On the Use of Repair Methods in Differential Evolution for Dynamic Constrained Optimization

Maria-Yaneli Ameca-Alducin(✉), Maryam Hasani-Shoreh, and Frank Neumann

Optimisation and Logistics, School of Computer Science,
The University of Adelaide, Adelaide, SA 5005, Australia
{maria-yaneli.ameca-alducin,maryam.hasanishoreh,
frank.neumann}@adelaide.edu.au

Abstract. Dynamic constrained optimization problems have received increasing attention in recent years. We study differential evolution which is one of the high performing class of algorithms for constrained continuous optimization in the context of dynamic constrained optimization. The focus of our investigations are repair methods which are crucial when dealing with dynamic constrained problems. Examining recently introduced benchmarks for dynamic constrained continuous optimization, we analyze different repair methods with respect to the obtained offline error and the success rate in dependence of the severity of the dynamic change. Our analysis points out the benefits and drawbacks of the different repair methods and gives guidance to its applicability in dependence on the dynamic changes of the objective function and constraints.

Keywords: Repair methods · Dynamic constrained optimization
Constraint-handling techniques · Differential evolution

1 Introduction

Differential evolution (DE) is known as one of the most competitive, reliable and versatile evolutionary algorithm for optimization on the continuous spaces [1]. DE has shown successful results in a variety of ranges of optimization problems including multi-objective [2], multi-modal [3], large-scale [4], expensive [5], constrained [6] and dynamic optimization problems [7,8]. Among these, constrained optimization problems have a great importance, since in the real world problems, most of the optimization problems have inequality and/or equality constraints. Constrained optimization problems are usually harder to tackle than unconstrained ones, and evolutionary algorithms require a constrained handling technique to deal with the constraints. A review of the different constraint handling techniques can be found in [9].

Another area that have attracted researcher's attention in recent years is dynamic optimization and DE has been regarded as a high performing algorithm in this area [7,10,11]. Considering constraints and dynamism simultaneously (known as dynamic constrained optimization problems: DCOPs) has been even more challenging for an algorithm, as it will be harder to track the global optimum solution when the constraints or the objective function change over-time. The algorithms that solve these kind of problems need to incorporate some mechanisms to deal with the changes in the environment. Different mechanisms have been proposed in the literature on DCOPs including change detection (re-evaluation of solutions, and decreasing the quality of solutions) [12], introducing diversity (increase the mutation) [13], maintaining diversity (adding random solutions called random immigrants) [14], memory-based approaches [15], and the population-based approaches [16].

In order to handle the constraints in these problems, different constraint handling techniques have been applied including penalty functions [13,17], feasibility rules [8], and repair methods [7,11,18]. The last technique has not only been suitable to deal with the constraints, but also has been able to improve the algorithm performance when has been used in dynamic environments. The reason is because this technique is not only choosing between the solutions in the selection, but also moves the solution toward the feasible region by the repair operator. Indeed, the main idea of a repair method is to convert infeasible solutions into feasible ones. Based on the competitive results that these methods have shown, we carry out investigations on the behaviour of these repair methods for DCOPs.

Based on the literature on the current repair methods applied in DCOPs, four types of repair methods including (i) reference-based repair [19], (ii) offspring-repair [7,20], (iii) mutant-repair [11,21] and (iv) gradient-based repair [18] have been distinguished. (i) uses reference solutions in order to convert an infeasible solution to a feasible one. In (ii) the repair method is similar to (i), the only difference between these two methods is that choosing the feasible reference solution in (i) is completely random, while in (ii) the nearest feasible reference solution is selected. (iii) is a repair method which does not require feasible solutions to operate, and is inspired by the differential mutation operator. (iv) is based on gradient information derived from the constraint set to systematically repair infeasible solutions.

Our main focus is to investigate the specifications of each of these methods on a recent benchmark set for DCOPs [19] when applying DE. For the comparison of the effectiveness of each method, the offline error [19] and two newly proposed measures are used. The analysis shows that the gradient-based method outperforms other repair methods based on almost of the measures. However, this method can not be used like a black-box, since it should be known if the constraints have derivative. On the contrary, based on offline error, the worst method seem to be mutant repair method, but this method repairs the solutions very fast after only a few tries. Although, these small number of tries for repairing a solution in this method is mostly because in this benchmark, most of the problems have a huge feasible area. Finally, based on the analysis, the benefits and drawbacks of each method are pointed out and directions for future work are given.

The rest of this paper is organized as follows. In Sect. 2, we define our notion of dynamic constrained optimization problems and provide an introduction into differential evolution together with the different repair methods investigated in this paper. In Sect. 3, the experimental investigations regarding the effectiveness of repair methods with respect to different performance measures are described and the experimental results are divided in offline error analysis and success rate analysis and are presented in Sect. 4 and Sect. 5 respectively. Finally, we finish with some conclusions and directions for future work.

2 Preliminaries

In this section, first we define the problem statement for DCOPs, then we give a brief description of DE algorithm and the mechanism that we have added to it in order to deal with the changes in the environment (called dynamic DE) and finally we present different repair methods that have been applied.

2.1 Problem Statement

A dynamic constrained optimization problem (DCOP) is an optimization problem where the objective function and/or the constraints can change over time [19,22]. Formally, a DCOP can be defined as follows.
Find $\boldsymbol{x}$, at each time t, which:

$$\min_{\boldsymbol{x} \in F_t \subseteq [L,U]} f(\boldsymbol{x}, t) \tag{1}$$

where $f : S \rightarrow \mathbb{R}$ is a single objective function, $\boldsymbol{x} \in \mathbb{R}^D$ is a solution vector and $t \in N^+$ is the current time,

$$[L, U] = \{\boldsymbol{x} = (x_1, x_2, ..., x_D) \mid L_i \leq x_i \leq U_i, i = 1 \ldots D\} \tag{2}$$

is called the search space (S), where L_i and U_i are the lower and upper boundaries of the ith variable,
subject to:

$$\begin{aligned} F_t = \{\boldsymbol{x} \mid \boldsymbol{x} \in [L, U], g_i(\boldsymbol{x}, t) \leq 0, i = 1, \ldots, m, \\ h_j(\boldsymbol{x}, t) = 0, j = 1, \ldots, p\} \end{aligned} \tag{3}$$

is called the feasible region at time t, where m is the number of inequality constraints and p is the number of equality constraints at time t.

