Soft Comput (2018) 22:541–570
https://doi.org/10.1007/s00500-016-2353-1

METHODOLOGIES AND APPLICATION

Dynamic differential evolution with combined variants and a repair method to solve dynamic constrained optimization problems: an empirical study

María-Yaneli Ameca-Alducin[1] · **Efrén Mezura-Montes**[1] · **Nicandro Cruz-Ramírez**[1]

Published online: 27 September 2016

Abstract An empirical study of the algorithm dynamic differential evolution with combined variants with a repair method (DDECV + Repair) in the solution of dynamic constrained optimization problems is presented. Unexplored aspects of the algorithm are of particular interest in this work: (1) the role of each one of its elements, (2) its sensitivity to different change frequencies and change severities in the objective function and the constraints, (3) its ability to detect a change and recover after it, besides its diversity handling (percentage of feasible and infeasible solutions) during the search, and (4) its performance with dynamism present in different parts of the problem. Seven performance measures, eighteen recently proposed test problems and eight algorithms found in the specialized literature are considered in four experiments. The statistically validated results indicate that DDECV + Repair is robust to change frequency and severity variations, and that it is particularly fast to recover after a change in the environment, but highly depends on its repair method and its memory population to provide competitive results. DDECV + Repair shows evidence on the convenience of keeping a proportion of infeasible solutions in the population when solving dynamic constrained optimization problems. Finally, DDECV + Repair is highly competitive particularly when dynamism is present in both, objective function and constraints.

Communicated by V. Loia.

✉ María-Yaneli Ameca-Alducin
yaneliameca@gmail.com

Efrén Mezura-Montes
emezura@uv.mx

Nicandro Cruz-Ramírez
ncruz@uv.mx

1 Artificial Intelligence Research Center, University of Veracruz, Sebastián Camacho 5 Centro, Xalapa, Veracruz, Mexico

Keywords Differential evolution · Constraint handling · Dynamic optimization · Dynamic constrained optimization problem

1 Introduction

In the specialized literature on constrained numerical optimization problems with meta-heuristics, evolutionary algorithms (EAs) stand out as a valid option to solve them (Coello Coello 2002; Mezura-Montes 2009; Michalewicz and Schoenauer 1996). However, in recent years, the presence of some kind of dynamism in the objective function and/or the constraints has raised the interest of researchers and practitioners (Mezura-Montes and Coello 2011; Nguyen et al. 2012; Nguyen and Yao 2012, 2013). This type of problem is known as the dynamic constrained optimization problem (DCOP) (Nguyen et al. 2012; Nguyen and Yao 2009, 2012, 2013; Singh et al. 2009). A DCOP could be considered as a single search problem in which the objective function and/or the constraints change through time. Given those conditions, traditional EAs must be adapted to identify changes in the fitness landscape and/or in the feasible region so as to be able to find new feasible optimal solutions (Nguyen et al. 2012; Nguyen and Yao 2012, 2013).

The specialized literature on EAs shows a significant amount of research in dynamic unconstrained optimization, e.g., multimodal functions (Rohlfshagen and Yao 2013; Filipiak and Lipinski 2014; Li et al. 2014, 2015; Mukherjee et al. 2016; Umenai et al. 2016; Zhang et al. 2016; Yu and Wu

2016; Pekdemir and Topcuoglu 2016) and multi-objective optimization (Liu et al. 2014; Azzouz et al. 2015; Martínez-Peñaloza and Mezura-Montes 2015; Jiang and Yang 2016). Nevertheless, in the presence of dynamic constraints in continuous search spaces, the research is still scarce as mentioned in a survey on evolutionary constrained optimization (Mezura-Montes and Coello 2011). In a recent review on EAs to solve DCOPs, the genetic algorithm (GA) appears as the most popular algorithm. However, there are new proposals based on other bio-inspired algorithms, e.g., differential evolution (DE) (Ameca-Alducin et al. 2014), gravitational search algorithm (GSA) (Pal et al. 2013b), evolutionary algorithm (EA) (Sharma and Sharma 2012b), T cell artificial immune system (Aragón et al. 2013) and multi-population algorithm (Bu et al. 2016). To deal with a constrained dynamic space, two main types of mechanisms have been added to the above-mentioned algorithms: (1) introduction and maintenance of diversity as in a GA with elitism and random immigrants (RIGAElit) (Grefenstette 1992), a GA with elitism and hypermutation (HyperMElit) (Cobb 1990), a dynamic constrained T cell (DCTC) (Aragón et al. 2013), intelligent constraint handling evolutionary algorithm (ICHEA) (Sharma and Sharma 2012b), a dynamic species-based particle swam optimization (DSPSO) (Bu et al. 2016) and a DE with two variants (DE/rand/1/bin and DE/best/1/bin) called dynamic differential evolution with combined variants (DDECV) (Ameca-Alducin et al. 2014) and (2) repair mechanisms within a GA (GA + Repair) (Nguyen and Yao 2012), within DE (DE + Repair) (Pal et al. 2013a) and also within GSA (GSA + Repair) (Pal et al. 2013b).

Recently, an improved version of DDECV, called DDECV + Repair (Ameca-Alducin et al. 2015a, b), was proposed, where a simple but effective repair method based on the differential mutation operator and resampling was proposed. The novelty of this repair method with respect to the above-mentioned is the fact that no feasible solutions are required. In contrast, the usage of the differential mutation with random vectors is emphasized so as to generate feasible solutions.

Even though DDECV + Repair was already proposed, its empirical validation was very limited [i.e., just a couple of measures were adopted in the experiments (Ameca-Alducin et al. 2015a, b)]. Therefore, this work aims to provide an in-depth analysis of DDECV + Repair, where the following unexplored issues are investigated:

1. The role of each one of its elements in the search behavior and expected performance.
2. The effects of (a) the change frequency and (b) the severity of the change in both, the objective function and the constraints, in its overall performance.
3. Its ability to detect a change and recover after it, and the way diversity is handled (i.e., suitable infeasible solutions maintenance during the search).
4. Its performance when solving problems where dynamism is present only in the objective function, only in the constraints or dynamism in both of them.

For each one of those four issues an experiment is designed. The empirical evidence provided in this work comprises both, direct and indirect comparisons. seven performance measures are computed (offline error, recovery rate, absolute recovery rate, percentage of infeasible solutions, detected change count, best error before change and feasible offline error), and statistical tests are applied to validate the findings observed in the samples of runs carried out. Eight approaches found in the specialized literature are adopted for comparison purposes. The contribution of this work aims to add knowledge about the capabilities of this particular DE-based algorithm to deal with a dynamic numerical constrained search space.

The rest of the paper is divided as follows. In Sect. 2 the problem of interest is stated, while Sect. 3 briefly introduces DE, DECV (the base algorithm) and details DDECV + Repair. Section 4 presents the four experiments and their corresponding results, where a recently proposed benchmark is solved (Nguyen and Yao 2012). Finally, Sect. 5 includes the conclusions and directions regarding future research.

2 Problem statement

A DCOP can be seen as a search problem where its fitness landscape and feasible region change through time. Without loss of generality, a DCOP can be defined as to:
Find $\mathbf{x}$, at each time t, which:

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

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

$$[L,U] = \{\mathbf{x} = (x_1, x_2, \ldots, x_D)\,|L_i \le x_i \le U_i,\\ i = 1 \ldots D\}$$

is the search space,
subject to:

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

is called the feasible region at time t.

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

Four types of DCOPs are defined: (1) a static objective function and static constraints (i.e., a static constrained optimization problem), (2) a dynamic objective function and static constraints, (3) a static objective function and dynamic constraints and (4) a dynamic objective function and dynamic constraints.

3 DDECV + Repair

3.1 Differential evolution

DE is a stochastic search algorithm which operates with a population of solutions called vectors (Price et al. 2005). The population is represented as: $\mathbf{x}_{i,G}, i = 1, \dots, NP$, where $\mathbf{x}_{i,G}$ represents vector i at generation G, and NP is the population size. Each target vector $\mathbf{x}_{i,G}$ generates one trial vector $\mathbf{u}_{i,G}$ by using a mutant vector $\mathbf{v}_{i,G}$. The mutant vector is obtained as in Eq. (1), where $\mathbf{x}_{r0,G}$, $\mathbf{x}_{r1,G}$ and $\mathbf{x}_{r2,G}$ are vectors chosen at random from the current population ($r0 \neq r1 \neq r2 \neq i$); $\mathbf{x}_{r0,G}$ is known as the base vector, and $\mathbf{x}_{r1,G}$ and $\mathbf{x}_{r2,G}$ are the difference vectors. $F > 0$ is a scale factor defined by the user.

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

After the mutant vector $\mathbf{v}_{i,G}$ is generated, it is combined with the target vector $\mathbf{x}_{i,G}$ to generate the trial vector $\mathbf{u}_{i,G}$ by applying a crossover operator as shown in Eq. (2).

$$u_{i,j,G} = \begin{cases} v_{i,j,G} & \text{if } (\text{rand}_j \leq \text{CR}) \text{ or } (j = J_{\text{rand}}) \\ x_{i,j,G} & \text{otherwise} \end{cases} \tag{2}$$

where $\text{CR} \in [0, 1]$ defines the similarity between the trial vector and the mutant vector, rand_j generates a random real number with uniform mutation between 0 and 1, $j \in \{1, \dots, D\}$ is the jth variable of the D-dimensional vector, $J_{\text{rand}} \in [1, D]$ is an random integer which prevents a target vector copy as its trial vector.

Finally, the best vector, based on its objective function value, between the target and trial vector is chosen to remain in the population for the next generation as shown in Eq. (3) (assuming minimization):

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G} & \text{if } \left(f(\mathbf{u}_{i,G}) \leq f(\mathbf{x}_{i,G})\right), \\ \mathbf{x}_{i,G} & \text{otherwise} \end{cases} \tag{3}$$

This DE variant is known as DE/rand/1/bin, where "rand" means the criterion to choose the base vector $\mathbf{x}_{r0,G}$, "1" indicates the number of vector differences, and "bin" is the type of crossover (binomial in this case, as in Eq. (2)).

Another DE variant is DE/best/1/bin, where the only difference with respect to DE/rand/1/bin is that the best vector in the current population, represented as $\mathbf{x}_{\text{best},G}$, is the base vector for all differential mutations [see Eq. (4)]. Details of other DE variants, particularly for constrained optimization, can be found in (Mezura-Montes et al. 2010)

$$\mathbf{v}_{i,G} = \mathbf{x}_{\text{best},G} + F(\mathbf{x}_{r1,G} - \mathbf{x}_{r2,G}) \tag{4}$$

3.2 Differential evolution with combined variants (DECV)

DDECV + Repair algorithm is based on differential evolution with combined variants (DECV) (Mezura-Montes et al. 2010), where DE/rand/1/bin is used at the beginning of the search, and after a percentage of feasible vectors (PFV), defined by the user, is found, DE/best/1/bin is used instead. DECV was initially proposed to solve static constrained optimization problems (SCOPs) (Mezura-Montes et al. 2010). The feasibility rules proposed by Deb (2000) are used in DECV as selection criteria in Eq. (3), and also every time the best vector is selected in DE/best/1/bin. The three rules are as follows:

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.

To favor a self-contained paper, the complete pseudocode of DECV is detailed in Algorithm 1

Algorithm 1 DECV algorithm

```
1: G=0
2: Create a randomly generated initial population $\mathbf{x}_{i,G}\ \forall i, i = 1, \dots, NP$
3: Evaluate each $\mathbf{x}_{i,G}\ \forall i, i = 1, \dots, NP$
4: $eval = eval + NP$
5: while $eval \leq Max\_eval$ do
6:    Compute feasiblePercent
7:    for $i \leftarrow 1$ to $NP$ do
8:       if feasiblePercent ≤ PFV then
9:          Generate $\mathbf{u}_{i,G}$ with Equations 1 and 2
10:      else
11:         Generate $\mathbf{u}_{i,G}$ with Equations 4 and 2
12:      end if
13:      if $f(\mathbf{u}_{i,G})$ is better than $f(\mathbf{x}_{i,G})$ based on the feasibility rules then
14:         $\mathbf{x}_{i,G+1} = \mathbf{u}_{i,G}$
15:      else
16:         $\mathbf{x}_{i,G+1} = \mathbf{x}_{i,G}$
17:      end if
18:   end for
19:   $G = G+1$
20: end while
```

3.3 DDECV + Repair

To deal with DCOPs, DDECV + Repair was added with a change detection mechanism which covers modifications in the objective function and the constraints. When a change is detected, DDECV + Repair utilizes DECV's two-variant combination to promote exploration in the dynamic constrained search space. As it is important to promote exploration after a change, approaching faster to the, maybe different, feasible region, is essential as well. Therefore, a repair mechanism based on the differential mutation and resampling is applied to infeasible vectors. Finally, a set of randomly generated vectors called immigrants are added to increase diversity in the population. In the following subsections, those four DDECV + Repair elements are detailed (Ameca-Alducin et al. 2015a, b).

3.3.1 Change detection

A timely change detection of the objective function and/or the constraints of a DCOP is the desired starting point to deal with a dynamic search space (du Plessis 2012; Richter 2009b). Therefore, DDECV+Repair uses a solution (i.e., vector) re-evaluation, also known as sensor-based detection (Richter 2009a). At each generation, before the first target vector and the target vector at the middle of the current population generate their corresponding trial vector, they are evaluated again and their objective function values and constraint values are compared against their previous values. If any value is different, an indicator is activated and the best vector in the current population is stored in an archive, called the memory population. Furthermore, all vectors in the current population and also those in the memory population are re-evaluated so as to get them updated. The memory population keeps promising solutions, based on the DCOP features before the detected change, that can be used afterward if similar conditions return later in the dynamic search (Nguyen et al. 2013). Using the first vector and the one located at the middle of the population for change detection purposes look to decrease the chance of missing a change as such a mechanism operates twice during a single generation and just two extra solution evaluations are computed. The pseudocode of the change detection mechanism (Ameca-Alducin et al. 2015b) is detailed in Algorithm 2.

Algorithm 2 Change_detection_mechanism

```
Require: $\mathbf{x}_{i,t-1}$
1: Evaluate $\mathbf{x}_{i,t}$ 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 $\mathbf{x}_{\text{best},t}$ to the memory population
4:     Re-evaluate all vectors in the current population and also in the memory population
5:     $eval = eval + current\_population\_size + memory\_population\_size$
6: end if
```

3.3.2 Exploration promotion

DDECV + Repair, as in the original DECV, starts using DE/rand/1/bin. Once a change is detected (see Algorithm 2), then the exploration promotion mechanism is activated as follows: the DE variant is changed to DE/best/1/bin, whose usage will last a number of generations defined by the user (Gen_{best}), and the F value is increased during such period of time to favor larger movements promoting exploration. Furthermore, considering that DE/best/1/bin is used, the best vector can be chosen from either the current population or the memory population (the best vectors found in previous environments). The idea of using DE/best/1/bin with larger F values to promote diversity in a constrained search space was concluded in Mezura-Montes et al. (2010), and it is adopted in this work. The details of the exploration promotion mechanism can be seen in Algorithm 3.

