A Repair Method for Differential Evolution with Combined Variants to Solve Dynamic Constrained Optimization Problems

María-Yaneli Ameca-Alducin, Efrén Mezura-Montes and Nicandro Cruz-Ramírez
Artificial Intelligence Research Center, University of Veracruz
Xalapa, Veracruz, México
yaneliameca@gmail.com, {emezura,ncruz}@uv.mx

ABSTRACT

Repair methods, which usually require feasible solutions as reference, have been employed by Evolutionary Algorithms to solve constrained optimization problems. In this work, a novel repair method, which does not require feasible solutions as reference and inspired by the differential mutation, is added to an algorithm which uses two variants of differential evolution to solve dynamic constrained optimization problems. The proposed repair method replaces a local search operator with the aim to improve the overall performance of the algorithm in different frequencies of change in the constrained space. The proposed approach is compared against other recently proposed algorithms in an also recently proposed benchmark. The results show that the proposed improved algorithm outperforms its original version and provides a very competitive overall performance with different change frequencies.

CCS Concepts

•**Computing methodologies** → **Genetic algorithms;**
•**Mathematics of computing** → *Continuous functions;*

Keywords

Differential Evolution; Constraint-handling; Dynamic optimization

1. INTRODUCTION

Constrained optimization problems have been successfully solved by Evolutionary Algorithms (EAs) [6, 12, 16]. Nevertheless, among the current trends which have attracted the interest of researchers and practitioners is the presence of dynamism in the fitness function and/or the constraints of the optimization problem [13, 18]. This type of problems is known as the Dynamic Constrained Optimization Problem (DCOP) [17, 18, 20, 22]. A DCOP can be seen as a search problem where its fitness landscape and feasible region change through time. In their original versions, EAs were not designed to deal with dynamic search spaces because they lacked mechanisms to detect search space changes [18, 17]. Without loss of generality, a DCOP can be defined as to:

Find $\vec{x}$, at each time t, which:

$$\min_{\vec{x} \in F_t \subseteq [L,U]} f(\vec{x}, t)$$

where $t \in N^+$ is the current time,

$$[L, U] = \{\vec{x} = (x_1, x_2, ..., x_D) | L_i \leq x_i \leq U_i, i = 1 \dots D\}$$

is the search space,
subject to:

$$F_t = \{\vec{x} | \vec{x} \in [L, U], g_i(\vec{x}, t) \leq 0, i = 1 \dots m, \\ h_j(\vec{x}, t) = 0, j = 1 \dots p\}$$

is called the feasible region at time t.

$\forall \vec{x} \in F_t$ if there exists a solution $\vec{x}^* \in F_t$ such that $f(\vec{x}^*, t) \leq f(\vec{x}, t)$, then $\vec{x}^*$ is called a feasible optimal solution and $f(\vec{x}^*, t)$ is called the feasible optima value at time t.

Four types of DCOPs are defined: i) a static objective function and static constraints (i.e. a static constrained optimization problem), ii) a dynamic objective function and static constraints, iii) a static objective function and dynamic constraints, and iv) a dynamic objective function and dynamic constraints.

In the specialized literature on DCOPs, the Genetic Algorithm (GA) is the most popular EA, where different mechanisms to deal with fitness landscape and feasible region changes have been considered, such as diversity maintenance [4, 5, 17] and solution repair [18, 20, 22]. On the other hand, recent meta-heuristic approaches like the Dynamic Constrained T-Cell (DCTC), which is inspired by the T-Cell model of the immune system[2], has been used to tackle DCOPs. Another bio-inspired algorithm recently adapted to solve DCOPs is the Gravitational Search Algorithm (GSA), where a repair mechanism [22, 24] was also considered. Furthermore, in [21], an algorithm that combined Differential Evolution (DE) and a repair method was proposed in order to produce better results. In another DE-based algorithm, the combination of two DE variants, called Dynamic Differential Evolution with Combined Variants (DDECV), was proposed to solve DCOPs [1]. Among its added mechanisms there was a local search operator to promote convergence to

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

the dynamic feasible region of the also dynamic search space. However, such operator showed some limitations to generate feasible solutions after a change [1]. On the other hand, the above mentioned repair methods require feasible solutions as reference to convert infeasible solutions into feasible ones. This is the main motivation of this paper, where a novel repair method, which does not require feasible solutions to operate, and inspired by the differential mutation operator, is proposed to promote a better overall performance of the approach. A set of experiments focused on the frequency of the change in the search space are carried out, because the repair method has an important role for the algorithm to recover after a change. The results obtained are compared against several state-of-the-art approches to solve DCOPs.

The rest of the paper is divided as follows: Section 2 details DDECV, while Section 3 introduces the proposed repair method. Section 4 presents the experiments and results obtained by the algorithm in a benchmark recently proposed [18]. Finally, Section 5 includes the conclusions and directions regarding future research.