$\forall \boldsymbol{x} \in F_t$ if there exists a solution $\boldsymbol{x}^* \in F_t$ such that $f(\boldsymbol{x}^*, t) \leq f(\boldsymbol{x}, t)$, then $\boldsymbol{x}^*$ is called a feasible optimal solution and $f(\boldsymbol{x}^*, t)$ is called the feasible optima value at time t. The objective function and the constrains can be linear or nonlinear.

2.2 Dynamic Differential Evolution

Differential evolution (DE) was first introduced in [23] as a stochastic search algorithm that is simple, reliable and fast. Each vector $\boldsymbol{x}_{i,G}$ in the current population (called at the moment of the reproduction as target vector) generates one trial vector $\boldsymbol{u}_{i,G}$ by using a mutant vector $\boldsymbol{v}_{i,G}$. The mutant vector is created applying $\boldsymbol{v}_{i,G} = \boldsymbol{x}_{r0,G} + F(\boldsymbol{x}_{r1,G} - \boldsymbol{x}_{r2,G})$, where $\boldsymbol{x}_{r0,G}$, $\boldsymbol{x}_{r1,G}$, and $\boldsymbol{x}_{r2,G}$ are vectors chosen at random from the current population ($r0 \neq r1 \neq r2 \neq i$); $\boldsymbol{x}_{r0,G}$ is known as the base vector and $\boldsymbol{x}_{r1,G}$, and $\boldsymbol{x}_{r2,G}$ are the difference vectors and $F > 0$ is a parameter called scale factor. Then the trial vector is created by the recombination of the target vector and mutant vector using a probability crossover $CR \in [0, 1]$.

In this paper DE/rand/1/bin variant is adopted [24], where "rand" indicates how the base vector is chosen (at random in our case), "1" represents how many vector differences (vector pairs) will contribute in differential mutation, and "bin" is the type of crossover (binomial in our case).

In a DCOP an important task is to verify that the solutions' information is correct during the search process. Because when a new change occurs in the environment, the values of the objective function and/or the constraints may change. For this reason a change detection mechanism is required to detect the changes in the objective function and/or the constraints [11,12]. A general overview of DDE algorithm is presented in Algorithm 1.

Algorithm 1. Dynamic differential evolution (DDE)

```
1: Create and evaluate a randomly initial population $\boldsymbol{x}_{i,G}\ \forall i, i = 1, \ldots, NP$
2: for $G \leftarrow 1$ to $MAX\_GEN$ do
3:   for $i \leftarrow 1$ to $NP$ do
4:     Change detection mechanism $(\boldsymbol{x}_{i,G})$
5:     Randomly select $r0 \neq r1 \neq r2 \neq i$
6:     $J_{rand} = randint[1, D]$
7:     for $j \leftarrow 1$ to $D$ do
8:       if $rand_j \leq CR$ Or $j = J_{rand}$ then
9:         $u_{i,j,G} = x_{r1,j,G} + F(x_{r2,j,G} - x_{r3,j,G})$
10:      else
11:        $u_{i,j,G} = x_{i,j,G}$
12:      end if
13:    end for
14:    if $u_{i,j,G}$ is infeasible then
15:      Use the repair method
16:    end if
17:    if $f(\boldsymbol{u}_{i,G}) \leq f(\boldsymbol{x}_{i,G})$ then
18:      $\boldsymbol{x}_{i,G+1} = \boldsymbol{u}_{i,G}$
19:    else
20:      $\boldsymbol{x}_{i,G+1} = \boldsymbol{x}_{i,G}$
21:    end if
22:  end for
23: end for
```

2.3 Repair Methods

Repair methods have shown competitive results compared to other constraint handling methods in constrained optimization. The main idea of a repair method is to use a transformation process to convert an infeasible solution into a feasible one. Although, there is no need for special operators or any modifications of the fitness function in this method like other constraint handling methods, in some repair methods, reference feasible solutions are required [7,19,20,25]. However, the repair methods presented in [11,18] does not require feasible reference solutions. Repair methods used in dynamic constrained optimization problems have had an important role in the algorithm's recovery after a change since they help to move the infeasible solutions toward feasible region. Basically on the related literature of dynamic constrained optimization problems, there have been four repair methods utilized for constraint handling as follows.

Reference-based repair method: This method was originally proposed in [25], and [19] utilized this method with a simple genetic algorithm for solving DCOPs. In this method, firstly, a reference feasible population (R) is created. If an individual of the search population (S) is infeasible, a new individual is generated on the straight line joining the infeasible solution and a randomly chosen member of R. This process will continue until the infeasible solution is repaired or a repair limit (RL $= 100$) attempts are computed. If the new feasible solution has better fitness value, it will be replaced by the selected reference individual. An overview of this method used for our investigations is presented in Algorithm 2, however the boldface part is only used for offspring-repair method.

Offspring-repair method: This method was applied in DCOPs in [7,20]. In this method, a reference feasible population (R) is generated. For any infeasible solution of the search population (S), a new individual is generated on the straight line joining the infeasible solution and the nearest member of the reference population R based on Euclidean distance. This process will continue until the infeasible solution is repaired or a repair limit (RL $= 100$) attempts are computed. If the new feasible solution has better fitness value, it will be replaced by the selected reference individual. This method is similar to the reference-based repair method [19]. The only difference is in the process of selecting the reference solution. An overview of this method is presented in Algorithm 2.

