Evolutionary Adaptation of the Differential Evolution Control Parameters

M.G. Epitropakis, V.P. Plagianakos, and M.N. Vrahatis

Abstract— **This papers proposes a novel self–adaptive scheme for the evolution of crucial control parameters in Evolutionary Algorithms. More specifically, we suggest to utilize the Differential Evolution algorithm to endemically evolve its own control parameters. To achieve this, two simultaneous instances of Differential Evolution are used, one of which is responsible for the evolution of the crucial user–defined mutation and recombination constants. This self–adaptive Differential Evolution algorithm alleviates the need of tuning these user–defined parameters while maintains the convergence properties of the original algorithm. The evolutionary self–adaptive scheme is evaluated through several well–known optimization benchmark functions and the experimental results indicate that the proposed approach is promising.**

I. Introduction

Evolutionary Algorithms (EAs) are well-established nature inspired optimization methods [1]. The broad class of EAs has demonstrated numerous methods that have been effectively and successfully applied to numerous difficult, real-world optimization problems [1]–[3]. One of the well known and widely used EAs is the Differential Evolution (DE) algorithm [4]–[6]. DE is capable of handling non-differentiable, nonlinear, multimodal and noisy objective functions. Many comparative studies confirm its robust and effective capabilities, also it is stated that in many cases DE outperforms many other well known Evolutionary Computational approaches in terms of convergence speed and quality of solutions [5], [7]. DE has three crucial user–defined control parameters: (a) the *mutation constant* (F) controlling the mutation strength, (b) the *recombination constant* (CR), and (c) the *population size* (NP). The originally proposed DE keeps all three control parameters fixed during the evolution process.

Although, in their original study, Storn and Price state that the control parameters of DE are easy to choose, several recent works [8]–[11] indicate that effectiveness, efficiency and robustness of the DE algorithm strongly depend on their values. Moreover, their optimal values are affected by the objective function and the computational time and accuracy requirements. The sensitivity of the DE algorithm to its control parameters can lead to significant performance deterioration. Numerous studies have shown that both the convergence rate and speed of the DE algorithm depends on the control parameters (especially the mutation constant). Additionally, due to the mutation operator of the DE, incorrect (i.e. very small) values of the mutation constant lead to diversity loss, since the new individuals computed by the mutation operator do not substantial differ from their parents. On the other hand, very large values of the mutation constant excessively amplify the parents, leading to convergence problems.

To overcome the aforementioned problems, the user has to choose appropriate values either by a preliminary testing and hand–tuning or by employing a (self–)adaptive procedure. Self–adaptation approaches have proved to be very gainful in evolutionary algorithm literature (see for example [1], [12], [13]). Self–adaptation is the procedure of allowing an evolutionary task to adapt itself to a given class of problems without any user interaction. In Differential Evolution literature, self-adaptive schemes are usually applied on the mutation and/or the recombination control parameters. The automatic adaptation of those parameters solves problems stemming from inappropriate parameter values and may have the effect of increased convergence rates [9], [14], [15].

The main objective of this study is to propose a new evolutionary self–adaptive scheme that can be incorporated to evolve the mutation constant of the Differential Evolution algorithm. The proposed self–adaptive DE algorithm alleviates the need of tuning the user–defined mutation and recombination constants, while maintains the convergence properties of the original algorithm. A similar approach that incorporates the DE algorithm to manipulate critical heuristic parameters of the Particle Swarm Optimization (PSO) method has been proposed in [16] and has been empirically demonstrated to be efficient. Other self–adaptive DE approaches include: Zaharie's self-adaptive scheme [17], the Differential Evolution with Self Adapting Populations (DESAP) [8], the Fuzzy Adaptive Differential Evolution (FADE) [18], the Self–adaptive Differential Evolution algorithm (SaDE) [14], [15], the Self-adaptive Pareto DE (SPDE) [19], and the recently proposed self–adapting DE algorithms by Brest *et al.* (jDE) [10] and by Salman *et al.* (SDE) [9].

The rest of the paper is organized as follows. In Section II the DE algorithm is briefly described. In Section III we propose the self–adaptive Differential Evolution algorithm, while Section IV is devoted to the presentation and the discussion of the experimental results. The paper ends with concluding remarks and some pointers for future work.

II. The Differential Evolution Algorithm

Differential Evolution [4] is a stochastic parallel direct search method, which utilizes concepts borrowed from the broad class of EAs. The method typically requires few

All the authors are with Computational Intelligence Laboratory (CI Lab), Department of Mathematics, University of Patras Artificial Intelligence Research Center (UPAIRC), University of Patras, GR–26110 Patras, Greece. (email: {mikeagn, vpp, vrahatis}@math.upatras.gr).

control parameters. Experimental results have shown that DE has good convergence properties and outperforms other well known EAs [4], [6], [7], [20].

