Self-Adaptive Mutation in the Differential Evolution

Self-* Search

Rodrigo C. Pedrosa Silva
Department of Electrical Engineering
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
rcpsilva@gmail.com

Rodolfo A. Lopes
Department of Electrical Engineering
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
rodolfo.ufop@gmail.com

Frederico G. Guimarães
Department of Electrical Engineering
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
fredericoguimaraes@ufmg.br

ABSTRACT

The Differential Evolution (DE) algorithm is an efficient and powerful evolutionary algorithm (EA) for solving optimization problems. However the success of DE in solving a specific problem is closely related to appropriately choosing its control parameters. Parameter tuning leads to additional computational costs because of time-consuming trial-and-error tests. Self-adaptation, in contrast, allows the algorithm to reconfigure itself, automatically adapting to the problem being solved. There are in the literature some self-adaptive versions of differential evolution, however they do not align completely with self-adaptation concepts. In this paper, some self-adaptive versions of DE in the literature are described and discussed, and then a new Self-Adaptive Differential Evolution with multiple mutation strategies is proposed; it is called Self-adaptive Mutation Differential Evolution (SaMDE) and aims at preserving the essential characteristics of self-adaptation. Some computational experiments which illustrate algorithm behaviour and a comparative test with the classical DE and with an important self-adaptive DE are presented. The results suggest that SaMDE is a very promising algorithm.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Stochastic programming, Global optimization*

General Terms

Algorithms

Keywords

Differential Evolution, Self-Adaptation, Numerical Optimization, Evolutionary Algorithms

1. INTRODUCTION

GECCO'11, July 12–16, 2011, Dublin, Ireland.

The Differential Evolution (DE) algorithm is a simple yet powerful evolutionary algorithm (EA) proposed in the mid 1990s for optimization with continuous variables [23, 24]. In this context, it is an important optimizer for both single objective optimization [18, 13, 6] and multiobjective problems [28, 2], and, more recently, it has also been successfully applied to combinatorial optimization problems [16, 17, 15].

Like most evolutionary algorithms, DE uses a population of solutions and has mutation, recombination and selection operators. The algorithm has some parameters that must be defined: the population size, the scale factor of the perturbations generated by mutation and the recombination probability. As shown in [5], the choice of suitable parameter settings is critical on DE performance and depends on the problem being solved. Usually this may lead to additional computational costs due to the time-consuming trial-and-error parameter tuning process. In this context, self-adaptation has proven to be highly beneficial in automatically and dynamically adjusting evolutionary parameters [3, 22].

In Self-adaptation the control parameters are encoded into the genotype of the individuals and undergo the actions of the genetic operators. The idea is that good values of these encoded parameters tend to lead to better individuals which are more likely to survive and, hence, more likely to propagate these good parameter values. The self-adaptation allows the algorithm to adapt to any kind of problem reconfiguring itself without the need of user interaction.

There are, in the literature, some works related to self-adaptation in differential evolution [5, 19, 25, 26, 27]. However, when dealing with self-adaptation, the basic premise is that the self-adapted parameters must be implicitly or indirectly evaluated by the selection operator, which means that they should influence the fitness of the individual to whom they are related. Additionally, there should be some learning capability in the adaptation process, in which previous parameter values are taken into account in the production of new values. These characteristics are not always observed in self-adaptation mechanisms available in the literature.

In this paper, three self-adaptive versions of differential evolution are briefly described and analyzed, then a new self-adaptive Differential Evolution method with multiple mutation strategies is proposed, called Self-adaptive Mutation Differential Evolution (SaMDE). The new approach aims at preserving the essential characteristics of self-adaptation. The proposed method self-adapts parameters that can be

implicitly evaluated during the selection procedure and uses the "knowledge" acquired by the algorithm during the search process for building new parameter values. In the end some computational tests in well known test functions are presented. The results suggest that SaMDE is a very promising algorithm for general-purpose optimization and show that an adequate use of self-adaptation strategies can indeed lead to improved performance in differential evolution.

This paper is organized as follows: Section 2 presents the basic DE and reviews other self-adaptive approaches for differential evolution proposed in the literature; Section 3 describes the new self-adaptive approach; Section 4 presents a picture of the new approach behaviour; Section 5 presents the results obtained in a benchmark of test functions; and Section 6 concludes the paper.

2. BACKGROUND

2.1 Basic Differential Evolution

In this section, the basic DE algorithm is briefly reviewed, see [23, 24, 18] for additional details. Like other evolutionary algorithms, the original DE algorithm works with a population of candidate solutions randomly generated within the domain region of the problem, usually described as:

$$\mathcal{X} = \left\{ \mathbf{x} \in \mathbb{R}^n : x_k^{min} \leq x_k \leq x_k^{max}, k = 1, \ldots, D \right\} \quad (1)$$

where $x_k^{\min}$ and $x_k^{\max}$ are respectively the low and upper limits of each variable and D is the problem dimension, i.e., the number of variables in the problem.