Mutant-repair method: The mutant-repair method (see Algorithm 3) is based on the differential mutation operator, and does not require reference solutions [11]. For each infeasible solution, three new and temporal solutions are generated at random and a differential mutation operator similar to the one used in DE is applied. This repair method is applied until the infeasible solution is repaired or a specific number of unsuccessful trials to obtain a feasible solution have been carried out (RL).

Gradient-based repair method: The gradient-based repair method (see Algorithm 4) was first applied into a simple GA [26] to handle constraints in a static optimization problem and in [18] was applied for solving DCOPs. In this method,

Algorithm 2. Reference-based and offspring-repair methods

```
Require: $\boldsymbol{u}_{i,G}$ {trial vector}
    $counter = 0$
 2: while $\boldsymbol{u}_{i,G}$ is infeasible and $counter \leq$ RL do
        Select the reference individual $r \in R$ based on:
 4:     $\begin{cases} \text{Randomly} & \text{<reference-based>} \\ \text{Min distance between } \boldsymbol{u}_{i,G} \text{ and } r & \text{<offspring>} \end{cases}$
 6:     Create random number $a = U[0,1]$
        Create a new individual in the segment between $\boldsymbol{u}_{i,G}$ ($s \in S$) and $r$
 8:     $\boldsymbol{u}_{i,G} = a * r + (1 - a) * \boldsymbol{u}_{i,G}$
        if $\boldsymbol{u}_{i,G}$ is infeasible then
10:         go to step 2
        else
12:         Update reference population if the repaired solution has better fitness value
            than $R$
        end if
14:     $counter = counter + 1$
    end while
16: Return $\boldsymbol{u}_{i,G}$
```

Algorithm 3. Mutant-repair method

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

the gradient information of the constraints are utilized to repair the infeasible solutions [26]. For this purpose the gradient of the constraints based on the solution vector (that represent the rate of change of constraints based on each variable) will be calculated. At the next step the constraint violations are calculated and based on this amount and the vector of gradient, the solutions will move toward the feasible region with the proportional quantity. The constraints that are non-violated are not considered in these calculations. In this method the main idea is to only change the effective variables over the constraints that have a violation. More detail about this method can be found in [18].

3 Experimental Investigations

In this section, the utilized test problems, the performance measures and the experimental setup are presented.

Algorithm 4. Gradient-based repair method

```
Require: $\boldsymbol{u}_{i,G}$ {trial vector}
   $counter = 0$
2: while $\boldsymbol{u}_{i,G}$ is infeasible and $counter \leq$ RL do
      Calculate the constraint violation
4:    Calculate the amount of solution movement $\Delta \boldsymbol{u}_{i,G}$ based on the current constraint violation and the gradient information
      $\boldsymbol{u}_{i,G} = \boldsymbol{u}_{i,G} + \Delta \boldsymbol{u}_{i,G}$
6:    $counter = counter + 1$
   end while
8: Return $\boldsymbol{u}_{i,G}$
```

3.1 Test Problems and Performance Measures

The chosen benchmark problem originally has 18 functions [19], however in this work, 10 functions among them were used for the experiments. The reason for this selection was that part of these functions were not constrained and part of them did not have derivative for the constraints and could not be applied in Gradient-based method. The test problems in this benchmark consist a variety of characteristics like (i) disconnected feasible regions (1–3), (ii) the global optima at the constraints' boundary or switchable between disconnected regions, or (iii) the different shape and percentage of feasible area. In the experiments, for the objective function, only medium severity is considered ($k = 0.5$), while different change severities are considered for the constraints ($S = 10$, 20 and 50). Based on the definition of the constrains in this benchmark [19], $S = 10$ represents for large severity, $S = 20$ for medium severity and $S = 50$ for the small severity of changes on the constraints. The frequency of change (f_c) is considered equal to 1000 evaluations (only in the objective function). Worth to mention that, in the repair methods, the constraints evaluations are not considered as extra evaluations when using for DCOPs [21]. More details on the benchmark can be found in [19].

For the purpose of comparing the effectiveness of each repair method, the following performance measures were used:

Offline error (off_e) [27]**:** This measurement is equal to the average of the sum of errors in each generation divided by the total number of generations. The zero value of offline error indicates a perfect performance [22]. This measure is defined as:

$$off\_e = \frac{1}{G_{max}} \sum_{G=1}^{G_{max}} e(G) \tag{4}$$

where G_{max} is the number of generations computed by the algorithm and $e(G)$ denotes the error in the current iteration G (see 5):

$$e(G) = |f(\boldsymbol{x}^*, t) - f(\boldsymbol{x}_{best,G}, t)| \tag{5}$$

where $f(\boldsymbol{x}^*, t)$ is the feasible global optima[1] at current time t, and $f(\boldsymbol{x}_{best,G}, t)$ represent the best solution (feasible or infeasible) found so far at generation G at current time t.

Success rate: This measure is calculated such that considers how many of the infeasible solutions were successful to be repaired after 100 iterations. For each infeasible solution, a repair is needed and at the end of repair iteration (Maximum 100 tries), if the solution is feasible a counter is increased. In another words, it is considered a success if before achieving to the maximum number of allowed iterations for repair (100 in our case) a solution is feasible. The total number of these successful repaired solutions (s) divided by the total number of solutions that need repair (n_T) is equal to success rate percentage. Based on this, the repair methods with success rate values equals to 100%, are able to convert all the solutions.

$$s_r = \frac{s}{n_T} \tag{6}$$

Required number of iterations: In order to distinguish the difference between the number of evaluations that each method consumes for repairing the solution, a measurement is defined called as required number of iterations (rn_i). In this way, it is possible to compare the efficiency of each repair method. The range of values of this measure is $\in (1 - 100)$. The more efficient method uses lower number of evaluations in order to repair an infeasible solution. The final amount for this measurement value is the average between the number of tries taken to convert each infeasible solution into feasible one.

3.2 Experimental Setup

The experimental results are divided as (i) offline error analysis and (ii) success rate and required number of iterations. In these experiments we investigate the behaviour of different repair methods in DDE algorithm based on the previous defined measures. In the analysis, the effects of different severities on the constraints are considered for these ten test problems. We do not bring the results for changes of frequency since it does not have any effect in the behaviour of the repair methods.