2. DDECV ALGORITHM

DDECV is based on the combination of two well-known DE variants, DE/rand/1/bin (base variant) and DE/best/-1/bin (alternative variant) [1]. DE is a stochastic search algorithm based on a set of solutions called vectors [23]. Such set is called population and each one of its vectors can be represented as: $\vec{x}_{i,G},\ i = 1, \ldots, NP$, where $\vec{x}_{i,G}$ is vector i at generation G, and NP represents the population size. For both variants, each target vector $\vec{x}_{i,G}$ generates one trial vector $\vec{u}_{i,G}$ by using a mutant vector $\vec{v}_{i,G}$. For DE/rand/1/bin, the mutant vector is computed as in Equation 1, where $\vec{x}_{r0,G}$, $\vec{x}_{r1,G}$, and $\vec{x}_{r2,G}$ are vectors chosen at random from the current population ($r0 \neq r1 \neq r2 \neq i$. $\vec{x}_{r0,G}$ is known as the base vector and $\vec{x}_{r1,G}$, and $\vec{x}_{r2,G}$ are the difference vectors. $F > 0$ is a scale factor defined by the user. For DE/best/1/bin the mutant vector is generated as in Equation 2, where the only difference is the fact that the base vector is the best solution in the current population (represented as $\vec{x}_{best,G}$).

$$\vec{v}_{i,G} = \vec{x}_{r0,G} + F(\vec{x}_{r1,G} - \vec{x}_{r2,G}) \tag{1}$$

$$\vec{v}_{i,G} = \vec{x}_{best,G} + F(\vec{x}_{r1,G} - \vec{x}_{r2,G}) \tag{2}$$

After the mutant vector $\vec{v}_{i,G}$ is generated, for the two DE variants discussed in this paper, it is combined with the target vector $\vec{x}_{i,G}$ by applying a crossover operator as shown in Equation 3.

$$u_{i,j,G} = \begin{cases} v_{i,j,G} & \text{if}(rand_j \leq CR) \text{ or } (j = J_{rand}) \\ x_{i,j,G} & \text{otherwise} \end{cases} \tag{3}$$

where $CR \in [0, 1]$ defines the similarity between the trial vector and the mutant vector, $rand_j$ generates a random number between 0 and 1, and $j \in \{1, \ldots, D\}$ is the j-th variable of the D- dimensional vector. $J_{rand} \in [1, D]$ is an integer random number which ensures that at least one element of the mutant vector is copied to the trial vector so as to prevent copies of the target vector.

To end the process, the best vector, based on fitness, between the target and trial vector is chosen for the next generation as follows:

$$\vec{x}_{i,G+1} = \begin{cases} \vec{u}_{i,G} & \text{if}(f(\vec{u}_{i,G}) \leq f(\vec{x}_{i,G})), \\ \vec{x}_{i,G} & \text{otherwise} \end{cases} \tag{4}$$

DDECV, wich will be explained in detail so as to get a self-contained paper, has three main elements: (1) change detection, (2) exploration promotion, and (3) convergence promotion. The last is the source of motivation of this research and will be detailed later. Firstly, the change detection and the exploration promotion mechanisms are described below.

The change detection mechanism consists of solution reevaluation [25, 9]. At each generation, the first trial vector and the trial vector at the middle of the current population are evaluated and their objective function values and constraints values are compared against their previous values. If any value is different, an indicator is activated. Lastly, the best vector in the current population is stored in a memory file, called the memory population. This memory keeps promising solutions that can be used after a change. The goal of using the first vector and the one located at the middle of the population is to decrease the chances of missing a change during a given generation. The pseudocode of the change detection mechanism is detailed in Algorithm 1.

Algorithm 1 Change_detection_mechanism

Require: $\vec{x}_{i,t-1}$
1: Evaluate $\vec{x}_{i,t}$ at time t
2: **if** any value is not the same as those of its previous evaluation **then**
3: Copy the best vector in the population $\vec{x}_{best,t}$ to the memory population
4: Reevaluate all vectors in the current population and also in the memory population
5: $eval = eval + current\_population\_size + memory\_population\_size$
6: **end if**

The exploration promotion mechanism is activated after a successful change detection (see Algorithm 1), and here is where the two DE variants are switched because DDECV starts by using DE/rand/1/bin. However, after the change detection, DE/best/1/bin is used instead for a number of generations defined by the user (Gen_{best}) and the F value is increased (expanded F) during the same period of time so as to promote exploration [14]. Furthermore, considering that DE/best/1/bin is used, the best vector (which will be the base vector for the differential mutation) can be chosen from either the current population or the memory population. The exploration promotion mechanism is outlined in Algorithm 2.

To add more diversity to the population, a number of IB randomly generated solutions, called immigrants [26], are added to the population at the end of each generation. However, the number of immigrants is increased (IA) during the DE/best/1/bin operation so as to increase the diversiy after a change. The worst individuals are replaced by the immigrants to keep the population size fixed.

The set of three feasibility rules proposed by Deb [7] are used in DDECV as selection criteria in Equation 4 and also

Algorithm 2 Exploration_promotion_mechanism

if counter for using DE/best/1/bin $< Gen_{best}$ **then**
  Generate $\vec{u}_{i,t}$ with Equation 2 with expanded F value and Equation 3
else
  Generate $\vec{u}_{i,t}$ with Equations 1 and 3
end if
return $\vec{u}_{i,t}$

every time the best vector is selected. The rules are the following:

1. Between two feasible vectors, the one with the best objective function value is selected.
2. If one vector is feasible and the other one is infeasible, the feasible vector is selected.
3. If both vectors are infeasible, the one with the lowest sum of constraint violation is selected.

Finally, the convergence promotion is carried out by a local search operator [10], which is applied to a randomly chosen vector $\vec{x}_{rand,t}$ from the current population. A randomly chosen variable from $\vec{x}_{rand,t} = (x_{rand,1,t}, \ldots, x_{rand,D,t})$ is added with a random value $\delta \in [0, 1]$. Such value is also substracted to the variable, then two neighbors are generated. The interval $[0, 1]$ is suitable for the benchmark used in this work, but it can be changed for other problems. The best vector, based on the three feasibility criteria above mentioned, among the original vector and the two neighbors is chosen as the new starting point. The process is repeated ILS (Iterations for Local Search) times, (user-defined parameter). The vector obtained by this mechanism replaces the worst vector in the current population (see Algorithm 3).