We adopt in this paper the notation $x_{g,i,j}$ such that $g = 1, \ldots, G$ represents the generation counter; $i = 1, \ldots, NP$ represents the index of the individual in the population; and $j = 1, \ldots, D$ represents the variable index. A given individual is represented by:

$$\mathbf{x}_{g,i} = \left\langle x_{g,i,1}, x_{g,i,2}, x_{g,i,3}, ..., x_{g,i,D} \right\rangle \quad (2)$$

New individuals are generated by using the differential mutation. The mutation is based on the difference between two individuals randomly chosen from the current population. This differential vector is multiplied by a constant and added to a third individual, called base vector (base solution), leading to the so-called mutant vector:

$$\mathbf{v}_{g,i} = \mathbf{x}_{g,r_1} + F\left(\mathbf{x}_{g,r_2} - \mathbf{x}_{g,r_3}\right) \quad (3)$$

where $r_1 \neq r_2 \neq r_3 \in \{1, \ldots, NP\}$ are mutually distinct random indices, and F is a differential weight, a scale factor applied to the differential vector. For each $\mathbf{x}_{g,i}$ in the population a corresponding mutant solution $\mathbf{v}_{g,i}$ is generated.

A trial vector $\mathbf{u}_{g,i}$ is produced through recombination of $\mathbf{x}_{g,i}$ and $\mathbf{v}_{g,i}$. In the basic DE algorithm, the discrete recombination with probability CR is used, as we can see in the following scheme:

$$\mathbf{u}_{g,i,j} = \begin{cases} \mathbf{v}_{g,i,j}, & \text{if } \mathcal{U}_{[0,1]} \leq CR \\ \mathbf{x}_{g,i,j}, & \text{otherwise} \end{cases} \quad (4)$$

where $\mathcal{U}_{[a,b]}$ represents the sampling of a random variable with uniform distribution in the interval $[a, b]$. In this way, F and CR represent control parameters of the algorithm.

Finally, the trial vector $\mathbf{u}_{g,i}$ competes with the current solution $\mathbf{x}_{g,i}$ based on their objective function evaluations. If the trial solution is better or equal than the current solution, it replaces the current solution, otherwise the current solution survives while the trial one is eliminated, as described below:

$$\mathbf{x}_{g+1,i} = \begin{cases} \mathbf{u}_{g,i} & \text{if } f(\mathbf{u}_{g,i}) \leq f(\mathbf{x}_{g,i}) \\ \mathbf{x}_{g,i} & \text{otherwise} \end{cases} \quad (5)$$

Other variants of this basic scheme are presented and discussed in [18, 13, 6].

2.2 jDE algorithm

In order to avoid the trial-and-error method used for tuning the control parameters, Brest et al. proposed the jDE algorithm [4]. In jDE, the control parameters F and CR are encoded into the individual representation, which will have the following aspect:

$$\mathbf{x}_i = \left\langle x_{i,1}, x_{i,2}, x_{i,3}, ..., x_{i,D}, F_i, CR_i \right\rangle \quad (6)$$

Throughout the algorithm execution new control parameters $F_{g+1,i}$ and $CR_{g+1,i}$ are calculated as:

$$F_{g+1,i} = \begin{cases} F_l + F_u * \mathcal{U}_{[0,1]}, & \text{if } \mathcal{U}_{[0,1]} < \tau_1 \\ F_{g,i}, & \text{otherwise} \end{cases} \quad (7)$$

$$CR_{g+1,i} = \begin{cases} \mathcal{U}_{[0,1]}, & \text{if } \mathcal{U}_{[0,1]} < \tau_2 \\ CR_{g,i}, & \text{otherwise} \end{cases} \quad (8)$$

where τ_1 and τ_2 represent probabilities to adjust control parameters F and CR, respectively. The authors suggest the values $\tau_1 = 0.1$, $\tau_2 = 0.1$, $F_l = 0.1$ and $F_u = 0.9$. The new F takes a value from $[0.1, 1.0]$, and the new CR from $[0, 1]$ in a random manner. $F_{g+1,i}$ and $CR_{g+1,i}$ are obtained before the mutation is performed. In this way they influence the mutation, crossover and selection operations of the new vector $\mathbf{x}_{g+1,i}$.

Although jDE has proven to improve the robustness of the original DE [4], it does not perform a real self-adaptation of parameters. In jDE there is no learning, since the parameter update equations do not take into account the previous values, not using the knowledge gained during the search process. It merely resets the control parameters with probabilities τ_1 and τ_2. Notice that good values may eventually be lost.

