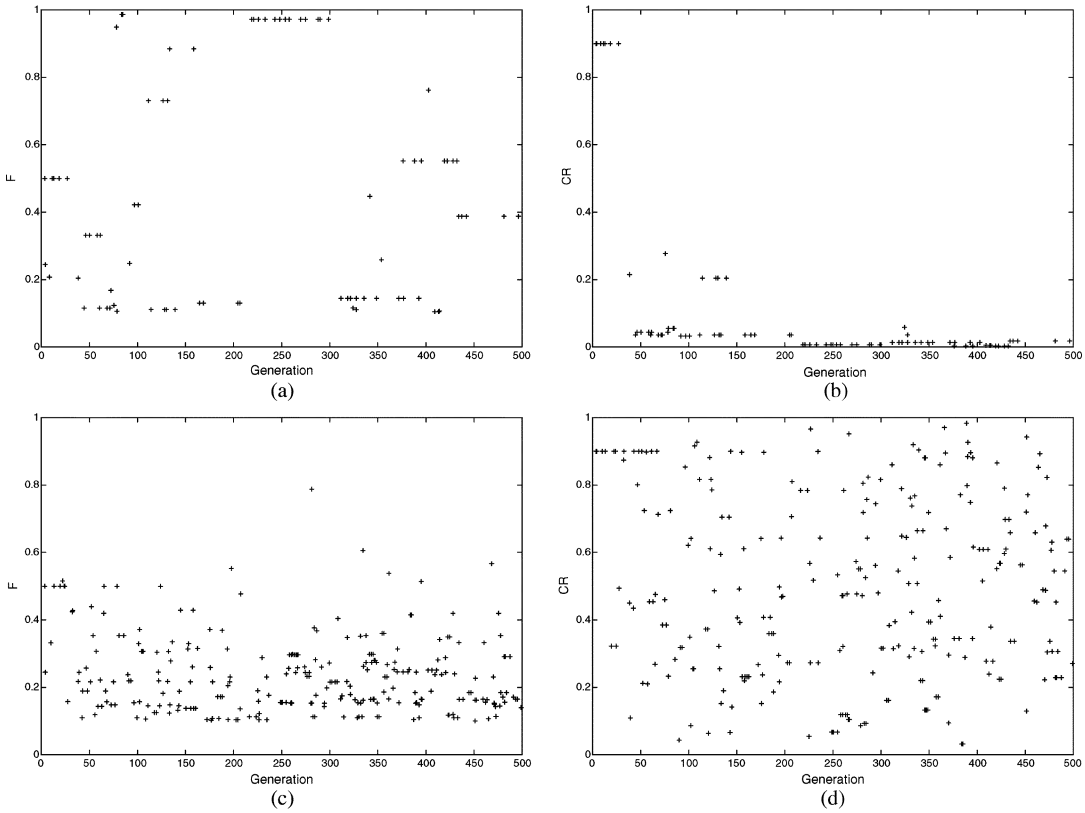


Fig. 6. CR and F values by self-adaptive DE for functions f_9 and f_8, respectively. Dot is plotted when best fitness value in generation is improved. (a) Test function f_9. (b) Test function f_9. (c) Test function f_{11}. (d) Test function f_{11}.

TABLE IV
EXPERIMENTAL RESULTS, AVERAGED OVER 50 INDEPENDENT RUNS, OF SELF-ADAPTIVE DE WITH DIFFERENT INITIAL F AND CR VALUES FOR SELECTED BENCHMARK FUNCTIONS

F	#Gen.	$F = 0.1, CR = 0.1$	$F = 0.5, CR = 0.5$	$F = 0.5, CR = 0.9$	$F = 0.9, CR = 0.9$	$F \in [0.1, 1.0], CR \in [0.0, 1.0]$
f_1	1500	3.2×10^{-29}	1.1×10^{-28}	1.1×10^{-28}	2.5×10^{-28}	1.9×10^{-28}
f_3	1500	8.5×10^{-2}	4.8×10^{-1}	9.0×10^{-2}	1.9×10^{-1}	2.3×10^{-1}
f_8	1500	-12569.5	-12569.5	-12569.5	-12569.5	-12569.5
f_9	1500	0	1.8×10^{-15}	1.6×10^{-15}	1.4×10^{-16}	4.2×10^{-15}
f_{21}	100	-10.5353	-10.5361	-10.5364	-10.5355	-10.5357

It is interesting to make a comparison of values for control parameters F and CR of Figs. 3 and 4 with Figs. 5 and 6, for each function, respectively. We can see that the values of control parameters obtained by self-adaptive DE algorithm are quite similar to (good) F and CR values obtained from the experiment in Section VII-B. But this time, good F and CR parameter values are not obtained by tuning, hence saving many runs.