More specifically, DE is a population–based stochastic algorithm that exploits a population of potential solutions, *individuals*, to effectively probe the search space. The population of individuals is randomly initialized in the optimization domain with *NP*, D–dimensional, vectors following a uniform probability distribution. Individuals evolve over successive iterations to explore the search space and locate the minima of the objective function. Throughout the execution process, the user–defined population size, *NP*, is fixed. At each iteration, called *generation*, new vectors are derived by the combination of randomly chosen vectors from the current population. This operation in our context can be referred to as *mutation*, while the outcoming vectors as *mutant individuals*. Each mutant individual is then mixed with another, predetermined, vector – the *target* vector – through an operation called *recombination*. This operation yields the so–called *trial* vector. Finally, the trial vector undergoes the *selection* operator, according to which it is accepted as a member of the population of the next generation only if it yields a reduction in the value of the objective function f relative to that of the target vector. Otherwise, target vector is retained in the next generation.

The search operators efficiently shuffle information among the individuals, enabling the search for an optimum to focus on the most promising regions of the solution space. Next, we briefly describe the search operators that were considered in this paper.

A. Mutation Operators

Here we describe the original mutation operators proposed in [4]. Specifically, for each individual x_g^i, $i = 1, \ldots,$ *NP*, where g denotes the current generation, the mutant individual v_{g+1}^i can be generated according to one of the following equations:

$$v_{g+1}^i = x_g^{\text{best}} + F(x_g^{r1} - x_g^{r2}), \quad (1)$$

$$v_{g+1}^i = x_g^{r1} + F(x_g^{r2} - x_g^{r3}), \quad (2)$$

$$v_{g+1}^i = x_g^i + F(x_g^{\text{best}} - x_g^i) + F(x_g^{r1} - x_g^{r2}), \quad (3)$$

$$v_{g+1}^i = x_g^{\text{best}} + F(x_g^{r1} - x_g^{r2}) + F(x_g^{r3} - x_g^{r4}), \quad (4)$$

$$v_{g+1}^i = x_g^{r1} + F(x_g^{r2} - x_g^{r3}) + F(x_g^{r4} - x_g^{r5}), \quad (5)$$

where x_g^{best} is the best member of the previous generation, $r_1, r_2, r_3, r_4, r_5 \in \{1, 2, \ldots, i-1, i+1, \ldots, NP\}$, are random integers mutually different and not equal to the running index i, and $F > 0$ is a real parameter, called *mutation or scaling factor*. The user–defined *mutation constant*, controls the amplification of the difference between two individuals, and is used to prevent the risk of stagnation, of the search process. It is also mainly responsible for the convergence rate of the algorithm. Thus, an inappropriate mutation constant value can cause deceleration of the algorithm and decrease of the population diversity.

Trying to rationalize the above equations, we observe that Eq. (2) is similar to the crossover operator employed by some Genetic Algorithms; while Eq. (1) is derived from Eq. (2), by substituting the best member of the previous generation, x_g^{best}, for the random individual x_g^{r1}. Eqs. (3), (4) and (5) are modifications obtained by the combination of Eqs (1) and (2). It is clear that more mutation operators can be generated using the above ones as building blocks, such as the trigonometric mutation operator [21] and recently proposed genetically programmed mutation operators [22].

The recently proposed trigonometric mutation operator [21] performs a mutation according to the following equation, with probability τ_μ:

$$v_{g+1}^i = (x_g^{r1} + x_g^{r2} + x_g^{r3})/3 + (p_2 - p_1)(x_g^{r1} - x_g^{r2}) + \\ + (p_3 - p_2)(x_g^{r2} - x_g^{r3}) + (p_1 - p_3)(x_g^{r3} - x_g^{r1}), \quad (6)$$

and with probability $(1 - \tau_\mu)$, the mutation is performed according to Eq. (2). Here, τ_μ is a user defined parameter, typically set around 0.1. The values of p_m, $m = \{1, 2, 3\}$ and p' are obtained through the following equations:

$$\begin{aligned} p_1 &= \left|f(x_g^{r1})\right| / p', \\ p_2 &= \left|f(x_g^{r2})\right| / p', \\ p_3 &= \left|f(x_g^{r3})\right| / p', \text{ and} \\ p' &= \left|f(x_g^{r1})\right| + \left|f(x_g^{r2})\right| + \left|f(x_g^{r3})\right|. \end{aligned}$$

For the rest of the paper, we call DE_1, DE_2, …, DE_6 the DE algorithm that uses Eq. (1), Eq. (2), …, Eq. (6) as the mutation operator, respectively.

B. Recombination and Selection Operators

Having performed the mutation, the recombination operator is subsequently applied to further increase the diversity of the population. To this end, the mutant individuals are combined with other predetermined individuals, called the target individuals. Specifically, for each component l ($l = 1, 2, \ldots, D$) of the mutant individual v_{g+1}^i, we randomly choose a real number r in the interval $[0, 1]$. Then, we compare this number with the user–defined *recombination constant*, *CR*. If $r \leqslant CR$, then we select, as the l–th component of the trial individual u_{g+1}^i, the l–th component of the mutant individual v_{g+1}^i. Otherwise, the l–th component of the target vector x_g^i becomes the l–th component of the trial vector. This operation yields the trial individual. It is evident that if the value of the recombination constant is too small (close to zero) the effect of the mutation operator is cancelled, since the target (and not the mutant) vector will become the new trial vector.