The configurations for the experiments are as follows. The number of runs in the experiments are 50, and number of considered times for dynamic perspective of the test algorithm is $5/k$ ($k = 0.5$). Parameters relating to DDE algorithm are as follows: DE variant is DE/rand/1/bin, population size is 20, scaling factor (F) is a random number $\in [0.2, 0.8]$, and crossover probability is 0.2. In the experiments, four repair methods including Reference-based, Offspring, Mutant and Gradient-based as explained in Sect. 2.3 have been applied for handling the constraint in DDE algorithm.

[1] This global optima is an approximation, which is the best solution found by DE in 50 runs for the current time.

Table 1. Average and standard deviation of offline error values obtained by all the repairs methods with $k = 0.5$, $S = 10$, 20 and 50, and $f_c = 1000$. Best results are remarked in boldface.

Algorithms			$S = 10$		
	G24_1	**G24_f**	**G24_2**	**G24_3**	**G24_3b**
Reference	0.07(±0.029)	0.029(±0.022)	0.394(±0.212)	0.041(±0.025)	0.058(±0.027)
Offspring	0.07(±0.053)	0.036(±0.036)	0.451(±0.317)	0.068(±0.056)	0.073(±0.048)
Mutant	0.271(±0.051)	0.095(±0.048)	0.29(±0.021)	0.159(±0.031)	0.193(±0.041)
Gradient	**0.043(±0.028)**	**0.004(±0.003)**	**0.259(±0.012)**	**0.01(±0.004)**	**0.033(±0.015)**
	G24_3f	**G24_4**	**G24_5**	**G24_7**	**G24_8b**
Reference	0.007(±0.004)	0.071(±0.035)	0.071(±0.024)	0.12(±0.088)	0.105(±0.062)
Offspring	0.04(±0.083)	0.067(±0.031)	0.089(±0.035)	0.253(±0.128)	0.114(±0.056)
Mutant	0.046(±0.019)	0.187(±0.045)	0.126(±0.021)	0.208(±0.034)	0.338(±0.048)
Gradient	**0.002(±0.003)**	**0.032(±0.013)**	**0.024(±0.007)**	**0.021(±0.008)**	**0.031(±0.009)**
Algorithms			$S = 20$		
	G24_1	**G24_f**	**G24_2**	**G24_3**	**G24_3b**
Reference	0.078(±0.042)	0.026(±0.019)	0.406(±0.328)	0.02(±0.009)	0.06(±0.036)
Offspring	0.086(±0.061)	0.03(±0.025)	0.416(±0.321)	0.039(±0.033)	0.035(±0.023)
Mutant	0.246(±0.047)	0.1(±0.05)	0.296(±0.02)	0.156(±0.033)	0.207(±0.031)
Gradient	**0.048(±0.026)**	**0.004(±0.004)**	**0.258(±0.009)**	**0.004(±0.002)**	**0.035(±0.017)**
	G24_3f	**G24_4**	**G24_5**	**G24_7**	**G24_8b**
Reference	0.008(±0.005)	0.06(±0.032)	0.075(±0.033)	0.107(±0.045)	0.108(±0.041)
Offspring	0.023(±0.029)	0.043(±0.038)	0.092(±0.048)	0.213(±0.075)	0.12(±0.069)
Mutant	0.05(±0.019)	0.218(±0.033)	0.132(±0.024)	0.267(±0.039)	0.333(±0.044)
Gradient	**0.002(±0.002)**	**0.033(±0.015)**	**0.029(±0.013)**	**0.021(±0.009)**	**0.033(±0.008)**
Algorithms			$S = 50$		
	G24_1	**G24_f**	**G24_2**	**G24_3**	**G24_3b**
Reference	0.069(±0.031)	0.031(±0.024)	0.371(±0.232)	0.011(±0.005)	0.045(±0.025)
Offspring	0.06(±0.032)	0.039(±0.037)	0.39(±0.187)	0.037(±0.069)	0.029(±0.025)
Mutant	0.26(±0.051)	0.1(±0.047)	0.298(±0.023)	0.1(±0.023)	0.161(±0.024)
Gradient	**0.043(±0.018)**	**0.003(±0.003)**	**0.257(±0.01)**	**0.002(±0.002)**	**0.027(±0.011)**
	G24_3f	**G24_4**	**G24_5**	**G24_7**	**G24_8b**
Reference	0.008(±0.009)	0.053(±0.042)	0.062(±0.018)	0.084(±0.024)	0.096(±0.041)
Offspring	0.03(±0.049)	0.038(±0.046)	0.08(±0.032)	0.2(±0.078)	0.111(±0.065)
Mutant	0.046(±0.018)	0.162(±0.022)	0.145(±0.025)	0.289(±0.037)	0.351(±0.04)
Gradient	**0.003(±0.003)**	**0.026(±0.011)**	**0.033(±0.012)**	**0.026(±0.011)**	**0.031(±0.007)**

4 Offline Error Analysis

The results obtained for the four repair methods using offline error are summarized in Table 1. Furthermore, for the statistical validation, the 95%-confidence Kruskal-Wallis (KW) test and the Bergmann-Hommels post-hoc test, as suggested in [28] are presented (see Table 2). Non-parametric tests were adopted because the samples of runs did not fit to a normal distribution based on the Kolmogorov-Smirnov test. Based on the results, for the constraint's change severity $S = 10$, the gradient-based repair outperformed almost all of the other methods in nine test problems (G24_f, G24_2, G24_3, G24_3b, G24_3f, G24_4, G24_5, G24_7 and G24_8b) except one test problem (G24_1) that in which offspring-repair has similar performance. For this severity, reference-based repair and offspring-repair performed almost the same for nine test problems (G24_1, G24_f, G24_2, G24_3, G24_3b, G24_3f, G24_4, G24_5 and G24_8b) except one test

Table 2. Statistical tests on the offline error values in Table 1. "$X^{(-)}$" means that the corresponding algorithm outperformed algorithm X. "$X^{(+)}$" means that the corresponding algorithm was dominated by algorithm X. If algorithm X does not appear in column Y means no significant differences between X and Y.

