Saving Evaluations in Differential Evolution for Constrained Optimization

Efrén Mezura-Montes and Carlos A. Coello Coello
Evolutionary Computation Group (EVOCINV) at CINVESTAV-IPN
Electrical Engineering Department. Computer Science Section
Av. IPN No. 2508 Col. San Pedro Zacatenco México D.F. 07300, MÉXICO
`emezura@computacion.cs.cinvestav.mx`
`ccoello@cs.cinvestav.mx`

Abstract

Generally, evolutionary algorithms require a large number of evaluations of the objective function in order to obtain a good solution. This paper presents a simple approach to save evaluations, applied to a competitive differential evolution algorithm used to solve constrained optimization problems. The idea is based on the way in which differential evolution finds new promising areas of the search space. This allows to randomly assign a zero fitness to some offspring newly generated in order to avoid its evaluation and, as a secondary effect, to slow down convergence. The approach is tested using different percentages of individuals from the population, providing a competitive performance. Besides, the effect that the elimination of individuals has on convergence is also analyzed. Finally, to remark behavior differences, the approach is tested against a version with a smaller population and against a version with a simple fitness approximation method. The results obtained are discussed and some conclusions are drawn.

1. Introduction

We are interested in the general non linear programming problem in which we want to:Find $\vec{x}$ which optimizes $f(\vec{x})$ subject to: $g_i(\vec{x}) \leq 0, \;\; i = 1, \ldots, m$ $h_j(\vec{x}) = 0, \;\; j = 1, \ldots, p$ where $\vec{x} \in \mathbb{R}^n$ is the vector of solutions $\vec{x} = [x_1, x_2, \ldots, x_n]^T$, where each $x_i, \;\; i = 1, \ldots, n$ is bounded by lower and upper limits $L_i \leq x_i \leq U_i$; m is the number of inequality constraints and p is the number of equality constraints (in both cases, constraints could be linear or nonlinear). If we denote with $\mathcal{F}$ to the feasible region and with $\mathcal{S}$ to the whole search space, then it should be clear that $\mathcal{F} \subseteq \mathcal{S}$.

Evolutionary Algorithms (EAs) are widely used as powerful approaches to solve optimization problems [5]. However, in order to obtain a reasonably good solution, they typically require to perform a large number of evaluations of the objective function and also of the constraints of the problem, in case of constrained optimization (we will call OFE to the number of evaluations of the objective function and the constraints of a problem in the remainder of the paper). For many real-world problems, the evaluation of the objective function is very time-consuming. Therefore, several fitness approximation models have been proposed in order to replace some costly evaluations by objective function estimates [3]. Most of this research is focused on approximation models like polynomial models, kriging models, neural networks and support vector machines [3]. These models have proven to be effective in certain applications [9]. However, despite avoiding the evaluation of the objective function of a problem, they add an extra computational cost related to the construction and updating of the model. On the other hand, we hypothesized that, another way of saving OFEs can be based on the way the EA works. Motivated by this idea, we found that, observing the way Differential Evolution [6] finds promising regions in the search space when using its mutation operator to solve optimization problems, it is possible to sentence to death (assign a zero fitness), based on a user-defined ratio, to some offspring newly created in order to save OFEs and also to decrease the convergence speed (which may lead to premature convergence) by favoring other zones (not necessarily the best at the current time) of the search space.

This paper is organized as follows: In Section 2 we present a brief introduction to DE and our specific DE-based approach. In Section 3, a detailed description of our scheme is provided. Afterwards, in Section 4 we present the results obtained in the four phases of our experimental design and we discuss them. In Section 5 we summarize our findings.

```
Begin
    G=0
    Create a random initial population $\vec{x}^i_G \; \forall i, i = 1, \ldots, NP$
    Evaluate $f(\vec{x}^i_G) \; \forall i, i = 1, \ldots, NP$
    For G=1 to MAX_GENERATIONS Do
        F=rand[0.3,0.9]
        For i=1 to NP Do
            For k=1 to $n_o$ Do
                Select randomly $r_1 \neq r_2 \neq r_3 \neq i$
                $j_{rand}$ = randint$(1, D)$
                For j=1 to $n_o$ Do
                    If (rand$_j [0,1) < CR$ or $j = j_{\text{rand}}$) Then
                        child$_j = x^{r_3}_{j,G} + F(x^{r_1}_{j,G} - x^{r_2}_{j,G})$
                    Else
                        child$_j = x^i_{j,G}$
                    End If
                End For
⇒               If flip$(IR)$ Then
⇒                   $f(\vec{\text{child}}_i) = -\infty$
⇒                   sumviol$(\vec{\text{child}}_i) = \infty$
⇒               Else
⇒                   evaluate child with original objective function
⇒               End If
                If $k > 1$ Then
                    If (child is better than $\vec{u}^i_{G+1}$
                     (based on the three selection criteria)) Then
                        $\vec{u}^i_{G+1}$=child
                    End If
                Else
                    $\vec{u}^i_{G+1}$=child
            End For
            If flip$(S_r)$ Then
                If $(f(\vec{u}^i_{G+1}) \leq f(\vec{x}^i_G))$ Then
                    $\vec{x}^i_{G+1} = \vec{u}^i_{G+1}$
                Else
                    $\vec{x}^i_{G+1} = \vec{x}^i_G$
                End If
            Else
                If ($\vec{u}^i_{G+1}$ is better than $\vec{x}^i_G$
                 (based on the three selection criteria)) Then
                    $\vec{x}^i_{G+1} = \vec{u}^i_{G+1}$
                Else
                    $\vec{x}^i_{G+1} = \vec{x}^i_G$
                End If
            End If
        End For
        $G = G + 1$
    End For
End
```