Finally, the trial individual is accepted for the next generation only if it reduces the value of the objective function (selection operator):

$$u_{g+1}^i = \begin{cases} v_{g+1}^i & \text{if } f(v_{g+1}^i) < f(x_g^i) \\ x_g^i & \text{otherwise} \end{cases}. \quad (7)$$

III. Evolutionary Self–Adaptive Differential Evolution

This section briefly describes the proposed approach. The Evolutionary Self–Adaptive Differential Evolution (ESADE)

algorithm is a self–adaptive scheme that adjust the two crucial DE's control parameters, namely the *mutation constant* (*F*) and the *recombination constant* (*CR*). The *population size* (*NP*) is fixed during the evolution process. More specifically, a separate Differential Evolution algorithm is utilized to adapt the mutation constant of the main algorithm. To this end, we utilize two different Differential Evolution levels; the first one for evolving the mutation constant and the second one for actually optimizing the objective function.

More specifically, in the first evolutionary level, Differential Evolution uses one–dimensional individuals $x_g = \{F_g\}$ to initialize its population, where F_g is a possible mutation constant value used by the second evolutionary process. In practice, the fixed value of the mutation constant is usually in the range $(0.1, 1.0]$. Although, smaller values lead to better exploitation of the local neighborhood, may also lead to premature convergence to a local minimum. On the other hand, larger values result in better exploration of the search space, but also in slower convergence rates. One could choose to initialize the mutation constant with a relatively large value and gradually decrease it, but this approach tends to be inefficient and extremely problem dependent.

Thus, the proposed evolutionary adaptation scheme initializes the one–dimensional individuals with random values from the normal distribution, with mean value 0.5 and standard deviation 0.3. This choice has been experimentally proved to be a good starting point [14], [15].

After the initialization of the individuals of the first evolutionary level, one generation of the second evolutionary level is performed to determine the fitness value of each one. More specifically, the objective function value of the best individual of the second evolutionary level (i.e. $f(x_g^{\text{best}})$) is assigned as the fitness value of the respective individual of the first evolutionary level. It must be noted that the first evolutionary level evolves the mutation constant and, simultaneously, the second evolutionary level minimizes the objective function using that mutation constant.

It is clear that incorporating a relatively large population size for the first evolutionary level will drastically increase the objective function evaluations needed, while may hinder the fast evolution of the mutation constant. To this end, we have employed the smallest possible population having only six individuals.

Regarding the optimal value of the recombination constant (*CR*), it must be noted that is much more sensitive to the properties and complexity of the optimization problem (e.g. dimensionality, multimodality, etc.). A proper choice of the recombination constant may lead to improved performance, while a wrong choice usually results in severe performance deterioration. Moreover, experimental results indicate that optimal values for the recombination constant usually fall within a small range, in which the algorithm can perform consistently well on complex problems [14], [15]. One can try to evolve the recombination constant utilizing the proposed evolutionary process used for the mutation constant. However, the Differential Evolution algorithm seems

THE ESADE ALGORITHM

Begin
Initialize the two populations Pop_1, Pop_2 (six and *NP* individuals respectively)
Evaluate the fitness of the Pop_1
Repeat
  For $i = 1$ **to** 6 **Do** /**level one*/*
  /* *level two*/*
    Evolve for one generation Pop_2 using DE with $F = F_i$ and $CR = N(0.6, 0.1)$
    Mutation$(F_g^i) \rightarrow \text{Mutant}_g^i$
    Recombination$(\text{Mutant}_g^i) \rightarrow \text{Trial}_g^i$
    Evaluate F_i with $f(F_i) = f(\text{best}_{(\text{Pop}_2)})$
    If f$(\text{Trial}_g^i) \leqslant f(F_g^i)$ **Then**
      accept Trial_g^i for the next generation
    EndIf
  EndFor
Until the termination criteria are satisfied
End

more sensitive to inappropriate recombination constant values and the experimental results are rather random; there exist instances where rapid convergence is achieved and instances where the algorithm exhibits significantly decreased convergence speed and success rate. Thus, the proposed algorithm employs a simple adaptation scheme and resets the recombination constant at every generation of the first evolutionary level. Specifically, a random value from the normal distribution (with mean value 0.6 and standard deviation 0.1) is assigned and then, if needed, is restricted in the range $[0.0, 1.0]$. Experimental results indicate that this specific range aids DE to successfully tackle many different optimization tasks. The proposed approach is outlined in the above algorithmic scheme.

To depict the evolution of the mutation constant, we have applied ESADE on the Levy No. 5 test function (see IV-A.8 below). At the top of Figure 1 the values of the mutation constant are demonstrated, while at the bottom the fitness of the best individual is illustrated. It is clear that, ESADE initially explores the parameter's search space and then converges to values around 0.6.