Algorithm 3 Exploration_promotion_mechanism

```
1: if counter for using DE/best/1/bin $< Gen_{\text{best}}$ then
2:     Generate $\mathbf{u}_{i,t}$ with Equation 4 with expanded F value and Equation 2
3: else
4:     Generate $\mathbf{u}_{i,t}$ with Equations 1 and 2
5: end if
6: return $\mathbf{u}_{i,t}$
```

3.3.3 The repair method

Repairing, in the context of constrained optimization, is understood as the process of converting an infeasible solution into a feasible one. Reference feasible solutions are then required for that purpose (Michalewicz and Nazhiyath 1995; Nguyen and Yao 2012, 2009; Pal et al. 2013a, b). However, the repair method used in DDECV + Repair does not use feasible vectors as reference. It is a resampling approach based on the differential mutation operator which works as follows (Ameca-Alducin et al. 2015b):

After each trial vector is generated, if it is infeasible, three new and temporal vectors are randomly generated with uniform distribution in order to apply the differential mutation operator [see Eq. (1)] in a similar way as a mutant vector is created in DE. Feasibility is then checked on the obtained vector. The process repeats until either a feasible vector is generated or Repair_Limit iterations are computed. Regardless of the feasibility of the vector obtained after the repair process, it is considered as the trial vector to increase diversity in the population. As only constraints are evaluated by the repair method, such evaluations are not added to the total evaluations required by DDECV + Repair. Algorithm 4 includes the repair details.

Algorithm 4 Repair_Method

```
Require: $\mathbf{u}_{i,G}$ {trial vector}
1: $counter = 0$
2: while $\mathbf{u}_{i,G}$ is infeasible and $counter \leq$ Repair_Limit do
3:     Generate three random vectors ($\mathbf{u}_{r0,G}$, $\mathbf{u}_{r1,G}$ and $\mathbf{u}_{r2,G}$)
4:     $\mathbf{u}_{i,G} = \mathbf{u}_{r0,G} + F(\mathbf{u}_{r1,G} - \mathbf{u}_{r2,G})$
5:     $counter = counter + 1$
6: end while
7: Return $\mathbf{u}_{i,G}$
```

3.3.4 The random immigrants

To add more diversity to the current population in DDECV + Repair, a number of IB immigrants (vectors generated at random with uniform distribution) are added to the population at the end of each generation. IB stands for "Immigrants Before a change". Moreover, during the period of time the exploration promotion is working (controlled by the Gen_{best} parameter), such number of immigrants (IA, "Immigrants After a change") is increased. In both cases, the immigrants replace the worst vectors in the current population.

The pseudocode of DDECV + Repair is shown in Algorithm 5.