Functions	$S = 10$			
	Reference(1)	**Offspring(2)**	**Mutant(3)**	**Gradient(4)**
G24_1 (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_f (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_2 (44.2%)	$4^{(+)}$	$4^{(+)}$	$4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3 (7.1-49.21%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3b (7.1-49.21%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3f (7.1%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_4 (0-44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_5 (0-44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_7 (0-44.2%)	$2^{(-)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(+)}$ and $4^{(+)}$	$1^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_8b (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
	$S = 20$			
	Reference(1)	**Offspring(2)**	**Mutant(3)**	**Gradient(4)**
G24_1 (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_f (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_2 (44.2%)	$4^{(+)}$		$4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_3 (7.1-49.21%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3b (7.1-49.21%)	$2^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_3f (7.1%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_4 (4.75-44.2%)	$2^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_5 (4.75-44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_7 (4.75-44.2%)	$2^{(-)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(+)}$ and $4^{(+)}$	$1^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_8b (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
	$S = 50$			
	Reference(1)	**Offspring(2)**	**Mutant(3)**	**Gradient(4)**
G24_1 (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_f (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_2 (44.2%)	$3^{(+)}$		$1^{(+)}$ and $4^{(+)}$	$3^{(-)}$
G24_3 (7.1-18.63%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_3b (7.1-18.63%)	$2^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_3f (7.1%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_4 (28.9-44.2%)	$2^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$ and $3^{(-)}$
G24_5 (28.9-44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_7 (28.9-44.2%)	$2^{(-)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$
G24_8b (44.2%)	$3^{(-)}$ and $4^{(+)}$	$3^{(-)}$ and $4^{(+)}$	$1^{(+)}$, $2^{(+)}$ and $4^{(+)}$	$1^{(-)}$, $2^{(-)}$ and $3^{(-)}$

problem (G24_7) where reference-based repair outperformed offspring-repair. As Table 2 illustrates, mutant-repair is the worst between all the methods for eight test problems (G24_1, G24_f, G24_3, G24_3b, G24_3f, G24_4, G24_5 and G24_8b) except two test problems in which has similar results with reference-based repair (G24_2) and offspring-repair (G24_2 and G24_7).

For the constraint's change severity $S = 20$, the gradient-repair excelled almost all the other methods in seven test problems (G24_1, G24_f, G24_3, G24_3f, G24_5, G24_7 and G24_8b) with exceptions including G24_2, G24_3b and G24_4, that in which offspring-repair had similar performance. For this change severity, reference-based repair and offspring-repair performed almost the same for seven test problems (G24_1, G24_f, G24_2, G24_3, G24_3f, G24_5 and G24_8b) except three test problems (G24_3b, G24_4 and G24_7). For these three problems, while in two test problems (G24_3b and G24_4) offspring-repair had better results, in one test problem (G24_7) reference-based repair outperformed the offspring-repair. Mutant-repair had the worst results between all the methods for eight test problems (G24_1, G24_f, G24_3, G24_3b, G24_3f, G24_4, G24_5 and G24_8b) except two test problems in which had similar results with reference-based repair (G24_2) and offspring-repair (G24_2 and G24_7).

For the constraint's change severity $S = 50$, the gradient-repair excelled the other methods in six test problems (G24_f, G24_3, G24_3f, G24_5, G24_7 and G24_8b) with exceptions of having similar performance with offspring-repair (G24_1, G24_2, G24_3b and G24_4) and reference-based repair (G24_2). For this change severity, reference-based repair and offspring-repair performed almost the same for seven test problems (G24_1, G24_f, G24_2, G24_3, G24_3f, G24_5 and G24_8b) except three test problems (G24_3b, G24_4 and G24_7). For these three problems, while in two test problems (G24_3b and G24_4) offspring-repair had better results, in one test problem (G24_7) reference-based repair outperformed the offspring-repair. Mutant-repair had the worst results between all the methods for nine test problems (G24_1, G24_f, G24_3, G24_3b, G24_3f, G24_4, G24_5, G24_7 and G24_8b) except one test problem (G24_2) in which showed similar results with offspring-repair.

Gradient-repair for all severities outperformed other methods, because in this work, all of the test problems have the global optimum on the constraints' boundaries, and since this method moves slowly toward the feasible area its less probable that loses the information of global optima in the boundaries by crossing it. Although, this method cannot be applied for the functions that do not have derivative for their constraints. For this reason, the four functions G24_6a, G24_6b, G24_6c and G24_6d (that are functions inside this set of benchmark), were not used in our experiments. Therefore, for this method an understanding about the behaviour of the constraints is specifically needed. Changes in severity do not decrease the performance of this method. Even if for severity $S = 50$, it outperformed other methods in less test problems, this is because offspring-repair performance increased for some test problems for this severity. Similar behaviour in reference-based repair and offspring-repair based on offline error for all the severities is due to the similar procedure (uniform crossover in GA) that they use for repairing the infeasible solutions. The only difference is the way that they choose the reference solution.

5 Analysis of Success Rate and Required Number of Iterations for Repairing Solutions

Regardless of severity the total number of infeasible solutions (n_T) that needed repair for different functions were in the range between 1882 and 2981. The n_T values were increased for the functions that had dynamic constraints like G24_3, G24_3b, G24_4, G24_5 and G24_7. The reason is because, when the constraints

Table 3. Average and standard deviation of: (i) Success rate(s_r), (ii) required number of iterations (rn_i) for each of the repairs methods with $k = 0.5$, $S = 10$, 20 and 50, and $f_c = 1000$. Best results are remarked in boldface.