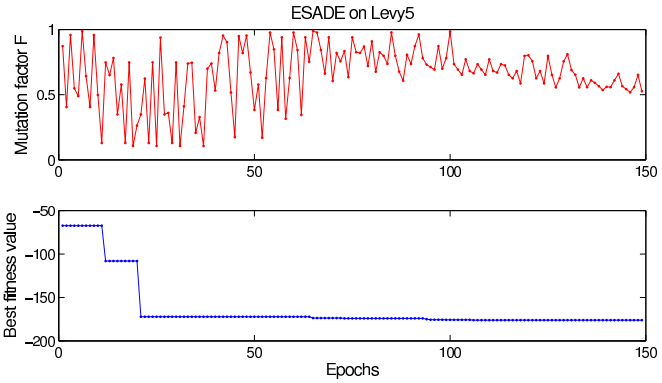

Fig. 1. ESADE: (Top) Evolution of the mutation constant, (Bottom) Fitness value of the best individual (Levy No. 5)

IV. EXPERIMENTAL RESULTS

We implemented and tested the proposed Evolutionary Self–adaptive DE scheme on a large number of real parame-

ter optimization benchmark functions. In this study we report experimental results from ten well–known minimization test functions.

The computational experiments were performed utilizing a DE interface developed in C++, using GNU compiler collection (gcc) version 3.4.6 on a Debian GNU Linux operating system. For each test function and each mutation operator, we have conducted 100 independent runs. The mutation and crossover constants of the first evolutionary level and for all six classic DE mutation operators, have fixed values $F = 0.5$ and $CR = 0.7$, respectively. In Table I the parameter setup used in the numerical experiments conducted is summarized. Specifically, D denotes the dimensionality of the problem, *NP* stands for the population size used for each function, while $MaxGen$ is the maximum number of generations allowed.

Notice that both ESADE and DE algorithms had the same termination criteria; the algorithm was stopped: i) when it reached the Maximum Number of Function Evaluations (MNFE), or ii) when the solution was computed with the prespecified accuracy. MNFE can be calculated through the following equation: $\mathrm{MNFE} = NP \cdot MaxGen$.

No	Test function	D	NP	$MaxGen$
1	Sphere function	30	50	5000
2	Rosenbrock's saddle	2	30	1000
3	Step function	5	20	1000
4	Quartic function	30	100	2000
5	Shekel's foxholes	2	30	1000
6	Corana's parabola	4	15	2000
7	Griewangk's function	10	50	10000
8	Levy No.5 function	2	40	1000
9	Rastrigin function	10	40	3000
10	Ackley function	30	40	3000

TABLE I
PARAMETER SETUP VALUES

Next, we will briefly report the benchmark optimization functions used along with their global minima and minimizers in the search space.

A. Test Functions

The interested reader can find detailed information about the test functions used here in [4], [23]–[27].

1) Sphere:

$$f_1(x) = \sum_{j=1}^{30} x_j^2, \qquad x_j \in [-5.12, 5.12]. \tag{8}$$

The sphere test function is a considered to be a simple minimization problem. The minimum is $f_1^*(0, 0, \ldots, 0) = 0$.

2) Rosenbrock's Saddle:

$$\begin{aligned} f_2(x) &= 100 \cdot (x_1^2 - x_2)^2 + (1 - x_1)^2, \\ & x_j \in [-2.048, 2.048]. \end{aligned} \tag{9}$$

This is a two–dimensional test function, which is known to be relatively difficult to minimize. The minimum is $f_2^*(1, 1) = 0$.

3) Step Function:

$$f_3(x) = 30 + \sum_{j=1}^{5} \lfloor x_j \rfloor, \quad x_j \in [-5.12, 5.12]. \tag{10}$$

The minimum of this function is $f_3^*(-5-\xi, \ldots, -5-\xi) = 0$, where $\xi \in [0, 0.12]$. This function exhibits many flat regions that can cause search stagnation.

4) Quartic Function:

$$f_4(x) = \sum_{j=1}^{30} \left(j \cdot x_j^4 + \eta \right), \tag{11}$$

where $x_j \in [-1.28, 1.28]$. This is test function is designed to evaluate the behavior of minimization algorithms in the presence of noise. To this end, η is a random variable following the uniform distribution in the range $[0, 1]$. The inclusion of η makes f_4 more difficult to optimize. The functional minimum of the function is $f_4^*(0, 0, \ldots, 0) \leqslant 30 \cdot E[\eta] = 15$, where $E[\eta]$ is the expectation of η.

5) Shekel's Foxholes:

$$f_5(x) = \frac{1}{0.002 + \psi_1(x)}, \; x_j \in [-65.536, 65.536], \tag{12}$$

where, $\psi_1(x) = \sum_{i=0}^{24} 1/(1 + i + \sum_{j=1}^{2}(x_j - a_{ij})^6)$. The parameters for this function are:

$$\begin{aligned} a_{i1} &= \{-32, -16, 0, 16, 32\}, \text{ where} \\ & \quad i = \{0, 1, 2, 3, 4\} \text{ and } a_{i1} = a_{i \bmod 5,1} \\ a_{i2} &= \{-32, -16, 0, 16, 32\}, \text{where} \\ & \quad i = \{0, 5, 10, 15, 20\} \text{ and} \\ & \quad a_{i2} = a_{i+k,2}, \; k = \{1, 2, 3, 4\}. \end{aligned}$$

The global minimum of $f_5^*(-32, -32) = 0.998004$.