Algorithm 5 DDECV + Repair algorithm

```
1: G=0
2: Create a randomly generated initial population $\mathbf{x}_{i,G}\ \forall i, i = 1, \ldots, NP$
3: Evaluate each $\mathbf{x}_{i,G}\ \forall i, i = 1, \ldots, NP$
4: $eval = eval + NP$
5: while $eval \leq Max\_eval$ do
6:   for $i \leftarrow 1$ to $NP$ do
7:     if $i = 1$ or $i = NP/2$ then
8:       Change_detection_Mechanism ($\mathbf{x}_{i,G}$) {Algorithm 2}
9:       $eval = eval + 1$
10:    end if
11:    $\mathbf{u}_{i,G}$ = Exploration_promotion_mechanism {Algorithm 3}
12:    if $\mathbf{u}_{i,G}$ is infeasible then
13:      Repair_Method($\mathbf{u}_{i,G}$) {Algorithm 4}
14:    end if
15:    $eval = eval + 1$
16:    if $f(\mathbf{u}_{i,G})$ is better than $f(\mathbf{x}_{i,G})$ based on the feasibility rules then
17:      $\mathbf{x}_{i,G+1} = \mathbf{u}_{i,G}$
18:    else
19:      $\mathbf{x}_{i,G+1} = \mathbf{x}_{i,G}$
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 design

Recalling from Sect. 1, the aim of this paper is to deepen into the empirical analysis of DDECV + Repair by considering (1) the role of each one of its elements, (2) how affected is with different change frequencies and severities, (3) its ability to detect a change and recover after it and its diversity handling and (4) its performance with dynamism in different parts of the problem. Therefore, four experiments were designed:

– A comparison of DDECV + Repair against own versions, each one without one of its elements (exploration promotion, repair method and random immigrants).
– A comparison of DDECV + Repair against recent approaches to solve DCOPs by varying the change frequency and severity.
– A comparison of DDECV + Repair against recent approaches to solve DCOPs by measuring changes detected, recovery rate and balance between feasible and infeasible vectors in the population.
– A comparison of DDECV + Repair against two recent approaches (ICHEA and DCTC) to analyze the presence of dynamism in different parts of the problem, i.e., dynamic objective function and static constraints, static objective function and dynamic constraints and dynamic objective function and dynamic constraints.

The four experiments solved a recently proposed benchmark for DCOPs (Nguyen and Yao 2012), which contains eighteen problems. The main features of those problems are summarized in Table 1, and the details can be found in Nguyen and Yao (2012, 2013). The parameters used for DDECV are those in Table 3. Such values were taken from (Ameca-Alducin et al. 2015b) and were obtained by using the iRace tool for parameter tuning (López-Ibáñez and Stützle 2012). The parameters used for the benchmark problems are detailed in Table 2, where different change frequencies and severities were considered.

4.1.1 Performance measures

The following seven performance measures were used in this work:

1. *Offline error* (Branke and Schmeck 2003) the most popular measure in the specialized literature of DCOPs (Nguyen and Yao 2012; Pal et al. 2013b). It is defined as the average of the computed errors at each one of the generations covering the total number of times. The offline error is always greater than or equal to zero. This lat-

Table 1 Main features of the test problems (Nguyen and Yao 2012)

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

Table 2 Parameter values for the test problems taken from Nguyen and Yao (2012)

Benchmark problem settings	
Number of runs	50
Number of changes	5/k
Change frequencies	250, 500, 1000, 2000 and 4000 Evals.
Obj. function change severity k	0.25 (small) 0.5 (medium) and 1.0 (large)
Constraint change severity S	10 (small), 20 (medium) and 50 (large)

Table 3 DDECV + Repair parameter values taken from Ameca-Alducin et al. (2014, 2015b)

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
Repair_Limit	100

ter value indicates a perfect performance (Nguyen et al. 2012). This measure is defined as in Eq. (5):

$$\text{offline\_error} = \frac{1}{G_{\max}} \sum_{G=1}^{G_{\max}} e(G) \tag{5}$$

where $G_{\max}$ is the number of generations computed by the algorithm and $e(G)$ denotes the error in the current generation G [see Eq. (6)]:

$$e(G) = |f(\mathbf{x}^*, t) - f(\mathbf{x}_{\text{best},G}, t)| \tag{6}$$

where $f(\mathbf{x}^*, t)$ denotes the feasible global optima in current time t, and $f(\mathbf{x}_{\text{best},G}, t)$ is the best solution (feasible or infeasible) found so far at generation G in current time t. Using the absolute value of the error mitigates those problems related to the presence of infeasible solutions with a better objective function value with respect to the feasible optimum of the corresponding time.

2. *Detected_change count* each detected change is counted to verify whether the algorithm has the ability to detect all of them during the search process.
3. *Recovery rate (RR)* (Nguyen and Yao 2012) this measure was designed to analyze how quickly an algorithm recov-

ers after a change and starts converging to the new best solution before the next change occurs. Such new best solution is not necessarily the global optimum. The RR value would be 1 in the best case where the algorithm is able to recover and converge to the best solution immediately after a change. A value closer to zero means the algorithm is unable to recover [see (Eq. 7)]:

$$\mathrm{RR} = \frac{1}{t_{\max}} \sum_{t=1}^{t_{\max}} \frac{\sum_{G=1}^{G_{\max}(t)} \left[f(\mathbf{x}_{\mathrm{best},G}, t) - f(\mathbf{x}_{\mathrm{best},1}, t) \right]}{G_{\max}(t) \left[f(\mathbf{x}_{\mathrm{best},G}, t) - f(\mathbf{x}_{\mathrm{best},1}, t) \right]} \tag{7}$$

where $t_{\max}$ is the total number of times (changes) in the environment and every change occurs at time t, $G_{\max}(t)$ is the total number of generations at time t, $f(\mathbf{x}_{\mathrm{best},G}, t)$ is the objective function value of the best *feasible* solution found at current generation G in current time t, and $f(\mathbf{x}_{\mathrm{best},1}, t)$ is the objective function value of the best feasible solution at the first generation of the new time (i.e., the first best feasible solution obtained by the algorithm after a change).

4. *Absolute recovery rate (ARR)* (Nguyen and Yao 2012) it is similar to RR, but used to analyze how fast an algorithm starts converging to the global optimum before the next change occurs. The ARR value would be 1 in the best case when the algorithm is able to recover and converge to the global optimum immediately after a change and would be zero in case the algorithm is unable to recover [see Eq. (8)]:

$$\mathrm{ARR} = \frac{1}{t_{\max}} \sum_{t=1}^{t_{\max}} \frac{\sum_{G=1}^{G_{\max}(t)} [f(\mathbf{x}_{\mathrm{best},G}, t) - f(\mathbf{x}_{\mathrm{best},1}, t)]}{G_{\max}(t)[f(\mathbf{x}^{*}, t) - f(\mathbf{x}_{\mathrm{best},1}, t)]} \tag{8}$$

where $f(\mathbf{x}^{*}, t)$ is the feasible global optima in current time t. $t_{\max}$, t, $G_{\max}(t)$, G, $f(\mathbf{x}_{\mathrm{best},G}, t)$ and $f(\mathbf{x}_{\mathrm{best},1}, t)$ were defined in Eq. (7) for RR.

5. *Percentage of selected infeasible individuals* this measure computes the average of the number of infeasible vectors at each one of the $G_{\max}$ generations. A value greater than zero is desirable for most of the search, as the algorithm is able to maintain infeasible solutions to avoid feasible local optima.

6. *Best error before change* this measure, proposed in Trojanowski and Michalewicz (1999), is calculated as the average of the smallest errors accomplished at the end of each period of time right before a change of time occurs [see Eq. (9)]:

$$E_{\mathrm{best}} = \frac{1}{t_{\max}} \sum_{t=1}^{t_{\max}} e_{\mathrm{best}}(G_{\max}(t)) \tag{9}$$

where $t_{\max}$ is the total number of times (changes) in the environment and every change occurs at time t, $G_{\max}(t)$ is the total number of generations at time t, and $e_{\mathrm{best}}(G_{\max}(t))$ is the difference between the feasible global optima in current time t and the best solution found so far at generation G in current time t [see (Eq. 6)].

7. *Feasible offline error* (Aragón et al. 2013) it is similar to offline error [see Eq. (5)], but only the feasible solutions are computed. If an infeasible solution is found, then nothing is added.

4.1.2 Compared algorithms

The results obtained by DDECV + Repair were compared with those obtained by the following eight state-of-the-art algorithms in dynamic constrained optimization, which are briefly described below.

GAElit is a traditional genetic algorithm (GA) with elitism. In this algorithm, nonlinear ranking parent selection, arithmetic crossover and uniform mutation are employed. To cope with constraints, a static penalty function is used. To avoid obsolete data, the change detection mechanism (see Algorithm 2) is applied when elitism takes place.

HyperMElit is similar to GAElit, but with an adaptive mechanism to switch between two mutation rates: (1) low (standard mutation) rate and (2) high (hypermutation) rate, to increase diversity. If the best solution gets worse, the hypermutation is applied for an user-defined number of generations (Cobb 1990).

RIGAElit is also similar to GAElit. However, after the mutation operator is applied, a fraction of the current population is replaced with randomly generated individuals (random immigrants). This fraction is determined by a random immigrant rate (also named replacement rate). In the experiments of this work, the third part of the current population is replaced by random immigrants in order to maintain diversity throughout the search process (Grefenstette 1992).

GA + Repair was proposed in Nguyen and Yao (2009). This algorithm is similar to GAElit but with a repair method based on GENOCOP III (Michalewicz and Nazhiyath 1995). The repair method converts infeasible solutions in the population (called search population) into feasible ones by using feasible solutions as reference. Those feasible solutions are in the so-called reference population, where only feasible solutions are allowed. To avoid obsolete data, the change detection mechanism is applied (see Algorithm 2).

DE+Repair is based on DE/rand/1/bin with a change detection mechanism, in which the solutions are re-evaluated in order to detect modifications in the environment. A modified repair method proposed by Michalewicz and Nazhiyath (1995) is used as a constraint handler, whose application is based on closeness in the variable space between the reference (feasible) solution and the infeasible solution to be repaired (Pal et al. 2013a).

GSA + Repair is based on the gravitational search algorithm (GSA) (Rashedi et al. 2009) and shares the same change detection mechanism and repair method with DE + Repair (Pal et al. 2013b).

ICHEA is a variation of an EA. This approach uses a intermarriage crossover operator which employs knowledge from constraints rather than blindly searching the solution. This crossover is intended to make a generic offspring that tries to satisfy more than one constraint because its parents are selected from two different feasible regions (Sharma and Sharma 2012a). The algorithm favors those offspring which satisfy more constraints by using Deb's rules (Deb 2000). To avoid the loss of diversity in the population, this algorithm has a diversity management and a stagnant local optimal solutions management which works like tabu search algorithm (Sharma and Sharma 2012b).

Dynamic constrained T cell (DCTC) is an algorithm inspired on the T cell model, which operates on four populations, corresponding to the groups in which the T-cells are divided: (1) virgin cells to provide diversity, (2) effector cells CD4 to explore the search space, (3) effector cells CD8 to use real numbers representation and (4) memory cells to explore the neighborhood of the best found solutions. Furthermore, a change detection mechanism in the environment through the re-evaluation of solutions is added. The feasibility criteria are used as constraint handler (Aragón et al. 2013).

The parameter values used for the abovementioned compared algorithms, where direct comparisons were carried out (experiment 3), are given in Table 4.

4.2 Results

4.2.1 Experiment 1. DDECV + Repair element analysis

The first experiment compared DDECV+Repair against its own versions, where in each one of them, a single element was deactivated, as follows:

- *DDE rand + Repair* A version without the exploration promotion mechanism (i.e., without the combined variants and memory population), where only DE/rand/1/bin is used.
