Alexandria Engineering Journal (2023) 66, 827–843

HOSTED BY

Alexandria University

Alexandria Engineering Journal

www.elsevier.com/locate/aej
www.sciencedirect.com

ORIGINAL ARTICLE

Evolutionary approach for dynamic constrained optimization problems

Noha Hamza [*], Ruhul Sarker, Daryl Essam, Saber Elsayed

School of Engineering and Information Technology, University of New South Wales, sACT, Australia

Received 5 August 2022; revised 15 October 2022; accepted 29 October 2022
Available online 18 November 2022

KEYWORDS

Dynamic optimization;
Constrained optimization;
Evolutionary algorithms

Abstract The number of research works on dynamic constrained optimization problems has been increasing rapidly over the past two decades. In this domain, many real-life decision problems need to be solved repeatedly with changing data and parameters. However, no research on dynamic problems with changes in the coefficients of the constraint functions has been reported. In this paper, to deal with such problems, a new evolutionary framework with multiple novel mechanisms is proposed. The new mechanisms are for (1) dealing with both linear and non-linear components in the constraint functions, (2) identifying the rate of change in the coefficients of the variables and (3) updating the population efficiently after every change occurs in the problem. To evaluate the performance of the proposed algorithm, we designed a new set of 13 dynamic benchmark problems, each of which consists of 20 dynamic changes and 3 different scenarios. The results demonstrate that the proposed algorithm significantly contributes in achieving good quality solutions, high feasibility rates and fast convergence in rapidly changing environments. In addition, the framework shows its capability of using different meta-heuristics to solve dynamic problems.

1. Introduction

Many disciplines, such as computer science, cybersecurity, decision sciences and engineering, deal with optimization problems with dynamic constraints that may change over time [1,2]. Such a problem is well-known as a dynamic constrained optimization problem (DCOP) in which a change may occur in its objective function and/or constraints. Specifically, it may be on the right-hand side of the constraints and/or the coefficients of the decision variables in the objective and constraint functions. DCOPs are more complicated than their unconstrained counterparts as their final solutions have to be feasible (i.e., satisfy all their constraints). Solving them is further complicated if the feasible region is small and/or the problem contains multiple disjointed feasible areas. Also, a quick change in an environment adds another barrier to solving them efficiently. Note that we call each instance of a problem an environment(t).

In this paper, we are concerned with changes in the coefficients of the decision variables in the constraint functions of DCOPs. This type of problem can be found in many practical decision-making situations [3], such as those currently faced

* Corresponding author.
E-mail addresses: n.hamza@unsw.edu.au (N. Hamza), r.sarker@adfa.edu.au (R. Sarker), d.essam@unsw.edu.au (D. Essam), s.elsayed@unsw.edu.au (S. Elsayed).

Peer review under responsibility of Faculty of Engineering, Alexandria University.

https://doi.org/10.1016/j.aej.2022.10.072
1110-0168 © 2022 THE AUTHORS. Published by Elsevier BV on behalf of Faculty of Engineering, Alexandria University.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

due to Covid-19; for example, consider a company that has a limited budget to produce a certain number of items, each of which has its price and cost (the cost, i.e., hourly pay rate, can be a coefficient in a budget constraint). Given a sudden Covid-19 outbreak and a strict lockdown (or a worker testing positive who needs to be in quarantine), the hourly pay rate (a kind of incentive) may need to be increased to encourage other workers to work longer hours to meet demands on time; for example, temporary increases were introduced by Home Depot (one of the largest home improvement retailers in the United States) and Costco to reward their current workers until their staffing problems were sorted out [1]. Such a change could happen more frequently (depending on the number of Covid-19 cases), i.e., the hourly pay rate could be either further increased or decreased to an earlier one (i.e., cyclic change). We believe that handling such problems quickly is very important in such situations.

In the literature, evolutionary algorithms (EAs) are popular choices for solving static optimization problems due to their capability to deal with those that have both standard and non-standard mathematical properties [4–6]. Although they are increasingly used to solve DCOPs, mechanisms for maintaining diversity have played a key role in solving them. Due to the weaknesses of EAs, such as their slow convergence rates and other performance factors, new techniques, including prediction methods [7–11], memory-based concepts [1,12] and multi-population approaches [13,14], have been combined with them. Although they are commonly used for unconstrained dynamic problems, they can be applied to constrained ones by incorporating appropriate mechanisms that can facilitate reaching feasibility, such as repairing and/or tracking the feasible region. However, some are computationally expensive [15,16] and/or developed for only specific types of changes [17] and may lead to loss of diversity [18].

Recently, Hamza et al. [5] proposed a sensitivity-based approach that detects the active constraints and subsequently takes action when a change in the right-hand side of a constraint function is found. According to the literature, most research work has been conducted on changes in the objective function but very little on those in the coefficients of the constraint functions in DCOPs. Any change in an objective function may change the location of its optimal solution but any in a constraint one will influence both the dynamics of the feasible space and the location of its new optimal solution. Therefore, this research topic is crucial for DCOPs. It could be challenging when the constraint functions are non-linear with either a tiny feasible space or multiple disjointed feasible regions while a quick environmental change adds extra complexity.

To advance the literature on DCOPs with changes in their constraint functions, in this paper, an approach based on EAs and a number of mechanisms for handling DCOPs is proposed. For any change in the coefficients of the constraint functions, some variables' values must be changed to maintain feasibility and reach the new optima. To achieve this, the proposed approach must identify the changed variables and the constraints affected. Changes in the variables' values will be quite different depending on their linear or non-linear effects. Therefore, each variable's contribution to each constraint and the change rate should be identified. This information will be helpful in designing the proposed approach which, in turn, will help to evolve a population for maintaining feasibility or converging quickly to reasonable solutions. The algorithm is tested on 13 newly designed DCOPs, with 20 environmental changes and three different scenarios. Its results demonstrate its efficiency in enhancing the performances of EAs and its capability to outperform existing methods. Note that new test problems are generated as no existing benchmark problems consider dynamic changes in the coefficients of the decision variables in the constraint functions.

In summary, the contributions of this paper are introducing: (1) an efficient evolutionary framework for dealing with both linear and non-linear DCOPs; (2) a mechanism capable of identifying dynamic changes in the coefficients of the decision variables; (3) a methodology for determining the rates of change in the decision variables and, in turn, adapting its population efficiently after every change in an environment; and (4) a new set of benchmark problems that encapsulate the new type of DCOPs. Finally, it is worth mentioning that this work is significantly different from that of Hamza et al. [5] which deals with changes in only the right-hand sides of the constraint functions.

This paper is organized as follows. In Section 2, a brief review of related work on DCOPs is provided. The mathematical model of the DCOPs considered in this paper is shown in Section 3. In Section 4, the components of the proposed approach are detailed. The computational results are discussed in Section 5 and the method adopted using other EAs in Section 6. Comparisons of the proposed and other state-of-the-art methods are presented in Section 7, with conclusions and suggested future work in Section 8.

2. Literature review

There are many research studies in the literature on solving unconstrained dynamic optimization problems (DOPs) [19,20]. The methodologies proposed for solving DCOPs are usually extensions of those for DOPs with some additional mechanisms for handling constraints and repairing feasibility. In them, EAs are commonly used as basic search algorithms [18,21].

Related work can be classified in different groups, that is, (1) methods that use a repair mechanism to move to the feasible region, (2) methods that try to maintain diversity (i.e., using multiple populations), (3) memory-based approaches, (4) techniques that adapt an EA designed for static problems to deal with DCOPs, (5) prediction-based approaches, (6) methods that use multiple populations and (7) hybrid approaches that combine more than one concept under a general framework. However, as reported in [5], they all suffer from various shortcomings. Repair-based ones are computationally expensive, especially if the feasible region is small. Memory-based ones experience difficulty defining the optimal memory size and handling environments with no historical similarity while prediction ones fail to deal with problems with large random changes. Furthermore, existing works do not consider DCOPs with complex mathematical properties, i.e., non-linearity, a disjointed feasible region or rotation [22,19,23]. Below are summaries of some of these methods.

[1] this was covered in the Wall Street Journal on July 10, 2021 (Link).

In [21], a straightforward mechanism called the infeasibility-driven EA (IDEA) algorithm is adopted to solve DCOPs. It simply re-evaluates the entire population after every change and uses information from infeasible solutions to guide the search process. It performs well in comparison with other state-of-the-art algorithms for two DCOPs [24]. However, as the changes detected are based on a random individual, they could be inaccurate and affect the overall detection performance.

The abstract memory strategy in [17] [12] uses two memory schemes (blending and censoring) to store promising solutions and their information. Although the algorithm performs well, it is not efficient for cyclic environments. This work is extended in [25] to test more search strategies using memories to tackle dynamic environments.

A gravitational search algorithm (GSA) with a repair method is used in [26]. It tries to move infeasible points towards their closest feasible ones. Ameca et al. [15,16,27] use a mutant repair method to enhance infeasible solutions, but it is too computationally expensive for problems with tiny feasible regions. Some advantages of using repairing methods are highlighted in [22].

To help an optimizer find multiple local solutions, a speciated evolution with a local search (SELS) is introduced in [28]. It combines a change detection mechanism and random immigrants (RI) [29] with a feasibility rule constraint-handling technique (CHT) [30]. However, it is noted that it fails to maintain sufficient diversity.

To track a moving feasible region, in[14], a dynamic species-based particle swarm optimization (DSPSO) algorithm [31] is used, with a mix of different mechanisms, to locate and track multiple feasible regions. This approach is capable of maintaining good diversity during the search process.

A framework for a dynamic benchmark generator with linear constraint changes is developed in [23]. To solve the problems, the differential evolution (DE) algorithm adopted uses a change detection mechanism and different CHTs. However, the benchmarks generated introduce dynamic changes into the environments for only linear constraints.

To help EAs predict the directions and estimate the number of movements of solutions after every change in an environment, in [5], a sensitive constraint-detection approach is proposed. Although the algorithm shows promising results, changes occur on only the right-hand sides of some of the constraints. However, we assume changes occur in the coefficients of the constraint functions.