Figure 1. Our SDDE. The steps of our saving mechanism are marked with an arrow. "∞" means to assign the worst value for the individual. randint(min,max) returns an integer number between min and max. rand$[0, 1)$ returns a real number between 0 and 1, both with uniform probability distribution. flip(W) returns 1 with probability W.

Finally, in Section 6, we provide some conclusions and we define our future work.

2. Previous Work

DE [6] is a population-based evolutionary algorithm with a simple mutation mechanism and a crossover operator that performs a linear recombination of a number of individuals (normally three) and one parent (which is the subject to be replaced) to create one child. Selection is deterministic between the parent and the child (i.e., the best of them remains in the next population). DE shares similarities with traditional EAs. However, it does not use binary encoding as the simple genetic algorithm [2] and it does not use a probability density function to self-adapt its parameters as the evolution strategy [8].

Mezura-Montes et al. [4] proposed a novel approach based on DE to solve constrained optimization problems called Diversity Differential Evolution (DDE). This constraint-handling technique consists on a comparison method based on feasibility. Also two additional modifications to the DE approach were added: (1) a diversity mechanism based on allowing the survival for the next generation of those individuals with a good value of the objective function (regardless of feasibility) and (2) allowing to each parent to generate more than one offspring, with the aim of increasing the probability of getting a better offspring, besides promoting the collaboration of more individuals with one parent to generate new solutions. The approach was compared against state-of-the-art approaches and the results were very competitive at a low computational cost, measured by the number of OFEs. The approach performed $225,000$ OFEs.

Most of the research on fitness approximation is focused in unconstrained optimization [1]. However, to the best of our knowledge, very few of these approaches have been used in constrained optimization. Won and Ray [9] compared Kriging and Cokriging surrogate models with radial basis Function models using a set of five unconstrained problems and also two well-known constrained engineering design problems. Their results showed the viability of using these models on constrained problems. Runarsson [7] used nearest neighborhood regression to solve a set of benchmark problems in constrained optimization. His results showed that, for the general nonlinear programming problem, using only the information of the closest solution to approximate the value of a new one is better than using a set of the closest solutions to average the approximation.

3. Our approach

The main motivation of this work was to find a way of reducing the computational cost of a competitive approach used to solve constrained optimization problems. However, instead of incorporating computationally expensive approximation models, the goal was to use features related to the search engine adopted in order to perform this cost reduction.

From previous research [4] and some observations, we found, based on the DE mutation and crossover operator, that the offspring generated using them is located near one of the three random individuals selected for reproduction (about 80% of the time) instead of being close to its parent. See rows 9 to 17 in Figure 1 for details. Therefore, promising regions found by the offspring are basically chosen depending on the distribution of these three individuals. If the offspring finds a more promising area than its parent, it will remain in the population. Based on this way of operation, we propose to randomly "sentence to death" some offspring in order to save its OFE. This will also allow other offspring, which could be less fit at the moment, to survive. In addition, it will promote the exploration of other regions slowing down the convergence of the approach. In this way we achieve two goals: (1) to reduce the computational cost (based on the OFEs performed by the approach) without using an approximation model and (2) to allow the algorithm to explore other regions of the search space in order to avoid premature convergence. The pseudocode of the approach is presented in Figure 1. Pointed with arrows are the steps added for the saving mechanism; we call it SDDE for "Saving Diversity Differential Evolution".

4. Experiments and Results

In the following experiments, we used 13 well-known benchmark problems for constrained optimization. Details of each problem can be found in [4]. A summary of their main features are presented in Table 1. There are problems with different features (dimensionality, type of objective function, estimated size of the feasible region with respect to the whole search space and also the number and type of constraints).

The experimental design has four phases: (1) to evaluate the effect of different ratios of the saving mechanism on the quality (best solution found overall) and robustness (best mean and standard deviation values) of the approach, (2) to analyze the effect of the saving mechanism in the convergence behavior of the algorithm, (3) to establish differences between the SDDE against the original approach with a reduced population size (this experiment aims to emphasize that saving evaluations during the evolutionary process is not equal to just reducing the population size and performing less OFEs), and (4) to compare the SDDE against a simple fitness approximation approach.