6) Corana Parabola:

$$f_6(x) = \sum_{j=1}^{4} \begin{cases} \psi_2(x_j), \text{ if } |x_j - z_j| < 0.05, \\ \psi_3(x_j), \text{ otherwise.} \end{cases} \tag{13}$$

where $\psi_2(x_j) = 0.15 \left(z_j - 0.05\mathrm{sign}(z_j)\right)^2 d_j$, $\psi_3(x_j) = d_j x_j^2$, $z_j = \lfloor 5|x_j| + 0.49999 \rfloor \mathrm{sign}(x_j) 0.2$ and $d_j = \{1, 1000, 10, 100\}$. The function is characterized by a multitude of local minima, increasing in depth as one moves closer to the origin. The global minimum of the function is $f_6^*(x) = 0$, for $x_j^* \in (-0.05, 0.05)$.

7) Griewangk's Function:

$$\begin{aligned} f_7(x) &= \sum_{j=1}^{10} \frac{x_j^2}{4000} - \prod_{j=1}^{10} \cos\left(\frac{x_j}{\sqrt{j}}\right) + 1, \\ & x_j \in [-400, 400]. \end{aligned} \tag{14}$$

This test function is riddled with local minima. The global minimum of the function is $f_7^*(0, 0, \ldots, 0) = 0$.

8) Levy No.5 Function:

$$f_8(x) = \sigma_1\sigma_2 + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2, \quad (15)$$

where $x_i \in [-10, 10], i = 1, 2$, and σ_1 and σ_2 are given by:

$$\sigma_1 = \sum_{i=1}^{5} \Big[i \ \cos\big((i-1)x_1 + i\big) \Big],$$

$$\sigma_2 = \sum_{j=1}^{5} \Big[j \ \cos\big((j+1)x_2 + j\big) \Big].$$

There exist about 760 local minima and one global minimum with function value $f_8^*(x) = -176.1375$, located at $x^* = (1.3068, 1.4248)$. The large number of local optimizers makes it difficult for any method to locate the global minimizer.

9) Rastrigin Function:

$$f_9(x) = A \cdot n + \sum_{i=1}^{n} x_i^2 - A \cdot \cos(\omega \cdot x_i) \quad (16)$$
$$A = 10 \ ; \ \omega = 2 \cdot \pi \ ; \ x_i \in [-5.12, 5.12].$$

The Rastrigin Function is a typical example of non-linear multimodal function. This function is a fairly difficult problem due to its large search space and its large number of local minima. The global minimum of the function is $f_9^*(0, 0, \ldots, 0) = 0$.

10) Ackley Function:

$$f_{10}(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \quad (17)$$
$$- \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e,$$

where $x_i \in [-32.768, 32.768]$. The global minimum of the function is $f_{10}^*(0, 0, \ldots, 0) = 0$.

B. Presentation of the Results

To evaluate the proposed ESADE algorithm, we compared its performance against the six classic DE mutation operators presented in Section II on the ten test functions described above. During the development phase of the proposed algorithm several different approaches have been considered. The obtained experience can be summarized below. Preliminary experiments included the evolution of the recombination constant alone, and the evolution of both the mutation and recombination constants. The obtained results for these approaches were dissatisfying. The success rates were rather low, while the location of the global minimum demanded a significantly increased amount of function evaluations. The main problem was that the first evolutionary level usually assigned values to the control parameters (especially to the recombination constant) that spread all over the allowed range. Thus, the poorly selected values prevented the convergence of the DE algorithm.

To utilize the proposed algorithm, one can apply any DE mutation strategy for the evolution of the mutation constant at the first evolutionary level. Extensive experimental results demonstrated that the utilization of an explorative mutation strategy, such as DE/rand/1 or DE/rand/2, enhances the efficiency and effectiveness of the algorithm [28]. Additionally, a hybrid approach that balances the explorative and the exploitive behavior of the original DE mutation strategies can also be employed [29]. The usage of such approach, can firstly explore for promising regions of the search space and then exploit the aforementioned region for an optimal value. In this study, due to space restrictions, we report experimental results of the application of the DE_5 mutation strategy (i.e. DE/rand/2) for the first evolutionary level.

We performed 100 independent runs for each algorithm and each problem. The following notation is used in the Tables: Min indicates the minimum number of function evaluations for the experiments that reached a solution (i.e. a global minimum); Max is the maximum number of function evaluations; $Mean$ is the average function evaluations number and $St.D.$ is the standard deviation. Finally, the last column, $Success$ is the percentage of experiments that reached a solution.

Tables II-XI summarize the experimental results. The experimental results on the test functions indicate that the proposed approach is promising and that exhibits success rate equal or better than the original DE algorithm, at the expense of an increase of the average function evaluations required.

Finally, Figure 2 exhibits boxplots summarizing the last iteration values of the mutation constant for all the test problems. Each boxplot depicts the obtained values for the mutation constant for all the experiments. The box has lines at the lower quartile, median, and upper quartile values. The lines extending from each end of the box (whiskers) exhibit the range covered by the remaining data. Notches represent a robust estimate of the uncertainty about the median. It is clear that, for all benchmark functions, the mutation constant takes values around the 0.5 value.