3. Mathematical model

As previously mentioned, in this research paper, we are concerned with changes in the coefficients of the decision variables in the constraint functions of DCOPs which can be formulated as

$$\min_{\vec{x} \in F(t) \subseteq S} f(\vec{x}, t)$$

subject to

$$g_k\left(\vec{\widetilde{C}} \cdot \vec{x}, t\right) \leqslant b_k(t), \forall k = 1, 2, \ldots, K$$

$$h_e\left(\vec{\widetilde{C}} \cdot \vec{x}, t\right) = 0, \forall e = 1, 2, \ldots, E$$

$$l_j(t) \leqslant x_j(t) \leqslant u_j(t) \quad (1)$$

where $f(\vec{x}, t)$ is the fitness value of $\vec{x}$ at a discrete time (t), $t = \{1, 2, \ldots, T\}$, $\vec{x} = [x_1, x_2, \ldots, x_D]$ a vector with D decision variables, where $l_j(t)$ and $u_j(t)$ are the lower and upper limits of x_j, respectively, $j = 1, 2, \ldots, D$, $F(t) = \{\vec{x} | \vec{x} \in R^n, g_k\left(\vec{\widetilde{C}} \cdot \vec{x}, t\right) \leq 0, h_e\left(\vec{\widetilde{C}} \cdot \vec{x}, t\right) = 0\}$ the feasible space that varies depending on the time variable (t), b_k the boundary of $g_k, (b_k = 0)$, $S = \prod_{j=1}^{D} [l_j, u_j]$ the search space, $\vec{\widetilde{C}} = \left[\widetilde{C}_1, \widetilde{C}_2, \ldots, \widetilde{C}_D\right]$ the coefficients of $\vec{x}$ which may change over $t, g_k\left(\vec{\widetilde{C}} \cdot \vec{x}, t\right)$ the k^{th} inequality constraint, $h_e\left(\vec{\widetilde{C}} \cdot \vec{x}, t\right)$ the e^{th} equality constraint, and $b_k(t)$ the boundary of g_k. The "·" symbol in $\left(\vec{\widetilde{C}} \cdot \vec{x}\right)$ refers to the dot product of both vectors which results in $\widetilde{C}_1 x_1 + \widetilde{C}_2 x_2 + \widetilde{C}_3 x_3 + \ldots + \widetilde{C}_D x_D$.

It is assumed that each change happens in the coefficient $\left(\widetilde{C}_j(t)\right)$ of a constraint function at each instant of a problem changes over time as $\widetilde{C}_j(t) = \widetilde{C}_j(t = 1) \times r_j(t)$, where $r_j(t)$ is a time-dependent parameter that determines how the coefficients will change.

4. Proposed algorithm

To achieve the overall objective of this research, that is, solving DCOPs in quickly-changing environments, an evolutionary approach for changing constrained optimization (EACCO) is introduced. EACCO tries to detect the variables that change, their contributions, whether there are linear or non-linear effects and the amount of change that occurs (i.e., the change in $\widetilde{C}_j$). We believe that attaining these goals can help EACCO adapt its population quickly to adapt rapidly to a new environment and, subsequently, obtain good solutions. Note that EACCO does not have access to the mathematical properties of the problems at hand.

4.1. EACCO

As presented in Fig. 1 and Algorithm 1, EACCO begins by generating an initial random population (X) of size NP solutions, such that $X = \left(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{NP}\right)$, where each solution $(\vec{x})$ is generated as

$$x_j = l_j + \ rand \ \times \ \left(u_j - l_j\right) \quad (2)$$

where *rand* is a uniformly distributed random number $(\in [0, 1])$. Then, a multi-operator DE (MODE)[4] is used to evolve X (as explained in Section 4.3) until a change in the environment occurs, i.e., $t = t + 1$. Consequently, once the optimization process for the current t is completed (i.e., by reaching the maximum number of fitness evaluations (FEs)

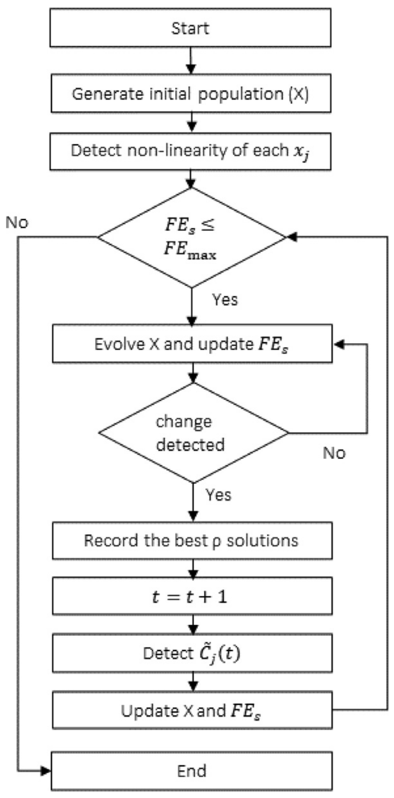
Start
Generate initial population (X)
Detect non-linearity of each x_j
$FE_s \leq FE_{max}$
No
Yes
Evolve X and update FE_s
change detected
No
Yes
Record the best ρ solutions
$t = t + 1$
Detect $\tilde{C}_j(t)$
Update X and FE_s
End

Fig. 1 steps of the proposed EACCO.

or a dynamic change is detected), the best ρ solutions in the current environment are recorded.

Subsequently, the algorithm detects which $\widetilde{C}_j$ has changed as follows. Firstly, after every change in the environment (subSection 4.2), it finds the constraints that have changed. Secondly, it determines the common variables in these constraints and calculates every new $\widetilde{C}_j$. As some variables may be non-linear, which affects calculations of the changes in $\widetilde{C}_j$, it also detects the non-linearity of each variable per constraint. This process can be performed only once, i.e., before the start of the evolutionary process (at $t = 1$) (Section 4.2.1).

After detection of the new $\widetilde{C}_j$, the action taken to update X falls in two categories: (1) if this change occurred previously, i.e., the $\widetilde{C}_j(t)$ of all the variables changed is similar to the corresponding $\widetilde{C}_j$ in a previous environment $(1, \ldots, t-1)$, the worst ρ solutions in the current X_t are replaced with the best ones from those similar environments; or (2) if this change is new, the amount of change in $\widetilde{C}_j(t)$ is detected with the best ρ solutions from previous environments passed to the new environment under the condition that their values are changed based on the change rate in $\widetilde{C}_j(t)$ (Section 4.2.3).

As MODE evolves a new population until another change is detected, the above steps are conducted until no more changes occur or the maximum number of FEs is reached.

Algorithm 1. EACCO

```
Algorithm 1 EACCO
$t \leftarrow 1$ (current environment);
$FEs \leftarrow 0$ (current fitness evaluation);
$FEs_{max} \leftarrow$ total number of fitness evaluations;
$X \leftarrow$ randomly generated population;
Check/record non-linearity of each $x_j$ (section 4.2.1);
while $FEs \leq FEs_{max}$ do
    Evolve $X$ (section 4.3);
    Update $FEs$;
    if change detected then
        Record best $\rho$ solutions;
        $t \leftarrow t + 1$;
        Detect change in $\widetilde{C}_j(t)$ (Algorithm 2);
        Update $X$ (section 4.2.3);
        Update $FEs$;
    end
end
```

Algorithm 2. Detection of variables and calculation of coefficients

Algorithm 2 Detection of variables and calculation of coefficients

```
$D \leftarrow$ total number of decision variables;
$\overrightarrow{x} = [x_1, x_2, \ldots, x_D]^T$;
$g_k(\overrightarrow{C} \cdot \overrightarrow{x}, t) \leftarrow k^{th}$ inequality constraint at $t$;
$CL \leftarrow \{\}$ %list of changed constraints;
$MVL \leftarrow \{\}$; %mutual variables in $CL$;
for $k = 1 : K$ // Identify changed constraints do
    Calculate $g_k(\overrightarrow{x}_{best}, t)$;
    if $g_k\left(\overrightarrow{C} \cdot \overrightarrow{x}_{best}, t\right) - g_k\left(\overrightarrow{C} \cdot \overrightarrow{x}_{best}, t-1\right) \neq 0$ then
        $CL \leftarrow CL \cup k$;
    end if
end
foreach $k \in CL$ // Find mutual variables list ($MVL$) do
    for $j = 1 : D$ do
        $\overrightarrow{V}_1 \leftarrow [1e-04, 1e-04, ..., 1e-04]$;
        $\overrightarrow{V}_2 \leftarrow \overrightarrow{V}_1$ except $\overrightarrow{V}_{2,j}$ is set to a large value;
        if $g_k(\overrightarrow{V}_1, t) \neq g_k(\overrightarrow{V}_2, t)$ then
            $MVL \leftarrow MVL \cup j$;
    end
end
foreach $j \in MVL$ //Calculate coefficient values do
    $\overrightarrow{V}_3 \leftarrow \overrightarrow{V}_2$;
    foreach $k \in CL$ do
        Calculate $g_k(\overrightarrow{V}_3, t = 1)$ and $g_k(\overrightarrow{V}_3, t)$;
        Calculate $\widetilde{C}_j(t)$ (equation 6);
    end
end
```

4.2. Detecting change

One of the essential considerations in this paper is that the algorithm does not have detailed information about the constraint functions. Instead, it can obtain information about the quality of solutions in terms of their fitness and constraint values. Therefore, to detect a dynamic change in an environment, the best individual in X is evaluated to obtain these values which are compared with those calculated for the same solution in the previous generation. If they are not the same, it is concluded that there is a change in the environment, and the following steps are carried out.

4.2.1. Detecting non-linearity

One of the contributions of this paper is the capability of the proposed approach to identify the non-linearity of each variable in each constraint. The motivation behind this is that correctly detecting non-linearity will result in better calculations of $\widetilde{C}$ which is our focus. This fact can be explained using a straightforward example. Assume that there is a cubic function $\left(\widetilde{C}x^3\right)$ where $\widetilde{C} = 2$ which is not known to the optimizer as it is a black-box problem. Assume that it is necessary to calculate the value of $\widetilde{C}$ at $x = 2$; if unaware of the non-linearity of x, a wrong detection may be made, that is, the value of the above-mentioned function $x = 2$ is $2 \times 2^3 = 16$. If assumed to be a linear function, the detection $\widetilde{C}$ is $\frac{16}{2} = 8$ which is wrong. However, if the exponent of x can be detected, $\widetilde{C}$ can be correctly predicted as $\frac{16}{2^3} = 2$.

Bearing in mind we deal with black-box problems, to detect the non-linearity of a decision variable, firstly, for each x_j, its value is set as a large one (i.e., $1e+50$) while the values of all the other decision variables are set to a small value, i.e., for $(x_{j=1}), \overrightarrow{x}_{base,1} = [1e+50, 1e-20\ldots, 1e-20]$.Then, the corresponding constraint values of this $\overrightarrow{x}_{base,j}$ are calculated, that is, $g_k(\overrightarrow{x}_{base,j}), \forall k = 1,2,\ldots,K$. Finally, the exponent (γ) of the j^{th} decision variable in each constraint is calculated by

$$\gamma_{j,k} = \left\lfloor log \mid \left(\frac{g_k(\overrightarrow{x}_{base,j}) \mid}{log(x_j)}\right)\right\rfloor \tag{3}$$

Below is an example that clarifies how this process works.