(1) Different ratios: For the first phase we used the following parameters for the SDDE: population size $NP = 36$, number of generations = 500, $CR = 0.9$, $F \in [0.3, 0.9]$ randomly chosen at each generation, number of offspring per parent $n_o = 5$ (90,036 OFEs), selection ratio $S_r = 0.45$, tolerance for equality constraints $\epsilon = 0.0001$. The saving mechanism ratio called SM was assigned with five different values $SM \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and also $SM = 0$ (original approach without saving mechanism) in order to assess its effect in the search. The statistical results of 30 independent runs are summarized in Table 2.

Problem	n	Function	ρ	LI	NI	LE	NE
g01	13	quadratic	0.0003%	9	0	0	0
g02	20	nonlinear	99.9973%	1	1	0	0
g03	10	nonlinear	0.0026%	0	0	0	1
g04	5	quadratic	27.0079%	0	6	0	0
g05	4	nonlinear	0.0000%	2	0	0	3
g06	2	nonlinear	0.0057%	0	2	0	0
g07	10	quadratic	0.0000%	3	5	0	0
g08	2	nonlinear	0.8581%	0	2	0	0
g09	7	nonlinear	0.5199%	0	4	0	0
g10	8	linear	0.0020%	3	3	0	0
g11	2	quadratic	0.0973%	0	0	0	1
g12	3	quadratic	4.7697%	0	9^3	0	0
g13	5	nonlinear	0.0000%	0	0	0	3

Table 1. Main features of the 13 test problems chosen. n is the number of decision variables, and for the constraints LI is the number of linear inequalities, NI the number of nonlinear inequalities, LE is the number of linear equalities and NE is the number of nonlinear equalities. ρ is the ratio between the feasible region and the whole search space.

Ratio	OFEs performed	OFEs saved
0.0	90036	0
0.1	81049	8987
$\mathbf{0.2}$	$\mathbf{72078}$	$\mathbf{17958}$
0.3	63087	26949
0.4	54117	35919
0.5	45118	44918

Table 3. Average OFE (out of 90036) performed and saved by the SDDE in the 13 test problems. A result in boldface indicates the percentage whose results were the best overall.

It is important to emphasize that, depending of the value of the SM parameter, there is a number of saved OFEs; these savings are presented in Table 3.

From the results in Table 2, several interesting issues are observed. There is a very competitive performance of the SDDE when the saving mechanism parameter range is $SM \in [0.1, 0.4]$. When $SM = 0.5$, the performance is severely affected (no feasible solutions are found in some runs when solving problems g01, g03, g05 and g13), and less competitive results are obtained in problems g02 and g10. There is a set of problems where the saving mecha-