- *DDE best + Repair* A version without the exploration promotion mechanism (i.e., without the combined variants and memory population), where only DE/best/1/bin is used.
- *DDECV − Repair* A version with all the elements with the exception of the repair method.
- *DDECV − Imm + Repair* A version with all the elements with the exception of the random immigrants.
- *DDECV − Mem + Repair* A version with all the elements with the exception of the memory population (the best vectors in the previous times).

Table 4 Parameter values for the tested algorithms (GAElit, RIGAElit, HyperMElit and GA + Repair), taken from Nguyen and Yao (2012)

All algorithms	
Pop size	25
Elitism	Elitism is applied ever
Selection	Nonlinear ranking
Mutation	Uniform, $P = 0.15$
Crossover	Arithmetic, $P = 0.1$
RIGAElit	
Immigrants rate	$P = 0.3$
HyperMElit	
Triggered mutate	Uniform, $P = 0.5$
GA + Repair	
Search pop size	Pop size (4/5)
Reference pop size	Pop size (1/5)
Replacement rate	0

The eighteen test problems were solved by each one of the six versions, and the offline error measure was computed. The change frequency was 1000 evaluations, and the severity of the change was medium (i.e., $k = 0.50$ and $S = 20$). Those values are the most used in the specialized literature of DCOPs (Nguyen and Yao 2012). The results obtained are summarized in Table 5. The statistical validation was made with the 95 % confidence Kruskal–Wallis (KW) test and the Bergmann–Hommels post hoc test, as suggested in Derrac et al. (2011). Such tests indicated no significant differences among DDECV + Repair versions by considering all the eighteen test problems.

To get a closer look, each test problem was analyzed separately. In this way, the 95 % confidence Wilcoxon rank-sum test was applied for each test problem in pairwise comparisons between DDECV + Repair and each one of its five versions. The results are presented in Table 6.

Based on such results, DDECV + Repair outperformed DDE_rand +Repair in fourteen test problems (g24_u, g24_1, g24_2, g24_3, g24_3b, g24_4, g24_5, g24_6a, g24_6b, g24_6c, g24_6d, g24_7, g24_8a and g24_8b), while DDE_rand + Repair was better in just one (unconstrained) test problem (g24_2u). In three test problems (g24_f, g24_uf

Table 5 Average and standard deviation offline error values obtained by DDECV + Repair and its five incomplete versions with a change frequency of 1000 evaluations and a medium severity of change ($k = 0.50$ and $S = 20$)

Algorithms	Functions		
	G24_u	G24_1	G24_f
DDE_rand + Repair	0.046 (±0.007)	0.08 (±0.014)	0.023 (±0.008)
DDE_best + Repair	0.044 (±0.005)	0.063 (±0.013)	**0.013 (±0.004)**
DDECV − Repair	**0.037 (±0.005)**	0.094 (±0.024)	0.031 (±0.01)
DDECV − Imm + Repair	0.101 (±0.025)	0.067 (±0.014)	0.02 (±0.007)
DDECV − Mem + Repair	0.04 (±0.006)	0.075 (±0.015)	0.025 (±0.007)
DDECV + Repair	0.039 (±0.007)	**0.061 (±0.01)**	0.021 (±0.006)
	G24_uf	G24_2	G24_2u
DDE_rand + Repair	0.012 (±0.004)	0.081 (±0.011)	0.035 (±0.003)
DDE_best + Repair	**0.006 (±0.002)**	0.065 (±0.011)	0.033 (±0.002)
DDECV − Repair	0.011 (±0.005)	0.088 (±0.016)	0.031 (±0.001)
DDECV − Imm + Repair	0.01 (±0.004)	0.067 (±0.014)	0.46 (±0.249)
DDECV − Mem + Repair	0.011 (±0.004)	0.062 (±0.011)	**0.031 (±0.001)**
DDECV + Repair	0.009 (±0.002)	**0.062 (±0.006)**	0.036 (±0.001)
	G24_3	G24_3b	G24_3f
DDE_rand + Repair	0.074 (±0.006)	0.146 (±0.019)	0.013 (±0.003)
DDE_best + Repair	0.054 (±0.005)	0.118 (±0.013)	**0.009 (±0.003)**
DDECV − Repair	0.061 (±0.009)	0.131 (±0.021)	0.025 (±0.008)
DDECV − Imm + Repair	**0.042 (±0.003)**	0.095 (±0.012)	0.011 (±0.003)
DDECV − Mem + Repair	0.045 (±0.005)	0.101 (±0.014)	0.012 (±0.004)
DDECV + Repair	0.046 (±0.006)	**0.084 (±0.006)**	0.01 (±0.002)
	G24_4	G24_5	G24_6a
DDE_rand + Repair	0.144 (±0.016)	0.092 (±0.012)	0.044 (±0.005)
DDE_best + Repair	0.117 (±0.013)	0.079 (±0.008)	0.034 (±0.004)
DDECV − Repair	0.131 (±0.02)	0.122 (±0.023)	0.065 (±0.026)
DDECV − Imm + Repair	0.094 (±0.009)	0.38 (±0.162)	**0.032 (±0.005)**
DDECV − Mem + Repair	0.099 (±0.014)	0.082 (±0.012)	0.033 (±0.007)
DDECV + Repair	**0.088 (±0.011)**	**0.078 (±0.008)**	0.036 (±0.005)
	G24_6b	G24_6c	G24_6d
DDE_rand + Repair	0.055 (±0.01)	0.051 (±0.007)	0.099 (±0.009)
DDE_best + Repair	0.041 (±0.008)	0.039 (±0.006)	0.094 (±0.009)
DDECV − Repair	0.049 (±0.012)	0.051 (±0.014)	0.115 (±0.024)
DDECV − Imm + Repair	**0.037 (±0.006)**	**0.035 (±0.005)**	0.111 (±0.014)
DDECV − Mem + Repair	0.045 (±0.01)	0.042 (±0.009)	0.082 (±0.011)
DDECV + Repair	0.041 (±0.01)	0.041 (±0.01)	**0.079 (±0.006)**
	G24_7	G24_8a	G24_8b
DDE_rand + Repair	0.129 (±0.018)	0.215 (±0.02)	0.145 (±0.042)
DDE_best + Repair	0.107 (±0.012)	0.138 (±0.019)	0.09 (±0.034)
DDECV − Repair	0.169 (±0.027)	0.159 (±0.023)	0.13 (±0.035)
DDECV − Imm + Repair	**0.099 (±0.013)**	0.283 (±0.064)	0.076 (±0.024)
DDECV − Mem + Repair	0.114 (±0.015)	0.18 (±0.021)	0.098 (±0.029)
DDECV + Repair	0.107 (±0.011)	**0.138 (±0.015)**	**0.074 (±0.025)**

Best results are remarked in boldface

Table 6 95 % confidence Wilcoxon rank-sum test results on pairwise comparisons between DDECV + Repair against each one of its five incomplete versions based on the offline error results in Table 5

Functions	Algorithms				
	DDE_rand + Repair	DDE_best + Repair	DDECV − Repair	DDECV − Imm + Repair	DDECV − Mem + Repair
g24_u	+	+	−	+	+
g24_1	+	=	+	=	+
g24_f	=	−	+	−	+
g24_uf	=	−	=	=	+
g24_2	+	=	+	=	+
g24_2u	−	−	=	+	−
g24_3	+	+	+	−	+
g24_3b	+	+	+	+	+
g24_3f	=	−	+	+	+
g24_4	+	+	+	+	+
g24_5	+	=	+	+	+
g24_6a	+	−	+	=	−
g24_6b	+	+	=	−	+
g24_6c	+	−	+	−	+
g24_6d	+	+	+	+	+
g24_7	+	+	+	−	=
g24_8a	+	+	=	+	+
g24_8b	+	+	+	+	+

"+" means that DDECV + Repair outperformed the version in the corresponding column. "−" means that the version in the corresponding column outperformed DDECV + Repair. No significant differences between DDECV + Repair and the version in the corresponding columns are indicated with "="

and g24_3f), all of them static and the second of them unconstrained, no significant differences were observed.

DDECV + Repair outperformed DDE_best + Repair in nine test problems (g24_u, 24_3, g24_3b, g24_4, g24_6b, g24_6d, g24_7, g24_8a and g24_8b). On the other hand, DDE_best + Repair had a better performance in six test problems (g24_f, g24_uf, g24_2u, g24_3f, g24_6a and g24_6c test). No significant differences were observed in three test problems (g24_1, g24_2 and g24_5). It is important to note that the problems where DDE_best + Repair provided a better performance have static constraints or they are unconstrained ones.

Regarding DDECV − Repair, the results indicate that DDECV + Repair outperformed it in thirteen test problems (g24_1, g24_f, g24_2, g24_3, g24_3b, g24_3f, g24_4, g24_5, g24_6a, g24_6c, g24_6d, g24_7 and g24_8b). In contrast, DDECV − Repair was better in just one unconstrained test problem (g24_u). No significant differences were observed in four test problems (g24_uf, g24_2u, g24_6b and g24_8a), where the first two are unconstrained. Those results highlight the importance of the repair method in DDECV + Repair.

Furthermore, DDECV + Repair obtained better results with respect to DDECV − Imm + Repair in nine test problems (g24_u, g24_2u, g24_3b, g24_3f, g24_4, g24_5, g24_6d, g24_8a and g24_8b), while DDECV − Imm + Repair surpassed the results of DDECV + Repair in five test problems (g24_f, g24_3, g24_6b, g24_6c and g24_7). In four test problems (g24_1, g24_uf, g24_2, and g24_6a) no significant differences were obtained. The common feature of three test problems where the version with no immigrants was better than DDECV + Repair (g24_f, g24_3 and g24_7) is that the global optimum is in the boundaries of the feasible region, while in the other two test problems (g24_6b and g24_6c), the feasible global optimum is in the boundaries of the search space. Therefore, the addition of those randomly generated solutions into the current population seems to affect the convergence to those solutions located in the boundaries of either the feasible region or the search space.

Finally, DDECV + Repair outperformed DDECV − Mem + Repair in fifteen test problems (g24_u, g24_1, g24_f, g24_uf, g24_2, g24_3, g24_3b, g24_3f, g24_4, g24_5, g24_6b, g24_6c, g24_6d, g24_8a and g24_8b), while DDECV − Mem + Repair had a better performance in two test problems (g24_2u and g24_6a), where the first problem is unconstrained and the second has static constraints. No significant differences were observed in just one test problem (g24_7), where the main feature is that the global optimum is in the boundaries of the feasible region. The results suggest the importance of the memory population, because the previous search conditions appear later in the search.

Table 7 Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 250 evaluations and a medium change severity ($k = 0.5$ and $S = 20$)

Algorithms	Functions		
	G24_u	G24_1	G24_f
GAElit	0.265 (±0.05)	0.781 (±0.168)	0.326 (±0.161)
RIGAElit	0.316 (±0.056)	0.706 (±0.12)	0.351 (±0.129)
HyperMelit	0.274 (±0.045)	0.667 (±0.12)	0.34 (±0.109)
GA + Repair	0.674 (±0.099)	0.414 (±0.099)	0.125 (±0.052)
DDECV + Repair	**0.179 (±0.013)**	**0.269 (±0.021)**	**0.097 (±0.027)**
	G24_uf	G24_2	G24_2u
GAElit	0.177 (±0.055)	0.469 (±0.081)	0.234 (±0.066)
RIGAElit	0.202 (±0.06)	0.386 (±0.055)	0.195 (±0.035)
HyperMelit	0.185 (±0.069)	0.434 (±0.061)	0.198 (±0.045)
GA + Repair	0.463 (±0.152)	0.412 (±0.06)	0.46 (±0.071)
DDECV + Repair	**0.039 (±0.018)**	**0.245 (±0.011)**	**0.189 (±0.01)**
	G24_3	G24_3b	G24_3f
GAElit	0.646 (±0.179)	0.857 (±0.188)	0.35 (±0.188)
RIGAElit	0.655 (±0.126)	0.797 (±0.094)	0.413 (±0.12)
HyperMelit	0.562 (±0.13)	0.732 (±0.108)	0.364 (±0.163)
GA + Repair	0.115 (±0.027)	0.382 (±0.098)	0.059 (±0.022)
DDECV + Repair	**0.11** (±0.013)	**0.28** (±0.03)	**0.038 (±0.011)**
	G24_4	G24_5	G24_6a
GAElit	0.844 (±0.112)	0.452 (±0.07)	1.179 (±0.232)
RIGAElit	0.773 (±0.114)	0.398 (±0.062)	0.775 (±0.138)
HyperMelit	0.738 (±0.105)	0.398 (±0.048)	0.83 (±0.095)
GA + Repair	**0.252 (±0.058)**	0.263 (±0.046)	0.785 (±0.192)
DDECV + Repair	0.277 (±0.029)	**0.25** (±0.031)	**0.191 (±0.014)**
	G24_6b	G24_6c	G24_6d
GAElit	0.814 (±0.088)	0.77 (±0.097)	0.842 (±0.099)
RIGAElit	0.624 (±0.071)	0.649 (±0.084)	0.75 (±0.1)
HyperMelit	0.67 (±0.057)	0.708 (±0.056)	0.723 (±0.081)
GA + Repair	0.738 (±0.101)	0.673 (±0.076)	0.622 (±0.118)
DDECV + Repair	**0.237** (±0.028)	**0.227 (±0.021)**	**0.437 (±0.029)**
	G24_7	G24_8a	G24_8b
GAElit	0.631 (±0.141)	0.404 (±0.033)	0.926 (±0.115)
RIGAElit	0.771 (±0.089)	0.526 (±0.046)	0.913 (±0.086)
HyperMelit	0.619 (±0.105)	0.46 (±0.023)	0.923 (±0.053)
GA + Repair	0.211 (±0.047)	**0.389 (±0.05)**	0.511 (±0.126)
DDECV + Repair	**0.169** (±0.03)	0.48 (±0.039)	**0.349 (±0.021)**

Best results are remarked in boldface

The overall results of this first experiment lead to the following conclusions:

– The most important mechanism in DDECV + Repair is precisely the repair mechanism. However, even without it, the algorithm can provide competitive results but only in unconstrained dynamic problems.
– The absence of the diversity promotion mechanism also affects DDECV + Repair's performance, but such impact is less significant. In fact, even without it, if DE/best/1/bin

Table 8 Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 500 evaluations and a medium change severity ($k = 0.5$ and $S = 20$)

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	0.5 (±0.059)	0.264 (±0.024)	0.077 (±0.011)
DDECV + Repair	**0.086 (±0.009)**	**0.123 (±0.015)**	**0.04 (±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 + Repair	**0.019 (±0.005)**	**0.137 (±0.008)**	**0.089 (±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 + Repair	0.064 (±0.008)	**0.143 (±0.012)**	**0.024 (±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 + Repair	0.145 (±0.012)	**0.139 (±0.014)**	**0.08 (±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 + Repair	**0.097 (±0.01)**	**0.093 (±0.011)**	**0.193 (±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	0.43 (±0.062)	0.355 (±0.028)	0.71 (±0.071)
GA + Repair	**0.134 (±0.017)**	0.341 (±0.032)	0.38 (±0.068)
DDECV + Repair	0.135 (±0.023)	**0.262 (±0.031)**	**0.176 (±0.033)**

Best results are remarked in boldface

is used, problems with a dynamic objective function but with static constraints can still be solved.