Assume that there are the two constraints:

$$g_1(\overrightarrow{x}) = 10x_1^2 + x_2 \tag{4}$$

$$g_2(\overrightarrow{x}) = x_1^3 + x_2^4 \tag{5}$$

Firstly, γ_1 is identified by setting $\overrightarrow{x}_{base,1} = [1e+50, 1e-20]$. As this makes $g_1 = (1e+101) + (1e-20) \approx 1e+101$ and $g_2 \approx 1e+150, \gamma_{j=1,k=1} = \lfloor\frac{log|(1e+101)|}{log(1e+50)}\rfloor = 2$ and $\gamma_{j=1,k=2} = \lfloor\frac{log|(1e+150)|}{log(1e+50)}\rfloor = 3$.

Similarly, to identify $\gamma_2, \overrightarrow{x}_{base,2}$ is set to $[1e-20, 1e+50]$. Consequently, $g_1 = (1e-20)^2 + (1e+50) \approx 1e+50$ and $g_2 \approx (1e+50)^4 = 1e+200$. Therefore, $\gamma_{j=2,k=1} = \left\lfloor\frac{log|(1e+50)|}{log(1e+50)}\right\rfloor = 1$ and $\gamma_{j=2,k=2} = \left\lfloor\frac{log|(1e+200)|}{log(1e+50)}\right\rfloor = 4$.

4.2.2. Calculating $\widetilde{\boldsymbol{C}}_j(\boldsymbol{t})$

This process has three consecutive steps: (1) identifying the changed constraints (CL); (2) finding the mutual variables list (MVL) in those constraints; and (3) predicting the new $\widetilde{C}_j(t)$ for every variable in the MVL.

1. identifying the changed constraints is achieved by calculating the value of each constraint before $\left(g_k\left(\overrightarrow{x}_{best}(t-1)\right)\right)$ and after $\left(g_k\left(\overrightarrow{x}_{best}(t)\right)\right), \forall k = 1,2,\ldots,K$ when a change occurs. For every constraint (k), if $g_k\left(\overrightarrow{x}_{best}(t)\right) - g_k\left(\overrightarrow{x}_{best}(t-1)\right) \neq 0$, the constraint index ($k$) is added to the list of changed constraints ($CL = \{\}$), i.e., $CL = CL \cup k$.
2. Finding MVL involves defining two vectors ($\overrightarrow{V}_1$ and $\overrightarrow{V}_2$) for each variable, where all the values of the former are set to small ones, i.e., $1e-04$ and those for $\overrightarrow{V}_2$ to $1e-04$, except for the j^{th} element which is set to a large one. Then, for each $k \in CL$, the constraint values for both vectors are calculated, i.e., $g_k\left(\overrightarrow{V}_1, t\right)$ and $g_k\left(\overrightarrow{V}_2, t\right)$. If these values are different, this means that the j^{th} variable is potentially active and, therefore, is added to the MVL, i.e., $MVL = MVL \cup j$.
3. Predicting $\widetilde{C}_{j,k}(t)$, for every variable in the MVL, a new vector ($\overrightarrow{V}_3$) is defined with all its values set to small ones except for the j^{th} element which is set to a large one. Next, the constraint values of each $j \in MVL$ and $k \in CL$ at $t = 1$ $(g_k\left(\overrightarrow{V}_3, t=1\right))$ and the current t $(g_k\left(\overrightarrow{V}_3, t\right))$ are calculated. Subsequently, $\widetilde{C}_{j,k}(t)$ is calculated as follows, noting that, for simplicity, its value is rounded to 2 decimal places.

$$\widetilde{C}_{j,k}(t) = \frac{\left(g_k\left(\overrightarrow{V}_3, t\right)\right)^{\frac{1}{\gamma_{j,k}}} - \left(g_k\left(\overrightarrow{V}_3, t=1\right)\right)^{\frac{1}{\gamma_{j,k}}}}{\left(g_k\left(\overrightarrow{V}_3, t=1\right)\right)^{\frac{1}{\gamma_{j,k}}}} \tag{6}$$

As previously discussed, it is assumed that, at each instant of time (t), the values of the changes in the coefficient ($\widetilde{C}_j$) of the same variable (j) are the same in each changed constraint. However, if the detection mechanism developed detects different $\widetilde{C}_j$ in different constraints, a consensus is reached by setting $\widetilde{C}_j$ to the value of the changes detected in most of them; for example, if the method detects a change in $\widetilde{C}_1$ with values of 1.5, 2.5, 2.5 and 2.5 for four constraints, the final change value considered is 2.5 as it occurs in three of the constraints.

4.2.3. Updating population

Once the new $\widetilde{C}_j$ values are detected, a similarity check is carried out to find a previous environment that has the same $\widetilde{C}_j$. If a similar environment is detected, its best ρ solutions are transferred to the current population (X_t) and replace the worst ones. However, if this change is new (i.e., no similar previous environments are detected), the amount of change in $\widetilde{C}_j(t)$ is detected and the best ρ solutions from every previous environment passed to the new one (by replacing the worst solutions in the current population) under the condition that their values are updated based on the new $\widetilde{C}_j$, that is, $x_{i,j}(t) = \frac{\widetilde{C}_j(\lambda)\times x_{b,j}(\lambda)}{\widetilde{C}_j(t)} \forall b \in \{1,2,\ldots,\rho\}, \lambda \in \{1,2,\ldots,t-1\}$, where $i = NP - b \times \lambda + 1$ (i.e., the worst solutions in the current population are replaced) and $j \in MVL$.

4.3. Optimization method: multi-operator DE (MODE)

Given its success in solving constrained problems, MODE [4] (an earlier algorithm proposed by the authors) is used to evolve X while EACCO is generic and can be adopted with any other optimization algorithm.

In short, MODE starts with NP randomly generated solutions. Then, X is randomly divided into two groups of solutions of equal size. The first is evolved by the rand-to-$p$best/bin DE variant and the other by the current-to-$p$best/bin one (where bin refers to a binomial crossover). Subsequently, the size of each group is updated based on how successful each DE variant is at achieving good solutions (i.e., considering the two criteria of diversity and quality). MODE also uses an information-sharing scheme among both groups of solutions which enhances its performance in attaining better solutions and a self-adaptive mechanism to update both the F and Cr parameters. Finally, the survival of solutions in MODE is based on three well-known conditions: (1) of the feasible solutions, the one with the lowest fitness value is always preferred; (2) a feasible point is considered superior to an infeasible one; and; (3) for infeasible solutions, the one with the least sum of constraint violations is considered the best.

In the current work, we propose a mechanism for detecting dynamic changes and a new way of updating X after every change which improves the algorithm's convergence to good solutions.

4.4. Determining time complexity

- The time complexity involved in detecting the non-linearity of each variable (subsubSection 4.2.1) is $O(KD)$.
- Algorithm 2 has three main components: (1) $O(K)$ is the time complexity of detecting which constraint has changed; (2) the computational cost of finding the mutual variables in CL is $O(KD)$; and (3) the time complexity for calculating the new $\widetilde{C}_{j,k}(t)$ is $O(KD)$. This makes the time complexity equal to $O(K) + 2 \times O(KD)$. By focusing on the dominant term and removing any constants, the overall time complexity is $O(KD)$.
- To update X, $O(tD)$ is the time complexity for finding a similar environment. The computational cost of copying solutions from previous environments is $O(\rho tD)$. As $\rho \times t$ cannot exceed the size of the population for any scenario, the time complexity is $O(NP \times D)$ which is equal to the cost of initializing a population of solutions using any optimization algorithm.

Given these values ($O(KD)$ (point 1 above) and $O(KD)$ (point 2 above)), the extra time complexity the proposed mechanism will add to any optimizer is $O(KD)$. Assuming that any optimizer needs to re-initialize its population after every change, the extra time is not added to the complexity of point 3 above. It should be recalled that the above steps are only carried out at the start of every environmental change.

5. Experimental Study

To the best of authors' knowledge, no existing benchmark problems involve dynamic changes in the coefficients of the decision variables in the constraint functions. Therefore, we transform 13 test problems from the CEC2006 COPs [32] into dynamic ones, as discussed in subSection 5.1. They are $G_{01}, G_{02}, G_{04}, G_{06}, G_{07}, G_{08}, G_{09}, G_{10}, G_{12}, G_{16}, G_{18}, G_{19}$ and G_{24} which are selected because they have different mathematical properties.

5.1. Changes in coefficients of constraints

The proposed change is applied to the coefficients of the inequality constraints as follows.

Given the general static form of each problem, it is set as

$$\min_{\vec{x} \in F \subseteq S} f(\vec{x})$$

subject to

$$g_k\left(\vec{\widetilde{C}} \cdot \vec{x}\right) \leqslant b_k$$

$$h_e\left(\vec{\widetilde{C}} \cdot \vec{x}\right) = 0 \tag{7}$$

where the feasible space $(F) = \{\vec{x} | \vec{x} \in R^n, g_k(\vec{x}) \leqslant b_k, h_e(\vec{x}) = 0\}$ and $b_k = 0$. The coefficient of the inequality constraint $\widetilde{C}_j(t)$ changes, as explained in Section 4. Every dynamic parameter $(r_j(t))$ has a different effect on a constraint, e.g., a change in $r_j(t)$ would re-scale it by shrinking or magnifying it over time.

Different types of dynamic drivers can be used to generate dynamic changes, such as linear, cyclic, decay oscillation, random and chaotic ones [24,33]. In this paper, a cyclic change is applied to the coefficients of the decision variables. In this type of change, the environment periodically appears based on its cycle length (i.e., one environment may be repeated only once). Note that, although no knowledge of such changes is assumed, the algorithm tries to detect a change automatically and if there are any similar previous environments.

A detailed description of each test problem and the settings of its dynamic parameters are provided in the supplementary document in Appendix 1.

The following three scenarios are considered.

- Scenario 1: the coefficient $(\widetilde{C}_j(t))$ of only one variable is changed by $r_j(t)$ at each instant of time.
- Scenario 2: the coefficient of two variables is changed at each instant.
- Scenario 3: while similar to Scenario 1, a change between environments follows a complicated pattern (see Table 1 in the supplementary document in Appendix 1) which adds more complexity to the problem.