Problem & Best Known Sol.	Stats	Different saving percentages tested against NO saving					
		original DDE	10%-SDDE	20%-SDDE	30%-SDDE	40%-SDDE	50%-SDDE
g01 −15.000	best	**−15.000**	**−15.000**	**−15.000**	−14.999	−14.999	∗ − 14.999
	mean	**−15.000**	**−15.000**	−14.999	−14.999	−14.999	∗ − 10.679
	worst	**−15.000**	−14.999	−14.999	−14.999	−14.998	∗ − 1.914
	St. Dev	**0**	**0**	1.0E-5	7.0E-6	4.1E-4	6.0E+0
g02 0.803619	best	**0.803585**	0.803579	0.803571	0.803568	0.803521	0.803404
	mean	0.754213	0.751654	**0.763916**	0.754336	0.760721	0.754366
	worst	0.661897	0.590035	**0.699599**	0.604008	0.660187	0.599550
	St. Dev	2.9E-2	4.6E-2	**2.6E-2**	4.5E-2	4.3E-2	4.4E-2
g03 1.000	best	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	†0.998
	mean	**1.000**	**1.000**	**1.000**	**1.000**	0.999	†0.983
	worst	**1.000**	**1.000**	0.999	**1.000**	0.997	†0.923
	St. Dev	4.1E-5	**3.8E-5**	1.3E-4	6.8E-5	8.4E-4	1.8E-2
g04 −30665.539	best	**−30665.539**	**−30665.539**	**−30665.539**	**−30665.539**	**−30665.539**	**−30665.539**
	mean	**−30665.539**	**−30665.539**	**−30665.539**	**−30665.539**	**−30665.539**	−30664.495
	worst	**−30665.539**	**−30665.539**	**−30665.539**	**−30665.539**	**−30665.539**	−30634.223
	St. Dev	**0**	**0**	**0**	**0**	**0**	5.6E+0
g05 5216.498	best	**5126.497**	**5126.497**	**5126.497**	**5126.497**	**5126.497**	**‡5126.497**
	mean	**5126.497**	**5126.497**	**5126.497**	**5126.497**	**5126.497**	**‡5126.497**
	worst	**5126.497**	**5126.497**	**5126.497**	**5126.497**	**5126.497**	**‡5126.497**
	St. Dev	**0**	**0**	**0**	**0**	**0**	9.3E-6
g06 −6961.814	best	**−6961.814**	**−6961.814**	**−6961.814**	**−6961.814**	**−6961.814**	**−6961.814**
	mean	−6960.400	**−6961.814**	−6952.279	−6960.667	**−6961.814**	**−6961.814**
	worst	−6939.776	**−6961.814**	−6780.019	−6927.400	**−6961.814**	**−6961.814**
	St. Dev	4.6E+0	**0**	3.6E+2	6.2E+0	**0**	**0**
g07 24.306	best	**24.306**	**24.306**	**24.306**	**24.306**	24.307	24.319
	mean	24.307	24.307	**24.306**	24.307	24.310	24.348
	worst	24.309	24.308	**24.307**	24.312	24.324	24.437
	St. Dev	7.7E-4	4.5E-4	**2.5E-4**	1.4E-3	3.6E-3	2.7E-2
g08 0.095825	best	**0.095825**	**0.095825**	**0.095825**	**0.095825**	**0.095825**	**0.095825**
	mean	**0.095825**	**0.095825**	**0.095825**	**0.095825**	**0.095825**	**0.095825**
	worst	**0.095825**	**0.095825**	**0.095825**	**0.095825**	**0.095825**	**0.095825**
	St. Dev	**0**	**0**	**0**	**0**	**0**	**0**
g09 680.63	best	**680.630**	**680.630**	**680.630**	**680.630**	**680.630**	**680.630**
	mean	**680.630**	**680.630**	**680.630**	**680.630**	**680.630**	**680.630**
	worst	**680.630**	**680.630**	**680.630**	**680.630**	**680.630**	**680.630**
	St. Dev	**0**	**0**	**0**	**0**	**0**	2.4E-5
g10 7049.248	best	**7049.249**	**7049.249**	7049.262	7050.044	7071.104	7146.630
	mean	**7049.278**	7049.653	7051.473	7098.588	7209.375	7431.165
	worst	**7049.390**	7052.776	7061.144	7408.632	7639.454	8330.030
	St. Dev	**3.1E-2**	9.5E-1	3.4E+1	7.5E+1	1.4E+2	2.8E+2
g11 0.75	best	**0.75**	**0.75**	**0.75**	**0.75**	**0.75**	**0.75**
	mean	**0.75**	**0.75**	**0.75**	**0.75**	**0.75**	**0.75**
	worst	**0.75**	**0.75**	**0.75**	**0.75**	**0.75**	**0.75**
	St. Dev	**0**	**0**	**0**	**0**	**0**	**0**
g12 1.000	best	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	mean	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	worst	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	St. Dev	**0**	**0**	**0**	**0**	**0**	**0**
g13 0.053950	best	**0.053942**	**0.053942**	**0.053942**	**0.053942**	**0.053942**	§0.053942
	mean	0.123123	0.105921	**0.067300**	0.105306	0.079599	§0.082457
	worst	0.441444	0.439278	**0.438803**	0.439180	**0.438803**	§0.438883
	St. Dev	1.4E-1	1.3E-1	**6.9E-2**	1.3E-1	9.6E-2	1.0E-1

Table 2. Statistical results obtained by the original DDE with respect to the SDDE using different saving percentages in 30 independent runs. A result in boldface means a better (or best) solution obtained.The symbols in column 8 mean that in only a fraction of the 30 runs feasible solutions were found. "*"=$\frac{4}{30}$, "†"=$\frac{28}{30}$, "‡"=$\frac{29}{30}$ and "§"=$\frac{27}{30}$.

nism helped SDDE to obtain even better results than the original DDE; these problems are g02, g06, g07 and g13. On the other hand, the quality and robustness of the results provided when solving problems g01, g03 and g10 were affected by the use of the saving mechanism. Meanwhile, for some test functions there was no evident decrease in the performance by the use of the mechanism (g04, g05, g08, g09, g11 and g12). Finally, we observed the most competitive performance of the SDDE when $SM = 0.2$ and, based on the values reported in Table 3, we are obtaining equally competitive results by the SDDE but with only $\mathbf{72078}$ OFEs on average (instead of 90036 OFEs required by the original approach).

(2) Convergence effect: Regarding the way in which the saving mechanism affects the convergence of the approach, we decided to show the convergence graph (the best objective function value found so far at each generation) using the SDDE and also the original DDE. In each graph, two lines will be found: the first one refers to the 20%-SDDE (with the saving mechanism) and the second one is the original DDE (without saving mechanism). The parameters used in this experiment were set in order to allow both approaches to perform almost the same number of OFEs ($75,000$). The idea is to promote a fair comparison based on equal computational cost. The parameters are exactly the same used in the previous experiment for both approaches except for the following: The SDDE used $SM = 0.2$; and for the DDE: population size $NP = 30$ in order to perform 75,000 OFEs and, obviously, $SM = 0.0$. We selected two test functions for each effect of the saving mechanism (positive, negative or no effect) found in the previous experiment. Convergence graphs from the 500 generations are presented in Figure 2.