Functions	Success rate(s_r)				Required number of iterations(rn_i)			
	Reference	**Offspring**	**Mutant**	**Gradient**	**Reference**	**Offspring**	**Mutant**	**Gradient**
	$S = 10$							
G24_1	99.95(±0.08)	99.94(±0.08)	**100.00(±0.00)**	99.72(±0.15)	67.98(±7.99)	79.78(±4.33)	**2.26(±0.03)**	4.30(±0.24)
G24_f	99.97(±0.05)	99.97(±0.05)	**100.00(±0.00)**	99.71(±0.17)	64.77(±8.41)	74.95(±8.33)	**2.26(±0.03)**	3.96(±0.23)
G24_2	99.99(±0.02)	99.96(±0.08)	**100.00(±0.00)**	99.78(±0.18)	51.20(±10.65)	70.18(±5.94)	**2.26(±0.04)**	3.80(±0.26)
G24_3	99.98(±0.04)	99.96(±0.06)	**99.98(±0.02)**	95.19(±1.73)	60.35(±7.92)	70.91(±6.07)	**4.74(±0.13)**	8.03(±1.64)
G24_3b	99.97(±0.07)	99.97(±0.06)	**99.99(±0.03)**	95.96(±1.38)	66.51(±7.95)	71.81(±7.50)	**4.74(±0.11)**	7.60(±1.30)
G24_3f	99.88(±0.11)	99.85(±0.12)	**99.94(±0.05)**	93.33(±2.21)	84.66(±3.47)	89.90(±3.49)	14.04(±0.25)	**10.26(±2.10)**
G24_4	99.98(±0.04)	99.94(±0.07)	**99.99(±0.02)**	95.45(±1.75)	62.23(±9.58)	72.54(±6.27)	**4.77(±0.15)**	8.05(±1.66)
G24_5	**95.76(±3.55)**	74.16(±11.24)	71.44(±0.69)	74.31(±1.90)	47.28(±6.52)	72.00(±3.66)	38.09(±0.74)	**28.79(±1.78)**
G24_7	**92.79(±6.18)**	75.81(±9.57)	72.40(±0.66)	75.55(±2.06)	63.57(±6.47)	73.20(±4.76)	37.30(±0.73)	**27.44(±1.97)**
G24_8b	99.97(±0.06)	99.95(±0.06)	**100.00(±0.00)**	99.75(±0.13)	63.29(±6.01)	69.54(±6.26)	**2.26(±0.03)**	3.94(±0.19)
	$S = 20$							
G24_1	99.97(±0.05)	99.94(±0.08)	**100.00(±0.00)**	99.75(±0.11)	66.87(±7.25)	78.58(±5.85)	**2.25(±0.04)**	4.22(±0.24)
G24_f	99.98(±0.04)	99.97(±0.07)	**100.00(±0.00)**	99.67(±0.19)	65.19(±8.69)	76.88(±6.91)	**2.26(±0.04)**	4.02(±0.26)
G24_2	99.97(±0.05)	99.95(±0.10)	**100.00(±0.00)**	99.83(±0.11)	50.61(±10.83)	69.63(±6.42)	**2.26(±0.03)**	3.75(±0.22)
G24_3	99.96(±0.07)	99.93(±0.09)	**100.00(±0.01)**	92.37(±2.48)	69.44(±5.67)	74.51(±4.81)	**6.11(±0.14)**	11.09(±2.37)
G24_3b	99.96(±0.07)	99.89(±0.11)	**100.00(±0.01)**	92.12(±2.83)	71.79(±6.88)	79.61(±3.70)	**6.08(±0.14)**	11.60(±2.68)
G24_3f	99.90(±0.11)	99.84(±0.13)	**99.94(±0.04)**	92.96(±2.17)	85.00(±3.61)	90.97(±2.55)	14.12(±0.23)	**10.63(±2.04)**
G24_4	99.96(±0.07)	99.89(±0.10)	**99.99(±0.02)**	92.54(±2.23)	72.50(±6.14)	79.75(±3.84)	**6.12(±0.12)**	11.22(±2.11)
G24_5	97.47(±1.19)	82.19(±9.70)	**100.00(±0.00)**	92.46(±2.64)	39.24(±11.07)	69.08(±4.56)	**4.96(±0.11)**	10.85(±2.53)
G24_7	96.09(±1.84)	86.39(±5.93)	**100.00(±0.00)**	93.03(±2.25)	56.24(±6.08)	69.03(±4.75)	**4.90(±0.11)**	10.41(±2.17)
G24_8b	99.97(±0.05)	99.95(±0.07)	**100.00(±0.00)**	99.70(±0.20)	61.55(±5.67)	68.86(±7.18)	**2.27(±0.04)**	3.95(±0.26)
	$S = 50$							
G24_1	99.97(±0.05)	99.91(±0.10)	**100.00(±0.00)**	99.70(±0.15)	68.09(±8.06)	78.75(±4.43)	**2.26(±0.04)**	4.28(±0.23)
G24_f	99.97(±0.05)	99.95(±0.07)	**100.00(±0.00)**	99.69(±0.15)	64.98(±10.06)	74.94(±7.47)	**2.26(±0.04)**	4.02(±0.27)
G24_2	99.99(±0.02)	99.93(±0.11)	**100.00(±0.00)**	99.84(±0.08)	50.01(±10.05)	69.64(±5.59)	**2.27(±0.04)**	3.76(±0.17)
G24_3	99.92(±0.09)	99.89(±0.13)	**99.99(±0.01)**	92.90(±2.42)	75.99(±4.35)	80.88(±4.04)	**8.63(±0.15)**	10.62(±2.31)
G24_3b	99.92(±0.10)	99.86(±0.12)	**99.99(±0.02)**	93.04(±2.19)	77.14(±5.02)	83.60(±2.87)	**8.63(±0.18)**	10.82(±2.12)
G24_3f	99.91(±0.09)	99.84(±0.11)	**99.95(±0.04)**	93.18(±2.16)	84.12(±3.39)	90.11(±3.96)	14.01(±0.25)	**10.40(±2.08)**
G24_4	99.90(±0.10)	99.85(±0.13)	**99.99(±0.02)**	93.36(±2.01)	78.17(±4.62)	84.28(±2.89)	**8.66(±0.15)**	10.51(±1.91)
G24_5	97.64(±1.30)	86.30(±8.46)	**100.00(±0.00)**	91.57(±3.13)	40.28(±9.01)	66.52(±6.61)	**2.83(±0.05)**	11.66(±2.99)
G24_7	96.88(±1.65)	89.71(±6.01)	**100.00(±0.00)**	93.04(±2.82)	55.12(±6.69)	66.17(±5.68)	**2.82(±0.04)**	10.38(±2.70)
G24_8b	99.97(±0.06)	99.97(±0.07)	**100.00(±0.00)**	99.77(±0.16)	63.97(±7.20)	71.18(±6.15)	**2.27(±0.03)**	3.89(±0.24)