The best-known solutions for each test problem using each scenario are shown in Table A in the supplementary document in Appendix 2. Also, the percentages of the feasible regions for all the test problems in scenario 2 for 20 environments are reported in Table 2 in the supplementary document in Appendix 1.

5.2. Experimental settings

Each problem was run 25 times for 20 environments ($T = 20$). The frequency of change (FE_{change}) in each environment was set to 1000 FEs, the change severity of the constraint (k) to 1 and the cycle length to 10.

5.3. Parameter settings

The same settings for MODE as recommended in [4] were used, with $NP = 100$ and $\rho = 3$ the best solutions. Note that ρ is the only extra parameter introduced in this paper and is analyzed in subsubSection 5.5.2.

5.4. Performance measures

The well-known performance measures, the average fitness value and average feasibility rate FR were used to measure the algorithms' performances. It is worth noting that, if all the algorithms compared could not obtain feasible solutions, the average sum of their constraint violations was considered a measure of the quality of their solutions. Also, the non-parametric Wilcoxon signed-rank test [34,35] with a 5% significance level was used.

All the algorithms were compared based on (1) the modified offline errors of the DCOPs ('offline error' for short) and (2) their normalized scores. The former was calculated as

$$e = \frac{1}{T}\sum_{t=1}^{T}(\vec{x}^{*}(t) - \vec{x}_{best}(t)),$$

where $\vec{x}_{best}(t)$ represents the best solution (feasible or infeasible) found so far by an algorithm in the current (t) environment, $\vec{x}^{*}(t)$ the global solution in that environment and T the total number of environments.

The normalized score was calculated as:

$$S_{norm}(I) = \frac{1}{G}\sum_{m=1}^{G}\frac{|e_{max}(m) - e(I,m)|}{|e_{max}(m) - e_{min}(m)|}, I = 1, 2, \ldots, N$$

where G is the number of test problems, N the number of algorithms compared, $e(I,m)$ the offline error of algorithm I in problem m and $e_{max}(m)$ and $e_{min}(m)$ the largest and smallest offline errors, respectively. The value of the normalized score for each algorithm varied between zero and one, with the algorithm having the highest score considered to exhibit the best performance.

5.5. Analyses

In this subsection, the benefits of the proposed approach and its components are discussed and analyzed.

5.5.1. Ascertaining benefits of proposed approach (EACCO)

To achieve the aim of showing the benefits of the proposed method, three variants of MODE were run: (1) using EACCO to handle dynamic changes (denoted as '*MODE-EACCO*'); (2) adopting all the solutions obtained before a change as the new ones for MODE to evolve in a new environment ('*MODE*');); and (3) re-initializing the entire population after every change ('*MODE-reinitialization*'). The results for scenarios 1 and 2 are shown in Tables B and C, respectively, in the supplementary document in Appendix 2, with summaries of the comparisons based on the average results obtained provided in Table 1.

The results in Table 1 indicate the capability of MODE-EACCO to outperform the other two variants when a change occured in the coefficient(s) of one or more variables. This was because it could move quickly to feasible solutions once there was a change in the environment. Table 2 shows that MODE-EACCO had the best average numbers of offline errors for all the test problems. Also, readers can observe that the re-initialization mechanism was not a good choice for such problems given the rapid changes in the environments and the times required to converge to good solutions.

The FR results shown in Fig. 2 demonstrate that the proposed approach attained higher rates of feasible solutions than the other two. Also, it is evident in Table 3 that it obtained better-normalized errors than them.

The key reason for the good performance of EACCO was its efficiency in detecting the variable(s) that changed, their non-linearity effects and amounts of change.

5.5.2. Analyzing effect of ρ

As previously discussed, ρ represents the best solutions recorded at the end of each environment and was initially set to 3. Its effect was analyzed by changing its value to

Table 1 Summary of comparisons of MODE-EACCO, MODE and MODE-reinitialization based on average results.

	MODE-EACCO vs. MODE					MODE-EACCO vs. MODE-reinitialization				
Scenarios	Better	Equal	Worse	p-value	Decision	Better	Equal	Worse	p-value	Decision
Scenario 1	195	13	52	8.09E-14	+	247	13	0	4.81E-34	+
Scenario 2	158	13	89	0.003403	+	246	13	1	2.44E-39	+

Table 2 Average numbers of offline errors obtained by MODE-EACCO, MODE and MODE-reinitialization for scenarios 1 and 2.

Scenarios	Scenario 1			Scenario 2		
Algorithms	MODE-reinitialization	MODE-EACCO	MODE	MODE-reinitialization	MODE-EACCO	MODE
G01	1.17E + 01	**1.64E + 00**	1.78E + 00	1.54E + 01	**2.05E + 00**	2.23E + 00
G02	5.14E-01	**1.63E-01**	1.98E-01	5.18E-01	**1.81E-01**	1.92E-01
G04	2.48E + 02	**7.30E + 01**	8.09E + 01	4.53E + 02	**3.01E + 02**	3.08E + 02
G06	3.42E + 02	**5.57E + 01**	6.95E + 01	2.49E + 02	**4.17E + 01**	8.19E + 02
G07	7.00E + 02	**1.93E + 01**	2.27E + 01	6.64E + 02	**2.91E + 01**	3.94E + 01
G08	4.88E-01	4.48E-03	**3.55E-03**	1.86E + 00	**8.45E-02**	3.88E-01
G09	1.49E + 02	**7.13E + 00**	8.17E + 00	1.39E + 02	**8.96E + 00**	1.10E + 01
G10	5.83E + 03	**1.14E + 03**	1.34E + 03	6.75E + 03	**5.38E + 02**	6.20E + 02
G12	2.12E-03	**3.50E-05**	2.30E-04	4.93E-03	**7.36E-04**	2.44E-03
G16	4.42E-01	**8.72E-02**	1.17E-01	2.38E + 00	**2.93E-01**	3.03E-01
G18	7.26E + 01	**1.66E + 00**	1.72E + 00	8.76E + 01	**2.31E + 00**	2.60E + 00
G19	8.57E + 02	**9.60E + 01**	1.08E + 02	7.40E + 02	**7.72E + 01**	8.25E + 01
G24	7.72E-03	**3.00E-03**	3.32E-03	1.26E-02	**6.62E-03**	1.71E-02

$1, 2, \ldots, 10$. The fitness value results for scenarios 1 and 2 are reported in Tables D and E, respectively, in the supplementary document in Appendix 2. A Friedman test was carried out to rank these variants, with a summary of all the rankings based on the average results shown in Table 4. They indicate that setting the values of ρ to 6 and 4 were best for scenarios 1 and 2, respectively. As no one value suited both of them, based on their average rankings, it was decided that setting to a value of 4 achieved their best rankings.

5.5.3. *Determining effect of rapid change*

To check the performances of the MODE-EACCO approach for quickly changing environments, EACCO was run with $FE_{change} = 500$ FEs, and its results compared with those obtained by MODE and MODE-reinitialization. Table 5 presents the average numbers of offline errors obtained by all three variants. It can be seen that MODE-EACCO had the best values for all the test problems. Not surprisingly, in such quickly changing environments, MODE outperformed the same algorithm using a re-initialization method (MODE-reinitialization).

As previously reported, the superiority of the proposed approach (MODE-EACCO) was due to its capability to detect the rates of change that occurred and, in turn, quickly converge to the feasible region, as confirmed by the average FR of each algorithm (Fig. 5). Consequently, all these promising results led it to be ranked best based on the normalized score each algorithm obtained, as reported in Table 6. More detailed results for scenarios 1 and 2 are shown in Tables F and G, respectively, in the supplementary document in Appendix 2. (see Table 7)

Looking at the convergence pattern depicted in Fig. 3 for G06 with $FE_{change} = 500$, it is clear that MODE-EACCO can achieve better solutions than MODE. Note that MODE-re-initialization could not obtain any feasible solutions in any environment. Furthermore, considering $FE_{change} = 1000$, Fig. 4 also demonstrates the superiority of MODE-EACCO, while MODE re-initialization is still the inferior method.

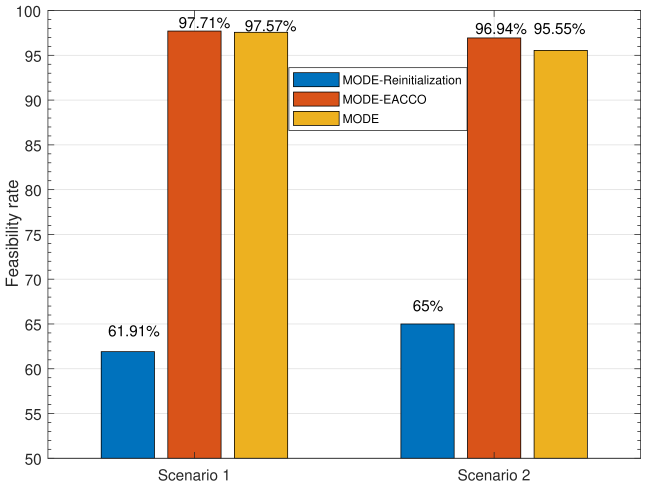

Fig. 2 The rate of feasible solutions, over 25 runs, achieved by MODE-reinitialization, MODE-EACCO and MODE for scenarios 1 and 2, with $FE_{change} = 1000$.

Table 3 Normalized scores obtained by MODE-reinitialization, MODE-EACCO and MODE for scenarios 1 and 2.

Scenarios	MODE-reinitialization	MODE-EACCO	MODE
Scenario 1	0.00E + 00	**1.00E + 00**	9.60E-01
Scenario 2	8.96E-02	**1.00E + 00**	7.90E-01

5.5.4. *Analyzing effect of complex change patterns*

In the previous subsubsection, if the amount of change was not very large, it might have led to good solutions in the current environment if they were close to those of the immediate previous environment. Another scenario with more severe changes was used to evaluate the proposed approach (Table 1 in the supplementary document in Appendix 1). These changes meant that there was often a big change in $\left(\widetilde{C}\right)$ in consecutive environments. Note that it was assumed that a change occurred to only $\widetilde{C}$ when $FE_{change} = 500$ and $FE_{change} = 1000$.

The results presented Fig. 6 show that MODE-EACCO was still better than the other methods for most test problems with $FE_{change} = 500$ and 1000. Its average FR confirmed it was more effective than the others. Again, its normalized scores (0.987 and 0.98) for $FE_{change} = 500$ and 1000, respectively) placed it first (Table 8). The fitness values for both $FE_{change} = 500$ and 1000 are listed in Tables H and I, respectively, in the supplementary document in Appendix 2.