Since, values for the mutation constant $F \approx 0.5$ are

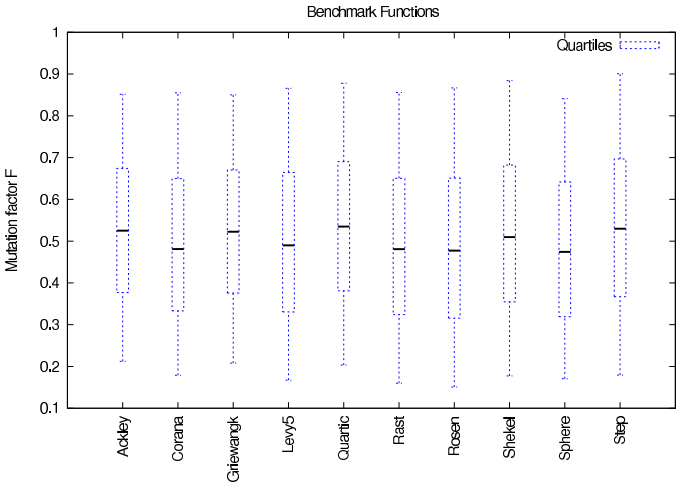

Fig. 2. The evolution of the mutation constant

known to be efficient, the experimental results indicate that the proposed algorithm is capable to automatically compute appropriate values for the control parameters of the DE algorithm. So, for an unknown optimization problem the ESADE algorithm is capable to reach high success rates at the expense of a slight increase of the function evaluations required. Thus, with the application of the ESADE algorithm locating optima becomes feasible on a first–time basis for a given unknown problem, without the need for user intervention.

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	76120	81760	89320	2636.03	100
$ESADE_2$	112360	117001	120000	1837.47	83
$ESADE_3$	104920	109142	112600	1583.95	100
$ESADE_4$	98680	111209	120000	4274.81	98
$ESADE_5$	N/A	N/A	N/A	N/A	N/A
$ESADE_6$	97960	103170	111640	2449.98	100
DE_1	59720	62703.2	65520	905.9	99.9
DE_2	92280	95449.3	98560	959.3	100
DE_3	69680	72583.3	76680	904.1	100
DE_4	93520	98079.6	101680	1138.8	100
DE_5	N/A	N/A	N/A	N/A	N/A
DE_6	79400	81931.0	84920	870.5	100

TABLE II

COMPARATIVE RESULTS FOR THE ACKLEY TEST FUNCTION

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	915	1391.0	2265	240.9	76
$ESADE_2$	1635	2467.0	3255	309.0	98
$ESADE_3$	1995	2434.9	3435	229.6	98
$ESADE_4$	1275	2244.6	24855	2537.6	84
$ESADE_5$	2355	3228.9	4335	466.0	100
$ESADE_6$	1725	2359.5	3165	289.3	99
DE_1	705	1099.1	1845	174.4	67.0
DE_2	1485	2055.8	2700	208.9	98.8
DE_3	1005	1776.2	3765	226.9	98.1
DE_4	1155	1791.7	2415	189.2	97.9
DE_5	1770	2894.6	3885	282.4	100
DE_6	1395	1966.5	3375	191.2	99.1

TABLE III

COMPARATIVE RESULTS FOR THE CORANA TEST FUNCTION

V. CONCLUSIONS

The performance of the DE algorithm is sensitive to inappropriate values of the control parameters. Poor values can lead to significant performance deterioration and/or slow convergence. Thus, the incorporation of an intelligent adaptation method is compulsory.

In this study, we presented an evolutionary self–adaptive scheme for adjusting Differential Evolution's mutation and recombination constants. A separate Differential Evolution algorithm is employed to evolve the mutation constant *F*, while the recombination constant *CR* takes values from the normal distribution.

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	14750	23850	31250	6607.2	6
$ESADE_2$	39950	55715	74750	7892.8	100
$ESADE_3$	56150	108038	222950	28834.9	99
$ESADE_4$	38450	64631	95150	10865.1	66
$ESADE_5$	73550	116453	156650	15163.1	100
$ESADE_6$	28550	50000	66650	8180.5	100
DE_1	10100	18764.0	28450	3629.1	7.5
DE_2	29450	41799.1	57550	4921.1	100
DE_3	25350	56206.1	460000	28581.9	89.8
DE_4	25000	52086.1	76150	6875.6	91.9
DE_5	64750	85696.6	107550	7734.4	100
DE_6	22900	36154.5	53100	4783.7	100

TABLE IV

COMPARATIVE RESULTS FOR THE GRIEWANGK TEST FUNCTION

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	760	1665.1	2680	400.7	70
$ESADE_2$	2680	3695.2	4840	542.4	100
$ESADE_3$	3160	4619.2	7480	1058.9	100
$ESADE_4$	1720	2959.1	4840	680.0	92
$ESADE_5$	3400	5521.6	9160	1073.1	100
$ESADE_6$	2440	3455.2	5560	495.3	100
DE_1	920	1425.7	2520	234.9	70.8
DE_2	1920	3091.2	4560	384.6	100
DE_3	1720	3307.0	18360	933.7	97.3
DE_4	1760	2813.3	6000	621.8	97.8
DE_5	3120	4864.9	8480	781.9	100
DE_6	1840	2987.4	4400	386.2	100