2.3 SaDE algorithm

The self-adaptive differential evolution (SADE) was first proposed in [20]. In addition to automatically adapting the parameters, it also adapts multiple differential mutation strategies. Those variations are presented and discussed in [13]. In a recent version of this algorithm presented in [19] the individual has the following aspect:

$$x_i = \left\langle x_{i,1}, ..., x_{i,D}, F_i, CR_i^1, CR_i^2, CR_i^3, CR_i^4, p_i^1, p_i^2, p_i^3, p_i^4 \right\rangle \quad (9)$$

where p_i^k for $k = 1, 2, 3, 4$ is the probability of using a given mutation strategy in the generation of the mutant individual, with $\sum_{k=1}^{4} p_i^k = 1$. As we can see each individual has a CR for each strategy and a common F for all strategies.

Initially the probabilities p_i^k of each individual are set as 0.25, after LP generations (LP refers to learning period), these probabilities will be updated at each subsequent generation based on the success and failure "memories". For

example, at the generation G, the probability of choosing the kth strategy is updated by:

$$p_{i,G}^k = \frac{S_{i,G}^k}{\sum_{k=1}^{4} S_{i,G}^k} \quad (10)$$

where:

$$S_{i,G}^k = \frac{\sum_{g=G-LP}^{G-1} n_{s,G}^k}{\sum_{g=G-LP}^{G-1} n_{s,G}^k + \sum_{g=G-LP}^{G-1} n_{f,G}^k} + 0.01 \quad (11)$$

$S_{i,G}^k$ represents the success rate of trial vectors generated by the kth strategy and it is calculated by means of $n_{s,G}^k$, which is the number of trial vectors generated by the kth strategy that had a better fitness value than their parent, and $n_{f,G}^k$, which is the number of trial vectors generated by the kth strategy that had not a better fitness value than their parent.

During the first LP generations the control parameter CR of each strategy is updated as follows:

$$CR_i^k \leftarrow CR_i^k + \mathcal{N}_{[0,0.1]} \quad (12)$$

where $\mathcal{N}_{[a,b]}$ represents the sampling of a random variable with normal distribution, with mean a and standard deviation b.

During the LP generations the set CR_s^k is created. The set CR_s^k is composed by the CR_i^k values that generated individuals which were better than their parents. Once this set is ready, the CR_i^k values are updated as follows:

$$CR_i^k = \mathcal{N}_{[\text{median}(CR_s^k),0.1]} \quad (13)$$

Finally, F is updated as follows, notice that the parameter update is done before the operators application:

$$F_i = \mathcal{N}_{[0.5,0.3]} \quad (14)$$

The main contribution of this approach, is the use of multiple mutation strategies, what has proven to be beneficial [20, 3, 19]. However, the mechanism chosen to select between strategies takes into account only the number of better solutions created by a specific strategy. This can be a poor measure once it does not see the "amount of improvement" generated.

Other points that should be highlighted in this approach are:

- The proposed mechanism adds at least four new parameters in the algorithm, the standard deviations in equations (12)-(14) and the mean value in (14). Although the authors have presented them as fixed values, their influence is not known in the search process.
- As in jDE, the mechanism adopted to vary F can destroy good parameter values and it is not based on learning or previous values;
- It has just one F shared by all strategies. Given that the vectors distribution and sizes generated by each strategy are different it seems intuitive to have different values of F for each one.

2.4 DESAP algorithm

The Differential Evolution with Self-adapting Population (DESAP) was proposed in [25]. It has the self-adaptation of the population size, given by the parameter NP, as its main characteristic. As well as the parameter CR a new parameter M is also self-adapted. The individual representation is shown bellow:

$$\mathbf{x}_i = \langle x_{i,1}, x_{i,2}, x_{i,3}, ..., x_{i,D}, M_i, CR_i, NP_i \rangle \quad (15)$$

In each generation, each individual $\mathbf{x}_i$ enters a phase called by the author as recombination. In this phase the control parameters and the problem variables are varied by means of differential mutation, see (3), with probability CR_i. F is maintained fixed at the value 1.0 throughout the algorithm execution.

After recombination, the individual $\mathbf{x}_i$ enters a mutation phase with probability M_i, in which the control parameters and problem variables are varied by means of perturbations with normal distribution.

The DESAP algorithm has a real self-adaptive mechanism to update the control parameters. It takes into account the previous values and learns gradually, good parameter values. However the following cons may be highlighted:

- The control parameters are updated after the execution of the operators, with this, if a good solution is generated the parameters that will be disseminated on the population are not the parameters that generated that solution, what can create good solutions with poor encoded parameters, as reported in [7];
- The NP parameter does not seem to be a good candidate for self-adaptation, once its contribution for an isolated individual fitness is irrelevant, which means that this parameter can not be evaluated implicitly by the selection operator;
- There is no adaptation of the parameter F, to which the DE performance is very sensitive as shown in [5].