– The random immigrants have a positive effect on DDECV + Repair. However, the presence of random solutions at each generation may affect the algorithm's performance when the feasible global optimum is located at the boundaries of either the feasible region or the search space.

– Other important mechanism in DDECV + Repair is the memory population. Without it, the performance of DDECV + Repair is affected.

Table 9 Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 1000 evaluations and a medium change severity ($k = 0.5$ and $S = 20$)

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 + 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 (±0.05)	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 + 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 + Repair	0.046 (±0.006)	0.084 (±0.006)	0.01 (±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)	0.24 (±0.035)	0.333 (±0.05)
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 + 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)	0.51 (±0.075)
RIGAElit	0.309 (±0.039)	0.325 (±0.029)	0.342 (±0.057)
HyperMelit	0.39 (±0.039)	0.394 (±0.051)	0.456 (±0.041)
GA + Repair	0.427 (±0.048)	0.39 (±0.038)	0.354 (±0.038)
DE + Repair	0.101 (±0.012)	0.79 (±0.01)	0.91 (±0.011)
GSA + Repair	0.047 (±0.003)	0.045 (±0.004)	**0.037 (±0.007)**
DDECV + Repair	**0.041 (±0.01)**	**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 + Repair	0.107 (±0.011)	**0.138 (±0.015)**	**0.074 (±0.025)**

Best results are remarked in boldface

Table 10 Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 2000 evaluations and a medium change severity ($k = 0.5$ and $S = 20$)

Algorithms	Functions		
	G24_u	G24_1	G24_f
GAElit	0.065 (±0.011)	0.332 (±0.074)	0.092 (±0.052)
RIGAElit	0.11 (±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 + Repair	**0.023 (±0.003)**	**0.036 (±0.01)**	**0.011 (±0.004)**
	G24_uf	G24_2	G24_2u
GAElit	0.032 (±0.01)	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 (±0.04)
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)	0.32 (±0.058)	0.072 (±0.032)
RIGAElit	0.208 (±0.026)	0.262 (±0.024)	0.1 (±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)**	0.01 (±0.003)
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 (±0.05)
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)	0.3 (±0.054)
DDECV + Repair	0.057 (±0.005)	**0.041 (±0.004)**	**0.02 (±0.004)**
	G24_6b	G24_6c	G24_6d
GAElit	0.274 (±0.028)	0.282 (±0.033)	0.318 (±0.059)
RIGAElit	0.21 (±0.025)	0.213 (±0.027)	0.242 (±0.027)
HyperMelit	0.234 (±0.019)	0.249 (±0.034)	0.281 (±0.03)
GA + Repair	0.306 (±0.03)	0.287 (±0.042)	0.263 (±0.024)
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	**0.05 (±0.015)**	0.247 (±0.02)	0.136 (±0.035)
DDECV + Repair	0.057 (±0.005)	**0.075 (±0.015)**	**0.041 (±0.012)**

Best results are remarked in boldface

4.2.2 Experiment 2: Change frequency and severity analysis

The second experiment analyzed the effects of different change frequencies and also different change severities in the performance provided by DDECV+Repair. The six algorithms already described were used for comparison purposes, and their results were taken from their corresponding documents (Cobb 1990; Cobb and Grefenstette 1993; Nguyen et al. 2012; Nguyen and Yao 2010; Pal et al. 2013a, b). The eighteen test problems, as in Experiment 1, were solved.

Table 11 Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 4000 evaluations and a medium change severity ($k = 0.5$ and $S = 20$)

Algorithms	Functions		
	G24_u	G24_1	G24_f
GAElit	0.038 (±0.009)	0.192 (±0.052)	0.069 (±0.036)
RIGAElit	0.061 (±0.013)	0.155 (±0.016)	0.073 (±0.034)
HyperMElit	0.046 (±0.01)	0.195 (±0.025)	0.066 (±0.044)
GA + Repair	0.332 (±0.017)	0.032 (±0.008)	0.012 (±0.003)
DDECV + Repair	**0.01** (**±0.001**)	**0.019** (**±0.005**)	**0.005** (**±0.002**)
	G24_uf	G24_2	G24_2u
GAElit	0.016 (±0.003)	0.141 (±0.021)	0.028 (±0.006)
RIGAElit	0.03 (±0.007)	0.12 (±0.013)	0.036 (±0.01)
HyperMElit	0.015 (±0.005)	0.138 (±0.034)	0.025 (±0.008)
GA + Repair	0.117 (±0.025)	0.096 (±0.01)	0.114 (±0.029)
DDECV + Repair	**0.002** (**±0.001**)	**0.021** (**±0.005**)	**0.01** (**±0.001**)
	G24_3	G24_3b	G24_3f
GAElit	0.119 (±0.034)	0.2 (±0.05)	0.057 (±0.016)
RIGAElit	0.148 (±0.019)	0.188 (±0.026)	0.054 (±0.02)
HyperMElit	0.104 (±0.019)	0.192 (±0.026)	0.052 (±0.02)
GA + Repair	**0.01** (**±0.002**)	**0.024** (**±0.004**)	0.005 (±0.001)
DDECV + Repair	0.029 (±0.003)	0.038 (±0.006)	**0.003** (**±0.001**)
	G24_4	G24_5	G24_6a
GAElit	0.193 (±0.029)	0.141 (±0.018)	0.226 (±0.04)
RIGAElit	0.218 (±0.039)	0.132 (±0.017)	0.148 (±0.016)
HyperMElit	0.209 (±0.034)	0.144 (±0.024)	0.169 (±0.022)
GA + Repair	**0.025** (**±0.004**)	0.086 (±0.017)	0.217 (±0.028)
DDECV + Repair	0.037 (±0.006)	**0.024** (**±0.003**)	**0.01** (**±0.002**)
	G24_6b	G24_6c	G24_6d
GAElit	0.166 (±0.02)	0.164 (±0.02)	0.183 (±0.024)
RIGAElit	0.141 (±0.019)	0.143 (±0.019)	0.152 (±0.019)
HyperMElit	0.151 (±0.017)	0.151 (±0.017)	0.17 (±0.023)
GA + Repair	0.205 (±0.025)	0.212 (±0.04)	0.191 (±0.02)
DDECV + Repair	**0.013** (**±0.003**)	**0.012** (**±0.002**)	**0.024** (**±0.002**)
	G24_7	G24_8a	G24_8b
GAElit	0.148 (±0.03)	0.205 (±0.011)	0.344 (±0.04)
RIGAElit	0.21 (±0.04)	0.232 (±0.01)	0.363 (±0.042)
HyperMElit	0.158 (±0.037)	0.215 (±0.013)	0.332 (±0.034)
GA + Repair	**0.026** (**±0.007**)	0.217 (±0.014)	0.068 (±0.019)
DDECV + Repair	0.034 (±0.005)	**0.041** (**±0.014**)	**0.018** (**±0.006**)

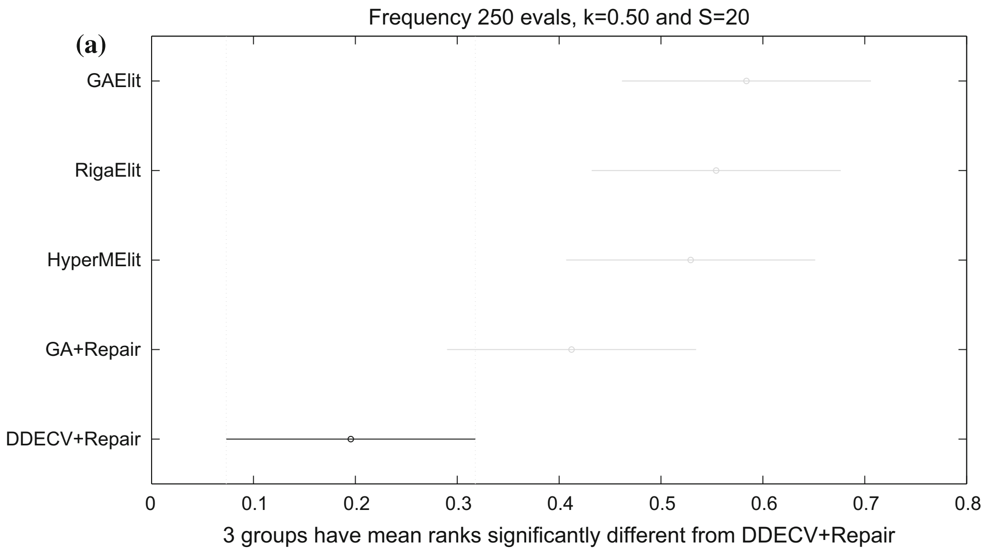
Best results are remarked in boldface

The first comparison of this second experiment focused on the change frequency. The results of the 95 % confidence Kruskal–Wallis test applied to the offline error results obtained by DDECV + Repair and the compared algorithms (GAElit, RIGAElit, HyperMElit, GA + Repair, DE + Repair and GSA + Repair) with five different change frequencies and a medium change severity ($k = 0.5$ and $S = 20$) are presented in Table 12 (250, 500, 2000 and 4000 evaluations)

Table 12 95 % confidence Kruskal–Wallis test results on the offline error values in Tables 7, 8, 10 and 11, obtained by DDECV+Repair, GAElit, RIGAElit, HyperMElit and GA + Repair with different change frequencies and a medium change severity ($k = 0.5$ and $S = 20$)

Frequency	Algorithms				
	GAElit (1)	RIGAElit (2)	HyperMElit (3)	GA + Repair (4)	DDECV + Repair (5)
250	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$		$1^{(+)}$, $2^{(+)}$ and $3^{(+)}$
500	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$ and $4^{(+)}$
2000	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$ and $4^{(+)}$
4000	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$ and $4^{(+)}$

"$X^{(+)}$" means that the algorithm in the corresponding column outperformed algorithm X. "$X^{(-)}$" means that the algorithm in the corresponding column was outperformed by algorithm X. If algorithm X does not appear in column, Y means no significant differences between X and Y



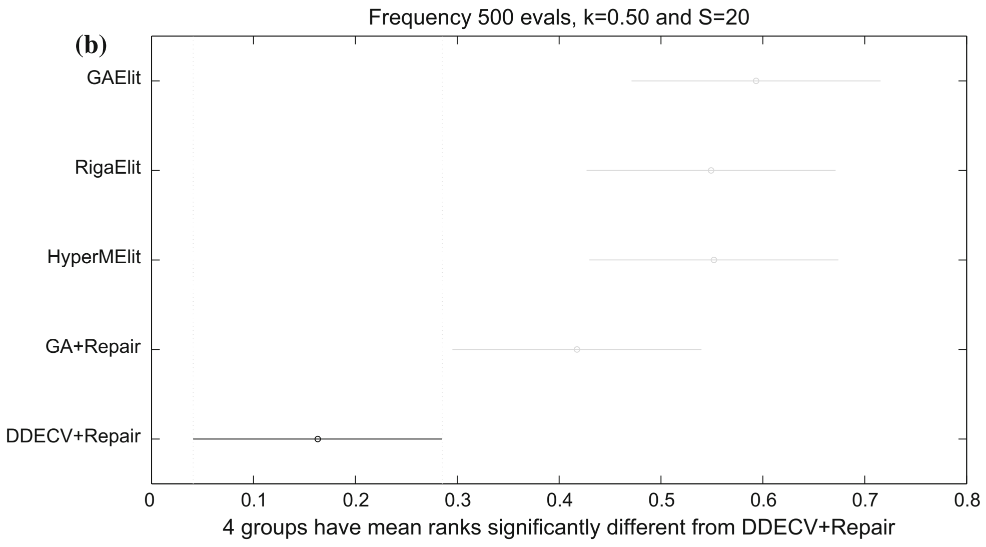


Fig. 1 Bonferroni–Dunn post hoc test results based on the offline error values obtained by DDECV + Repair and the compared algorithms with a medium change severity ($k = 0.5$ and $S = 20$), considering two change frequencies: **a** 250 evaluations and **b** 500 evaluations

and Table 13 (1000 evaluations, which was presented separately because DE + Repair and GSA + Repair only reported results for that change frequency).

The complete offline error values of this comparison are detailed in Tables 7, 8, 9, 10, and 11, for 250, 500, 1000, 2000 and 4000 evaluations, respectively.

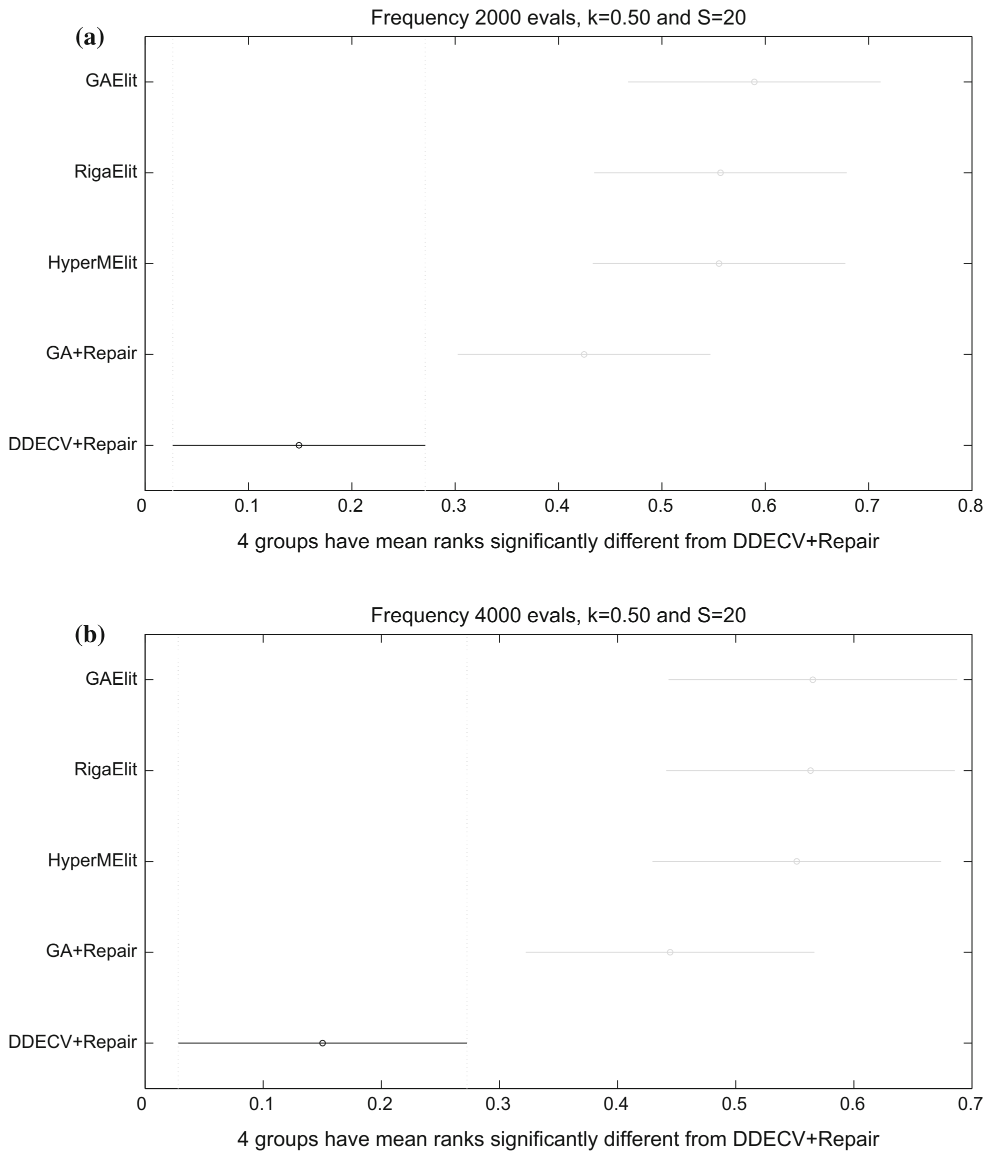


Fig. 2 Bonferroni–Dunn post hoc test results based on the offline error values obtained by DDECV + Repair and the compared algorithms with a medium change severity ($k = 0.5$ and $S = 20$), considering two change frequencies: **a** 2000 evaluations and **b** 4000 evaluations

From Table 12 it was observed that DDECV + Repair outperformed all algorithms in all change frequencies, with the exception of GA + Repair with 250 evaluations. The results of the Bonferroni–Dunn post hoc test for 250 and 500 evaluations in Fig. 1 and for 2000 and 4000 evaluations in Fig. 2 confirm such finding. Regarding the change frequency of 1000 evaluations (Table 13), DDECV + Repair outperformed again GAElit, RIGAElit, HyperMElit and GA + Repair. On the other hand, its performance was similar with respect to DE + Repair and GSA + Repair. Figure 3, with the Bonferroni–Dunn post hoc test results, confirms the abovementioned.

The second comparison of the second experiment aimed to analyze the impact of the change severity in DDECV + Repair. The results of the 95 % confidence Kruskal–Wallis test applied to the offline error results obtained by DDECV + Repair and the compared algorithms (GAElit, RIGAElit, HyperMElit and GA+Repair) with a low ($k = 0.25$ and $S = 10$) and a high ($k = 1.0$ and $S = 50$) change severities, both with 1000 evaluations as change frequency, are presented in Table 16. DE + Repair and GSA + Repair were omitted in this experiment because no results were found. The complete offline error values of this comparison are detailed in Tables 14 and 15, for the low change severity and the high change severity, respectively.

Based on the results in Table 16, DDECV + Repair outperformed the compared algorithms in both, low and high change severities. The only exception was GA + Repair when

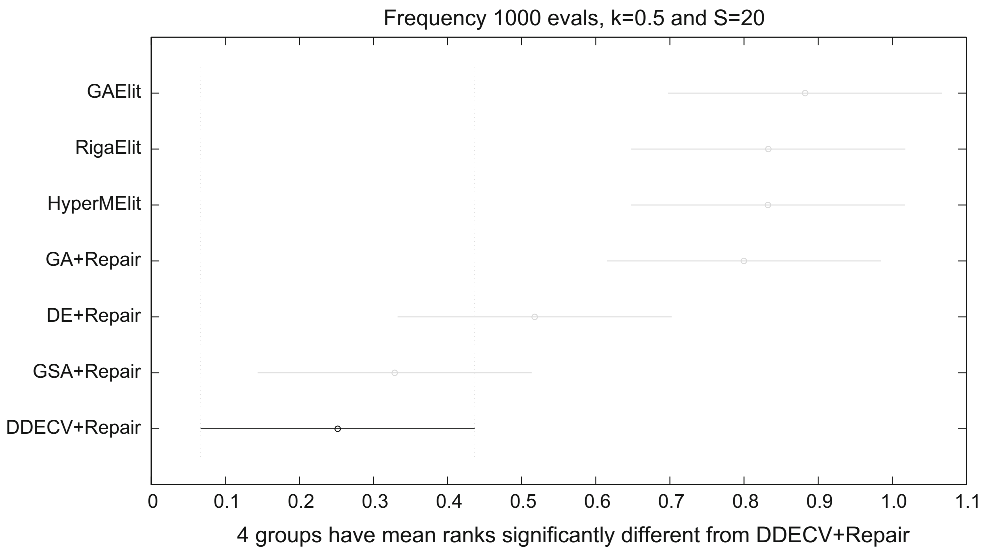


Fig. 3 Bonferroni–Dunn post hoc test results based on the offline error values obtained by DDECV + Repair and the compared algorithms considering 1000 evaluations as a change frequency and a medium change severity ($k = 0.5$ and $S = 20$)

Table 13 95 % confidence Kruskal–Wallis test results on the offline error values in Table 9, obtained by DDECV+Repair, GAElit, RIGAElit, HyperMElit, GA + Repair DE + Repair and GSA + Repair with a change frequency of 1000 evaluations and a medium change severity ($k = 0.5$ and $S = 20$)

GAElit (1)	RIGAElit (2)	HyperMElit (3)	GA + Repair (4)	DE + Repair (5)	GSA + Repair (6)	DDECV + Repair (7)
$6^{(-)}$ and $7^{(-)}$	$6^{(-)}$ and $7^{(-)}$	$6^{(-)}$ and $7^{(-)}$	$6^{(-)}$ and $7^{(-)}$		$1^{(+)}, 2^{(+)}, 3^{(+)}, 4^{(+)}$	$1^{(+)}, 2^{(+)}, 3^{(+)}, 4^{(+)}$

"$X^{(+)}$" means that the algorithm in the corresponding column outperformed algorithm X. "$X^{(-)}$" means that the algorithm in the corresponding column was outperformed by algorithm X. If algorithm X does not appear in column, Y means no significant differences between X and Y

the severity is low. In that case, both algorithms performed in a similar way. The results of the Bonferroni–Dunn post hoc test in Fig. 4a, b uphold that observation.

The second experiment provided the following research findings:

- DDECV + Repair was robust to different change frequencies. However, when the change frequency took medium values (i.e., 1000 evaluations), just comparable results were obtained with respect to two approaches with repair methods in their elements, DE + Repair and GSA + Repair.
- DDECV + Repair was not sensitive to low and high change severities in both, the objective function and the constraints of a DCOP.

4.2.3 *Experiment 3: Change detection, recovery and diversity analysis*

The third experiment studied the ability of DDECV + Repair to detect a change in the objective function and/or the constraints and its capacity to recover after it. Moreover, its diversity handling (feasible and infeasible solutions in the population) was also analyzed.

In Table 17, the average and standard deviation of the number of successful changes detected per each dynamic test problem in 50 independent runs by DDECV + Repair and HyperMElit are presented (static test problems were not considered in this experiment). Due to the fact that, among the compared algorithms, only HyperMElit uses a different detection mechanism (best objective function value decrease), it was selected for comparison purposes. The change frequency was set to 1000 evaluations, and the change severity was medium ($k = 0.5$ and $S = 20$).

Recalling that the total number of changes in a single run is 10, the results suggest that the re-evaluation of solutions used by DDECV + Repair is more effective with respect to the objective function value decrease when solving DCOPs (e.g., in G24_3 test problem HyperMElit was unable to detect changes in the dynamic constraints because the objective function is static). Furthermore, the feasible global optimum switches between disconnected regions. This performance was observed with the other change frequencies and severities.