TABLE V

COMPARATIVE RESULTS FOR THE LEVY. 5 TEST FUNCTION

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	4900	10366	16900	2192.3	100
$ESADE_2$	9100	15322	23500	2506.2	100
$ESADE_3$	9100	11998	17500	1887.8	100
$ESADE_4$	7300	13426	25300	3427.4	100
$ESADE_5$	9700	18730	27700	3689.1	100
$ESADE_6$	7900	13498	17500	1944.7	100
DE_1	4500	7236.5	11000	1010.1	100
DE_2	6300	11440.0	18600	1769.8	100
DE_3	3800	7088.1	9800	997.6	100
DE_4	4900	11159.0	16200	1776.6	100
DE_5	7500	15022.6	23600	2506.6	100
DE_6	5200	9801.3	14100	1388.0	100

TABLE VI

COMPARATIVE RESULTS FOR THE QUARTIC TEST FUNCTION

The performance of the proposed approach was evaluated on ten well–known benchmark optimization functions. The extensive experimental results of this paper provide evidence that the proposed approach is promising. The ESADE algorithm exhibits success rates equal or better than the original DE algorithm, at the expense of an increase of the average function evaluations required.

To conclude, we believe that ESADE is an attractive alternative choice to the original DE, especially on unknown real–

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	1860	3352.5	4740	605.6	89
$ESADE_2$	5820	7753.2	9420	699.0	100
$ESADE_3$	7620	10582.8	15180	1399.2	100
$ESADE_4$	3300	5348.4	9060	1028.0	100
$ESADE_5$	8340	10410.0	13020	1008.3	100
$ESADE_6$	4740	7386.0	8700	621.1	100
DE_1	2220	2871.1	3480	192.7	99.7
DE_2	5040	6282.7	7320	393.7	100
DE_3	3960	5697.1	6600	346.9	100
DE_4	3960	5029.5	6360	298.6	100
DE_5	6720	8448.4	9840	484.7	100
DE_6	4560	6047.7	7320	378.6	100

TABLE VII

COMPARATIVE RESULTS FOR THE RASTRIGIN TEST FUNCTION

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	570	1074.0	1830	275.5	100
$ESADE_2$	1110	2535.6	3810	477.0	100
$ESADE_3$	1290	2366.4	3990	437.1	100
$ESADE_4$	750	1655.4	3090	425.3	100
$ESADE_5$	2370	3462.6	4530	506.7	100
$ESADE_6$	1290	2870.4	6690	647.5	100
DE_1	390	821.5	1230	102.6	100
DE_2	1170	1960.4	8790	528.2	97.8
DE_3	960	1785.7	3120	301.9	100
DE_4	510	1417.2	2010	176.7	100
DE_5	1530	2542.0	3630	312.5	100
DE_6	1320	2168.9	8130	540.8	97.3

TABLE VIII

COMPARATIVE RESULTS FOR THE ROSENBROCK TEST FUNCTION

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	390	888.2	1470	230.2	56
$ESADE_2$	750	2651.0	3630	481.5	98
$ESADE_3$	1110	2791.2	7230	928.5	100
$ESADE_4$	570	1783.0	2730	493.3	92
$ESADE_5$	1650	3784.8	5790	747.6	100
$ESADE_6$	570	2473.4	3450	482.0	94
DE_1	180	741.4	1200	166.2	64.5
DE_2	600	1921.3	2940	328.2	98.1
DE_3	270	1273.1	11730	447.6	90.7
DE_4	60	1500.4	2400	287.5	99.6
DE_5	750	2670.3	4050	449.0	100
DE_6	630	1930.6	3270	326.2	98.2

TABLE IX

COMPARATIVE RESULTS FOR THE SHEKEL TEST FUNCTION

life optimization tasks, because of the fact that it does not require parameter tuning. In a future correspondence, we will investigate the performance of ESADE on high–dimensional and noisy benchmark functions. Additionally, we will further investigate the existence of an evolutionary adaptive scheme for all DE's crucial parameters, in an attempt to construct a totally self–adapting version of the DE algorithm.

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	55300	65614	78700	4087.8	100
$ESADE_2$	91300	101170	110500	3819.9	100
$ESADE_3$	82900	89776	94900	2059.8	100
$ESADE_4$	75100	90310	105700	7050.1	100
$ESADE_5$	112900	129580	144700	6221.3	100
$ESADE_6$	81700	90022	102100	3906.7	100
DE_1	48400	51016	53500	1012.6	100
DE_2	78800	82025	84500	1114.4	100
DE_3	56800	58834	60600	731.2	100
DE_4	78700	81326	83700	1161.3	100
DE_5	108600	112823	117000	1448.5	100
DE_6	68900	72001	74100	1012.6	100