Based on the experiment in this section, the necessity of changing control parameter during the optimization process is confirmed once again.

Initialization: The initial vector population is chosen randomly and there arises the question as to how to choose the initial F and CR control parameters for self-adaptive DE, since F and CR are encoded in the individuals (Fig. 1).

We performed an additional experiment to determine the initial F and CR values for our self-adaptive DE. Table IV shows the results obtained in our additional experiment only for the selected benchmark functions. The results do not differ (t-test does not show any significant differences); therefore, our self-adaptive DE is not sensitive to the initial F and CR values. This is an advantage of our algorithm.

E. Comparison of Self-Adaptive DE With Fuzzy Adaptive Differential Evolution Algorithm

Liu and Lampinen [16] introduce a new version of the differential evolution algorithm with adaptive control parameters, the fuzzy adaptive differential evolution (FADE) algorithm, which uses fuzzy logic controllers to adapt the search parameters for the mutation operation and crossover operation. The control inputs incorporate the relative objective function values and individuals of the successive generations.

The FADE algorithm was tested with a set of standard test functions, where it outperforms the original DE when the dimensionality of the problem is high [16].

In [16], ten benchmark functions are used, and nine of them are the same as the benchmark functions in [23] and in this

TABLE V
EXPERIMENTAL RESULTS, AVERAGED OVER 100 INDEPENDENT RUNS, OF SELF-ADAPTIVE DE AND FUZZY ADAPTIVE DE ALGORITHMS. "MEAN BEST" INDICATES AVERAGE OF MINIMUM VALUES OBTAINED AND "STD DEV" STANDS FOR STANDARD DEVIATION. t-TEST TESTS SELF-ADAPTIVE DE AGAINST OTHER ALGORITHMS, RESPECTIVELY

F	#Gen.	Self-Adaptive DE Mean Best (Std Dev)	Fuzzy Adaptive DE Mean Best (Std Dev)
f_1	5000	1.98×10^{-70} (1.56×10^{-70})	2.35×10^{-10} $(2.97 \times 10^{-21})^{\dagger}$
f_5	7000	0 (0)	$4.16 \times 10^{+1}$ $(1.82 \times 10^{-2})^{\dagger}$
f_6	5000	0 (0)	0 (0)
f_7	5000	4.25×10^{-3} (6.87×10^{-3})	$1.90 \times 10^{+1}$ $(2.92 \times 10^{-1})^{\dagger}$
f_9	10000	0 (0)	$2.58 \times 10^{+2}$ $(9.17 \times 10^{+1})^{\dagger}$
f_{10}	5000	6.8×10^{-15} (1.4×10^{-15})	5.9×10^{-2} $(1.23 \times 10^{-6})^{\dagger}$
f_{11}	5000	0 (0)	5.78×10^{-1} $(3.5 \times 10^{-3})^{\dagger}$
f_{14}	100	0.998004 (5.2×10^{-12})	0.9980 (2.5×10^{-26})
f_{18}	50	3.00001 (9.6×10^{-6})	3.0001 (3.35×10^{-7})

† The t value of 99 degree of freedom is significant at a 0.05 level of significance by two-tailed t-test.

paper. The following parameters were used in our experiment (the same parameter settings are used in [16]):

1) dimensionality of the problem $D = 50$;
2) population size $NP = 10 \cdot D$;
3) maximum number of generations: 5000 for f_1, f_6, f_7, f_{10}, and f_{11}, 7000 for f_5, 10 000 for f_9, 100 for f_{14}, and 50 for f_{18}.

Both algorithms use an approach to adapt mutation control parameter F and the crossover control parameter CR. The average results of 100 independent runs are summarized in Table V. The experimental results suggest that the proposed algorithm certainly performs better than the FADE algorithm. This is clearly reflected also by the t-test.