are changing, it is more probable that some feasible solutions be converted to infeasible ones after a change occurs.

The results for the success rate (s_r) and required number of iterations (nr_i) measures are presented in Table 3. Regarding to these results, some general observations can be concluded. The number of required iterations (nr_i), was the smallest for mutant-repair with a range between 2 to 8 and in second place is gradient-repair with the range between 4 to 10. An exception of this trend was seen in the function G24_3f in all the severities, and functions G24_5 and G24_7 for the severity $S = 10$, which gradient-repair excelled mutant-repair since the percentage of feasible area in these cases were small (see Table 2 for the feasibility percentages). Overall, in mutant-repair method since the process of producing a feasible solution is completely random and in this applied benchmark functions, the percentage of feasible area is huge, so this method achieved to feasible solutions after a few tries. In another words, this method is roughly dependent to the percentage of the feasible area. As mentioned before, the second smallest values for this measure was for gradient-based method; but in this case the reason was based on this method's wise selection and the fact that it only moves in the direction and with the amount of satisfying the constraint violations.

The worse results for this measure belonged to offspring-repair with an average number of required iterations ranging from 66 to 91 and reference-based repair with a range from 47 to 85. Compared to offspring-repair, reference-based repair required lower number of iterations, and this was because offspring-repair's step sizes are smaller and for this reason it needed more iterations to convert the infeasible solution to a feasible one. Other drawback in these two methods is that a number of evaluations is needed to produce feasible reference population. This can be expensive in high computational complex problems [5].

Generally based on considering all the measures, offspring and reference-based repair methods in most functions had similar behaviours. This is mostly because, the process of converting the infeasible solutions to feasible ones are approximately the same in these methods, and the only difference is the way that they choose the feasible reference solution. They do not loose the information of infeasible solution completely, as they use this individual to move in the direction of one of the feasible solutions (this is more evident in offspring as it uses the nearest reference feasible solution).

As regards to the third measure (success rate), although mutant-repair has the best values, but this is because in this set of benchmark, most of the functions

Table 4. Main features of each repair method

Method	Advantages	Disadvantages
Reference	(i) Maintain infeasible solution information, (ii) increase diversity	(i) Random behavior (ii) high number of required iterations and (iii) reference solutions needed
Offspring	(i) Maintain infeasible solution information	(i) High number of required iterations and (ii) roughly random behavior
Mutant	(i) High success rate, (ii) low iterations needed, (iii) no reference solution needed, (iv) increase diversity	(i) Not a good performance (offline error) (ii) loose the information and (iii) random behavior
Gradient	(i) Prominent performance when the optimal solution is in the boundaries of the feasible area, (ii) good performance (offline error), (iii) no reference solution needed, (iv) maintain infeasible solution information and (v) low iterations needed	(i) Knowledge about the characteristic of constraints needed and (ii) only can be applied when the constraints have derivate

has a huge percentage of feasible area. For this reason reaching to a feasible solution randomly after a few tries is easily possible based on this method. Obviously, for the cases of small percentage of feasible area, this method's efficiency will decrease. This was the case for the functions G24_5 and G24_7; as can be seen from Table 1, the values for this measure dropped drastically for this method as the percentage of feasible area is small for some time periods in these two functions. Reference-based and offspring were on the second place based on the values of this measure and the results of these two methods are roughly similar. Although, gradient-based method seemed to have worse results based on this measure, the differences between these values and the values for other methods were not significant. Moreover, practically, there is no need to convert all the infeasible solutions. In Table 4 a review of the advantages and disadvantages of each method is presented.

6 Conclusion and Future Work

In this paper, an investigation on different current repair methods in DCOPs were carried out. For the comparison, three different measures called: offline error, success rate and average number of required iterations for repairing the infeasible individuals, for each method were used. The results showed regardless of the change severities, in most cases gradient-based method outperformed the other methods based on offline error. This method especially performs much better than the other methods for the problems that have the optimal solution in the boundaries of the feasible area. Indeed, this method, moves very small steps and will not lose the optimal solution in the boundaries. Although, this method can not be applied for the functions that do not have derivative of the constraints. For the other measurement criteria, the number of required repair for mutant repair was the smallest and the second rank was for gradient-based method. Finally, based on the success rate, all of the repair methods were able to repair most of the infeasible solutions; such good performance was based on the fact that the feasible region of the main static test problem (G24) [29], occupy around 79% of the whole search space [11]. For future work, a combination of different repair methods can be investigated in order to make the most of each method. In addition, other constraint handling methods like ϵ-constrained, stochastic ranking and multi objective concepts that have not been applied in DCOPs can be applied and compared as well.

Acknowledgement. This work has been supported through Australian Research Council (ARC) grants DP140103400 and DP160102401.