5.6. *Discussion of performance of detection mechanism*

To demonstrate the capability of the proposed approach to detect the changed variables as well as their amounts of change and non-linearity effects, a summary of its detection accuracy

Table 4 Friedman rankings for MODE-EACCO algorithm with different values of ρ

ρ	1	2	3	4	5	6	7	8	9	10
Scenario 1	6.96	6.13	5.50	5.10	5.34	**4.53**	5.18	5.03	5.48	5.76
Scenario 2	6.35	5.59	5.36	**4.62**	4.57	5.23	5.10	5.43	6.28	6.48
Average ranking	6.66	5.86	5.43	**4.86**	4.95	4.88	5.14	5.23	5.88	6.12

Table 5 Average numbers of offline errors obtained by MODE-EACCO, MODE and MODE-reinitialization for scenarios 1 and 2 (500FEs).

Scenarios	Scenario 1			Scenario 2		
Algorithms	MODE-reinitialization	MODE-EACCO	MODE	MODE-reinitialization	MODE-EACCO	MODE
G01	4.40E + 01	**3.41E + 00**	3.97E + 00	1.54E + 01	**4.51E + 00**	6.02E + 00
G02	5.74E-01	**2.93E-01**	3.34E-01	5.18E-01	**3.09E-01**	3.37E-01
G04	4.01E + 02	**1.79E + 02**	2.08E + 02	4.53E + 02	**3.81E + 02**	4.09E + 02
G06	5.13E + 03	**1.21E + 02**	1.41E + 02	2.49E + 02	**8.94E + 01**	1.23E + 03
G07	5.84E + 01	4.78E + 01	**2.65E + 01**	6.64E + 02	**1.20E + 02**	1.68E + 02
G08	3.35E + 00	**1.67E-02**	4.68E-02	1.86E + 00	**2.28E-01**	9.92E-01
G09	3.16E + 02	**2.06E + 01**	2.47E + 01	1.39E + 02	**2.20E + 01**	3.38E + 01
G10	4.83E + 03	**3.00E + 03**	4.56E + 03	6.75E + 03	**1.61E + 03**	2.57E + 03
G12	8.76E-03	**2.02E-04**	1.80E-03	4.93E-03	**1.73E-03**	2.26E-02
G16	3.63E + 00	**1.53E-01**	2.45E-01	2.38E + 00	1.08E + 00	**1.07E + 00**
G18	1.81E + 02	**7.81E + 00**	9.12E + 00	8.76E + 01	**1.15E + 01**	1.38E + 01
G19	1.17E + 03	**2.35E + 02**	2.70E + 02	7.40E + 02	**2.12E + 02**	2.54E + 02
G24	2.41E-02	**4.04E-03**	7.92E-03	1.26E-02	**1.16E-02**	1.70E-02

rates for all three scenarios is provided in Fig. 7. It is clear that MODE-EACCO performed well in scenarios 1 and 3 as it could successfully detect the variables that changed, their non-linearity effects and types of changes. In scenario 2, where two variables changed, it was able to detect the right values in 11 of 13 problems but failed in two because it detected only one of the two changed variables.

Table 6 Normalized scores obtained by MODE-reinitialization, MODE-EACCO and MODE for scenarios 1 and 2 (500 FEs).

Scenarios	MODE-reinitialization	MODE-EACCO	MODE
Scenario 1	0.00E + 00	**9.49E-01**	8.75E-01
Scenario 2	1.93E-01	**9.99E-01**	6.45E-01

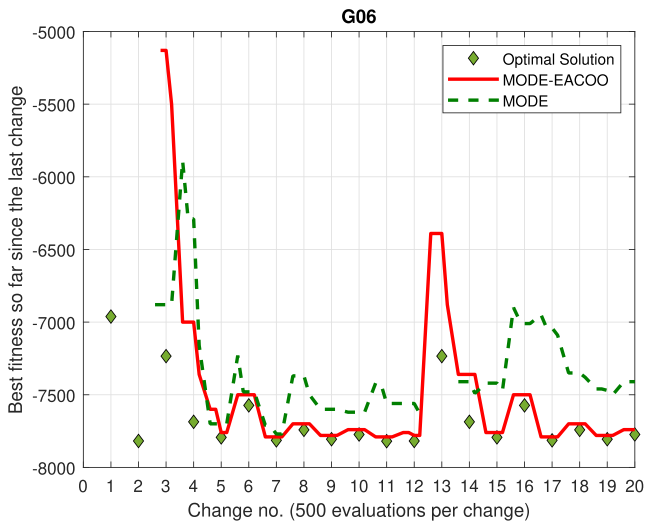

Fig. 3 Convergence pattern of MODE-EACCO and MODE in G06 for scenario 3 and $FE_{change} = 500$.

Table 7 Average offline errors obtained by MODE-EACCO, MODE and MODE-reinitialization for scenario 3 with different dynamic change rates.

FE_{change}	500			1000		
Problem	MODE-reinitialization	MODE-EACCO	MODE	MODE-reinitialization	MODE-EACCO	MODE
G01	4.51E + 01	**3.80E + 00**	4.37E + 00	1.17E + 01	2.01E + 00	**2.00E + 00**
G02	5.80E-01	**2.85E-01**	3.39E-01	5.18E-01	**1.63E-01**	2.00E-01
G04	3.93E + 02	**2.75E + 02**	3.80E + 02	2.15E + 02	1.79E + 02	**1.70E + 02**
G06	5.51E + 03	**1.11E + 02**	1.75E + 02	3.55E + 02	**4.12E + 01**	7.03E + 01
G07	**3.48E + 01**	3.95E + 01	6.18E + 01	5.23E + 02	**2.03E + 01**	2.28E + 01
G08	3.59E + 00	**6.26E-02**	1.70E-01	4.27E-01	**1.30E-02**	9.91E-02
G09	3.34E + 02	**2.85E + 01**	3.47E + 01	1.57E + 02	9.89E + 00	**9.02E + 00**
G10	3.65E + 03	**2.62E + 03**	3.06E + 03	4.21E + 03	**1.32E + 03**	1.51E + 03
G12	8.73E-03	**2.61E-04**	1.51E-02	2.20E-03	**8.88E-05**	5.98E-03
G16	4.88E + 00	**2.16E-01**	3.86E + 00	4.49E-01	**1.01E-01**	2.99E + 00
G18	1.81E + 02	**8.66E + 00**	1.17E + 01	7.32E + 01	**1.99E + 00**	2.22E + 00
G19	1.28E + 03	**2.49E + 02**	3.09E + 02	8.36E + 02	**1.09E + 02**	1.10E + 02
G24	1.42E-02	**7.03E-03**	1.25E-01	1.08E-02	**4.26E-03**	8.14E-03

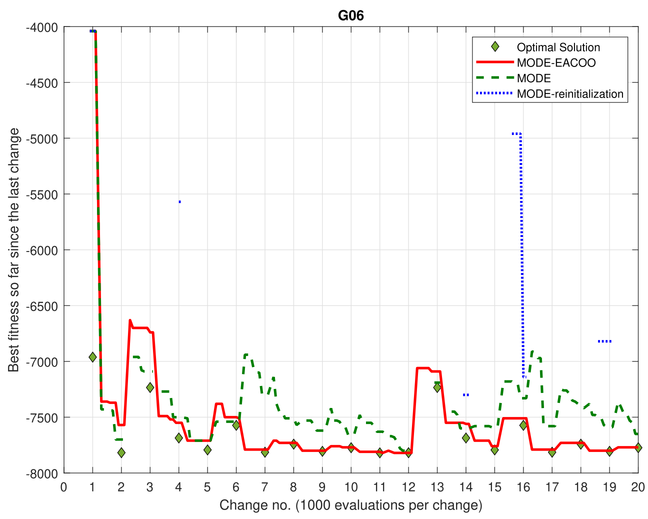

Fig. 4 Convergence pattern of MODE-EACCO and MODE in G06 for scenario 3 and $FE_{change} = 1000$.

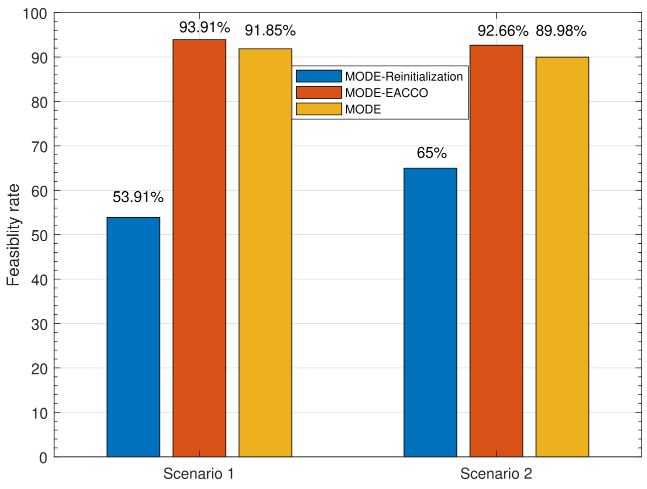

Fig. 5 Rate of the feasible solutions, over 25 runs, achieved by MODE-reinitialization, MODE-EACCO and MODE for scenarios 1 and 2, with $FE_{change} = 500$.

Table 8 Normalized scores of MODE-reinitialization, MODE-EACCO and MODE for scenario 3 with different dynamic change rates.

FE_{change}	MODE-reinitialization	MODE-EACCO	MODE
500	1.82E-01	**9.87E-01**	5.82E-01
1000	1.17E-01	**9.80E-01**	7.66E-01

6. Testing of Proposed Approach using Other EAs

As previously mentioned, the proposed approach is generic and can be adopted with other meta-heuristic techniques. To demonstrate this, the general framework was adopted with four algorithms: (1) rand-to-*p*best (DE1); (2) current-to-*p*best (DE2); (3) multi-parent crossover GA (MPC-GA) [36]; and (4) success history-based parameter adaptation DE (SHADE) [37]. DE1 and DE2 used the same parameter settings as those in [4] and SHADE and MPC-GA the same as those in [37,36], respectively. All the algorithms had $NP = 100$.