Table 14 Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 1000 evaluations and a low change severity ($k = 0.25$ and $S = 10$)

Algorithms	Functions		
	G24_u	G24_1	G24_f
GAElit	0.101 (±0.021)	0.319 (±0.06)	0.113 (±0.06)
RIGAElit	0.155 (±0.018)	0.246 (±0.043)	0.119 (±0.039)
HyperMelit	0.094 (±0.016)	0.273 (±0.047)	0.119 (±0.05)
GA + Repair	0.348 (±0.055)	0.067 (±0.011)	0.034 (±0.009)
DDECV + Repair	**0.027 (±0.005)**	**0.052 (±0.013)**	**0.023 (±0.006)**
	G24_uf	G24_2	G24_2u
GAElit	0.035 (±0.011)	0.157 (±0.03)	0.05 (±0.013)
RIGAElit	0.049 (±0.011)	0.152 (±0.025)	0.073 (±0.013)
HyperMelit	0.033 (±0.012)	0.151 (±0.029)	0.051 (±0.013)
GA + Repair	0.157 (±0.033)	0.141 (±0.023)	0.237 (±0.032)
DDECV + Repair	**0.01 (±0.004)**	**0.047 (±0.006)**	**0.022 (±0.001)**
	G24_3	G24_3b	G24_3f
GAElit	0.18 (±0.047)	0.313 (±0.038)	0.101 (±0.05)
RIGAElit	0.212 (±0.033)	0.301 (±0.041)	0.115 (±0.032)
HyperMelit	0.179 (±0.037)	0.303 (±0.059)	0.095 (±0.029)
GA + Repair	**0.015 (±0.003)**	**0.038 (±0.008)**	0.018 (±0.004)
DDECV + Repair	0.057 (±0.005)	0.076 (±0.012)	**0.011 (±0.003)**
	G24_4	G24_5	G24_6a
GAElit	0.367 (±0.06)	0.192 (±0.033)	0.652 (±0.104)
RIGAElit	0.344 (±0.025)	0.191 (±0.023)	0.358 (±0.029)
HyperMelit	0.33 (±0.051)	0.19 (±0.018)	0.452 (±0.047)
GA + Repair	**0.062 (±0.009)**	0.131 (±0.024)	0.442 (±0.064)
DDECV + Repair	0.077 (±0.012)	**0.075 (±0.012)**	**0.035 (±0.006)**
	G24_6b	G24_6c	G24_6d
GAElit	0.427 (±0.035)	0.435 (±0.031)	0.504 (±0.052)
RIGAElit	0.325 (±0.025)	0.326 (±0.021)	0.344 (±0.031)
HyperMelit	0.374 (±0.024)	0.392 (±0.039)	0.427 (±0.045)
GA + Repair	0.428 (±0.047)	0.409 (±0.044)	0.366 (±0.039)
DDECV + Repair	**0.043 (±0.01)**	**0.038 (±0.007)**	**0.083 (±0.011)**
	G24_7	G24_8a	G24_8b
GAElit	0.259 (±0.042)	**0.164 (±0.021)**	0.427 (±0.063)
RIGAElit	0.357 (±0.048)	0.271 (±0.023)	0.426 (±0.058)
HyperMelit	0.27 (±0.049)	0.171 (±0.018)	0.388 (±0.05)
GA + Repair	**0.063 (±0.015)**	0.191 (±0.031)	0.113 (±0.027)
DDECV + Repair	0.362 (±0.083)	0.2 (±0.02)	**0.07 (±0.015)**

Best results are remarked in boldface

After getting some knowledge about how effective is DDECV + Repair to detect a change, Figs. 5a–c and 6a, b depict the recovery rate (RR) and absolute recovery rate (ARR) diagrams of the following algorithms: GAElit, RIGAElit, HyperMElit, GA + Repair and DDECV + Repair, all of them with five change frequencies (250, 500, 1000, 2000 and 4000 evaluations) and a medium change severity ($k = 0.5$ and $S = 20$). Such results indicate that DDECV + Repair recovers faster than the compared algorithms, regardless of the change frequency. The same behavior is observed

Table 15 Average and standard deviation offline error values obtained by DDECV + Repair and the compared algorithms with a change frequency of 1000 evaluations and a high change severity ($k = 1.0$ and $S = 50$)

Algorithms	Functions		
	G24_u	G24_1	G24_f
GAElit	0.106 (±0.02)	0.574 (±0.114)	0.265 (±0.146)
RIGAElit	0.142 (±0.031)	0.481 (±0.044)	0.231 (±0.06)
HyperMelit	0.111 (±0.031)	0.502 (±0.069)	0.221 (±0.094)
GA + Repair	0.375 (±0.068)	0.222 (±0.045)	0.055 (±0.022)
DDECV + Repair	**0.082 (±0.013)**	**0.086 (±0.012)**	**0.022 (±0.007)**
	G24_uf	G24_2	G24_2u
GAElit	0.093 (±0.039)	0.474 (±0.094)	0.186 (±0.048)
RIGAElit	0.101 (±0.032)	0.392 (±0.06)	0.266 (±0.046)
HyperMelit	0.09 (±0.032)	0.428 (±0.077)	0.187 (±0.058)
GA + Repair	0.309 (±0.095)	0.441 (±0.068)	0.633 (±0.1)
DDECV + Repair	**0.009 (±0.004)**	**0.056 (±0.006)**	**0.037 (±0.001)**
	G24_3	G24_3b	G24_3f
GAElit	0.324 (±0.088)	0.615 (±0.114)	0.201 (±0.099)
RIGAElit	0.413 (±0.095)	0.482 (±0.07)	0.237 (±0.06)
HyperMelit	0.318 (±0.071)	0.54 (±0.097)	0.227 (±0.06)
GA + Repair	0.056 (±0.012)	0.17 (±0.041)	0.025 (±0.011)
DDECV + Repair	**0.026 (±0.003)**	**0.096 (±0.008)**	**0.011 (±0.003)**
	G24_4	G24_5	G24_6a
GAElit	0.62 (±0.06)	0.673 (±0.079)	0.719 (±0.183)
RIGAElit	0.528 (±0.098)	0.592 (±0.062)	0.353 (±0.052)
HyperMelit	0.55 (±0.098)	0.607 (±0.047)	0.44 (±0.074)
GA + Repair	0.102 (±0.046)	0.408 (±0.048)	0.452 (±0.093)
DDECV + Repair	**0.096 (±0.007)**	**0.087 (±0.01)**	**0.033 (±0.007)**
	G24_6b	G24_6c	G24_6d
GAElit	0.411 (±0.047)	0.491 (±0.066)	0.495 (±0.076)
RIGAElit	0.284 (±0.036)	0.307 (±0.034)	0.369 (±0.082)
HyperMelit	0.347 (±0.041)	0.357 (±0.046)	0.426 (±0.056)
GA + Repair	0.432 (±0.071)	0.438 (±0.059)	0.387 (±0.083)
DDECV + Repair	**0.044 (±0.01)**	**0.04 (±0.008)**	**0.082 (±0.008)**
	G24_7	G24_8a	G24_8b
GAElit	0.378 (±0.109)	0.364 (±0.037)	1.069 (±0.108)
RIGAElit	0.487 (±0.109)	0.339 (±0.034)	0.938 (±0.106)
HyperMelit	0.387 (±0.06)	0.383 (±0.039)	0.94 (±0.135)
GA + Repair	**0.09 (±0.025)**	0.384 (±0.047)	0.511 (±0.069)
DDECV + Repair	0.106 (±0.015)	**0.075 (±0.013)**	**0.111 (±0.016)**

Best results are remarked in boldface

in Fig. 7a, b, where a change frequency of 1000 evaluations coupled with a low change severity ($k = 0.25$ and $S = 10$) and a high change severity ($k = 1.0$ and $S = 50$), respectively, was considered. It was noted that the recovery ability showed by DDECV + Repair, as those of the compared algorithms, seems to be more affected by a fast change (i.e., 250 evaluations in Fig. 5a) than a high change severity (i.e., $k = 1.0$ and $S = 50$ in Fig. 7b).

Finally, for DDECV + Repair and the compared algorithms, the diversity handling, in the context of a constrained

Table 16 95 % confidence Kruskal–Wallis test results on the offline error values in Tables 14 and 15, obtained by DDECV + Repair, GAElit, RIGAElit, HyperMElit and GA+Repair with a low ($k = 0.25$ and $S = 10$) and a high ($k = 1.0$ and $S = 50$) change severities, both with 1000 evaluations as change frequency

The severity of change		Algorithms				
k	S	GAElit (1)	RIGAElit (2)	HyperMElit (3)	GA + Repair (4)	DDECV + Repair (5)
0.25	10	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$		$1^{(+)}$, $2^{(+)}$ and $3^{(+)}$
1.00	50	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$	$5^{(-)}$	$1^{(+)}$, $2^{(+)}$, $3^{(+)}$ and $4^{(+)}$

"$X^{(+)}$" means that the algorithm in the corresponding column outperformed algorithm X. "$X^{(-)}$" means that the algorithm in the corresponding column was outperformed by algorithm X. If algorithm X does not appear in column, Y means no significant differences between X and Y

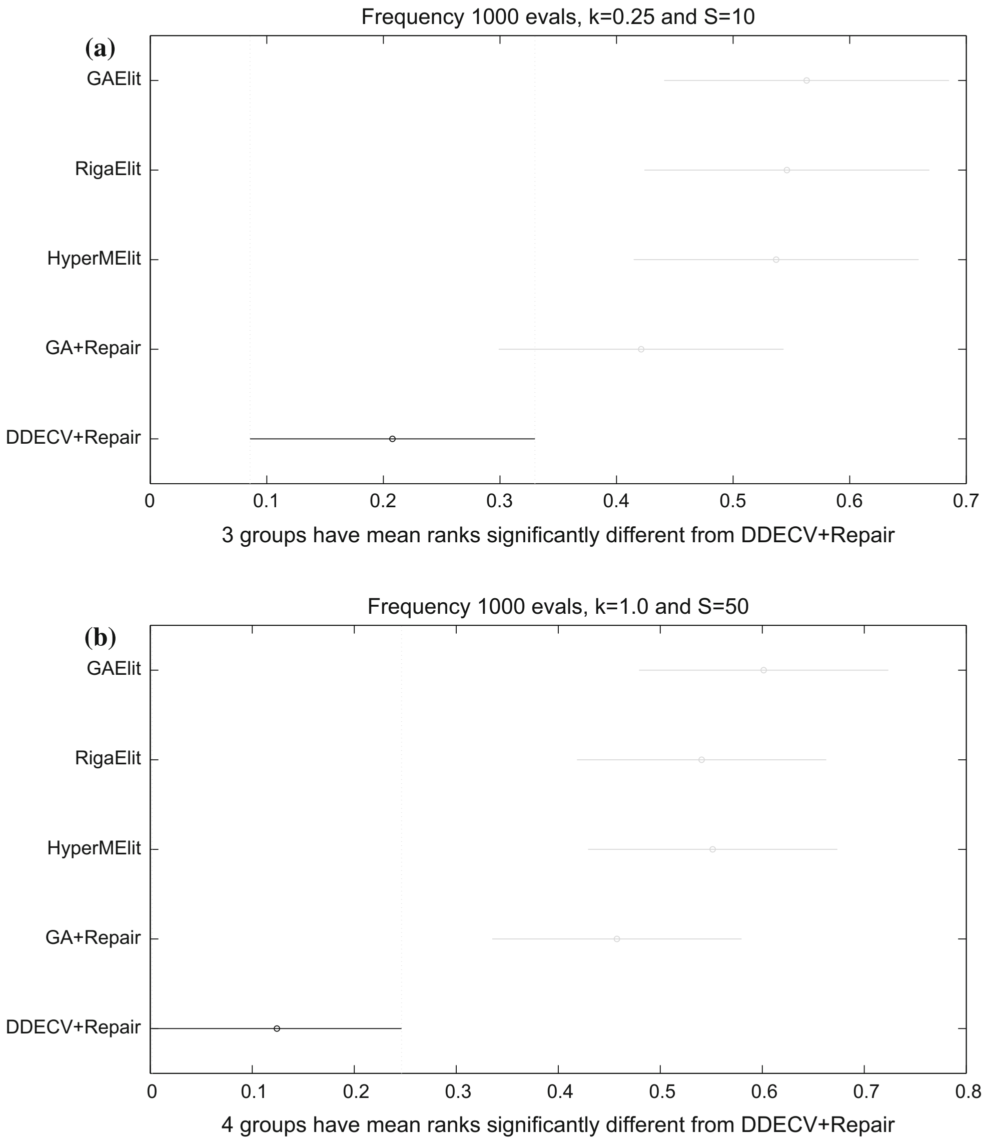


Fig. 4 Bonferroni–Dunn post hoc test results based on the offline error values obtained by DDECV + Repair and the compared algorithms with 1000 evaluations as the change frequency, considering two change severities: **a** low ($k = 0.25$ and $S = 10$) and **b** high ($k = 1.0$ and $S = 50$)

search space (i.e., the percentage of feasible and infeasible solutions in the population), was assessed by calculating the percentage of infeasible solutions in the population. The averages of infeasible solutions in the population considering the whole set of test problems with five different change frequencies (250, 500, 1000, 2000 and 4000 eval-

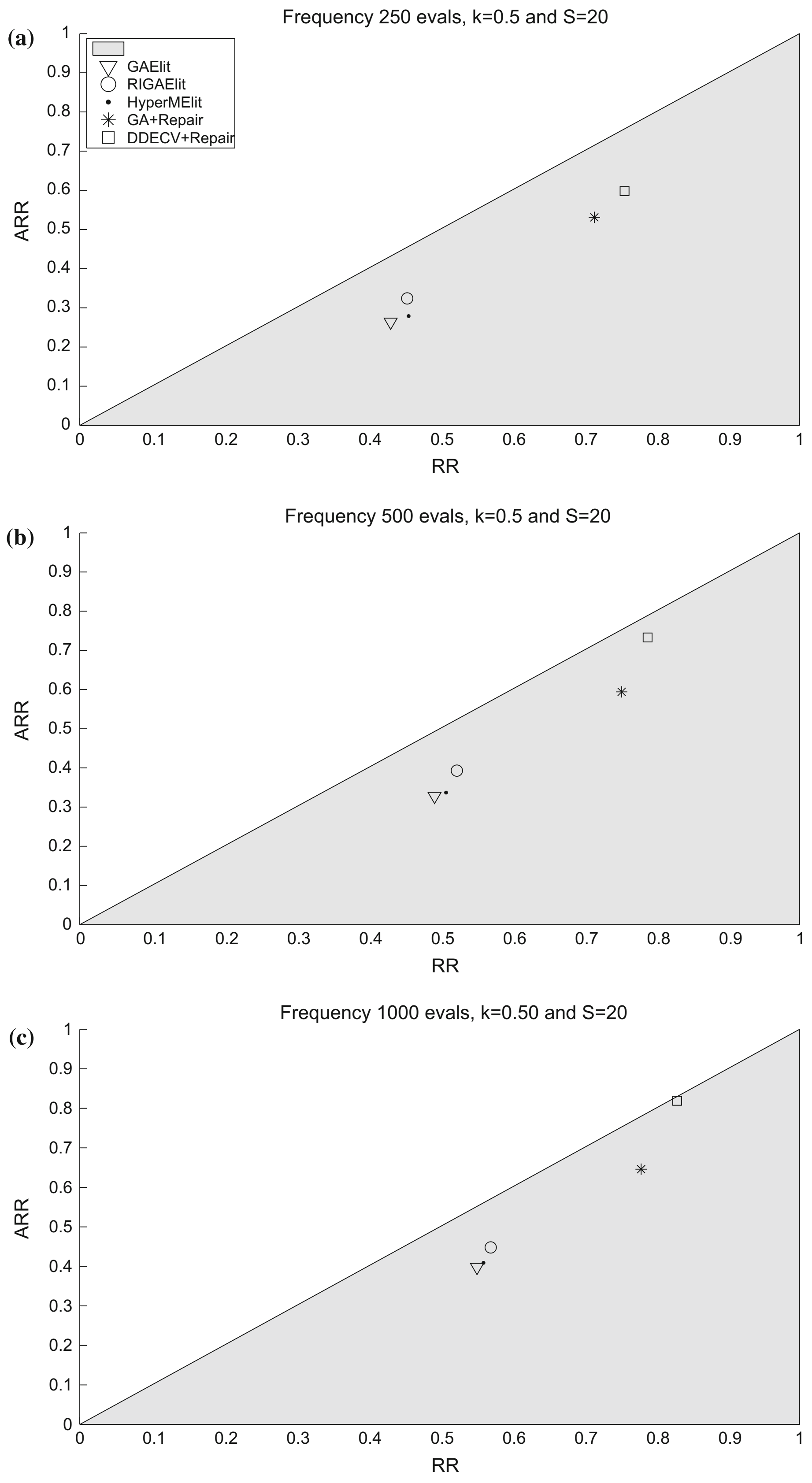


Fig. 5 Mapping of the RR/ARR scores of GAElit, RIGAElit, HyperMElit, GA + Repair and DDECV + Repair to the RR-ARR diagram for three change frequencies: **a** 250 evaluations, **b** 500 evaluations and **c** 1000 evaluations. If a point is closer to the right-hand side area of the graph, it indicates a faster recovery. Moreover, if the point lies on the *diagonal line*, the algorithm has been able to recover from the change and also converge to the new global optimum. The RR-ARR diagram was proposed in Nguyen and Yao (2012)

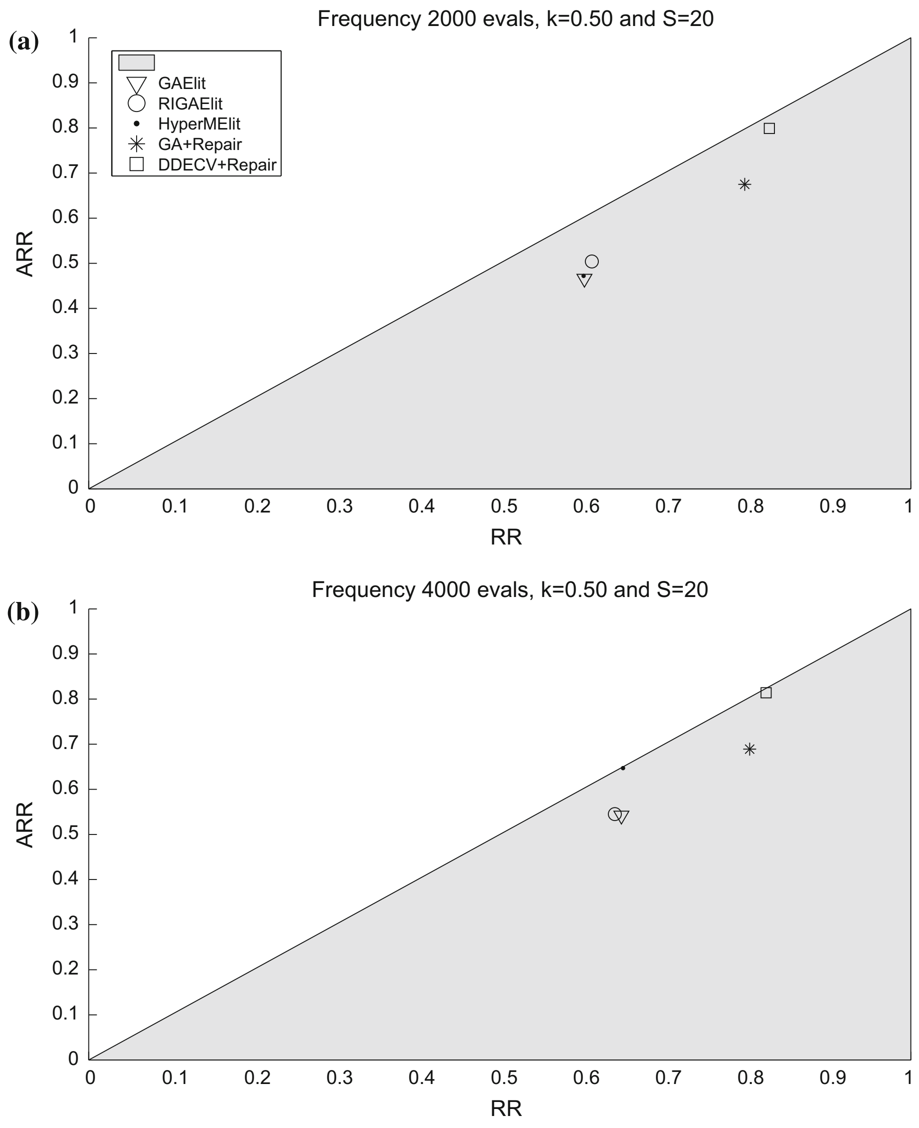


Fig. 6 Mapping of the RR/ARR scores of GAElit, RIGAElit, HyperMElit, GA + Repair and DDECV + Repair to the RR-ARR diagram for two change frequencies: **a** 2000 evaluations and **b** 4000 evaluations. If a point is closer to the right-hand side area of the graph, it indicates a faster recovery. Moreover, if the point lies on the *diagonal line*, the algorithm has been able to recover from the change and also converge to the new global optimum. The RR-ARR diagram was proposed in Nguyen and Yao (2012)

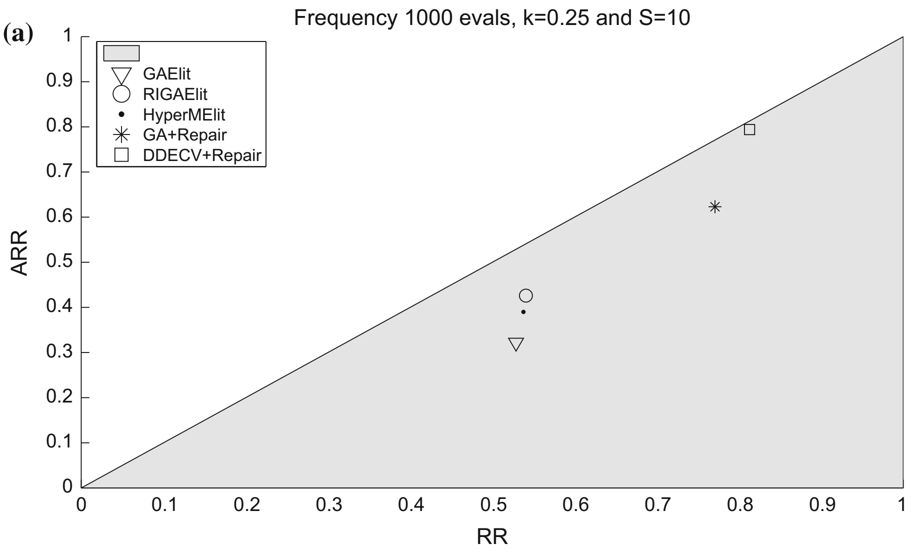
uations) and a medium change severity ($k = 0.5$ and $S = 20$) are shown in Table 18. The same information is presented in Table 19, with 1000 evaluations as the change frequency, but now considering a low ($k = 0.25$ and $S = 10$) and a high ($k = 1.0$ and $S = 50$) change severities.

In both tables, regardless of different change frequencies and severities, a similar behavior was observed, where the two algorithms with a repair method kept a lower percentage of feasible solutions than those approaches without it. Furthermore, it is worth noticing that, from both repair-based algorithms, GA + Repair and DDECV + Repair, the latter maintained a higher proportion of infeasible solutions (approximately between 8 and 14 %). Such ratio, considering the competitive overall results obtained by DDECV + Repair, seems to be suitable for an algorithm dealing with DCOPs. In fact, such finding agrees with the importance of keeping some feasible solutions to favor a successful search in the presence of constraints (Mezura-Montes and Coello 2011).

From this third experiment the following can be summarized:

- DDECV + Repair's change detection mechanism based on solution re-evaluation was particularly effective in the set of dynamic test problems solved in this study.
- DDECV + Repair provided the fastest recovery after a change. Moreover, it was found that faster changes affect this ability more than high severity changes.
- Similarly to static constrained optimization, keeping a low proportion of infeasible solutions (between 8 and 14 %), helps DDECV + Repair to provide competitive results when solving DCOPs.

(a)



(b)

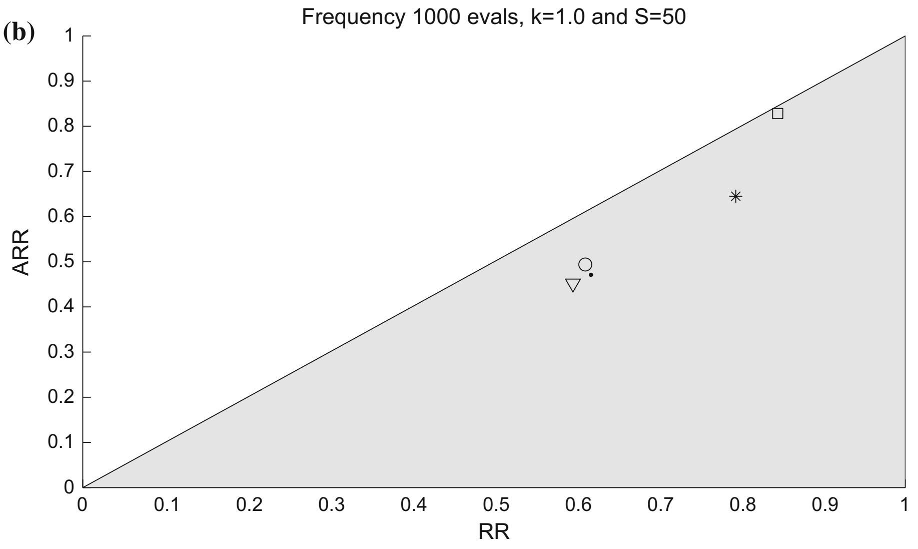


Fig. 7 Mapping of the RR/ARR scores of GAElit, RIGAElit, HyperMElit, GA + Repair and DDECV + Repair to the RR-ARR diagram with a change frequency of 1000 evaluations and: **a** a low change severity ($k = 0.25$ and $S = 10$) and **b** a high change severity ($k = 1.0$ and $S = 50$). If a point is closer to the right-hand side area of the graph, it indicates a faster recovery. Moreover, if the point lies on the *diagonal line*, the algorithm has been able to recover from the change and also converge to the new global optimum. The RR-ARR diagram was proposed in Nguyen and Yao (2012)

4.2.4 Experiment 4: Dynamism in different parts of the problem and best-error-before-change analysis

The fourth experiment studied the performance of DDECV + Repair with dynamism in different parts of the problem. ICHEA and DCTC algorithms, already described, were used for comparison purposes, and their results were taken from their corresponding documents (Sharma and Sharma 2012b; Aragón et al. 2013). The twelve out of eighteen test problems, as in Experiment 1, were solved.

The first comparison of this fourth experiment was between DDECV + Repair and ICHEA (Sharma and Sharma 2012a), and the Best-error-before-change measure was computed. The change frequency was 1000 evaluations, and the severity of the change was medium (i.e., $k = 0.50$ and $S = 20$). The results obtained are presented in Table 20. The statistical validation was made with the 95 % confidence Wilcoxon rank-sum test. Such test indicated no significant difference between DDECV + Repair and ICHEA algorithm by considering all the four test problems (g24_u, g24_1, g24_3 and g24_4). Therefore, DDECV + Repair has a similar performance with respect to ICHEA to reach the optimum solution before a change occurs.

The second comparison of this fourth experiment studied the effects of dynamism in different parts of the problem with several change frequencies and also different change severities in the performance provided by DDECV + Repair and DCTC (Aragón et al. 2013). The measure used in this comparison was the feasible offline error. The change frequencies were 250, 500 and 1000 evaluations and different change severities ($k = 0.25, 0.5$ and 1.0 and $S = 10, 20$ and 50). The statistical validation was made with the 95 %

Table 17 Average number of changes successfully detected by DDECV + Repair and HyperMElit with a change frequency of 1000 evaluations and a medium change severity ($k = 0.5$ and $S = 20$)

Functions	Change detection mechanism	
	HyperM	DDECV + Repair
g24_u	6.0 (±0.0)	10.0 (±0.0)
g24_1	6.0 (±0.0)	10.0 (±0.0)
g24_2	4.7 (±0.8)	10.0 (±0.0)
g24_2u	4.5 (±0.5)	10.0 (±0.0)
g24_3	0	10.0 (±0.0)