3. SAMDE: PROPOSED ALGORITHM

With the goal of improving DE robustness and adaptability to a bigger set of problems, without the need of expensive trial-and-error tests to set control parameters and learning strategies and in order to improve the existent "self-adaptive"[1] DE algorithms, here we present the Self-adaptive Mutation Differential Evolution (SaMDE), which makes use of multiple mutation strategies, and uses differential evolution itself to self-adapt both the control parameters and the odds of choosing a particular mutation strategy. First of all, in SaMDE, individuals have the following aspect:

$$\mathbf{x}_i = \langle x_{i,1}, ..., x_{i,D}, V_i^1, ..., V_i^k, F_i^1, CR_i^1, ..., F_i^k, CR_i^k \rangle \quad (16)$$

where $x_{i,d}$ are the problem variables, k is the number of different mutation strategies, $V_i^k \in [0,1]$ corresponds to a value attributed to the kth strategy (this will be used for the strategy selection), $F_i^k \in [0.1, 1]$ and $CR_i^k \in [0, 1.0]$ are the control parameters related to the kth strategy.

In the population initialization, the problem variables and parameters are chosen randomly. Then for each individual, first the values V are updated with differential mutation as follows:

$$V_i^k = V_{r1}^k + F'(V_{r2}^k - V_{r3}^k) \quad (17)$$

[1]As discussed in the previous section, some approaches are not truly self-adaptive. Some methods actually employ adaptive mechanisms or are based on randomly resetting parameters.

With the updated Vs , the strategy that will be applied on the current individual is chosen by means of the roulette wheel algorithm, where strategies with bigger values of V are more likely to be chosen.

Once the winner strategy w, $w \in \{1, 2, .., k\}$, is selected their related values F^w and CR^w are also updated by differential mutation, as follows:

$$F_i^w = F_{r1}^w + F'(F_{r2}^w - F_{r3}^w) \quad (18)$$

$$CR_i^w = CR_{r1}^w + F'(CR_{r2}^w - CR_{r3}^w) \quad (19)$$

Then the algorithm continues normally, operating the mutation with the winner strategy and doing the traditional recombination and selection steps. Notice that only the control parameters of the winner strategy should be updated, as they are the only ones to contribute to the trial solution fitness. $F' = \mathcal{U}_{[0.7,1]}$ in all cases. Although F' is an additional parameter of the self-adaptive mechanism, the performance of the algorithm is not very sensitive to this parameter in the indicated range, as shown in the results. For sake of clarity a pseudo-code of SaMDE is presented in Algorithm 1.

Algorithm 1: SaMDE

```
Initialize population;
while ¬ stop_condition do
    for each individual do
        Select a new F' ∈ [0.7, 1];
        Update(V_i^1, ..., V_i^k);
        for Each Strategy k do
            V_i^k = V_r1^k + F'(V_r2^k − V_r3^k) ;
        end
        Select a Strategy (w) by Roullete Wheel;
        Update winner strategy parameters;
        F_i^w = F_r1^w + F'(F_r2^w − F_r3^w);
        CR_i^w = CR_r1^w + F'(CR_r2^w − CR_r3^w);
        Apply the selected mutation strategy using F_i^w;
        Perform Recombination with CR_i^w;
        Apply Selection;
    end
end
```

Due to their search diversity, four mutation strategies were selected here, they are the same selected in [19] and are described below:

1. `rand/1` It is the original differential mutation (see [23]), usually demonstrates slow convergence speed and bears stronger exploration capability [19].

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,r1} + F(\mathbf{x}_{t,r2} - \mathbf{x}_{t,r3}) \quad (20)$$

2. `best/1` Usually has the fastest convergence speed and performs well when solving unimodal problems (see [13]).

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,best} + F(\mathbf{x}_{t,r2} - \mathbf{x}_{t,r3}) \quad (21)$$

3. `rand/2` In this strategy, the statistical distribution of the summation of all two-difference vectors have a bell shape, that is generally regarded as a better pertubation mode [19].

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,r1} + F(\mathbf{x}_{t,r2} - \mathbf{x}_{t,r3}) + F(\mathbf{x}_{t,r4} - \mathbf{x}_{t,r5}) \quad (22)$$

4. `current-to-rand/1` It is a rotation-invariant strategy. Its efectiveness has been verified when it was applied to solve multi-objective optimization problems [11].

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,i} + F(\mathbf{x}_{t,r1} - \mathbf{x}_{t,i}) + F(\mathbf{x}_{t,r2} - \mathbf{x}_{t,r3}) \quad (23)$$

In order to keep the updated control parameters in the boundaries, the mechanism proposed in [21] was used. In this mechanism the amount of violation is reflected back to the bound, as can be seen in the following equation.

$$x = \begin{cases} 2 \times x_{low} - x & \text{if } x < x_{low} \\ 2 \times x_{upp} - x & \text{if } x > x_{upp} \end{cases} \quad (24)$$

where x represents the updated control parameter, x_{low} and x_{upp} are the lower and the upper bounds, respectively.