Algorithm 3 Convergence_promotion_mechanism

Require: $\vec{x}_{rand,t}$
for $c \leftarrow 1$ to ILS **do**
  $\delta = random(0, 1)$
  $\vec{x}_{N_i+,t} = \vec{x}_{rand,t}$
  $\vec{x}_{N_i-,t} = \vec{x}_{rand,t}$
  Select randomly one variable $x_{rand,j,t}$ of $\vec{x}_{rand,t}$
  $x_{N_i+,j,t} += \delta$
  $x_{N_i-,j,t} -= \delta$
  $\vec{x}_{rand,t}$=best($\vec{x}_{rand,t}$, $\vec{x}_{N_i+,t}$, $\vec{x}_{N_i-,t}$)
end for
Replace the worst vector in the population with $\vec{x}_{rand,t}$

The pseudocode of the complete DDECV for solving DCOPs [1] is detailed in Algorithm 4.

3. DDECV+ REPAIR

Recalling from Section 1, the usage of repair methods has been reported as constraint-handlers for DCOPs. The main idea of a repair method is the use of a set with only feasible solutions to serve as reference for infeasible solutions of the current population and help them to become feasible [15, 18, 20, 21, 22].

In this work a novel but simple repair method is proposed and added to DDECV (DDECV + Repair). The main difference with respect to other repair methods lies in the fact

Algorithm 4 DDECV algorithm

G=0
Create a randomly-generated initial population $\vec{x}_{i,G}\, \forall i, i = 1, \ldots, NP$
Evaluate each $\vec{x}_{i,G}\, \forall i, i = 1, \ldots, NP$
$eval = eval + NP$
while $eval \leq Max\_eval$ **do**
  for $i \leftarrow 1$ to NP **do**
    if $i = 1$ or $i = NP/2$ **then**
      Change_detection_Mechanism ($\vec{x}_{i,G}$) {Algorithm 1}
      $eval = eval + 1$
    end if
    $\vec{u}_{i,G}$ = Exploration_promotion_mechanism {Algorithm 2}
    $eval = eval + 1$
    if $f(\vec{u}_{i,G})$ is better than $f(\vec{x}_{i,G})$ based on the feasibility rules **then**
      $\vec{x}_{i,G+1} = \vec{u}_{i,G}$
    else
      $\vec{x}_{i,G+1} = \vec{x}_{i,G}$
    end if
  end for
  Add IA or IB immigrants to the current population and evaluate them
  $eval = eval + IA(or + IB)$
  Convergence_promotion_mechanism {Algorithm 3}
  $eval = eval + 2 * ILS$
  $G = G + 1$
end while

that DDECV + Repair does not use feasible vectors as reference, it is just based on the differential mutation operator. For each infeasible vector, three new and temporal vectors are generated at random with the only aim to apply the differential mutation operator (see Equation 1) as if a mutant vector is created in DE. At each generation, before the application of the selection operator based on the feasibility rules (see Equation 4) between the target and trial vectors, if the trial vector is infeasible, the repair method is applied until it is repaired or Repair_Limit attempts are computed. This can be seen in Algorithm 5. Considering the fact that only the constraints are evaluated to check the feasibility of a vector, the evaluations computed by the repair method are not considered in the total evaluations made by the proposed algorithm.

DDECV + Repair does not consider the convergence promotion of its former version and the repair method is used instead. The details of DDECV + Repair are shown in Algorithm 6, where the repair method is remarked in boldface.

Algorithm 5 Repair_Method

Require: $\vec{u}_{i,G}$ {trial vector}
$counter = 0$
while $\vec{u}_{i,G}$ is infeasible and $counter \leq$ Repair_Limit **do**
  Generate three random vectors ($\vec{u}_{r0,G}$, $\vec{u}_{r1,G}$ and $\vec{u}_{r2,G}$)
  $\vec{u}_{i,G} = \vec{u}_{r0,G} + F(\vec{u}_{r1,G} - \vec{u}_{r2,G})$
  $counter = counter + 1$
end while
Return $\vec{u}_{i,G}$

Table 1: Main features of the test problems [18].

Problem	Obj. Function	Constraints	DFR	SwO	bNAO	OICB	OISB	Path
g24_u	Dynamic	No Constraints	1	No	No	No	Yes	N/A
g24_1	Dynamic	Static	2	Yes	No	Yes	No	N/A
g24_f	Static	Static	2	No	No	Yes	No	N/A
g24_uf	Static	No Constraints	1	No	No	No	Yes	N/A
g24_2*	Dynamic	Static	2	Yes	No	Yes and No	Yes and No	N/A
g24_2u	Dynamic	No Constraints	1	No	No	No	Yes	N/A
g24_3	Static	Dynamic	2-3	No	Yes	Yes	No	N/A
g24_3b	Dynamic	Dynamic	2-3	Yes	No	Yes	No	N/A
g24_3f	Static	Static	1	No	No	Yes	No	N/A
g24_4	Dynamic	Dynamic	2-3	Yes	No	Yes	No	N/A
g24_5*	Dynamic	Dynamic	2-3	Yes	No	Yes and No	Yes and No	N/A
g24_6a	Dynamic	Static	2	Yes	No	No	Yes	Hard
g24_6b	Dynamic	Static	1	No	No	No	Yes	N/A
g24_6c	Dynamic	Static	2	Yes	No	No	Yes	Easy
g24_6d	Dynamic	Static	2	Yes	No	No	Yes	Hard
g24_7	Static	Dynamic	2	No	No	Yes	No	N/A
g24_8a	Dynamic	No Constraints	1	No	No	No	No	N/A
g24_8b	Dynamic	Static	2	Yes	No	Yes	No	N/A