As it can be seen, the differences between SDDE and DDE for the three effects, are noted early in the evolutionary process (roughly before generation 60). However, for problem g02 the positive effect is clearly shown; the saving mechanism slows down the convergence of the approach, giving it a chance to explore other promising areas of the search space and allowing it to reach better results. In the remaining five plots, the effect is not clear at all. Therefore, we decided to re-plot the graphs but now showing the first 60 generations. They are presented in Figure 3.

These new graphs provide more evidence about the different behavior shown by the approach with the saving mechanism. For those functions where the behavior leads to a positive effect, there is a more accentuated oscillation of the best solution before generation 60 (functions g02 and g13 in Figure 3) when using SDDE (version with saving mechanism). On the other hand, for problems with negative effect of the saving mechanism, the oscillation is more stressed in the version with no saving mechanism (functions g01 and g10 in Figure 3). Finally, the oscillation is very similar between both versions in those problems where the

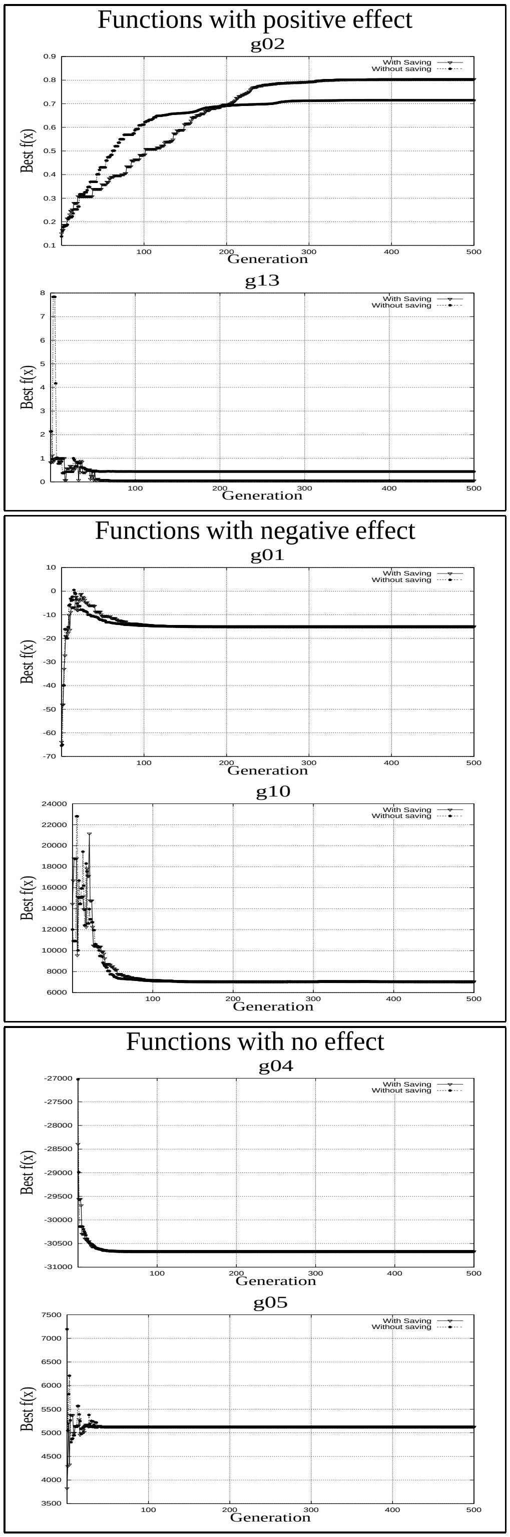

Figure 2. Convergence graphs (best f(x) at each generation) for test problems showing different effects with and without the saving mechanism.