The SaMDE algorithm employs different mutation strategies and self-adapts all its important control parameters. The population size is fixed and the parameters are updated before they are used for generating new solutions. This allows the operators to be influenced by the new updated parameter values, allowing them to be indirectly evaluated by the selection operator.

Furthermore differently from jDE, SaMDE presents learning capability, using the "knowledge" obtained during the search process to adapt control parameters and the use of mutation strategies. All this is done by introducing only one new parameter, F', which seems to be easy to set as will be seen in Section 4.1.

In the next two sections we provide two types of experimental results. The goal of the following experiment is to provide a picture of SaMDE behavior. In Section 5 comparative tests are performed in a benchmark of functions.

4. SAMDE BEHAVIOR

For these experiments, commonly used test functions were chosen from literature [5, 12, 20, 19] (see, Appendix A). Functions f_1 to f_3 are unimodal, functions f_4 to f_6 are multimodal and the number of local optima grows exponentially with the number of dimensions. They all have the number of dimensions D equal to 30. The global maximum of f_4 is -12569.5, and the minimum of all other functions is 0.

SaMDE algorithm was executed 30 times in each function with a stop criterion of 3000 generations for the unimodal functions and 6000 for the multimodal functions. The population size was set to 100 individuals. The goals here are: (i) evaluate the sensitivity of the algorithm to the parameter F' used for the self-adaptation and (ii) show the variation of the mutation strategies during the optimization process.

4.1 Study of the parameter F'

Here are presented the averages of the best values found in the 30 independent executions by SaMDE algorithm, varying the parameter F' in equations (17), (18) and (19). Table 1 shows the results for the unimodal functions and Table 2 shows the results for the multimodal functions.

As can be seen, in general, higher values of F' ($F' \in [0.7, 1.0]$) lead to better results, possibly because small values of F' reduce the capacity of adapting parameters, by diminishing the effect of the perturbations applied to current values. Based on these experiments, we have decided to vary F' only between 0.7 and 1 in the proposed algorithm.

Values of F'	f1	f2	f3
0.1	2.16	2.59	203.09
0.2	0.38	2.05	57.95
0.3	1.59E-29	1.01	2.98E-14
0.4	2.36E-122	0.63	1.77E-19
0.5	2.41E-156	0.16	2.98E-23
0.6	2.73E-162	0.04	6.21E-28
0.7	1.35E-157	2.20E-30	9.95E-30
0.8	1.34E-154	3.43E-47	1.14E-29
0.9	1.62E-146	2.53E-54	1.43E-31
1.0	1.05E-143	1.17E-61	8.15E-31

Table 1: Averages of the best value found in the unimodal functions by varying F'.

Values of F'	f4	f5	f6
0.1	-9478.40	63.259	0.1568
0.2	-9384.40	55.246	0.1226
0.3	-9431.60	47.157	0.0432
0.4	-9835.30	48.554	0.0232
0.5	-9750.80	45.801	0.0066
0.6	-9905.40	46.133	0.0179
0.7	-9867.30	41.722	0.0086
0.8	-10120.00	40.694	0.0095
0.9	-10418.00	40.694	0.0054
1.0	-10332.00	38.206	0.0167

Table 2: Averages of the best value found in the multimodal functions by varying F'.

4.2 Self-adaptation of the parameters V

This section illustrates the variation of strategies over the optimization process. For a given generation, the number of executions of each strategy was taken, then the average of the 30 independent runs was ploted.

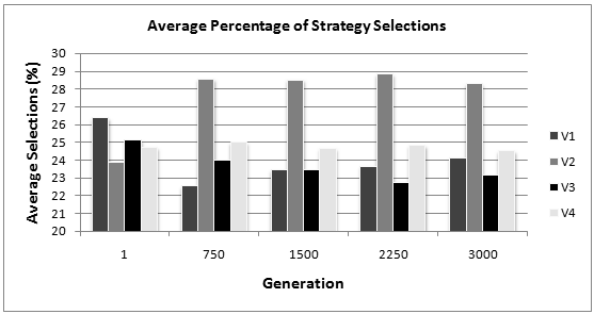
In Figures 1 and 2, the Vk, $k = 1, ..., 4$ refers to each mutation strategy, in the following sequence: rand/1, see (20); best/1, see (21); rand/2, see (22); and current-to-rand/1, see (23).

Figure 1 shows the variation in the unimodal functions. As was expected, there was a predominance of the best/1 strategy, usually known in the literature as the best option for this kind of function. This result indicates that the proposed mechanism is able to identify the best mutation strategy for a given problem.