DFR	Number of disconnected feasible regions
SwO	Switched global optimum between disconnected regions
bNAO	Better newly appear optimum without changing existing ones
OICB	Global optimum is in the constraint boundary
OISB	Global optimum is in the search boundary
Path	Indicate if it is easy or difficult to use mutation to travel between feasible regions
Dynamic	The function is dynamic
Static	There is no change
*	In some change periods, the landscape either is a plateau or contains infinite number of optima and all optima (including the existing optimum) lie in a line parallel to one of the axes

Algorithm 6 DDECV+Repair algorithm

```
1: G=0
2: Create a randomly-generated initial population x⃗_{i,G} ∀i, i = 1,...,NP
3: Evaluate each x⃗_{i,G} ∀i, i = 1,...,NP
4: eval = eval + NP
5: while eval ≤ Max_eval do
6:    for i ← 1 to NP do
7:       if i = 1 or i = NP/2 then
8:          Change_detection_Mechanism (x⃗_{i,G}) {Algorithm 1}
9:          eval = eval + 1
10:      end if
11:      u⃗_{i,G} = Exploration_promotion_mechanism {Algorithm 2}
12:      if u⃗_{i,G} is infeasible then
13:         Repair_Method(u⃗_{i,G}) {Algorithm 5}
14:      end if
15:      eval = eval + 1
16:      if f(u⃗_{i,G}) is better than f(x⃗_{i,G}) based on the feasibility rules then
17:         x⃗_{i,G+1} = u⃗_{i,G}
18:      else
19:         x⃗_{i,G+1} = x⃗_{i,G}
20:      end if
21:   end for
22:   Add IA or IB immigrants to the current population and evaluate them
23:   eval = eval + IA(or + IB)
24:   G = G + 1
25: end while
```

4. EXPERIMENTS AND RESULTS

4.1 Experimental setup

DDECV + Repair was tested on eighteen benchmark problems. The main features of those problems are summarized in Table 1 and the details can be found in [18, 19]. The results obtained by DDECV + Repair were compared with those from state-of-the-art EAs in dynamic constrained optimization: (1) a GA with elitism, nonlinear ranking parent selection, arithmetic crossover and uniform mutation (GAElit) [18], (2) a GA similar to GAElit but with random immigrants (RIGAElit) [18], (3) another version of GAElit but with hypermutation (HyperMElit) [18], (4) a GA with a traditional repair mechanism (GA + Repair) [18], (5) a traditional differential evolution with an also traditional repair mechanism (DE + Repair)[21], (6) the gravitational search algorithm with a traditional repair mechanism (GSA + Repair) [22], and (7) the original DDECV [1] to analyze the particular effect of the proposed repair method in this algorithm.

Table 2 shows the settings for the benchmark problems. Different change frequencies were tested (500, 1000 and 2000 evaluations), with a medium objective and constraint function change severity as suggested in [18]. DE + Repair and GSA + Repair were compared with 1000 evaluations as change frequency because no results were found for the remaining frequencies. The parameter values used by DDECV and DDECV + Repair are listed in Table 3 and were taken from [1] (with the exception of Repair_Limit).

The offline error [3] was employed to measure the performance of DDECV + Repair. The offline error is defined as the average of the sum of errors in each cycle divided by the sum of the number of cycles. The offline error is always greater than or equal to zero. This latter value indicates a perfect performance [17]. This measure is defined as:

Table 2: Parameter values for the test problems taken from [18]

Number of runs	50
Number of changes	12
Frequency change	500, 1000, 2000 Evals.
Obj. function severity k	0.5
Constraint severity S	20

Table 3: DDECV and DDECV + Repair parameter values taken from [1]

Pop size	25
Crossover	CR = 0.8399
F before change	F = 0.9644
F after change	FA = 1.0820
Immigrants before change	IB = 5
Immigrates after change	IA = 3
Gen_{best}	16
Iterations for local search	ILS = 8 (only DDECV)
Repair_Limit	100 (only DDECV + Repair)

$$offline\_error = \frac{1}{n}\sum_{j=1}^{n} e(j)$$

where n is the number of cycles so far and $e(j)$ denotes the best error since the last change gained by the algorithm at cycle j (see the next Equation).

$$e(j) = |f(\vec{x}^*, t) - f(\vec{x}, t)|$$

where $f(\vec{x}^*, t)$ denotes the feasible global optima at time t and $f(\vec{x}, t)$ is the best solution found at generation G.

4.2 Results

The average and standard deviation values of the offline error obtained per each compared algorithm in the set of eighteen test problems are presented in Tables 4, 5, and 6 for change frequencies of 500, 1000 and 2000 evaluations, respectively.

The statistical validation for all test functions was computed with the non-parametric 95%-confidence Friedman test and a post-hoc test (Bergmann-Hommels) [8]. Non-parametric tests were adopted because the samples of runs did not fit to a Gaussian distribution based on the Kolmogorov-Smirnov test.