The three approaches, MODEwith and without EACCO, and with re-initialization of the entire population after every change, were run using the four algorithms mentioned above. Each was tested under all three scenarios with $FE_{change} = 1000$

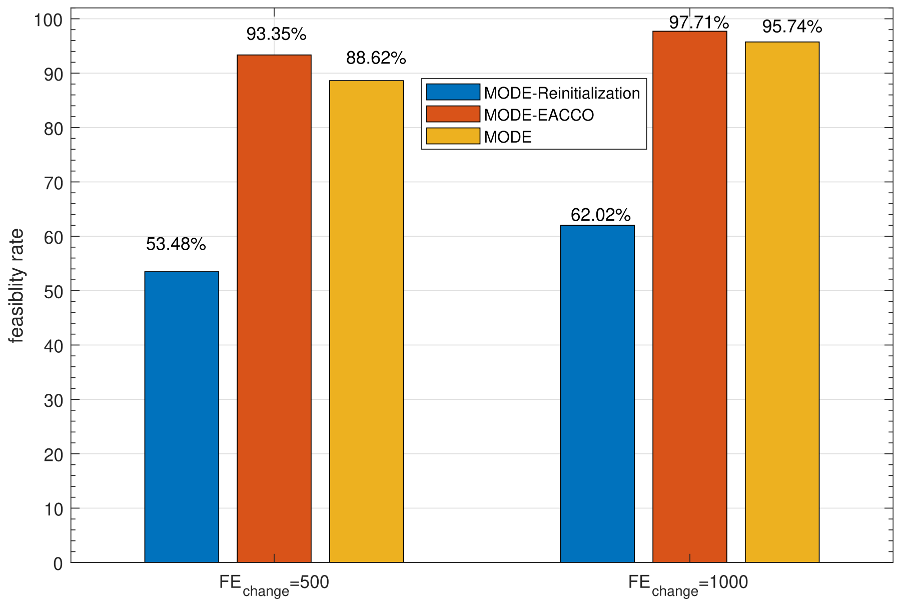

Fig. 6 Rate of the feasible solutions, over 25 runs, achieved by MODE-reinitialization, MODE-EACCO and MODE for scenario 3.

Table 9 Average feasibility rates (FR%) of different variants using DE1, DE2, MPC-GA and SHADE for all scenarios

Scenarios	Scenario1	Scenario2	Scenario3	Average FR
DE1	97.48%	95.48%	95.54%	96.09%
DE1-EACCO	**97.68%**	**96.94%**	**97.52%**	**97.29%**
DE1-reinitialization	62.25%	64.88%	62.49%	63.14%
DE2	97.42%	95.14%	95.71%	96.16%
DE2-EACCO	**97.63%**	**96.66%**	**97.57%**	**97.38%**
DE2-reinitialization	61.88%	65.63%	61.91%	63.21%
MPC-GA	87.83%	81.91%	82.58%	84.11%
MPC-GA-EACCO	**96.83%**	**95.88%**	**96.69%**	**96.47%**
MPC-GA-reinitialization	59.62%	61.68%	59.75%	60.35%
SHADE	91.82%	88.51%	89.28%	88.88%
SHADE-EACCO	**93.88%**	**93.00%**	**92.45%**	**92.60%**
SHADE-reinitialization	53.45%	59.75%	50.72%	53.99%

Table 10 Normalized scores for different variants using DE1, DE2, MPC-GA and SHADE for all scenarios.

Scenarios	Scenario1	Scenario2	Scenario3
DE1	9.45E-01	8.93E-01	7.62E-01
DE1-EACCO	**1.00E + 00**	**9.90E-01**	**9.98E-01**
DE1-reinitialization	0.00E + 00	0.00E + 00	9.11E-02
DE2	9.58E-01	8.53E-01	7.59E-01
DE2-EACCO	**9.98E-01**	**9.15E-01**	**1.00E + 00**
DE2-reinitialization	0.00E + 00	1.49E-01	1.04E-01
MPC-GA	5.58E-01	6.12E-01	4.39E-01
MPC-GA-EACCO	**9.38E-01**	**1.00E + 00**	**9.22E-01**
MPC-GA-reinitialization	2.50E-01	1.83E-01	4.22E-01
SHADE	8.89E-01	7.12E-01	5.44E-01
SHADE-EACCO	**9.40E-01**	**1.00E + 00**	**9.00E-01**
SHADE-reinitialization	0.00E + 00	1.03E-01	3.41E-01

FEs. Based on the detailed results presented in Tables J-1 to M-3 in the supplementary document in Appendix 2, comparative summaries are provided in Tables 4 to 7 in the supplementary document in Appendix 1.

These tables clearly show that the performances of the four EAs tested improved when using the proposed technique for handling dynamic changes, as confirmed by their average offline errors. Also, they were capable of achieving higher FRs (Table 9) than when using the other variants. Also, the normalized scores provided in Table 10 show that using the proposed approach was the best.

7. Comparisons of Proposed and Existing Methods

The performance of the proposed approach was compared with those of the two well-known algorithms (1) dynamic constrained optimization DE (DyCODE) [38] and (2) clustering particle swarm optimizer (CPSO) [39] [2]. Theses algorithms were run using the same settings reported in their corresponding papers, that is, $NP = 100$ and $FE_{change} = 500$ and 1000, under all the three scenarios previously mentioned. Detailed results are shown in Tables N-1 to N-6 in the supplementary document in Appendix 2.

In Tables 11,12, a summary of the average numbers of offline errors of the three approaches is presented. These results clearly reveal that MODE-EACCO was superior to both

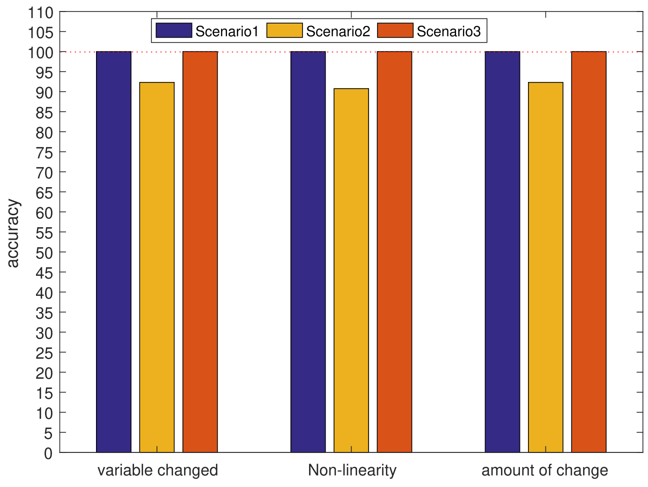

Fig. 7 Accuracy rates of detecting dynamic variables achieved by MODE-EACCO for all scenarios.

[2] the source code of both algorithms was provided by the authors of [38]

Table 11 Average numbers of offline errors obtained by MODE-EACCO and DyCODE with different dynamic change rates.

	Scenario 1				Scenario 2				Scenario 3			
FE_{change}	500		1000		500		1000		500		1000	
Problem	DyCODE	MODE-EACCO	DyCODE	MODE-EACCO	DyCODE	MODE-EACCO	DyCODE	MODE-EACCO	DyCODE	MODE-EACCO	DyCODE	MODE-EACCO
G01	1.29E + 01	**3.41E + 00**	8.34E + 00	**1.63E + 00**	1.33E + 01	**4.51E + 00**	9.59E + 00	**1.97E + 00**	1.25E + 01	**3.80E + 00**	7.43E + 00	**2.01E + 00**
G02	5.92E-01	**2.93E-01**	4.56E-01	**1.54E-01**	5.99E-01	**3.09E-01**	4.53E-01	**1.47E-01**	5.89E-01	**2.85E-01**	4.60E-01	**1.63E-01**
G04	3.54E + 02	**1.79E + 02**	1.49E + 02	**8.91E + 01**	**1.72E + 02**	3.81E + 02	**6.33E + 01**	2.97E + 02	**7.76E + 02**	2.75E + 02	**5.95E + 02**	1.79E + 02
G06	7.02E + 02	**1.21E + 02**	1.59E + 02	**5.06E + 01**	7.60E + 02	**8.94E + 01**	**4.05E + 01**	4.24E + 01	5.46E + 02	**1.11E + 02**	1.81E + 02	**4.12E + 01**
G07	5.58E + 02	**4.78E + 01**	7.86E + 01	**2.09E + 01**	1.38E + 03	**1.20E + 02**	1.03E + 02	**2.63E + 01**	**3.39E + 02**	3.95E + 01	1.81E + 02	**2.03E + 01**
G08	1.14E + 00	**1.67E-02**	2.81E-01	**4.06E-03**	**1.76E + 01**	2.28E-01	1.52E + 01	**5.34E-02**	1.03E + 01	**6.26E-02**	1.37E + 01	**1.30E-02**
G09	2.62E + 02	**2.06E + 01**	5.03E + 01	**8.03E + 00**	2.92E + 02	**2.20E + 01**	6.99E + 01	**1.02E + 01**	2.28E + 02	**2.85E + 01**	8.18E + 01	**9.89E + 00**
G10	6.24E + 03	**3.00E + 03**	3.62E + 03	**9.77E + 02**	1.39E + 04	**1.61E + 03**	9.90E + 03	**6.56E + 02**	1.48E + 03	**2.62E + 03**	2.87E + 01	**1.32E + 03**
G12	7.22E-03	**2.02E-04**	1.95E-03	**9.16E-05**	4.17E-02	**1.73E-03**	4.90E-02	**4.31E-04**	6.84E-02	**2.61E-04**	5.38E-02	**8.88E-05**
G16	6.25E-01	**1.53E-01**	2.86E-01	**8.09E-02**	1.86E + 00	**1.08E + 00**	**1.99E-01**	2.85E-01	4.55E + 00	**2.16E-01**	7.80E-01	**1.01E-01**
G18	8.76E + 00	**7.81E + 00**	3.24E + 00	**1.68E + 00**	**9.15E + 00**	1.15E + 01	3.53E + 00	**2.38E + 00**	**8.77E + 00**	8.66E + 00	3.28E + 00	**1.99E + 00**
G19	1.89E + 03	**2.35E + 02**	9.06E + 02	**9.13E + 01**	1.90E + 03	**2.12E + 02**	8.61E + 02	**8.41E + 01**	1.98E + 03	**2.49E + 02**	7.49E + 02	**1.09E + 02**
G24	2.71E-02	**4.04E-03**	1.94E-02	**3.03E-03**	1.53E + 00	**1.16E-02**	1.52E + 00	**8.57E-03**	1.48E + 00	**7.03E-03**	1.51E + 00	**4.26E-03**

Table 12 Average numbers of offline errors obtained by MODE-EACCO and CPSO with different dynamic change rates.