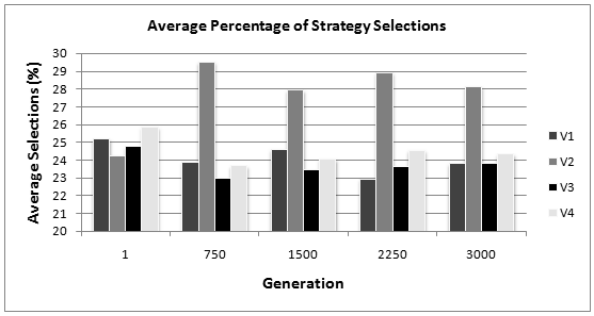
Figure 2 shows the variation in the multimodal functions. Although the best mutation strategy is not clear it is important to notice that the algorithm is able to switch to others mutation strategies during its execution. Additionally the results indicate that the best stategy changes throughout the various stages of the optimization process.

5. COMPARATIVE TESTS

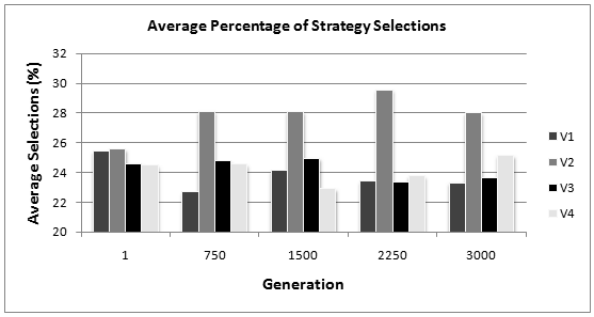
This section presents comparative results between the self-adaptive algorithms SaMDE and jDE and the original DE with fixed parameters ($F = 0.5$ and $CR = 0.9$ as suggested in [3]) on BBOB-2010 – the *Real-Parameter Black-Box Optimization Benchmark: noisyless functions* presented at the *Genetic and Evolutionary Computation Conference*



(a) Function f_1



(b) Function f_2



(c) Function f_3

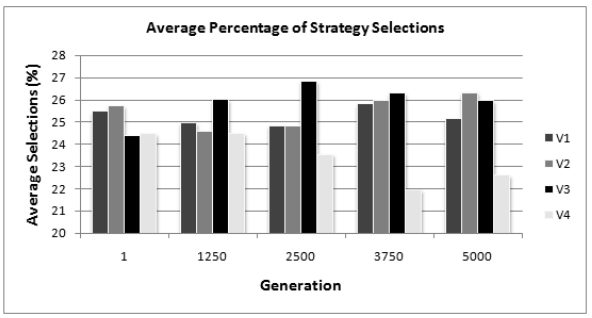
Figure 1: Avereged percentage of selection of each mutation strategy for unimodal function.

(GECCO) 2010, and described in [9]. The experimental procedure is detailed in [8]. It must be remarked that in order to perform a fair comparison, the various algorithms have been run with the same population size fixed in 100 individuals. The choice of jDE for comparison was based on the results presented in [14], in which jDE presented the best results in most test problems when compared with other modified structures of DE. Versions of jDE with multiple mutation strategies (see [3]) were not implemented, because no automatic mechanism for choosing between strategies was presented.

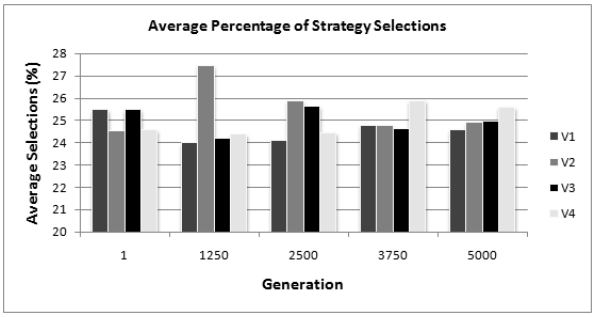
Figure 3 depicts the empirical cumulative distribution of runtimes[2] (RTs) of each algorithm on all functions $f1$-$f24$, grouped by number of dimensions (D). For each function with a given dimension, different target precision values[3]

[2]Runtime (RT) is the number of function evaluations until the optimal solution (with a certain precision) was reached.
[3]The target precision value $\Delta f_t = f_{target} - f_{optimun}$.

(a) Function f_4



(b) Function f_5



(c) Function f_6

Figure 2: Avereged percentage of selection of each mutation strategy for multimodal function.

are use within the range $\Delta f_t \in [10, 10^{-8}]$. Each graph in Figure 3 shows the proportion of solved problems, in a given dimension, by the runtime. For more detailed information about the graphs see [10].

As can be seen, SaMDE had the best performance of the three tested algorithms regardless of the dimensions. It was able to improve DE performance, in all groups of tests, solving a higher number of problems with a smaller RT. Although jDE and DE have had a similar performance among them, jDE seems to improve DE in problems with higher dimensions.

Despite still having a timid performance compared to the state-of-the-art algorithms (see, [1]), the experiments illustrate that the proposed approach for self-adapting the parameters of DE is promising and indeed improves the performance of the basic DE and jDE.

6. CONCLUSION

In this paper, a new Self-Adaptive Differential Evolution which aims at preserving the essential characteristics of self-adaptation is presented. The proposed approach uses multiple mutation strategies and in addition to self-adapting the control parameters F and CR for the different strategies, it also self-adapts the chance of using a particular mutation strategy.