It was observed from the results in Table 4 (500 evaluations for a change), and from the Friedman and Bergmann - Hommels tests, that DDECV + Repair outperformed all the compared algorithms but DDECV (i.e. no significant differences in the complete set of test problems were observed). Because of this, the 95%-confidence Wilcoxon test was used to determine statistical differences per each test problem between DDECV + Repair and DDECV. Based on such results, DDECV + Repair outperformed DDECV in fourteen test problems (g24_f, g24_2, g24_3, g24_3b, g24_3f, g24_4, g24_5, g24_6a, g24_6b, g24_6c, g24_6d, g24_7, g24_8a, g24_8b), while the original DDECV outperformed DDECV + Repair in four test problems (g24_u, g24_uf, g24_2u and g24_1). The main feature of the first three test problems (g24_u, g24_uf, g24_2u) is that they are unconstrained, and the fourth one (g24_1) has a disconnected feasible region. The better performance of the original DDECV in the first three test problems can be understood due to the fact that the local search used as convergence promotion (see Algorithm 3) is precisely designed to improve good solutions faster and the whole search space is feasible on those problems. However, in presence of constraints, DDECV + Repair clearly performed better (except in test problem g24_1).

Table 4: Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 500 evaluations. Significant best results, based on the 95%-confidence Wilcoxon test between DDECV and DDECV+Repair are remarked in boldface.

Algorithms	Functions		
	G24_u	**G24_1**	**G24_f**
GAElit	0.184(±0.035)	0.641(±0.057)	0.175(±0.083)
RIGAElit	0.235(±0.025)	0.496(±0.046)	0.266(±0.051)
HyperMElit	0.163(±0.026)	0.52(±0.065)	0.209(±0.053)
GA+Repair	0.500(±0.059)	0.264(±0.024)	0.077(±0.011)
DDECV	**0.082(±0.011)**	**0.109(±0.067)**	0.075(±0.019)
DDECV+Repair	0.086(±0.009)	0.117(±0.015)	**0.048(±0.009)**
	G24_uf	**G24_2**	**G24_2u**
GAElit	0.091(±0.022)	0.372(±0.05)	0.132(±0.017)
RIGAElit	0.125(±0.02)	0.325(±0.037)	0.146(±0.024)
HyperMElit	0.091(±0.012)	0.364(±0.043)	0.115(±0.016)
GA+Repair	0.358(±0.018)	0.298(±0.036)	0.354(±0.029)
DDECV	**0.010(±0.005)**	0.162(±0.032)	**0.065(±0.005)**
DDECV+Repair	0.016(±0.005)	**0.137(±0.008)**	0.088(±0.008)
	G24_3	**G24_3b**	**G24_3f**
GAElit	0.375(±0.049)	0.631(±0.084)	0.252(±0.058)
RIGAElit	0.436(±0.048)	0.545(±0.051)	0.264(±0.048)
HyperMElit	0.404(±0.05)	0.557(±0.088)	0.244(±0.051)
GA+Repair	0.063(±0.008)	0.184(±0.019)	0.035(±0.008)
DDECV	0.087(±0.024)	0.225(±0.07)	0.071(±0.025)
DDECV+Repair	**0.062(±0.008)**	**0.147(±0.012)**	**0.028(±0.007)**
	G24_4	**G24_5**	**G24_6a**
GAElit	0.646(±0.075)	0.367(±0.029)	1.038(±0.157)
RIGAElit	0.542(±0.047)	0.287(±0.035)	0.534(±0.05)
HyperMElit	0.573(±0.075)	0.324(±0.039)	0.694(±0.071)
GA+Repair	0.143(±0.015)	0.196(±0.024)	0.616(±0.074)
DDECV	0.233(±0.081)	0.195(±0.033)	0.267(±0.114)
DDECV+Repair	**0.143(±0.012)**	**0.140(±0.014)**	**0.083(±0.016)**
	G24_6b	**G24_6c**	**G24_6d**
GAElit	0.631(±0.057)	0.666(±0.052)	0.664(±0.075)
RIGAElit	0.436(±0.039)	0.443(±0.029)	0.512(±0.057)
HyperMElit	0.535(±0.039)	0.543(±0.051)	0.584(±0.041)
GA+Repair	0.567(±0.048)	0.518(±0.038)	0.475(±0.038)
DDECV	0.145(±0.029)	0.173(±0.048)	0.414(±0.083)
DDECV+Repair	**0.098(±0.01)**	**0.090(±0.011)**	**0.196(±0.015)**
	G24_7	**G24_8a**	**G24_8b**
GAElit	0.441(±0.053)	0.356(±0.028)	0.807(±0.056)
RIGAElit	0.565(±0.068)	0.405(±0.028)	0.758(±0.064)
HyperMElit	0.430(±0.062)	0.355(±0.028)	0.710(±0.071)
GA+Repair	0.134(±0.017)	0.341(±0.032)	0.380(±0.068)
DDECV	0.156(±0.038)	0.292(±0.035)	0.332(±0.108)
DDECV+Repair	**0.134(±0.023)**	**0.257(±0.031)**	**0.189(±0.033)**

Table 5: Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 1000 evaluations. Significant best results, based on the 95%-confidence Wilcoxon test between DDECV and DDECV+Repair are remarked in boldface.