g24_3b	6.0 (±0.0)	10.0 (±0.0)
g24_4	5.9 (±0.3)	10.0 (±0.0)
g24_5	6.5 (±0.5)	10.0 (±0.0)
g24_6a	5.0 (±0.0)	10.0 (±0.0)
g24_6b	5.0 (±0.0)	10.0 (±0.0)
g24_6c	5.0 (±0.0)	10.0 (±0.0)
g24_6d	5.0 (±0.0)	10.0 (±0.0)
g24_7	9.3 (±0.7)	10.0 (±0.0)
g24_8a	9.0 (±0.9)	10.0 (±0.0)
g24_8b	7.3 (±0.9)	10.0 (±0.0)

Static problems were discarded in this comparison. The total number of changes is 10 for all test problems

confidence Wilcoxon rank-sum test. The comparisons were divided by the kind of test problems as:

1. Dynamic objective function and dynamic constraints. In Table 21 the mean and standard deviation values of the feasible offline error are shown for DDECV + Repair and DCTC in three test problems (g24_3b, g24_4 and g24_5) with different change frequencies (250, 500 and 1000 evaluations) and different change severities ($k = 0.25$, 0.5 and 1.0 and $S = 10, 20$ and 50). Twenty-seven combinations were carried out by each test problem. The statistical test indicated that there were significant differences. Based on such results, DDECV + Repair outperformed DCTC in all combinations of the two test problems (g24_3b and g24_4). On the other hand, DDECV + Repair outperformed DCTC in nineteen combinations of one test problem (g24_5), while DCTC was better in eight combinations regardless of the frequency change with low and medium values of the severities of change in the objective function and high values of severity in the constraints.
2. Dynamic objective function and static constraints. In Table 22 the mean and standard deviation values of the feasible offline error are presented for DDECV + Repair and DCTC in three test problems (g24_1, g24_2 and g24_8b) with different change frequencies (250, 500 and 1000 evaluations) and different change severities in the objective function ($k = 0.25, 0.50$ and 1.0). Nine combinations were computed by each test problem. The statistical test indicated that there were significant differences. According to those results, DCTC was better in all combinations of two test problems (g24_1 and g24_2) and also eight combinations of one test problem (g24_8b). On the other hand, DDECV + Repair surpassed the results on just one combination when the frequency change was 250 evaluations and the severity change was high ($S = 50$). In Table 23, the mean and standard deviation values of the feasible offline error are presented for DDECV + Repair and DCTC in three test problems (g24_6a, g24_6c and g24_6d) with different change frequencies (250, 500 and 1000 evaluations) without a change severity. The results suggest that DDECV + Repair performance was similar with respect to DCTC, because no significant differences were observed.
3. Static objective function and dynamic constraints. In Table 24 the mean and standard deviation values of the feasible offline error are presented for DDECV + Repair and DCTC in two test problems (g24_3 and g24_7) with different change frequencies (250, 500 and 1000 evaluations) and different change severities in the constraints ($S = 10$, 20 and 50). Nine combinations were calculated by each test problem. Such results indicated that DDECV + Repair performance was similar with respect to DCTC, because no significant differences were observed.

The fourth experiment generated the following conclusions:

– DDECV + Repair outperformed the compared algorithms where the test problems had dynamism in the objective function and constraints.
– In the test problems with a dynamic objective function and static constraints, DCTC outperformed DDECV + Repair.
– Similar results were obtained between DDECV + Repair and DCTC in test problems with a static objective function and dynamic constraints.

5 Conclusions and future work

This paper presented an empirical assessment of the dynamic differential evolution with combined variants plus a repair method (DDECV + Repair) in the solution of dynamic constrained optimization problems (DCOPs). Four unexplored issues of DDECV + Repair were addressed: (1) the role of the exploration promotion mechanism, the repair method, the random immigrants and the memory in its performance, (2) its sensitivity to different change frequencies and severities,

Table 18 Average of percentage of selected infeasible individuals with change frequency of 250, 500, 1000, 2000 and 4000 evaluations

Frequency	Algorithms				
	GAElit (%)	RIGAElit (%)	HyperMElit (%)	GA + Repair (%)	DDECV + Repair (%)
250	53.7	39.7	54.5	5.0	13.2
500	52.4	39.8	53.1	2.0	12.9
1000	52.3	39.5	52.7	1.0	10.7
2000	51.9	39.5	52.2	0.0	9.5
4000	52.3	39.4	52.2	0.0	8.8

The severity of change is medium ($k = 0.5$ and $S = 20$). Only constrained problems are included to calculate this measure

Table 19 Average of percentage of selected infeasible individuals with frequency at 1000 evaluations and the severity of change is small and large ($k = 0.25,\ 1.0$ and $S = 20,\ 50$)

The severity of change		Algorithms				
k	S	GAElit (%)	RIGAElit (%)	HyperMElit (%)	GA + Repair (%)	DDECV + Repair (%)
0.25	10	50.8	40.9	52.8	3.6	13.5
1.00	50	53.4	39.6	53.7	3.6	10.9

Only constrained problems are included to calculate this measure

Table 20 Average and standard deviation *Best-error-before-change* values obtained by DDECV + Repair and ICHEA with a change frequency of 1000 evaluations and a medium severity of change ($k = 0.50$ and $S = 20$)

Algorithms	Functions			
	G24_u	G24_1	G24_3	G24_4
ICHEA	0.0051 (±0.004)	0.0333 (±0.005)	0.0187 (±0.003)	0.0799 (±0.006)
DDECV + Repair	**0.000** (**±0.000**)	**0.012** (**±0.010**)	**0.004** (**±0.001**)	**0.008** (**±0.006**)

Best results are remarked in boldface

(3) its ability to detect and recover after a change, besides its diversity handling during the search, and (4) its performance to solve problems with dynamism in both (the objective function and constraints) or dynamism in only one of them. Using seven performance measures, eight algorithms found in the specialized literature and a recently proposed set of test problems, four experiments were designed.

From the obtained results, statistically validated, it was found that the repair method is the most relevant mechanism in DDECV + Repair, while the presence of the random immigrants is positive, but may affect the results in problems where the feasible global optimum is located at the boundaries of either the feasible region or the search space. The memory mechanism plays an important role in DDECV + Repair because the optimal solution sometimes returns to the locations near its previous regions. Moreover, the sensitivity of DDECV + Repair to different change frequencies and severities was low, and from those two sources of difficulty, faster changes may affect its good ability to recover after a change. Furthermore, the re-evaluation-based change detection mechanism proved to be very effective. Moreover, it was showed that, as in the case of static constrained optimization, keeping a low ratio of feasible solutions during the search helps DDECV + Repair to reach competitive results in the resolution of DCOPs. Finally, DDECV + Repair had a better performance solving test problems with dynamism in both (the objective function and constraints) and a competitive performance in problems with dynamism in only one of them.

Part of the future work includes testing DDECV + Repair with random change frequencies and severities. Also, the immigrants technique will be revisited so as to analyze the negative effect of a feasible global optimum at the boundaries of either the search space or the feasible region. Furthermore, other modifications will be designed in order to improve the performance of DDECV + Repair to deal dynamism in the objective function and constraints separately. Finally, other test problems with smaller feasible regions will be sought so as to test the repair method in those situations.

Table 21 Average and standard deviation *Feasible offline error* values obtained by DDECV + Repair and DCTC with different change frequencies (250, 500 and 1000 evaluations) and several severities of change ($k = 0.25$, 0.5 and 1.0 and $S = 10$, 20 and 50), for test problems with dynamic objective function and dynamic constraints

Evals	Algorithms	Functions			Functions			Functions		
		G24_3b	G24_4	G24_5	G24_3b	G24_4	G24_5	G24_3b	G24_4	G24_5
	$k = 0.25$		$S = 10$			$S = 20$			$S = 50$	
250	DCTC	0.59 (±0.15)	0.43 (±0.07)	**0.21 (±0.02)**	0.54 (±0.15)	0.41 (±0.05)	0.18 (±0.02)	0.56 (±0.14)	0.28 (±0.04)	**0.11 (±0.02)**
	DDECV + Repair	**0.227 (±0.042)**	**0.222 (±0.04)**	0.235 (±0.027)	**0.213 (±0.018)**	**0.207 (±0.033)**	**0.18 (±0.026)**	**0.174 (±0.019)**	**0.172 (±0.019)**	0.171 (±0.028)
500	DCTC	0.54 (±0.16)	0.36 (±0.05)	0.18 (±0.01)	0.49 (±0.13)	0.35 (±0.02)	0.15 (±0.01)	0.49 (±0.1)	0.23 (±0.03)	**0.07 (±0.01)**
	DDECV + Repair	**0.138 (±0.02)**	**0.138 (±0.022)**	**0.143 (±0.022)**	**0.133 (±0.018)**	**0.131 (±0.019)**	**0.116 (±0.02)**	**0.094 (±0.013)**	**0.098 (±0.012)**	0.101 (±0.019)
1000	DCTC	0.47 (±0.12)	0.32 (±0.02)	0.17 (±0.02)	0.43 (±0.08)	0.32 (±0.02)	0.12 (±0.02)	0.39 (±0.06)	0.19 (±0.01)	**0.06 (±0.01)**
	DDECV + Repair	**0.083 (±0.014)**	**0.081 (±0.012)**	**0.096 (±0.014)**	**0.08 (±0.01)**	**0.081 (±0.011)**	**0.074 (±0.011)**	**0.053 (±0.008)**	**0.054 (±0.003)**	0.064 (±0.007)
	$k = 0.5$		$S = 10$			$S = 20$			$S = 50$	
250	DCTC	0.55 (±0.13)	0.71 (±0.08)	0.34 (±0.04)	0.57 (±0.13)	0.62 (±0.05)	0.31 (±0.04)	1.15 (±0.13)	0.28 (±0.06)	**0.12 (±0.04)**
	DDECV + Repair	**0.292 (±0.038)**	**0.3 (±0.035)**	**0.231 (±0.028)**	**0.291 (±0.033)**	**0.288 (±0.027)**	**0.248 (±0.027)**	**0.249 (±0.022)**	**0.244 (±0.021)**	0.262 (±0.027)
500	DCTC	0.49 (±0.14)	0.63 (±0.05)	0.28 (±0.02)	0.51 (±0.11)	0.55 (±0.04)	0.26 (±0.03)	1.03 (±0.1)	0.2 (±0.04)	**0.07 (±0.03)**
	DDECV + Repair	**0.159 (±0.023)**	**0.161 (±0.021)**	**0.143 (±0.02)**	**0.15 (±0.015)**	**0.149 (±0.015)**	**0.148 (±0.018)**	**0.117 (±0.01)**	**0.119 (±0.01)**	0.145 (±0.018)
1000	DCTC	0.41 (±0.12)	0.57 (±0.02)	0.25 (±0.01)	0.45 (±0.09)	0.5 (±0.02)	0.23 (±0.01)	0.98 (±0.09)	0.15 (±0.02)	**0.03 (±0.02)**
	DDECV + Repair	**0.087 (±0.016)**	**0.086 (±0.014)**	**0.087 (±0.014)**	**0.088 (±0.01)**	**0.089 (±0.01)**	**0.082 (±0.01)**	**0.061 (±0.006)**	**0.063 (±0.005)**	0.082 (±0.013)
	$k = 1.0$		$S = 10$			$S = 20$			$S = 50$	
250	DCTC	1.67 (±0.26)	1.53 (±0.13)	0.46 (±0.08)	1.09 (±0.29)	1.33 (±0.09)	**0.28 (±0.11)**	1.68 (±0.2)	1.33 (±0.11)	0.31 (±0.09)
	DDECV + Repair	**0.458 (±0.041)**	**0.465 (±0.04)**	**0.278 (±0.026)**	**0.46 (±0.03)**	**0.474 (±0.037)**	0.295 (±0.029)	**0.421 (±0.026)**	**0.42 (±0.026)**	**0.308 (±0.031)**
500	DCTC	1.59 (±0.23)	1.41 (±0.06)	0.38 (±0.04)	0.93 (±0.18)	1.26 (±0.08)	0.2 (±0.05)	1.54 (±0.16)	1.2 (±0.07)	0.25 (±0.07)
	DDECV + Repair	**0.236 (±0.015)**	**0.232 (±0.021)**	**0.16 (±0.018)**	**0.225 (±0.013)**	**0.225 (±0.014)**	**0.179 (±0.022)**	**0.195 (±0.014)**	**0.195 (±0.014)**	**0.184 (±0.019)**
1000	DCTC	1.44 (±0.18)	1.36 (±0.05)	0.34 (±0.03)	0.83 (±0.14)	1.17 (±0.05)	0.15 (±0.03)	1.45 (±0.08)	1.13 (±0.05)	0.2 (±0.03)
	DDECV + Repair	**0.108 (±0.011)**	**0.113 (±0.011)**	**0.09 (±0.011)**	**0.093 (±0.008)**	**0.094 (±0.008)**	**0.098 (±0.011)**	**0.093 (±0.008)**	**0.094 (±0.008)**	**0.098 (±0.011)**

Best results are remarked in boldface

Table 22 Average and standard deviation *Feasible offline error* values obtained by DDECV + Repair and DCTC with different change frequencies (250, 500 and 1000 evaluations) and several severities of change ($k = 0.25$, 0.5 and 1.0), for test problems with dynamic objective function and static constraints

Evals	Algorithms	Functions			Functions			Functions		
		G24_1	G24_2	G24_8b	G24_1	G24_2	G24_8b	G24_1	G24_2	G24_8b
			$k = 0.25$			$k = 0.50$			$k = 1.0$	
250	DCTC	**0.03 (±0.01)**	**0.08 (±0.03)**	**0.11 (±0.04)**	**0.05 (±0.03)**	**0.12 (±0.03)**	**0.25 (±0.08)**	**0.12 (±0.04)**	**0.18 (±0.1)**	0.58 (±0.19)
	DDECV + Repair	0.186 (±0.034)	0.159 (±0.016)	0.217 (±0.029)	0.27 (±0.032)	0.164 (±0.017)	0.377 (±0.045)	0.46 (±0.54)	0.252 (±0.022)	**0.532 (±0.065)**
500	DCTC	**0.0 (±0.0)**	**0.05 (±0.02)**	**0.04 (±0.02)**	**0.01 (±0.01)**	**0.06 (±0.03)**	**0.12 (±0.06)**	**0.03 (±0.02)**	**0.12 (±0.07)**	**0.29 (±0.13)**
	DDECV + Repair	0.113 (±0.028)	0.099 (±0.012)	0.127 (±0.024)	0.132 (±0.02)	0.131 (±0.014)	0.188 (±0.027)	0.215 (±0.026)	0.129 (±0.015)	0.297 (±0.048)
1000	DCTC	**0.0 (±0.0)**	**0.03 (±0.01)**	**0.01 (±0.01)**	**0.0 (±0.01)**	**0.03 (±0.03)**	**0.03 (±0.06)**	**0.0 (±0.0)**	**0.04 (±0.04)**	**0.07 (±0.07)**