References

1. Das, S., Mullick, S.S., Suganthan, P.N.: Recent advances in differential evolution – an updated survey. Swarm Evol. Comput. **27**, 1–30 (2016). http://www.sciencedirect.com/science/article/pii/S2210650216000146
2. Rakshit, P., Konar, A., Das, S., Jain, L.C., Nagar, A.K.: Uncertainty management in differential evolution induced multiobjective optimization in presence of measurement noise. IEEE Trans. Syst. Man. Cybern. Syst. **44**(7), 922–937 (2014)
3. Basak, A., Das, S., Tan, K.C.: Multimodal optimization using a biobjective differential evolution algorithm enhanced with mean distance-based selection. IEEE Trans. Evol. Comput. **17**(5), 666–685 (2013)
4. Omidvar, M.N., Li, X., Mei, Y., Yao, X.: Cooperative co-evolution with differential grouping for large scale optimization. IEEE Trans. Evol. Comput. **18**(3), 378–393 (2014)
5. Elsayed, S.M., Ray, T., Sarker, R.A.: A surrogate-assisted differential evolution algorithm with dynamic parameters selection for solving expensive optimization problems. In: 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 1062–1068. IEEE (2014)
6. Bu, C., Luo, W., Zhu, T.: Differential evolution with a species-based repair strategy for constrained optimization. In: 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 967–974. IEEE (2014)
7. Pal, K., Saha, C., Das, S.: Differential evolution and offspring repair method based dynamic constrained optimization. In: Panigrahi, B.K., Suganthan, P.N., Das, S., Dash, S.S. (eds.) SEMCCO 2013. LNCS, vol. 8297, pp. 298–309. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03753-0_27
8. Ameca-Alducin, M.Y., Mezura-Montes, E., Cruz-Ramirez, N.: Differential evolution with combined variants for dynamic constrained optimization. In: 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 975–982, July 2014
9. Mezura-Montes, E., Coello, C.A.C.: Constraint-handling in nature-inspired numerical optimization: past, present and future. Swarm Evol. Comput. **1**(4), 173–194 (2011)
10. Eita, M.A., Shoukry, A.A.: Constrained dynamic differential evolution using a novel hybrid constraint handling technique. In: 2014 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 2421–2426. IEEE (2014)
11. Ameca-Alducin, M.Y., Mezura-Montes, E., Cruz-Ramírez, N.: 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, GECCO 2015, ACM, New York, NY, USA, pp. 241–248 (2015). https://doi.org/10.1145/2739480.2754786
12. Richter, H.: Detecting change in dynamic fitness landscapes. In: IEEE Congress on Evolutionary Computation, CEC 2009, pp. 1613–1620 (2009)
13. Cobb, H.: An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical report, Naval Research Lab, Washington DC (1990)
14. Tins, R., Yang, S.: A self-organizing random immigrants genetic algorithm for dynamic optimization problems. Genet. Program. Evol. Mach. **8**(3), 255–286 (2007). https://doi.org/10.1007/s10710-007-9024-z
15. Richter, H., Yang, S.: Memory based on abstraction for dynamic fitness functions. In: Giacobini, M., et al. (eds.) EvoWorkshops 2008. LNCS, vol. 4974, pp. 596–605. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78761-7_65

16. Li, C., Nguyen, T.T., Yang, M., Yang, S., Zeng, S.: Multi-population methods in unconstrained continuous dynamic environments: the challenges. Inf. Sci. **296**, 95–118 (2015)
17. Grefenstette, J.: Genetic algorithms for changing environments. In: Parallel Problem Solving from Nature 2, pp. 137–144. Elsevier (1992)
18. Bu, C., Luo, W., Yue, L.: Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies. IEEE Trans. Evol. Comput. **PP**(99), 1 (2016)
19. Nguyen, T., Yao, X.: Continuous dynamic constrained optimization: the challenges. IEEE Trans. Evol. Comput. **16**(6), 769–786 (2012)
20. Pal, K., Saha, C., Das, S., Coello-Coello, C.: Dynamic constrained optimization with offspring repair based gravitational search algorithm. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 2414–2421 (2013)
21. Ameca-Alducin, M.Y., Mezura-Montes, E., Cruz-Ramírez, N.: Differential evolution with combined variants plus a repair method to solve dynamic constrained optimization problems: a comparative study. Soft Computing, pp. 1–30 (2016)
22. Nguyen, T., Yang, S., Branke, J.: Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evol. Comput. **6**, 1–24 (2012). http://www.sciencedirect.com/science/article/pii/S2210650212000363
23. Price, K., Storn, R., Lampinen, J.: Differential evolution a practical approach to global optimization, Natural Computing. Springer-Verlag, Heidelberg (2005). http://www.springer.com/west/home/computer/foundations?SGWID=4-156-22-32104365-0&teaserId=68063&CENTER_ID=69103
24. Mezura-Montes, E., Miranda-Varela, M.E., del Carmen Gómez-Ramón, R.: Differential evolution in constrained numerical optimization: an empirical study. Inf. Sci. **180**(22), 4223–4262 (2010)
25. Michalewicz, Z., Nazhiyath, G.: Genocop III: a co-evolutionary algorithm fornumerical optimization problems with nonlinear constraints. In: IEEE International Conference on Evolutionary Computation, vol. 2, pp. 647–651, November 1995
26. Chootinan, P., Chen, A.: Constraint handling in genetic algorithms using a gradient-based repair method. Comput. Oper. Res. **33**(8), 2263–2281 (2006). http://www.sciencedirect.com/science/article/pii/S030505480500050X
27. Branke, J., Schmeck, H.: Designing evolutionary algorithms for dynamic optimization problems. In: Ghosh, A., Tsutsui, S. (eds.) Advances in Evolutionary Computing. Natural Computing Series, pp. 239–262. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-642-18965-4_9
28. Derrac, J., García, S., Molina, D., Herrera, F.: 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 (2011). http://www.sciencedirect.com/science/article/pii/S2210650211000034
29. Liang, J.J., Runarsson, T., Mezura-Montes, E., Clerc, M., Suganthan, P., Coello Coello, C.A., Deb, K.: Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. Technical report, Nanyang Technological University, Singapore, Singapure, December 2005