Algorithms	Functions		
	G24_u	**G24_1**	**G24_f**
GAElit	0.106(±0.035)	0.459(±0.057)	0.154(±0.083)
RIGAElit	0.149(±0.025)	0.346(±0.046)	0.178(±0.051)
HyperMElit	0.111(±0.026)	0.384(±0.065)	0.149(±0.053)
GA+Repair	0.468(±0.059)	0.226(±0.024)	0.041(±0.011)
DE+Repair	0.099(±0.01)	0.151(±0.024)	0.039(±0.022)
GSA+Repair	0.049(±0.004)	0.132(±0.015)	0.029(±0.012)
DDECV	0.050(±0.006)	0.109(±0.033)	0.029(±0.010)
DDECV+Repair	**0.039(±0.007)**	**0.061(±0.01)**	**0.021(±0.006)**
	G24_uf	**G24_2**	**G24_2u**
GAElit	0.063(±0.022)	0.288(±0.050)	0.073(±0.017)
RIGAElit	0.069(±0.02)	0.246(±0.037)	0.091(±0.024)
HyperMElit	0.053(±0.012)	0.253(±0.043)	0.068(±0.016)
GA+Repair	0.218(±0.018)	0.281(±0.036)	0.294(±0.029)
DE+Repair	0.057(±0.019)	0.191(±0.014)	0.141(±0.012)
GSA+Repair	0.047(±0.009)	0.182(±0.019)	0.196(±0.012)
DDECV	**0.004(±0.002)**	0.126(±0.030)	0.054(±0.004)
DDECV+Repair	0.009(±0.002)	**0.062(±0.006)**	**0.036(±0.001)**
	G24_3	**G24_3b**	**G24_3f**
GAElit	0.289(±0.049)	0.457(±0.084)	0.158(±0.058)
RIGAElit	0.308(±0.048)	0.386(±0.051)	0.167(±0.048)
HyperMElit	0.243(±0.05)	0.394(±0.088)	0.128(±0.051)
GA+Repair	0.156(±0.008)	0.171(±0.019)	0.025(±0.008)
DE+Repair	0.091(±0.012)	0.121(±0.019)	0.013(±0.009)
GSA+Repair	0.028(±0.004)	0.076(±0.009)	0.009(±0.007)
DDECV	0.057(±0.018)	0.134(±0.033)	0.032(±0.011)
DDECV+Repair	**0.046(±0.006)**	**0.084(±0.006)**	**0.010(±0.002)**
	G24_4	**G24_5**	**G24_6a**
GAElit	0.453(±0.075)	0.266(±0.029)	0.674(±0.157)
RIGAElit	0.421(±0.047)	0.240(±0.035)	0.333(±0.050)
HyperMElit	0.426(±0.075)	0.248(±0.039)	0.491(±0.071)
GA+Repair	0.211(±0.015)	0.236(±0.024)	0.431(±0.074)
DE+Repair	0.121(±0.021)	0.121(±0.011)	0.047(±0.009)
GSA+Repair	0.073(±0.012)	0.153(±0.013)	0.033(±0.003)
DDECV	0.131(±0.032)	0.126(±0.019)	0.215(±0.067)
DDECV+Repair	**0.088(±0.011)**	**0.078(±0.008)**	**0.036(±0.005)**
	G24_6b	**G24_6c**	**G24_6d**
GAElit	0.408(±0.057)	0.441(±0.052)	0.510(±0.075)
RIGAElit	0.309(±0.039)	0.325(±0.029)	0.342(±0.057)
HyperMElit	0.390(±0.039)	0.394(±0.051)	0.456(±0.041)
GA+Repair	0.427(±0.048)	0.390(±0.038)	0.354(±0.038)
DE+Repair	0.101(±0.012)	0.790(±0.010)	0.910(±0.011)
GSA+Repair	0.047(±0.003)	0.045(±0.004)	0.037(±0.007)
DDECV	0.108(±0.016)	0.128(±0.025)	0.288(±0.055)
DDECV+Repair	**0.041(±0.010)**	**0.041(±0.01)**	**0.079(±0.006)**
	G24_7	**G24_8a**	**G24_8b**
GAElit	0.316(±0.053)	0.266(±0.028)	0.662(±0.056)
RIGAElit	0.416(±0.068)	0.304(±0.028)	0.598(±0.064)
HyperMElit	0.315(±0.062)	0.279(±0.028)	0.608(±0.071)
GA+Repair	0.181(±0.017)	0.496(±0.032)	0.391(±0.068)
DE+Repair	0.033(±0.009)	0.217(±0.033)	0.227(±0.039)
GSA+Repair	0.018(±0.002)	0.202(±0.041)	0.192(±0.034)
DDECV	**0.106(±0.022)**	0.141(±0.025)	0.151(±0.058)
DDECV+Repair	0.107(±0.011)	0.138(±0.015)	**0.074(±0.025)**

The results in Table 5 (1000 evaluations for a change) show that DDECV + Repair outperformed most of the compared algorithms with the exception of DDECV and GSA + Repair based on the Friedman and Bergmann - Hommels tests. Due to this, the 95%-confidence Wilcoxon test was used again to validate statistical differences per each test problem between DDECV + Repair and DDECV (the same comparison was not made against GSA + Repair because the results per each independent run were not available). According to those results, DDECV + Repair provided better results in fifteen test problems (g24_u, g24_1, g24_f, g24_2, g24_2u, g24_3, g24_3b, g24_3f, g24_4, g24_5, g24_6a, g24_6b, g24_6c, g24_6d, g24_8b) and the original DDECV was better in two test problems (g24_uf, g24_7). The first problem (g24_uf) is unconstrained, and the second one (g24_7) has a disconnected feasible region. In test problem g24_8a no significant differences were observed.

Table 6: Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 2000 evaluations. Significant best results, based on the 95%-confidence Wilcoxon test between DDECV and DDECV+Repair are remarked in boldface.