	Scenario 1				Scenario 2				Scenario 3			
FE_{change}	500		1000		500		1000		500		1000	
Problem	CPSO	MODE-EACCO	CPSO	MODE-EACCO	CPSO	MODE-EACCO	CPSO	MODE-EACCO	CPSO	MODE-EACCO	CPSO	MODE-EACCO
G01	3.73E + 00	**3.41E + 00**	3.93E + 00	**1.63E + 00**	6.58E + 00	**3.80E + 00**	3.77E + 00	**2.01E + 00**	**4.37E + 00**	4.51E + 00	4.73E + 00	**1.97E + 00**
G02	3.22E-01	**2.93E-01**	2.81E-01	**1.54E-01**	3.23E-01	**2.85E-01**	3.39E-01	**1.63E-01**	**2.99E-01**	3.09E-01	2.81E-01	**1.47E-01**
G04	2.76E + 02	**1.79E + 02**	2.34E + 02	**8.91E + 01**	3.67E + 02	**2.75E + 02**	2.75E + 02	**1.79E + 02**	4.75E + 02	**3.81E + 02**	3.92E + 02	**2.97E + 02**
G06	3.70E + 03	**1.21E + 02**	6.64E + 02	**5.06E + 01**	3.87E + 03	**1.11E + 02**	1.40E + 03	**4.12E + 01**	2.61E + 03	**8.94E + 01**	7.77E + 02	**4.24E + 01**
G07	7.24E + 01	**4.78E + 01**	**1.81E + 01**	2.09E + 01	4.22E + 01	**3.95E + 01**	3.09E + 01	**2.03E + 01**	**9.35E + 01**	1.20E + 02	5.45E + 01	**2.63E + 01**
G08	1.61E + 00	**1.67E-02**	5.01E-01	**4.06E-03**	3.45E + 00	**6.26E-02**	7.30E-01	**1.30E-02**	1.26E + 01	**2.28E-01**	2.38E + 00	**5.34E-02**
G09	3.90E + 01	**2.06E + 01**	1.85E + 01	**8.03E + 00**	3.79E + 01	**2.85E + 01**	2.11E + 01	**9.89E + 00**	8.08E + 01	**2.20E + 01**	4.14E + 01	**1.02E + 01**
G10	**2.69E + 03**	3.00E + 03	1.78E + 03	**9.77E + 02**	**1.67E + 03**	2.62E + 03	2.99E + 03	**1.32E + 03**	**1.36E + 03**	1.61E + 03	2.13E + 03	**6.56E + 02**
G12	5.21E-03	**2.02E-04**	9.90E-04	**9.16E-05**	4.18E-03	**2.61E-04**	1.28E-03	**8.88E-05**	7.89E-03	**1.73E-03**	3.81E-03	**4.31E-04**
G16	4.77E-01	**1.53E-01**	4.00E-01	**8.09E-02**	1.45E + 00	**2.16E-01**	4.56E-01	**1.01E-01**	4.68E + 00	**1.08E + 00**	2.43E + 00	**2.85E-01**
G18	**3.74E + 00**	7.81E + 00	1.93E + 00	**1.68E + 00**	**6.46E + 00**	8.66E + 00	2.40E + 00	**1.99E + 00**	**9.80E + 00**	1.15E + 01	4.52E + 00	**2.38E + 00**
G19	5.95E + 02	**2.35E + 02**	4.16E + 02	**9.13E + 01**	5.74E + 02	**2.49E + 02**	5.95E + 02	**1.09E + 02**	5.33E + 02	**2.12E + 02**	5.01E + 02	**8.41E + 01**
G24	1.48E-02	**4.04E-03**	3.80E-03	**3.03E-03**	1.53E-02	**7.03E-03**	4.31E-03	**4.26E-03**	**7.84E-03**	1.16E-02	**8.20E-03**	8.57E-03

Table 13 Normalized scores of MODE-EACCO, DyCODE and CPSO for different change frequencies

Scenarios	FE_{change}	DyCODE	MODE-EACCO	CPSO
Scenario 1	500	8.74E-02	**9.31E-01**	6.23E-01
	1000	0.00E + 00	**1.00E + 00**	6.57E-01
Scenario 2	500	2.71E-01	**8.63E-01**	6.50E-01
	1000	2.71E-01	**8.63E-01**	6.50E-01
Scenario 3	500	1.70E-01	**9.96E-01**	5.08E-01
	1000	1.46E-01	**9.67E-01**	6.12E-01

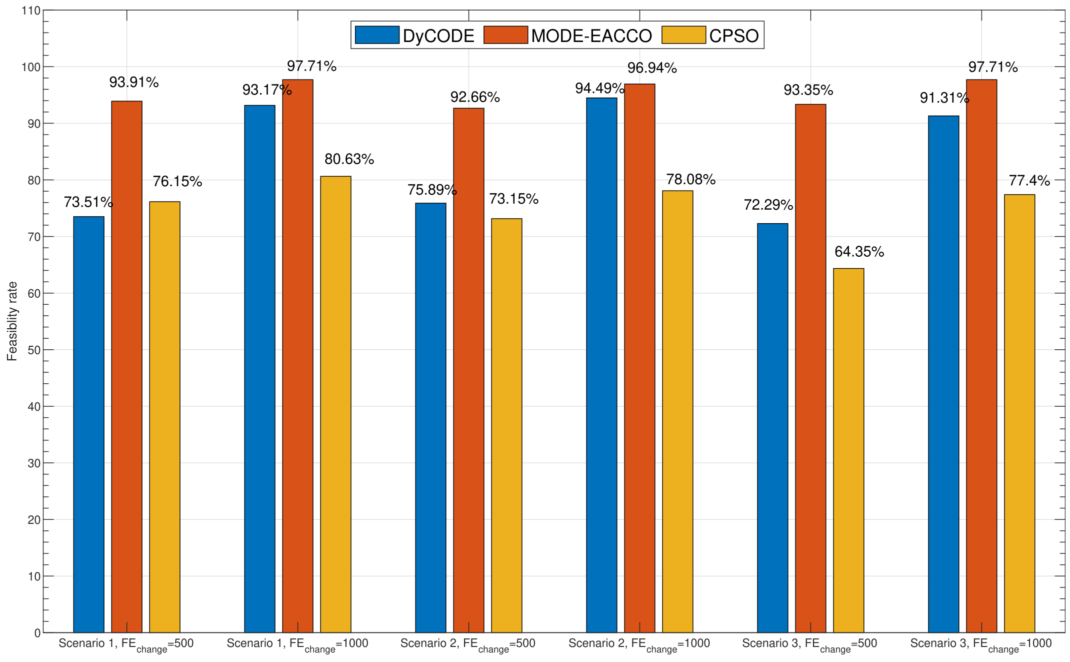

Fig. 8 Rate of the feasible solutions, over 25 runs, achieved by MODE-EACCO, DyCODE and CPSO, with different dynamic change rates.

Table 14 Summary of comparisons of MODE-EACCO, DyCODE and CPSO with different dynamic change rates.

Algorithms	Scenarios	FE_{change}	Better	Equal	Worse	p	Decision
MODE-EACCO vs DyCODE	Scenario 1	500	256	0	4	1.49E-35	+
		1000	252	0	8	2.20E-38	+
	Scenario 2	500	221	0	39	3.68E-24	+
		1000	212	0	48	8.78E-23	+
	Scenario 3	500	224	0	36	3.17E-21	+
		1000	212	0	48	3.61E-20	+
MODE-EACCO vs CPSO	Scenario 1	500	227	0	33	1.37E-26	+
		1000	251	0	9	3.704E-41	+
	Scenario 2	500	226	0	34	7.79E-26	+
		1000	205	0	55	1.794E-13	+
	Scenario 3	500	235	0	25	4.59E-31	+
		1000	241	0	19	1.847E-40	+

Table 15 average offline errors obtained by MODE-EACCO, MODE and MODE-reinitialization for solving engineering problems with different dynamic changes.

Problems	$FE_{change} = 500$			$FE_{change} = 1000$		
	MODE-reinitialization	MODE-EACCO	MODE	MODE-reinitialization	MODE-EACCO	MODE
Pressure vessel	6.7537E + 03	**8.7475E + 02**	1.1135E + 03	3.5618E + 03	**4.1126E + 02**	4.6559E + 02
Welded beam	1.6620E + 00	**5.7714E-01**	1.0230E + 00	1.0333E + 00	**2.9056E-01**	5.1688E-01
Coil spring	1.5215E-02	**2.0706E-03**	2.3389E-03	5.2309E-03	**6.6162E-04**	6.9271E-04
Speed reducer	3.9468E + 02	**1.4859E + 02**	1.9942E + 02	2.0081E + 02	**3.7446E + 01**	7.6001E + 01

Table 16 FR obtained by MODE-EACCO, MODE and MODE-reinitialization for solving engineering problems with different dynamic changes.

Problems	$FE_{change} = 500$			$FE_{change} = 1000$		
	MODE-reinitialization	MODE-EACCO	MODE	MODE-reinitialization	MODE-EACCO	MODE
Pressure vessel	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Welded beam	80.0%	**100.0%**	85.0%	95.0%	**100.0%**	**100.0%**
Coil spring	85.0%	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**
Speed reducer	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**

DyCODE and CPSO for scenario 1 with both $FE_{change} = 500$ and $FE_{change} = 1000$, except that CPSO solved only 2 and 1 problems with $FE_{change} = 500$ and $FE_{change} = 1000$, respectively. For both scenarios 2 and 3, MODE-EACCO still maintained better average offline errors than DYCODE and CPSO for most of the test problems.

The normalized scores shown in Table 13 confirm the superiority of MODE-EACCO for all the change frequencies.

Also, the calculated average FRs obtained by each algorithm (Fig. 8) demonstrate that MODE-EACCO achieved higher rates of feasibility than the others.

Considering the quality of solutions, it is apparent from Table 14 that MODE-EACCO was capable of obtaining better results than the other algorithms, with further statistical tests revealing that there were significant differences among them.

7.1. Solving Engineering Problems

In this section, the proposed approach is tested by solving four well-known engineering problems [40–42]. These problems are.

- Pressure Vessel design, where the design of a cylindrical pressure vessel capped at both ends by hemispherical heads needs to be optimized. The problem has four design variables.
- Welded beam design, which aims to minimize the fabrication cost of the welded beam. The problem has 4 continuous design variables.
- Compression spring design, where the objective function is to minimize the weight of the compression spring while considering a number of constraints. The problem has 3 decision variables.
- Speed Reducer design, where we need to design a speed reducer with a minimum weight subject to constraints on bending stress of the gear teeth, surface stress, transverse deflections of the shafts and stresses in the shafts. The problem has seven decision variables.