Based on the obtained results in this section, we can conclude that our self-adaptive method is very good in solving benchmark functions (yielding excellent results) and for determination of *good* values for control parameters of a DE.

VIII. CONCLUSION

Choosing the proper control parameters for DE is quite a difficult task because the best settings for the control parameters can be different for different functions. In this paper, the proposed self-adaptive method is an attempt to determine the values of control parameters F and CR.

Our self-adaptive DE algorithm has been implemented and tested on benchmark optimization problems taken from literature. The results show that our algorithm, with self-adaptive control parameter settings, is better or at least comparable to the standard DE algorithm and evolutionary algorithms from literature considering the quality of the solutions found. The proposed algorithm gives better results in comparison with the FADE algorithm.

Our self-adaptive method could be simply incorporated into existing DE algorithms, which are used to solve problems from different optimization areas.

We did not experiment with different population sizes, nor did we make population size adaptive. This remains a challenge for future work.

ACKNOWLEDGMENT

The authors would like to thank J. S. Versterstroem for letting us use his source code for most of the benchmark functions. The authors would also like to thank Prof. X. Yao, the anonymous associate editor, and the referees for their valuable comments that helped greatly to improve this paper. Simulation studies for the differential evolution strategy were performed with the C code downloaded from http://www.icsi.berkeley.edu/~storn/code.html.

REFERENCES

[1] R. Storn and K. Price, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimiz.*, vol. 11, pp. 341–359, 1997.
[2] J. Liu and J. Lampinen, "On setting the control parameter of the differential evolution method," in *Proc. 8th Int. Conf. Soft Computing (MENDEL 2002)*, 2002, pp. 11–18.
[3] T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. New York: Inst. Phys. and Oxford Univ. Press, 1997.
[4] M. H. Maruo, H. S. Lopes, and M. R. Delgado, "Self-adapting evolutionary parameters: Encoding aspects for combinatorial optimization problems," in *Lecture Notes in Computer Science*, G. R. Raidl and J. Gottlieb, Eds. Lausanne, Switzerland: Springer-Verlag, 2005, vol. 3448, Proc. Evol. Comput. Combinatorial Optimization, pp. 155–166.
[5] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 124–141, Jul. 1999.
[6] T. Krink and R. K. Ursem, "Parameter control using the agent based patchwork model," in *Proc. Congr. Evolutionary Computation*, La Jolla, CA, Jul. 6–9, 2000, pp. 77–83.
[7] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing. Berlin, Germany: Springer-Verlag, 2003.
[8] K. Deb, "A population-based algorithm-generator for real-parameter optimization," *Soft Computing—A Fusion of Foundations, Methodologies and Applications*, vol. 9, no. 4, pp. 236–253, 2005 [Online]. Available: http://springerlink.metapress.com/index/10.1007/s00500-004-0377-4
[9] M. M. Ali and A. Törn, "Population set-based global optimization algorithms: Some modifications and numerical studies," *Comput. Oper. Res.*, vol. 31, no. 10, pp. 1703–1725, 2004.
[10] T. Bäck, "Evolution strategies: An alternative evolutionary algorithm," in *Lecture Notes in Computer Science*, J. M. Alliott, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyders, Eds. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1063, Proc. Artificial Evolution: Eur. Conf., pp. 3–20.
[11] ——, "Adaptive business intelligence based on evolution strategies: some application examples of self-adaptive software," *Inf. Sci.*, vol. 148, pp. 113–121, 2002.
[12] L. Davis, Ed., *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.
[13] W. M. Spears, "Adapting crossover in evolutionary algorithms," in *Proc. 4th Annual Conf. Evolutionary Programming*, J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, Eds., 1995, pp. 367–384.
[14] M. A. Semenov and D. A. Terkel, "Analysis of convergence of an evolutionary algorithm with self-adaptation using a stochastic Lyapunov function," *Evol. Comput.*, vol. 11, no. 4, pp. 363–379, 2003.
[15] J. He and X. Yao, "Toward an analytic framework for analysing the computation time of evolutionary algorithms," *Artificial Intell.*, vol. 145, no. 1–2, pp. 59–97, 2003.
[16] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Computing—A Fusion of Foundations, Methodologies and Applications*, vol. 9, no. 6, pp. 448–462, 2005 [Online]. Available: http://springerlink.metapress.com/index/10.1007/s00500-004-0363-x