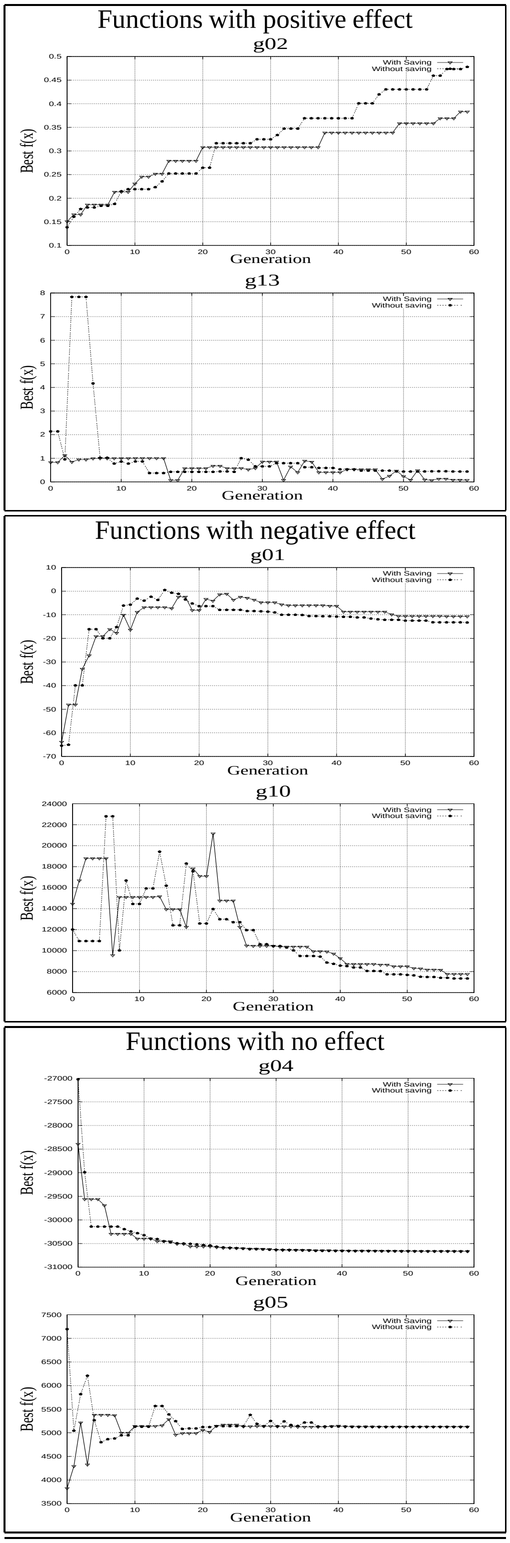

Figure 3. Details of convergence graphs (best f(x) at each generation) in the first 60 generations for test problems showing different effects with and without the saving mechanism.

saving mechanism provided no effect (functions g04 and g05 in Figure 3). The overall results suggest that the saving mechanism provides the approach with a slow-down convergence effect, whose oscillation in the first generations of the process allows the algorithm to explore other regions of the search space as to reach better final results. However, this oscillation is not enough for some problems in order to improve the obtained results. Finally, there is not a clear link between the features of a given problem (type of objective function, dimensionality, number and type of constraints) and the effect of the saving mechanism. This analysis is beyond the goals of the present work.

(3) Saving vs. small population: The effect of eliminating some individuals of the population can be considered analogous to using a smaller population from the beginning. Thus, we performed an experiment to compare the SDDE against DDE using a smaller population and, again, ensuring that both approaches performed the same number of OFEs. We decided to use the 40%-SDDE because it was the version with the lowest number of OFEs that produced competitive results ($\approx 55,000$. See Table 3) against a DDE performing the same number of evaluations. The parameters of both approaches are the same used in the first experiment. The only difference is the population size of the DDE (which had to be smaller) and was set to $NP = 22$ ($55,000$ OFEs) with, obviously, $SM = 0$. The statistical results of 30 independent runs are shown in Table 4.

The results show that the 40%-SDDE provided better quality and robust results in seven functions (g02, g03, g07, g09 and g13) and more robust results in functions g04, g05, g06, g09 and g11. Similar best, mean, worst and standard deviation results were obtained by both approaches in functions g08, g12. Finally, the DDE (with no saving mechanism and smaller population size) provided better results (quality and robustness) in functions g01 and g10. These results empirically imply a significant difference of the performance of the algorithm when using a mechanism that promotes ignoring some solutions during the evolutionary process, compared with just reducing the population size since the beginning, in order to save evaluations in both cases.

(4) Saving vs. fitness approximation: Finally, we compared our SDDE against a variation of a very simple fitness approximation approach called “nearest neighborhood regression” (NN) [7]. We used the basic idea of assigning the values of the objective function and the constraints of the closest individual in the decision space (decision variable space) to an individual to be approximated. We used the Euclidean distance measure in the decision space to compute closeness. As in SDDE, this mechanism is applied based on a user-defined ratio. We use the same DDE but now coupled with the NN. We decided to use this “very simple” approximation mechanism because we wanted to compare our approach, which, in fact is very simple, against the sim-