TABLE X

COMPARATIVE RESULTS FOR THE SPHERE TEST FUNCTION

Mutation Strategy	Function Evaluations Min	Mean	Max	St.D.	Total Success
$ESADE_1$	860	1210.7	1700	216.1	13
$ESADE_2$	1340	2176.4	5420	603.1	100
$ESADE_3$	1460	2792.8	12500	2131.0	28
$ESADE_4$	980	3413.3	17300	3944.8	72
$ESADE_5$	1580	2658.8	4580	458.4	100
$ESADE_6$	1340	2354.0	5420	819.6	100
DE_1	520	895.7	1440	184.9	3.8
DE_2	1000	1884.3	4560	365.8	100
DE_3	1020	1403.8	2040	212.1	2.1
DE_4	780	1728.1	11800	636.3	46.3
DE_5	1180	2477.6	4060	397.7	100
DE_6	1020	1994.5	9620	830.9	100

TABLE XI

COMPARATIVE RESULTS FOR THE STEP TEST FUNCTION

REFERENCES

[1] T. Baeck, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[2] V. P. Plagianakos and M. N. Vrahatis, "Neural network training with constrained integer weights," in *Congress of Evolutionary Computation (CEC'99)*, P. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, Eds. Washington D.C., U.S.A.: IEEE Press, 1999, pp. 2007–2013.

[3] ——, "Parallel evolutionary training algorithms for 'hardware–friendly' neural networks," *Natural Computing*, vol. 1, pp. 307–322, 2002.

[4] R. Storn and K. Price, "Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[5] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[6] U. K. Chakraborty, *Advances in Differential Evolution*. Springer Publishing Company, Incorporated, 2008.

[7] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *IEEE Congress on Evolutionary Computation (CEC 2004)*, vol. 2, 2004, pp. 1980–1987.

[8] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," *Soft Comput.*, vol. 10, no. 8, pp. 673–686, 2006.

[9] A. Salman, A. P. Engelbrecht, and M. G. Omran, "Empirical analysis of self-adaptive differential evolution," *European Journal of Operational Research*, vol. 183, no. 2, pp. 785–804, 2007.

[10] J. Brest, B. Boskovic, S. Greiner, V. Zumer, and M. S. Maucec, "Performance comparison of self-adaptive and adaptive differential evolution algorithms," *Soft Comput.*, vol. 11, no. 7, pp. 617–629, 2007.

[11] J. Liu and J. Lampinen, "On setting the control parameters of the differential evolution algorithm," in *Proceedings of MENDEL 2000, 8th International Mendel Conference on Soft Computing*, June 2002, pp. 11–18.

[12] T. Bäck, "Adaptive business intelligence based on evolution strategies: some application examples of self-adaptive software," *Inf. Sci. Appl.*, vol. 148, no. 1-4, pp. 113–121, 2002.

[13] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[14] A. Qin and P. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 2, pp. 1785–1791 Vol. 2, Sept. 2005.

[15] V. Huang, A. Qin, and P. Suganthan, "Self-adaptive differential evolution algorithm for constrained real-parameter optimization," *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pp. 17–24, 0-0 2006.

[16] K. E. Parsopoulos and M. N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," *Natural Computing: an international journal*, vol. 1, no. 2-3, pp. 235–306, 2002.

[17] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," in *Proceedings of MENDEL 2003, 9th International Mendel Conference on Soft Computing*, P. O. R. Matouek, Ed., 2003, pp. 41–46.

[18] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Comput.*, vol. 9, no. 6, pp. 448–462, 2005.

[19] H. Abbass, "The self-adaptive pareto differential evolution algorithm," *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 1, pp. 831–836, May 2002.

[20] R. Storn, "System design by constraint adaptation and differential evolution," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 22–34, 1999.

[21] H. Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," *Journal of Global Optimization*, vol. 27, pp. 105–129, 2003.

[22] N. G. Pavlidis, D. K. Tasoulis, V. P. Plagianakos, and M. N. Vrahatis, "Human designed vs. genetically programmed differential evolution operators," in *IEEE Congress on Evolutionary Computation (CEC 2006)*, 2006, pp. 1880–1886.

[23] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm," *ACM Transactions On Mathematical Software*, vol. 13, no. 3, pp. 262–280, 1987.

[24] K. D. Jong, "An analysis of the behaviour of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, 1975.

[25] A. Griewank, "Generalized descent for global optimization," *Journal of optimization theory and applications*, vol. 34, no. 1, pp. 11–39, 1981.

[26] A. Levy, A. Montalvo, S. Gomez, and A. Galderon, *Topics in Global Optimization*. Springer-Verlag, New York, 1981.

[27] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution," in *IEEE Congress on Evolutionary Computation (CEC 2004)*, 2004.

[28] D. Tasoulis, V. Plagianakos, and M. Vrahatis, "Clustering in evolutionary algorithms to efficiently compute simultaneously local and global minima," in *IEEE Congress on Evolutionary Computation (CEC 2005)*, vol. 2, Sept 2005, pp. 1847–1854.

[29] M. Epitropakis, V. Plagianakos, and M. Vrahatis, "Balancing the exploration and exploitation capabilities of the differential evolution algorithm," *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pp. 2686–2693, June 2008.