Note that to design those problems to be dynamic, we applied the dynamic changes used in Scenario 1 to the first decision variable in each problem, except for the Compression spring design, where the third decision variable was used.

Table 15 summarizes the average offline error achieved by MODE-EACCO, MODE and MODE-reinitialization on those problems with $FF_{change} = 500$ and 1000. The results clearly demonstrate the superiority of the proposed approach. This superiority is also demonstrated by achieving the 100% feasibility rate for all problems, as demonstrated in Table 16.

8. Conclusions and future work

In this paper, a new approach for solving DCOPs with changes in the coefficients of their constraint functions was proposed. In the proposed approach, we introduced (1) a new way of ascertaining the effects of the linear and non-linear components of the decision variables, (2) a new technique that could determine which variables and constraints change and the rates of change occurring to each decision variable, and (3) a mechanism to adapt the solutions to a new environment. In addition, a new set of 13 test problems (each of which has 20 environments/dynamic changes and 3 different scenarios) was designed. The proposed framework was initially incorporated with a differential evolution algorithm and tested on different setups.

The results demonstrated its superiority (in terms of the quality of solutions, feasibility rates and convergence) over a basic algorithm without any technique for handling dynamic constraints and one that used a re-initialization mechanism after every change. Furthermore, it was tested in rapidly changing environments and was consistently superior to the others. This shows that the algorithm has a better convergence rate than the other algorithms.

The algorithm was further tested on problems with large dynamic changes and consistently showed its superiority. A parametric analysis was also conducted to check the best number of best solutions to record at the end of each environment; the results showed that a value of 4 was good. Furthermore, when the framework was adapted with other meta-heuristics, it provided consistently better results than existing algorithms. The algorithm was also tested on four engineering problems and showed its superiority.

The results discussed in this paper reveal the following advantages of the proposed framework: (1) flexibility to be adapted to different algorithms, (2) ability to achieve high-quality solutions, especially for optimization problems with frequent dynamic changes, (3) ability to detect the dynamic decision variables and the rate of change. However, one limitation is that the algorithm was incapable of finding the exact values of the changes in a few non-linear cases where the variables were dependent. Therefore, further research in this direction will be conducted.

Also, it is intended to extend this approach to solving problems that involve dynamic changes in both their constraints and objective functions. Testing its scalability (by solving large-scale dynamic problems) is another potential direction for future exploration. A further possibility is to expand the benchmark problems to include more environments and apply changes to more decision variables. In addition, it is planned to investigate the performance of this algorithm when changes in the coefficients are different under different constraints.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to acknowledge the funding provided by the Australian Research Council Discovery Projects (DP210102939) to conduct this work.

Appendix A

The supplementary material document with all appendices can be found at THIS LINK.

References

[1] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, in: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Vol. 3, IEEE, 1999, pp. 1875–1882.

[2] R. Azzouz, S. Bechikh, L.B. Said, W. Trabelsi, Handling time-varying constraints and objectives in dynamic evolutionary multi-objective optimization, Swarm and Evolutionary Computation 39 (2018) 222–248.

[3] N. Jin, M. Termansen, K. Hubacek, J. Holden, M. Kirkby, Adaptive farming strategies for dynamic economic environment, 2007 IEEE Congress on Evolutionary Computation, IEEE, 2007, pp. 1213–1220.

[4] S. Elsayed, R. Sarker, C.C. Coello, Enhanced multi-operator differential evolution for constrained optimization, 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 4191–4198.

[5] N. Hamza, R. Sarker, D. Essam, Sensitivity-based change detection for dynamic constrained optimization, IEEE Access 8 (2020) 103900–103912.

[6] M. Braik, A. Hammouri, J. Atwan, M.A. Al-Betar, M.A. Awadallah, White shark optimizer: A novel bio-inspired meta-heuristic algorithm for global optimization problems, Knowl.-Based Syst. 243 (2022) 108457.

[7] A. Ahrari, S. Elsayed, R. Sarker, D. Essam, A new prediction approach for dynamic multiobjective optimization, 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2019, pp. 2269–2276.

[8] Y. Diao, C. Li, S. Zeng, M. Mavrovouniotis, S. Yang, Memory-based multi-population genetic learning for dynamic shortest path problems, 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2019, pp. 2277–2284.

[9] D. Gong, B. Xu, Y. Zhang, Y. Guo, S. Yang, A similarity-based cooperative co-evolutionary algorithm for dynamic interval multiobjective optimization problems, IEEE Trans. Evol. Comput. 24 (1) (2020) 142–156.

[10] M. Rong, D. Gong, Y. Zhang, Y. Jin, W. Pedrycz, Multidirectional prediction approach for dynamic multiobjective optimization problems, IEEE Transactions on Cybernetics 49 (9) (2019) 3362–3374.

[11] B. Xu, Y. Zhang, D. Gong, Y. Guo, M. Rong, Environment sensitivity-based cooperative co-evolutionary algorithms for dynamic multi-objective optimization, IEEE/ACM Trans. Comput. Biol. Bioinf. 15 (6) (2018) 1877–1890.

[12] H. Richter, S. Yang, Memory based on abstraction for dynamic fitness functions, in: Workshops on Applications of Evolutionary Computation, Springer, 2008, pp. 596–605.

[13] J. Branke, T. Kaußler, C. Smidt, H. Schmeck, A multi-population approach to dynamic optimization problems, in: Evolutionary Design and Manufacture, Springer, 2000, pp. 299–307.

[14] C. Bu, W. Luo, L. Yue, Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies, IEEE Trans. Evol. Comput. 21 (1) (2017) 14–33.

[15] M.-Y. Ameca-Alducin, E. Mezura-Montes, N. Cruz-Ramírez, A repair method for differential evolution with combined variants to solve dynamic constrained optimization problems, in: in: Proceedings of the 2015 annual conference on genetic and evolutionary computation, ACM, 2015, pp. 241–248.

[16] M.Y. Ameca-Alducin, E. Mezura-Montes, N. Cruz-Ramírez, Differential evolution with a repair method to solve dynamic constrained optimization problems, GECCO (Companion) (2015) 1169–1172.

[17] H. Richter, Memory design for constrained dynamic optimization problems, in: European Conference on the Applications of Evolutionary Computation, Springer, 2010, pp. 552–561.

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

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

[20] C. Li, T.T. Nguyen, M. Yang, M. Mavrovouniotis, S. Yang, An adaptive multipopulation framework for locating and tracking multiple optima, IEEE Trans. Evol. Comput. 20 (4) (2015) 590–605.

[21] H.K. Singh, A. Isaacs, T.T. Nguyen, T. Ray, X. Yao, Performance of infeasibility driven evolutionary algorithm (idea) on constrained dynamic single objective optimization

problems, 2009 IEEE congress on evolutionary computation, IEEE, 2009, pp. 3127–3134.

[22] M.-Y. Ameca-Alducin, M. Hasani-Shoreh, F. Neumann, On the use of repair methods in differential evolution for dynamic constrained optimization, in: in: International Conference on the Applications of Evolutionary Computation, Springer, 2018, pp. 832–847.

[23] M. Hasani-Shoreh, M.-Y. Ameca-Alducin, W. Blaikie, F. Neumann, M. Schoenauer, On the behaviour of differential evolution for problems with dynamic linear constraints, arXiv preprint arXiv:1905.04099.

[24] T.T. Nguyen, A proposed real-valued dynamic constrained benchmark set, School of Computer Science, Univesity of Birmingham, Tech. Rep.

[25] H. Richter, F. Dietel, Solving dynamic constrained optimization problems with asynchronous change pattern, in: European Conference on the Applications of Evolutionary Computation, Springer, 2011, pp. 334–343.

[26] K. Pal, C. Saha, S. Das, C.A.C. Coello, Dynamic constrained optimization with offspring repair based gravitational search algorithm, 2013 IEEE congress on evolutionary computation, IEEE, 2013, pp. 2414–2421.

[27] M.-Y. Ameca-Alducin, E. Mezura-Montes, N. Cruz-Ramírez, Dynamic differential evolution with combined variants and a repair method to solve dynamic constrained optimization problems: an empirical study, Soft. Comput. 22 (2) (2018) 541–570.

[28] X. Lu, K. Tang, X. Yao, Speciated evolutionary algorithm for dynamic constrained optimisation, in: International Conference on Parallel Problem Solving from Nature, Springer, 2016, pp. 203–213.

[29] J.J. Grefenstette, et al., Genetic algorithms for changing environments, in: PPSN, Vol. 2, 1992, pp. 137–144.

[30] E. Mezura-Montes, C.A.C. Coello, E.I. Tun-Morales, Simple feasibility rules and differential evolution for constrained optimization, Mexican International Conference on Artificial Intelligence, Springer, 2004, pp. 707–716.

[31] D. Parrott, X. Li, Locating and tracking multiple dynamic optima by a particle swarm model using speciation, IEEE Trans. Evol. Comput. 10 (4) (2006) 440–458.

[32] J. Liang, T.P. Runarsson, E. Mezura-Montes, M. Clerc, P.N. Suganthan, C.C. Coello, K. Deb, Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization, J. Appl. Mech. 41 (8) (2006) 8–31.

[33] A. Simões, E. Costa, Prediction in evolutionary algorithms for dynamic environments, Soft. Comput. 18 (8) (2014) 1471–1497.

[34] G. Corder, D. Foreman, Nonparametric statistics: an introduction, Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach, John Wiley & Sons, Hoboken, NJ, USA, 2009, pp. 101–111.

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

[36] S.M. Elsayed, R.A. Sarker, D.L. Essam, Ga with a new multi-parent crossover for solving ieee-cec2011 competition problems, 2011 IEEE congress of evolutionary computation (CEC), IEEE, 2011, pp. 1034–1040.

[37] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, 2013 IEEE congress on evolutionary computation, IEEE, 2013, pp. 71–78.

[38] Y. Wang, J. Yu, S. Yang, S. Jiang, S. Zhao, Evolutionary dynamic constrained optimization: Test suite construction and algorithm comparisons, Swarm and Evolutionary Computation 50 (2019) 100559.

[39] S. Yang, C. Li, A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments, IEEE Trans. Evol. Comput. 14 (6) (2010) 959–974.

[40] B. Kannan, S.N. Kramer, An augmented lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design.

[41] A.D. Belegundu, A study of mathematical programming methods for structural optimization, The University of Iowa, 1982.

[42] N. Hamza, R. Sarker, D. Essam, Evolutionary search from the interior of feasible space, 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2020, pp. 353–359.