Algorithms	Functions		
	G24-u	**G24-1**	**G24-f**
GAElit	0.065(±0.011)	0.332(±0.074)	0.092(±0.052)
RIGAElit	0.110(±0.014)	0.235(±0.038)	0.106(±0.037)
HyperMElit	0.072(±0.015)	0.289(±0.053)	0.084(±0.042)
GA+Repair	0.262(±0.04)	0.055(±0.012)	0.023(±0.006)
DDECV	0.030(±0.008)	0.066(±0.018)	0.016(±0.016)
DDECV+Repair	**0.023(±0.003)**	**0.036(±0.010)**	**0.011(±0.004)**
	G24-uf	**G24-2**	**G24-2u**
GAElit	0.032(±0.010)	0.183(±0.024)	0.049(±0.008)
RIGAElit	0.047(±0.015)	0.168(±0.023)	0.057(±0.011)
HyperMElit	0.028(±0.008)	0.172(±0.037)	0.044(±0.012)
GA+Repair	0.164(±0.054)	0.147(±0.022)	0.171(±0.040)
DDECV	**0.002(±0.001)**	0.071(±0.016)	0.031(±0.002)
DDECV+Repair	0.005(±0.001)	**0.035(±0.007)**	**0.018(±0.001)**
	G24-3	**G24-3b**	**G24-3f**
GAElit	0.164(±0.033)	0.320(±0.058)	0.072(±0.032)
RIGAElit	0.208(±0.026)	0.262(±0.024)	0.100(±0.026)
HyperMElit	0.168(±0.029)	0.288(±0.048)	0.082(±0.036)
GA+Repair	0.019(±0.004)	0.044(±0.009)	0.010(±0.003)
DDECV	0.032(±0.008)	0.078(±0.015)	0.017(±0.006)
DDECV+Repair	0.036(±0.002)	**0.063(±0.009)**	**0.006(±0.001)**
	G24-4	**G24-5**	**G24-6a**
GAElit	0.333(±0.074)	0.196(±0.026)	0.408(±0.050)
RIGAElit	0.309(±0.037)	0.174(±0.022)	0.236(±0.026)
HyperMElit	0.287(±0.067)	0.182(±0.019)	0.287(±0.036)
GA+Repair	0.044(±0.009)	0.111(±0.023)	0.300(±0.054)
DDECV	0.073(±0.014)	0.081(±0.011)	0.103(±0.032)
DDECV+Repair	**0.057(±0.005)**	**0.041(±0.004)**	**0.020(±0.004)**
	G24-6b	**G24-6c**	**G24-6d**
GAElit	0.274(±0.028)	0.282(±0.033)	0.318(±0.059)
RIGAElit	0.210(±0.025)	0.213(±0.027)	0.242(±0.027)
HyperMElit	0.234(±0.019)	0.249(±0.034)	0.281(±0.030)
GA+Repair	0.306(±0.030)	0.287(±0.042)	0.263(±0.024)
DDECV	0.053(±0.008)	0.063(±0.013)	0.139(±0.027)
DDECV+Repair	**0.026(±0.005)**	**0.022(±0.004)**	**0.044(±0.003)**
	G24-7	**G24-8a**	**G24-8b**
GAElit	0.217(±0.047)	0.232(±0.023)	0.499(±0.048)
RIGAElit	0.303(±0.043)	0.269(±0.017)	0.496(±0.042)
HyperMElit	0.253(±0.036)	0.237(±0.013)	0.463(±0.052)
GA+Repair	0.050(±0.015)	0.247(±0.020)	0.136(±0.035)
DDECV	0.062(±0.014)	0.072(±0.032)	0.078(±0.032)
DDECV+Repair	**0.057(±0.005)**	0.075(±0.015)	**0.041(±0.012)**

As it can be seen in Table 6 (2000 evaluations for a change) and based on the Friedman and Bergmann - Hommels tests, DDECV + Repair outperformed all the compared algorithms, except DDECV. As in the previous two cases, the 95%-confidence Wilcoxon test was used to determine statistical differences per each test problem between DDECV + Repair and DDECV. It was observed in the results that DDECV + Repair outperformed DDECV in fifteen test problems (g24_u, g24_1, g24_f, g24_2, g24_2u, g24_3b, g24_3f, g24_4, g24_5, g24_6a, g24_6b, g24_6c, g24_6d, g24_7, g24_8b) and the original DDECV was better in just one test problem (g24_uf). The main feature of this problem is that it is

unconstrained. In test problems g24_3 and g24_8a no significant differences were observed.

Finally, to get insights of the effectiveness of the repair method, the averages of successful applications, where a successful application means that a feasible solution was obtained from an infeasible one, for each test problem, and for each frequency, are reported in Table 7. In all the constrained problems the simple repair method based on the differential mutation, and with no help of feasible solutions, was very successful, even in those problems where the offline error was not as good as expected (i.e. test problems with a disjoint feasible region). Such good performance may be due to the fact that the feasible region of the original static test problem (g24) [11], which is the base of all the benchmark, covers about 79% of the whole search space.

Table 7: Average successful application of the proposed repair method with change frequency of 500, 1000 and 2000 evaluations. Only constrained problems are included.

Functions	Frequencies		
	500 Evals	1000 Evals	2000 Evals
g24_1	1.0	1.0	1.0
g24_f	1.0	1.0	1.0
g24_2	1.0	1.0	1.0
g24_3	0.99	0.99	0.99
g24_3b	0.99	0.99	0.99
g24_3f	0.99	0.99	0.99
g24_4	0.99	0.99	0.99
g24_5	0.99	0.99	0.99
g24_6a	1.0	1.0	1.0
g24_6b	1.0	1.0	1.0
g24_6c	1.0	1.0	1.0
g24_6d	1.0	1.0	1.0
g24_7	0.99	0.99	0.99
g24_8b	1.0	1.0	1.0

5. CONCLUSIONS

A simple repair method based on the differential mutation operator was proposed and added to the DDECV algorithm to solve dynamic constrained optimization problems. The main difference of the repair method is that it does not require feasible solutions to operate. Instead, random solutions are generated to apply the differential mutation operator a number of attempts defined by the user. Such repair method replaced the convergence promotion mechanisms in the original DDECV. DDECV + Repair was tested in a recently proposed benchmark with eighteen test problems at different change frequency values (500, 1000 and 2000 evaluations). The offline error was the value adopted to measure the performance of the proposed improved algorithm. Six algorithms (including GAs, DE, GSA, some of them with traditional repair methods), besides the original DDECV version, were used for comparison purposes.