Problem & Best Known Sol.		SDDE vs Small-population-DDE	
	Stats	**DDE** 40%	**DDE**
g01 −15.000	best	−14.999	**−15.000**
	mean	−14.999	**−15.000**
	worst	−14.998	**−14.999**
	St. Dev	4.1E-4	**0**
g02 0.803619	best	**0.803521**	0.790104
	mean	**0.760721**	0.658107
	worst	**0.660187**	0.460254
	St. Dev	**4.3E-2**	7.8E-2
g03 1.000	best	**1.000**	0.999
	mean	**0.999**	0.995
	worst	**0.997**	0.961
	St. Dev	**8.4E-4**	8.8E-3
g04 −30665.539	best	**−30665.539**	**−30665.539**
	mean	**−30665.539**	**−30665.539**
	worst	**−30665.539**	−30665.537
	St. Dev	**0**	3.5E-4
g05 5216.498	best	**5126.497**	**5126.497**
	mean	**5126.497**	5126.857
	worst	**5126.497**	5136.245
	St. Dev	**0**	1.7E+0
g06 −6961.814	best	**−6961.814**	**−6961.814**
	mean	**−6961.814**	−6836.516
	worst	**−6961.814**	−6016.000
	St. Dev	**0**	2.5E+2
g07 24.306	best	**24.307**	24.308
	mean	**24.310**	24.347
	worst	**24.324**	24.641
	St. Dev	**3.6E-3**	6.9E-2
g08 0.095825	best	**0.095825**	**0.095825**
	mean	**0.095825**	**0.095825**
	worst	**0.095825**	**0.095825**
	St. Dev	**0**	**0**
g09 680.63	best	**680.630**	**680.630**
	mean	**680.630**	680.636
	worst	**680.630**	680.665
	St. Dev	**0**	9.3E-3
g10 7049.248	best	7071.104	**7049.295**
	mean	7209.375	**7078.029**
	worst	7639.454	**7593.367**
	St. Dev	1.4E+2	**9.7E+1**
g11 0.75	best	**0.75**	**0.75**
	mean	**0.75**	**0.75**
	worst	**0.75**	**0.75**
	St. Dev	**0**	1.4E-5
g12 1.000	best	**1.000**	**1.000**
	mean	**1.000**	**1.000**
	worst	**1.000**	**1.000**
	St. Dev	**0**	**0**
g13 0.053950	best	**0.053942**	**0.053942**
	mean	**0.079599**	0.194396
	worst	**0.438803**	0.499903
	St. Dev	**9.6E-2**	1.8E-1

Table 4. Statistical results obtained by the 40%-SDDE against DDE with a reduced population size and no saving mechanism in 30 independent runs. A result in boldface means a better (or best) solution obtained.

plest version of a fitness-approximation-based approach. The parameters for both approaches are the same used in the first experiment (we used the same ratio of 20% for either the saving or the fitness approximation mechanism); therefore, both approaches performed the same number of OFEs on average ($\approx 72,000$). In Table 5, the results obtained in 30 independent runs are presented. It is very clear the the SDDE performed much better than the NN-DDE. SDDE found better results (based on quality and robustness) in functions g01, g02, g03, g05, g07, g10 and g13. SDDE was also clearly more robust in problems g04, g06, g09, g11 and g12. The only problem where NN-DDE was competitive was g08. In fact, in problems g01, g03, g06, 10 and g11, NN-DDE had difficulties to find feasible solutions in some runs. Furthermore, in functions g05 and g13 (both with equality constraints), no feasible solutions were found by the NN-DDE in any single run. Based on this discussion, we can suggest that the SDDE provides a clearly better performance than a fitness-approximation based approach whose complexity is similar.

5. Remarks

Based on the results obtained in our experiments, we highlight the following: (1) It is possible to save evaluations in an evolutionary algorithm based on analyzing its way of generating new solutions and not necessarily adopting a fitness approximation technique. (2) In this study, we saved about 20% of the OFEs to get a similar performance (or even better in some cases) to another competitive approach. Furthermore, we could save a 40% of the OFEs while getting just a small decrease of the overall performance of the approach. (3) For some problems, the convergence slowdown, derived from the saving mechanism, was enough as to maintain or even improve the quality of the results, despite the elimination of some individuals. However for a few problems, this small oscillation caused a performance decrease. (4) We empirically showed that the effect of eliminating some random individuals during the process does not provide the same effect of just reducing the population size since the beginning. (5) Finally, we also showed that our saving mechanism provided much better results than another (equally simple) fitness-approximation approach.

6. Conclusions and Future Work

We have presented a mechanism to save evaluations of the objective function and the constraints based on the way an EA generates new individuals instead of using a fitness approximation approach. The idea was to discard some new explored areas of the search space by assigning a zero fitness to some random offspring. The approach was tested using different variations, and provided competitive results