	DDECV + Repair	0.058 (±0.012)	0.049 (±0.006)	0.069 (±0.018)	0.066 (±0.015)	0.063 (±0.012)	0.09 (±0.028)	0.084 (±0.011)	0.061 (±0.008)	0.12 (±0.019)

Best results are remarked in boldface

Table 23 Average and standard deviation *Feasible offline error* values obtained by DDECV + Repair and DCTC with different change frequencies (250, 500 and 1000 evaluations), for test problems with dynamic objective function and static constraints

Evals	Algorithms	Functions		
		G24_6a	G24_6c	G24_6d
250	DCTC	0.26 (±0.38)	**0.12 (±0.05)**	**0.14 (±0.18)**
	DDECV + Repair	**0.183 (±0.018)**	0.214 (±0.031)	0.424 (±0.03)
500	DCTC	**0.06 (±0.12)**	**0.06 (±0.03)**	**0.04 (±0.14)**
	DDECV + Repair	0.082 (±0.01)	0.095 (±0.015)	0.186 (±0.016)
1000	DCTC	**0.02 (±0.02)**	**0.04 (±0.03)**	**0.0 (±0.0)**
	DDECV + Repair	0.035 (±0.007)	0.041 (±0.008)	0.079 (±0.009)

Best results are remarked in boldface

Table 24 Average and standard deviation *Feasible offline error* values obtained by DDECV + Repair and DCTC with different change frequencies (250, 500 and 1000 evaluations) and several severities of change ($S = 10, 20$ and 50), for test problems with static objective function and dynamic constraints

Evals	Algorithms	Functions		Functions		Functions	
		G24_3	G24_7	G24_3	G24_7	G24_3	G24_7
		$S = 10$		$S = 20$		$S = 50$	
250	DCTC	0.16 (±0.15)	**0.15 (±0.02)**	0.15 (±0.21)	**0.11 (±0.03)**	0.12 (±0.07)	**0.1 (±0.03)**
	DDECV + Repair	**0.130 (±0.019)**	0.240 (±0.038)	**0.105 (±0.018)**	0.16 (±0.03)	**0.072 (±0.008)**	0.124 (±0.032)
500	DCTC	0.13 (±0.14)	**0.12 (±0.02)**	0.1 (±0.13)	**0.07 (±0.02)**	0.1 (±0.11)	**0.06 (±0.02)**
	DDECV + Repair	**0.083 (±0.01)**	0.175 (±0.018)	**0.063 (±0.009)**	0.15 (±0.022)	**0.041 (±0.008)**	0.112 (±0.025)
1000	DCTC	0.11 (±0.03)	**0.1 (±0.02)**	0.05 (±0.03)	**0.05 (±0.01)**	0.05 (±0.04)	**0.04 (±0.01)**
	DDECV + Repair	**0.059 (±0.005)**	0.11 (±0.015)	**0.044 (±0.008)**	0.115 (±0.015)	**0.026 (±0.004)**	0.108 (±0.013)

Best results are remarked in boldface

Acknowledgements The first author acknowledges support from the Mexican National Council of Science and Technology (CONACyT) through a scholarship to pursue graduate studies at the University of Veracruz. The second author acknowledges support from CONACyT through Project No. 220522. This study was funded by the Mexican National Council of Science and Technology CONACyT (Grant No. 220522).

Compliance with ethical standards

Conflict of interest María-Yaneli Ameca-Alducin declares that she has no conflict of interest. Efrén Mezura-Montes declares that he has no conflict of interest. Nicandro Cruz-Ramírez declares that he has no conflict of interest.

Human and animal rights This article does not contain any studies with human participants or animals performed by any of the authors.

References

Ameca-Alducin MY, Mezura-Montes E, Cruz-Ramirez N (2014) Differential evolution with combined variants for dynamic constrained optimization. In: Evolutionary computation (CEC), 2014 IEEE congress on, pp 975–982. doi:10.1109/CEC.2014.6900629

Ameca-Alducin MY, Mezura-Montes E, Cruz-Ramírez N (2015a) Differential evolution with a repair method to solve dynamic constrained optimization problems. In: Proceedings of the companion publication of the 2015 on genetic and evolutionary computation conference. ACM, New York, GECCO companion '15, pp 1169–1172. doi:10.1145/2739482.2768471

Ameca-Alducin MY, Mezura-Montes E, Cruz-Ramírez N (2015b) A repair method for differential evolution with combined variants to solve dynamic constrained optimization problems. In: Proceedings of the 2015 on genetic and evolutionary computation conference. ACM, New York, GECCO '15, pp 241–248. doi:10.1145/2739480.2754786

Aragón V, Esquivel S, Coello C (2013) Artificial immune system for solving dynamic constrained optimization problems. In: Alba E, Nakib A, Siarry P (eds) Metaheuristics for dynamic optimization, studies in computational intelligence, vol 433. Springer, Berlin, pp 225–263. doi:10.1007/978-3-642-30665-5_11

Azzouz R, Bechikh S, Said LB (2015) A dynamic multi-objective evolutionary algorithm using a change severity-based adaptive population management strategy. Soft Comput. doi:10.1007/s00500-015-1820-4

Branke J, Schmeck H (2003) Designing evolutionary algorithms for dynamic optimization problems. In: Ghosh A, Tsutsui S (eds) Advances in evolutionary computing, Natural Computing Series. Springer, Berlin, pp 239–262. doi:10.1007/978-3-642-18965-4_9

Bu C, Luo W, Yue L (2016) Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies. IEEE Trans Evol Comput. doi:10.1109/TEVC.2016.2567644

Cobb H (1990) 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

Cobb H, Grefenstette J (1993) Genetic algorithms for tracking changing environments. In: Forrest S (ed) ICGA. Morgan Kaufmann, Los Altos, pp 523–530

Coello Coello CA (2002) Theoretical and numerical constraint handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Methods Appl Mech Eng 191(11–12):1245–1287

du Plessis M (2012) Adaptive multi-population differential evolution for dynamic environments, Ph.D. thesis. Faculty of Engineering, Built Environment and Information Technology, University of Pretoria

Deb K (2000) An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng 186(24):311–338

Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput 1(1):3–18. doi:10.1016/j.swevo.2011.02.002

Filipiak P, Lipinski P (2014) Univariate marginal distribution algorithm with Markov chain predictor in continuous dynamic environments. Springer, Cham, pp 404–411

Grefenstette J (1992) Genetic algorithms for changing environments. In: Parallel problem solving from nature 2. Elsevier, Amsterdam, pp 137–144

Jiang S, Yang S (2016) Evolutionary dynamic multiobjective optimization: benchmarks and algorithm comparisons. IEEE Trans Cybern. doi:10.1109/TCYB.2015.2510698

Li C, Yang S, Yang M (2014) An adaptive multi-swarm optimizer for dynamic optimization problems. Evol Comput 22(4):559–594

Liu R, Chen Y, Ma W, Mu C, Jiao L (2014b) A novel cooperative coevolutionary dynamic multi-objective optimization algorithm using a new predictive model. Soft Comput 18(10):1913–1929

Liu R, Chen Y, Ma W, Mu C, Jiao L (2014b) A novel cooperative coevolutionary dynamic multi-objective optimization algorithm using a new predictive model. Soft Comput 18(10):1913–1929

López-Ibáñez M, Stützle T (2012) Automatically improving the anytime behaviour of optimisation algorithms, Technical Report. TR/IRIDIA/2012-012, IRIDIA, Université Libre de Bruxelles, Belgium, published in European Journal of Operations Research Radulescu et al. (2013)

Martínez-Peñaloza MG, Mezura-Montes E (2015) Immune generalized differential evolution for dynamic multiobjective optimization problems. In: 2015 IEEE Congress on evolutionary computation (CEC), pp 1918–1925. doi:10.1109/CEC.2015.7257120

Mezura-Montes E (ed) (2009) Constraint-handling in evolutionary optimization, studies in computational intelligence, vol 198. Springer, Berlin

Mezura-Montes E, Coello CAC (2011) Constraint-handling in nature-inspired numerical optimization: past, present and future. Swarm Evol Comput 1(4):173–194

Mezura-Montes E, Miranda-Varela ME, del Carmen Gómez-Ramón R (2010) Differential evolution in constrained numerical optimization. An empirical study. Inf Sci 180(22):4223–4262

Michalewicz Z, Nazhiyath G (1995) Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In: Evolutionary computation, IEEE international conference on, vol 2, pp 647–651. doi:10.1109/ICEC.1995.487460

Michalewicz Z, Schoenauer M (1996) Evolutionary algorithms for constrained parameter optimization problems. Evol Comput 4(1):1–32

Mukherjee R, Debchoudhury S, Swagatam D (2016) Modified differential evolution with locality induced genetic operators for dynamic optimization. Eur J Oper Res 253(2):337–355

Nguyen TT, Yao X (2009) Benchmarking and solving dynamic constrained problems. In: Evolutionary computation, 2009. CEC '09. IEEE congress on, pp 690–697. doi:10.1109/CEC.2009.4983012

Nguyen T, Yao X (2010) Detailed experimental results of GA, RIGA, HYPERm and GA + Repair on the G24 set of benchmark problems. Technical report, School Computer Science, University of Birmingham, Birmingham. http://www.staff.livjm.ac.uk/enrtngu1/Papers/DCOPfulldata

Nguyen T, Yao X (2012) Continuous dynamic constrained optimization: the challenges. IEEE Trans Evol Comput 16(6):769–786. doi:10.1109/TEVC.2011.2180533

Nguyen T, Yao X (2013) Evolutionary optimization on continuous dynamic constrained problems—an analysis. In: Yang S, Yao X (eds) Evolutionary computation for dynamic optimization problems, studies in computational intelligence, vol 490. Springer, Berlin, pp 193–217. doi:10.1007/978-3-642-38416-5_8

Nguyen T, Yang S, Branke J (2012) Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evol Comput 6:1–24

Nguyen TT, Yang S, Branke J, Yao X (2013) chap Evolutionary dynamic optimization: methodologies. In: Evolutionary computation for dynamic optimization problems. Springer, Berlin, pp 39–64

Pal K, Saha C, Das S (2013a) Differential evolution and offspring repair method based dynamic constrained optimization. In: Panigrahi B, Suganthan P, Das S, Dash S (eds) Swarm, evolutionary, and memetic computing, Lecture notes in Computer Science, vol 8297. Springer, Berlin, pp 298–309. doi:10.1007/978-3-319-03753-0_27

Pal K, Saha C, Das S, Coello-Coello C (2013b) Dynamic constrained optimization with offspring repair based gravitational search algorithm. In: Evolutionary computation (CEC), 2013 IEEE congress on, pp 2414–2421. doi:10.1109/CEC.2013.6557858

Pekdemir H, Topcuoglu HR (2016) Enhancing fireworks algorithms for dynamic optimization problems. In: 2016 IEEE congress on evolutionary computation (CEC), pp 4045–4052

Price K, Storn R, Lampinen J (2005) Differential evolution: a practical approach to global optimization (Natural Computing Series). Springer, Secaucus

Radulescu A, López-Ibáñez M, Stützle T (2013) Automatically improving the anytime behaviour of multiobjective evolutionary algorithms. In: Purshouse R, Fleming P, Fonseca CM, Greco S, Shaw J (eds) Evolutionary multi-criterion optimization, Lecture notes in Computer Science, vol 7811. Springer, Berlin, pp 825–840. doi:10.1007/978-3-642-37140-0_61

Rashedi E, Nezamabadi H, Saryazdi S (2009) Gsa: a gravitational search algorithm. Inf Sci 179(13):2232–2248

Richter H (2009a) Change detection in dynamic fitness landscapes: an immunological approach. In: Nature biologically inspired computing, 2009. NaBIC 2009. World Congress on, pp 719–724. doi:10.1109/NABIC.2009.5393482

Richter H (2009b) Detecting change in dynamic fitness landscapes. In: Evolutionary computation. CEC '09. IEEE congress on, pp 1613–1620

Rohlfshagen P, Yao X (2013) chap Evolutionary dynamic optimization: challenges and perspectives. In: Evolutionary computation for dynamic optimization problems. Springer, Berlin, pp 65–84

Sharma A, Sharma D (2012a) chap ICHEA—a constraint guided search for improving evolutionary algorithms. In: Neural information processing: 19th international conference, ICONIP 2012, Doha, Qatar, Proceedings. Part I. Springer, Berlin, pp 269–279

Sharma A, Sharma D (2012b) chap Solving dynamic constraint optimization problems using ICHEA. In: Neural information processing: 19th international conference, ICONIP 2012. Doha, proceedings, Part III. Springer, Berlin, pp 434–444

Singh H, Isaacs A, Nguyen T, Ray T, Yao X (2009) Performance of infeasibility driven evolutionary algorithm (IDEA) on constrained dynamic single objective optimization problems. In: Evolutionary computation, 2009. CEC '09. IEEE Congress on, pp 3127–3134. doi:10.1109/CEC.2009.4983339

Trojanowski K, Michalewicz Z (1999) Searching for optima in non-stationary environments. In: Evolutionary computation, 1999. CEC 99. Proceedings of the 1999 Congress on, vol 3, p 1850. doi:10.1109/CEC.1999.785498

Umenai Y, Uwano F, Tajima Y, Nakata M, Sato H, Takadama K (2016) A modified cuckoo search algorithm for dynamic optimization problems. In: 2016 IEEE Congress on evolutionary computation (CEC), pp 1757–1764

Yu X, Wu X (2016) A multi-point local search algorithm for continuous dynamic optimization. In: 2016 IEEE Congress on evolutionary computation (CEC), pp 2736–2743

Zhang W, Yen GG, Wang X (2016) An immune inspired framework for optimization in dynamic environment. In: 2016 IEEE congress on evolutionary computation (CEC), pp 1800–1807