The overall results showed that DDECV + Repair improved the results of its previous version and also outperformed most of the compared al gorithms, regardless of the change frequency. The exception was with a frequency of 1000 evaluations, where DDECV + Repair was as good as GSA + Repair. However, GSA + Repair uses a traditional repair method which requires feasible solutions, while the proposed algorithm only uses the differential mutation with random solutions for the same purpose. Finally, some difficulties to get good offline error values were found in test problems with a disjoint feasible region.

For future work, the repair method will be re-visited so as to (1) analyze the negative effect of a disjoint feasible region, and (2) add it some knowledge to improve its performance. Finally, other test problems with smaller feasible regions will be sought so as to test the repair method in those situations.

6. ACKNOWLEDGMENTS

The authors acknowledge the valuable help and comments provided by Prof. Swagatam Das. The first author acknowledges support from the Mexican Council for Science and Technology (CONACyT) to pursue graduate studies at the University of Veracruz. The second author acknowledges support from CONACyT through project No. 220522.

7. REFERENCES

[1] M.-Y. Ameca-Alducin, E. Mezura-Montes, and N. Cruz-Ramirez. Differential evolution with combined variants for dynamic constrained optimization. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 975–982, July 2014.

[2] V. Aragón, S. Esquivel, and C. Coello. Artificial immune system for solving dynamic constrained optimization problems. In E. Alba, A. Nakib, and P. Siarry, editors, *Metaheuristics for Dynamic Optimization*, volume 433 of *Studies in Computational Intelligence*, pages 225–263. Springer Berlin Heidelberg, 2013.

[3] J. Branke and H. Schmeck. Designing evolutionary algorithms for dynamic optimization problems. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computing*, Natural Computing Series, pages 239–262. Springer Berlin Heidelberg, 2003.

[4] H. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical report, Naval Research Lab Washington DC, 1990.

[5] H. Cobb and J. Grefenstette. Genetic algorithms for tracking changing environments. In S. Forrest, editor, *ICGA*, pages 523–530. Morgan Kaufmann, 1993.

[6] C. A. Coello Coello. Theoretical and Numerical Constraint Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, January 2002.

[7] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(24):311–338, 2000.

[8] J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[9] M. du Plessis. *Adaptive Multi-Population Differential Evolution for Dynamic Environments*. PhD thesis, Faculty of Engineering, Built Environment and Information Technology, University of Pretoria, April 2012.

[10] S. Hernandez, G. Leguizamon, and E. Mezura-Montes. A hybrid version of differential evolution with two

differential mutation operators applied by stages. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 2895–2901, 2013.

[11] J. J. Liang, T. Runarsson, E. Mezura-Montes, M. Clerc, P. Suganthan, C. A. Coello Coello, and K. Deb. Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. Technical report, Nanyang Technological University, Singapore, December, 2005.

[12] E. Mezura-Montes, editor. *Constraint-Handling in Evolutionary Optimization*, volume 198 of *Studies in Computational Intelligence*. Springer-Verlag, 2009.

[13] E. Mezura-Montes and C. A. C. Coello. Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm and Evolutionary Computation*, 1(4):173–194, 2011.

[14] E. Mezura-Montes, M. E. Miranda-Varela, and R. del Carmen Gómez-Ramón. Differential evolution in constrained numerical optimization. an empirical study. *Information Sciences*, 180(22):4223–4262, 2010.

[15] Z. Michalewicz and G. Nazhiyath. Genocop iii: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 2, pages 647–651 vol.2, Nov 1995.

[16] Z. Michalewicz and M. Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[17] T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6(0):1 – 24, 2012.

[18] T. Nguyen and X. Yao. Continuous dynamic constrained optimization: The challenges. *IEEE Transactions on Evolutionary Computation*, 16(6):769–786, 2012.

[19] T. Nguyen and X. Yao. Evolutionary optimization on continuous dynamic constrained problems - an analysis. In S. Yang and X. Yao, editors, *Evolutionary Computation for Dynamic Optimization Problems*, volume 490 of *Studies in Computational Intelligence*, pages 193–217. Springer Berlin Heidelberg, 2013.

[20] T. T. Nguyen and X. Yao. Benchmarking and solving dynamic constrained problems. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 690–697, 2009.

[21] K. Pal, C. Saha, and S. Das. Differential evolution and offspring repair method based dynamic constrained optimization. In B. Panigrahi, P. Suganthan, S. Das, and S. Dash, editors, *Swarm, Evolutionary, and Memetic Computing*, volume 8297 of *Lecture Notes in Computer Science*, pages 298–309. Springer International Publishing, 2013.

[22] K. Pal, C. Saha, S. Das, and C. Coello-Coello. Dynamic constrained optimization with offspring repair based gravitational search algorithm. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 2414–2421, 2013.

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

[24] E. Rashedi, H. Nezamabadi, and S. Saryazdi. Gsa: A gravitational search algorithm. *Information Sciences*, 179(13):2232 – 2248, 2009.

[25] H. Richter. Detecting change in dynamic fitness landscapes. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 1613–1620, 2009.

[26] Y. Shengxiang. Memory-based immigrants for genetic algorithms in dynamic environments. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1115–1122, New York, NY, USA, 2005. ACM.