The experiments on well known test functions indicate that the proposed approach has a good potential in learning parameters and switching to a particular mutation strategy. On the comparative results the proposed algorithm presented the best performance on the tested functions. The self-adaptation of the mutation strategies and its parameters has shown to be beneficial to the algorithmic performance. In addition, the use of a unique mutation strategy seems to be inadequate once the best one dynamically changes during in the optimization process.

In future work, the authors intend to implement a local search method in order to make SaMDE performance comparable to the state-of-the-art algorithms. The authors also intend to run tests in noisy functions to analyze SaMDE behavior in this kind of problem.

7. ACKNOWLEDGMENTS

The authors would like to thank the Universidade Federal de Ouro Preto by courtesy structure and the following Brazilian agencies for the financial support: the National Council for Scientific and Technological Development (CNPq) and the Research Foundation of the State of Minas Gerais (FAPEMIG).

8. REFERENCES

[1] A. Auger, S. Finck, N. Hansen, and R. Ros. BBOB 2010: Comparison Tables of All Algorithms on All Noiseless Functions. Technical Report RT-388, INRIA, 09 2010.

[2] L. S. Batista, F. G. Guimarães, and J. A. Ramírez. A differential mutation operator for the archive population of multi-objective evolutionary algorithms. In *CEC'09: Proceedings of the Eleventh Congress on Evolutionary Computation*, pages 1108–1115, Piscataway, NJ, USA, 2009. IEEE Press.

[3] J. Brest, B. Boskovic, S. Greiner, V. Zumer, and M. Maucec. Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 11:617–629, 2007. 10.1007/s00500-006-0124-0.

[4] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, November 2006.

[5] J. Brest and M. S. Maucec. Control parameters in self-adaptive differential evolution, 2006.

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

[7] D. K. Gehlhaar and D. B. Fogel. Tuning evolutionary programming for conformationally flexible molecular docking. In *Evolutionary Programming*, pages 419–429, 1996.