[17] ——, "Adaptive parameter control of differential evolution," in *Proc. 8th Int. Conf. Soft Computing (MENDEL 2002)*, 2002, pp. 19–26.
[18] R. Storn and K. Price, Differential Evolution—A Simple and efficient adaptive scheme for global optimization over continuous spaces, Berkeley, CA, Tech. Rep. TR-95-012, 1995 [Online]. Available: citeseer.ist.psu.edu/article/storn95differential.html
[19] J. Vesterstroem and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proc. IEEE Congr. Evolutionary Computation*, Portland, OR, Jun. 20–23, 2004, pp. 1980–1987.
[20] J. Sun, Q. Zhang, and E. Tsang, "DE/EDA: A new evolutionary algorithm for global optimization," *Info. Sci.*, vol. 169, pp. 249–262, 2004.
[21] K. Price and R. Storn, "Differential evolution: A simple evolution strategy for fast optimization," *Dr. Dobb's J. Software Tools*, vol. 22, no. 4, pp. 18–24, Apr. 1997.
[22] R. Gämperle, S. D. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," WSEAS NNA-FSFS-EC 2002. Interlaken, Switzerland, WSEAS, Feb. 11–15, 2002 [Online]. Available: http://www.worldses.org/online/
[23] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, p. 82, Jul. 1999.
[24] C. Y. Lee and X. Yao, "Evolutionary programming using mutations based on the Lévy probability distribution," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, pp. 1–13, Feb. 2004.
[25] Z. Tu and Y. Lu, "A robust stochastic genetic algorithm (StGA) for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 5, pp. 456–470, Oct. 2004.
[26] F. Herrera and M. Lozano, "Adaptive genetic operators based on coevolution with fuzzy behaviors," *IEEE Trans. Evol. Comput.*, vol. 5, no. 2, pp. 149–165, Apr. 2001.
[27] A. Törn and A. Žilinskas, "Global optimization," in *Lecture Notes Computer Science*. Heidelberg, Germany: Spring-Verlag, 1989, vol. 350, pp. 1–24.

Janez Brest (M'02) received the B.S., M.Sc., and Ph.D. degrees in computer science from the University of Maribor, Maribor, Slovenia, in 1995, 1998, and 2001, respectively.

He has been with the Laboratory for Computer Architecture and Programming Languages, University of Maribor, since 1993. He is currently an Assistant Professor. His research interests include evolutionary computing, artificial intelligence, and optimization. His fields of expertise embrace programming languages, web-oriented programming, and parallel and distributed computing research.

Dr. Brest is a member of ACM.

Sašo Greiner received the B.S. and M.Sc. degrees in computer science from the University of Maribor, Maribor, Slovenia, in 2002 and 2004, respectively.

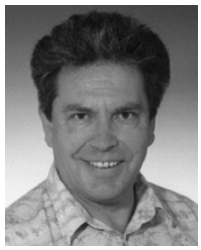
He is currently a Teaching Assistant at the Faculty of Electrical Engineering and Computer Science, University of Maribor. His research interests include object-oriented programming languages, compilers, computer architecture, and web-based information systems.

Borko Bošković received the B.S. degree, in 2003.

He is currently a Teaching Assistant at the Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia. He has worked in the Laboratory for Computer Architecture and Programming Languages, University of Maribor, since 2000. His research interests include web-oriented programming, evolutionary algorithms, and search algorithms for two players with perfect-information zero-sum games.

Marjan Mernik (M'95) received the M.Sc. and Ph.D. degrees in computer science from the University of Maribor, Maribor, Slovenia, in 1994 and 1998, respectively.

He is currently an Associate Professor in the Faculty of Electrical Engineering and Computer Science, University of Maribor. He is also an Adjunct Associate Professor in the Department of Computer and Information Sciences, University of Alabama, Birmingham. His research interests include programming languages, compilers, grammar-based systems, grammatical inference, and evolutionary computations. He is a member of ACM and EAPLS.



Viljem Žumer (M'77) is a full Professor in the Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia. He is the Head of Laboratory for Computer Architecture and Programming Languages and the Head of the Institute of Computer Science as well. His research interests include programming languages and computer architecture.