Problem & Best Known Sol.	Stats	SDDE against NN-DDE	
		20%-SDDE	20%-NN-DDE
g01 −15.000	best	**−15.000**	∗ − 14.819
	mean	**−14.999**	∗ − 13.290
	worst	**−14.999**	∗ − 9.408
	St. Dev	**1.0E-5**	1.4E+0
g02 0.803619	best	**0.803571**	0.764504
	mean	**0.763916**	0.442158
	worst	**0.699599**	0.288266
	St. Dev	**2.6E-2**	1.2E-1
g03 1.000	best	**1.000**	†0.863
	mean	**1.000**	†0.610
	worst	**0.999**	†0.221
	St. Dev	**1.3E-4**	1.9E-1
g04 −30665.539	best	**−30665.539**	**−30665.539**
	mean	**−30665.539**	−30620.403
	worst	**−30665.539**	−29782.133
	St. Dev	**0**	1.6E+2
g05 5216.498	best	**5126.497**	−
	mean	**5126.497**	−
	worst	**5126.497**	−
	St. Dev	**0**	−
g06 −6961.814	best	**−6961.814**	**‡−6961.814**
	mean	**−6952.279**	‡ − 6807.050
	worst	**−6780.019**	‡ − 3629.179
	St. Dev	**3.6E+2**	6.7E+2
g07 24.306	best	**24.306**	24.328
	mean	**24.306**	30.434
	worst	**24.307**	69.921
	St. Dev	**2.5E-4**	1.1E+1
g08 0.095825	best	**0.095825**	**0.095825**
	mean	**0.095825**	**0.095825**
	worst	**0.095825**	**0.095825**
	St. Dev	**0**	**0**
g09 680.63	best	**680.630**	**680.630**
	mean	**680.630**	685.973
	worst	**680.630**	769.016
	St. Dev	**0**	1.7E+1
g10 7049.248	best	**7049.262**	§7101.526
	mean	**7051.473**	§9026.246
	worst	**7061.144**	§25262.518
	St. Dev	**3.4E+1**	4.0E+3
g11 0.75	best	**0.75**	**⋆0.75**
	mean	**0.75**	**⋆0.75**
	worst	**0.75**	**⋆0.75**
	St. Dev	**0**	7.7E-4
g12 1.000	best	**1.000**	**1.000**
	mean	**1.000**	0.999
	worst	**1.000**	0.999
	St. Dev	**0**	8.0E-6
g13 0.053950	best	**0.053942**	−
	mean	**0.067300**	−
	worst	**0.438803**	−
	St. Dev	**6.9E-2**	−

Table 5. Statistical results obtained by the 20%-SDDE against the 20%-NN-DDE in 30 independent runs. A result in boldface means a better (or best) solution obtained. "-" means no feasible solutions were found. The symbols in column 4 mean that in only a fraction of the 30 runs feasible solutions were found. "∗"=$\frac{21}{30}$, "†"=$\frac{8}{30}$, "‡"=$\frac{25}{30}$, "§"=$\frac{23}{30}$ and "⋆"=$\frac{27}{30}$.

even when saving a 40% OFEs. Another effect of the saving mechanism was a slow-down of the convergence of the approach, which led, for some test problems, to improve the quality and robustness of the results. The approach was also compared against a version with no saving mechanism and a smaller population in order to empirically show that the effects are not similar. Finally, our proposal also outperformed a very simple fitness approximation approach. Our future paths of research consist on analyzing which features of a problem make it more suitable to be solved using the saving mechanism. We also want to explore the use of some criteria to select the individuals to be condemned to death (instead of just choosing them at random). Finally, we will perform comparisons against more complex fitness approximation approaches.

Acknowledgments: The first author acknowledges support from the Mexican Consejo Nacional de Ciencia y Tecnología (CONACyT) through a postdoctoral position at CINVESTAV-IPN's Electrical Engineering Department. The second author acknowledges support from CONACyT through project number 45683.

References

[1] J. Branke and C. Schmidt. Fast Convergence by Means of Fitness Estimation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9(1):13–20, January 2005.

[2] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Mass. USA, 1989.

[3] Y. Jin. A Comprehensive Survey of Fitness Approximation in Evolutionary Computation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9(1):3–12, January 2005.

[4] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello. Promising Infeasibility and Multiple Offspring Incorporated to Differential Evolution for Constrained Optimization. In H. Beyer et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2005)*, volume 1, pages 225–232, New York, 2005. ACM Press.

[5] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, Germany, 2nd edition, 2004.

[6] K. V. Price. An Introduction to Differential Evolution. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 79–108. Mc Graw-Hill, UK, 1999.

[7] T. P. Runarsson. Constrained Evolutionary Optimization by Approximate Ranking and Surrogate Models. In X. Y. et al., editor, *Proceedings of 8th Parallel Problem Solving From Nature (PPSN VIII)*, pages 401–410, Heidelberg, Germany, 2004. Springer-Verlag. LNCS 3242.

[8] H.-P. Schwefel, editor. *Evolution and Optimization Seeking*. John Wiley & Sons, New York, 1995.

[9] K.-S. Won and T. Ray. Performance of kriging and cokriging based Surrogate Models within the Unified Framework for Surrogate Assisted Optimization. In *Proceedings of the CEC'2004*, volume 2, pages 1577–1585, Piscataway, New Jersey, June 2004. IEEE Service Center.