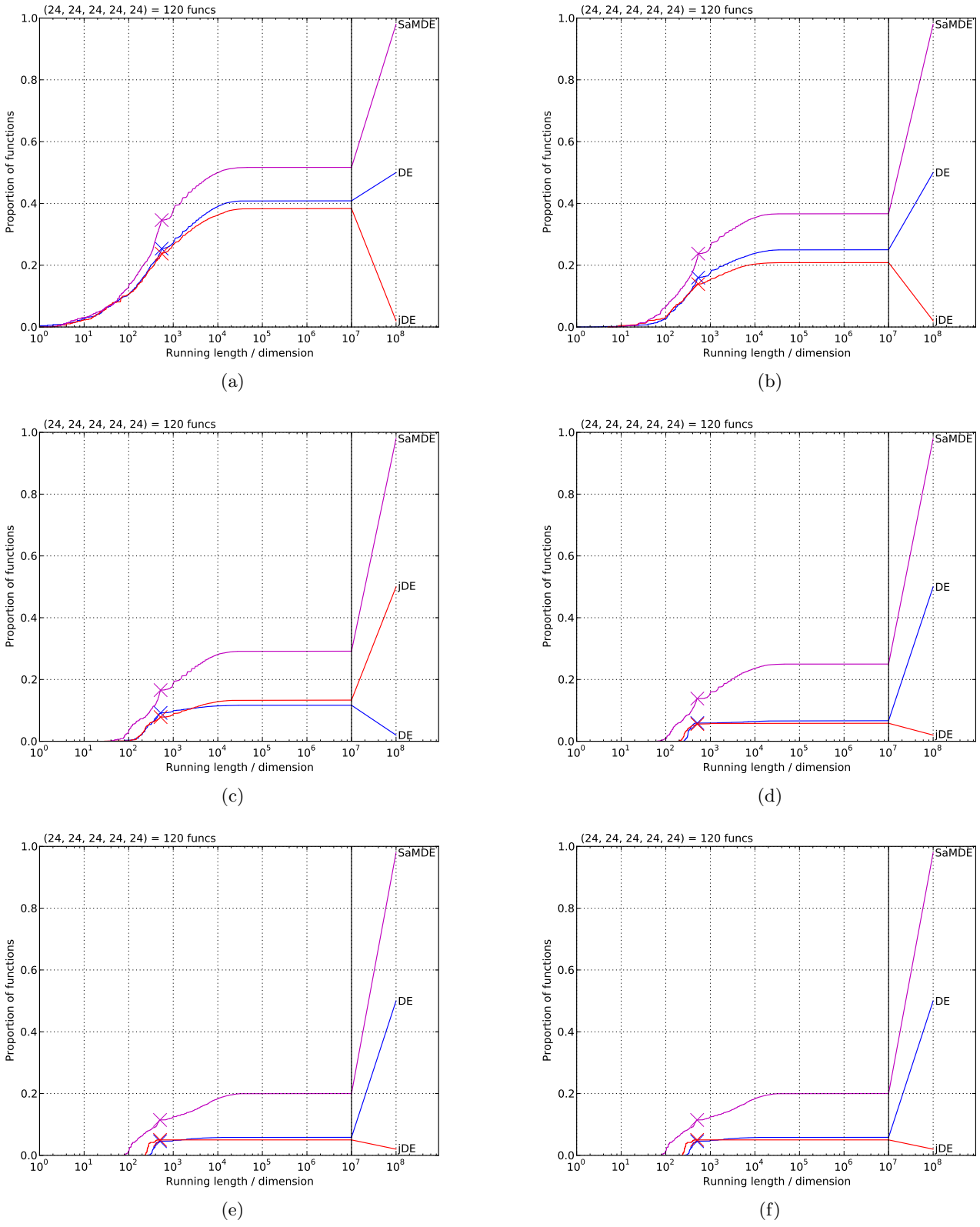


Figure 3: Empirical runtime distributions on all functions with $\Delta f_t \in [10, 10^{-8}]$ in dimension 2, 3, 5, 10, 20 e 40.

[8] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup. Research Report RR-7215, INRIA, 03 2010.

[9] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions. Technical report, INRIA, 2010.

[10] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, GECCO '10, pages 1689–1696, New York, NY, USA, 2010. ACM.

[11] A. W. Iorio and X. Li. Solving rotated multi-objective optimization problems using differential evolution. In *In AI 2004: Advances in Artificial Intelligence: 17th Australian Joint Conference on Artificial Intelligence*, pages 861–872. press, 2004.

[12] J. Liu and J. Lampinen. On setting the control parameter of the differential evolution method. In *8 th Int. Conf. Soft Computing (MENDEL2002)*, pages 11–18, 2002.

[13] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 485–492, New York, NY, USA, 2006. ACM.

[14] F. Neri and V. Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artif. Intell. Rev.*, 33(1-2):61–106, 2010.

[15] G. C. Onwubolu and D. Davendra. *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*. Springer Publishing Company, Incorporated, 2009.

[16] R. S. Prado, R. C. Pedros Silva, F. G. Guimaraes, and O. M. Neto. A new differential evolution based metaheuristic for discrete optimization. *International Journal of Natural Computing Research (IJNCR)*, 1:15–32, 2010.

[17] R. S. Prado, R. C. Pedrosa Silva, F. G. Guimaraes, and O. M. Neto. Using differential evolution for combinatorial optimization: A general approach. In *In 2010 IEEE International Conference on Systems Man and Cybernetics (SMC)*, pages 11–18. IEEE Press, 2010.

[18] K. V. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution A Practical Approach to Global Optimization*. Natural Computing Series. Springer-Verlag, Berlin, Germany, 2005.

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

[20] A. K. Qin and P. N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 1785–1791. IEEE Press, 2005.

[21] J. Ronkkonen, S. Kukkonen, and K. Price. Real-parameter optimization with differential evolution. In *The 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 506 –513 Vol.1, sept. 2005.

[22] H. Schwefel. *Numerical optimization of computer models*. Wiley, Chichester, WS, UK, 1981.

[23] R. Storn and K. Price. Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, 1995.

[24] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, 1997.

[25] J. Teo. Exploring dynamic self-adaptive populations in differential evolution. *Soft Comput.*, 10(8):673–686, 2006.

[26] V. Tirronen, F. Neri, and T. Rossi. Enhancing differential evolution frameworks by scale factor local search - part i. In *IEEE Congress on Evolutionary Computation*, pages 94–101, 2009.

[27] M. Weber, V. Tirronen, and F. Neri. Scale factor inheritance mechanism in distributed differential evolution. *Soft Comput.*, 14(11):1187–1207, 2010.

[28] F. Xue, A. C. Sanderson, and R. J. Graves. Pareto-based multi-objective differential evolution. In *Congress of Evolutionary Computation, 2003. CEC '03*, volume 2, pages 862–869, 2003.

APPENDIX

A. TEST FUNCTIONS

$$f_1(x) = \sum_{i=1}^{D} x_i^2 \ , x_i \in [-100, 100] \tag{25}$$

$$f_2(x) = \sum_{i=1}^{D} |x_i| + \prod_{i=1}^{D} |x_i| \ , x_i \in [-10, 10] \tag{26}$$

$$f_3(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2 \ , x_i \in [-100, 100] \tag{27}$$

$$f_4(x) = \sum_{i=1}^{D} -x_i \sin(\sqrt{|x_i|}) \ , x_i \in [-500, -500] \tag{28}$$

$$f_5(x) = \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i) + 10) \ , x_1 \in [-5.12, 5.12] \tag{29}$$

$$f_6(x) = \frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \ , x_i \in [-600, 600] \tag{30}$$