IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 16, 2020, accepted May 22, 2020, date of current version June 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2999161

Sensitivity-Based Change Detection for Dynamic Constrained Optimization

NOHA HAMZA, RUHUL SARKER, (Member, IEEE), AND DARYL ESSAM
School of Engineering and Information Technology, University of New South Wales at Canberra, Canberra, ACT 2600, Australia
Corresponding author: Noha Hamza (n.hamza@unsw.edu.au)

This work was supported by the Australian Research Council Discovery Projects under Grant DP170102416.

ABSTRACT In dynamic constrained optimization, changes may occur in either the objective function or constraint functions, or both. However, although research on dynamic optimization has been growing significantly, it is centered mainly around unconstrained problems. On the other hand, research on dynamic constrained problems has considered simple extensions of those conducted on unconstrained ones. These approaches are usually computationally expensive and exhibit slow convergence in changed environments. To develop an effective algorithm for dynamic constrained problems with changes occurring in only the constraint's space, in this research, we propose a sensitive constraint detection mechanism that provides valuable information for determining the movements of solutions in changing environments. As, by incorporating it with a search process, the convergence of an algorithm can be accelerated, it is adopted with a multi-operator evolutionary type of algorithm which is then tested on a set of constrained dynamic problems. The experimental results clearly demonstrate the benefits of this proposed approach.

INDEX TERMS Constrained optimization, dynamic optimization, evolutionary algorithms.

I. INTRODUCTION

Dynamic constrained optimization is a popular research topic in many disciplines, including, but not limited to, computer science, optimization and engineering [1], [2]. Without a loss of generality, a dynamic problem can be considered a series of static problem instances with a time gap between consecutive ones, that is, it is assumed that each is valid for a small duration of a time slot. Note that each instance is considered an environment of the problem. Mathematically, dynamic constrained optimization problems (DCOPs) with a single objective can be stated as

$$\begin{aligned} &\min f(\mathbf{x}, t) \\ &\text{subject to } g_k(\mathbf{x}, t) \leq 0, \quad \forall k = 1, 2, \ldots, K \\ &\qquad h_e(\mathbf{x}, t) = 0 \\ &\qquad L_j(t) \leq x_{t,j} \leq U_j(t) \end{aligned} \tag{1}$$

where $\mathbf{x} \in S(t) \subset R^{D(t)}$, $\mathbf{x} = [x_{t,1}, \ldots, x_{t,j}, \ldots, x_{t,D(t)}]^T$ is a vector with D-decision variables that changes over time t, $(f, g_k, h_e : R^n \rightarrow R$ and $E \leq n)$, $g_k(\mathbf{x}, t)$ the k^{th} inequality constraint and $h_e(\mathbf{x}, t)$ the e^{th} equality constraint, with each $x_{t,j}$ having has lower and upper limits ($L_j(t)$ and $U_j(t)$, respectively). If the feasible and search spaces for time t are denoted by $F(t)$ and $S(t)$, respectively, in which each point that satisfies the upper and lower limits is denoted by $F(t)(\subseteq S(t))$, a feasible solution for time t is a point that satisfies all constraints, with the set of all feasible points defined as:

$$x \in R^{D(t)} : h(\mathbf{x}, t) = 0, \quad g(\mathbf{x}, t) \leq 0 \tag{2}$$

In the literature, Evolutionary Algorithms (EAs) are well-known for their superior performances for solving static instances of optimization problems. To solve dynamic problems, the early trend was to modify the algorithms developed for static problems to adapt them for dynamic ones. In these modified algorithms, the main search process is usually the same as for static problems but is repeatedly used with changes commonly made in the initial populations, diversity mechanisms and stopping criteria. However, these approaches are recognized as computationally expensive, have slow convergence and may become stuck in local optima. To mitigate these issues, a few methods, such as memory-based [1], [3], multi-population [4], [5], and prediction-based [6]–[10], have been proposed. Note that most of these approaches were developed mainly for unconstrained problems in which, assuming a single objective,

The associate editor coordinating the review of this manuscript and approving it for publication was Victor Hugo Albuquerque.

the aim is to optimize the objective function, which changes from one environment to another, within variable bounds.

In addition to the objective function, in constrained problems, the constraint functions and their limits may also change from one environment to another. These problems are much more challenging than their unconstrained counterparts as it is necessary to determine the best solution within the feasible region. Note that this region is usually smaller than the search space and may have an irregular shape as it is bounded by complex constraint functions. However, if it is too small, which is the case when there are equality constraints, it becomes much more difficult to solve the problem for which it is known that there is often only a limited time due to the fast changing nature of a problem's environment. To deal with this, common approaches include mechanisms for repairing and/or tracking the feasible region. However, some are computationally expensive [11], [12] and/or developed for only specific types of changes [13] or loss of diversity [14].

In constrained optimization, it is well known that the best feasible solution (also known as optimal solution) can be determined by satisfying only the active constraints. In other words, the active constraints are sensitive to feasibility and optimality while the inactive ones are redundant. The active and inactive constraints are also known as binding (or tight) and non-binding (or loose) constraints, respectively. In dynamic problems, those that are active in one environment may either remain active in the next or become inactive. Similarly, an inactive one may either remain inactive or become active. In this case, if the changes in environment $t + 1$ are only in inactive constraint (as in environment t) and do not become active, there is no need to resolve the problem in $t+1$. Also, a change in the environment may change the shape and location feasible region which requires a new mechanism to converge to the new optimal solution quickly. As the changes in practical problem instances between environments are usually not huge, the sensitivity of the constraints and amount of changes can provide useful information for the design of an effective algorithm for solving dynamic optimization problems (DOPs). This is the main motivation for proposing a new algorithm for solving the practical dynamic constrained optimization problems.

The key contribution in this paper is the development of a sensitive constraint detection mechanism that helps to determine the directions and amount of potential movements of solutions in the search process when solving constrained optimization problems under changing environments. In this paper, we study single-objective constrained dynamic optimization problems with possible changes in their constraints' boundaries and propose a mechanism for detecting changes in the feasible region. In the sensitive-constraint detection mechanism, the sensitivity of each constraint is determined, based on its slack value given the solution in the current environment. The number of sensitive constraints in the previous environment are divided into groups, with the solutions in each then updated to move towards a new promising search area based on the slack value of the corresponding constraint. This process is useful for any changes in either the constraint's resources, or their usage rates or both. The constraint sensitivity is also critical for any changes in the objective function. To the best of our knowledge, this is the first such approach proposed. Two different algorithms incorporate the detection and movement mechanism and to judge their performances, we solved 13 simulated dynamic constrained problems, each with 7 environments. The experimental results showed the benefits of these approaches for obtaining better-quality solutions, feasibility rates and convergence speeds. A summary of comparisons with a well-known existing approach revealed the superiority of the proposed method.

This paper is organized as follows. After this introduction, section II reviews the related literature on DCOPs and the existing EAs used to solve them. Then, the details of the proposed algorithm are presented in section III and the experimental results and analysis in section IV. Finally, section VII provides the conclusions and suggestions for future work.

II. LITERATURE REVIEW

In the literature, many studies for solving unconstrained DOPs have been introduced [15], [16] but, to date, few of DCOPs have been undertaken.

Dynamic constrained techniques can be used to solve both single- and multi-objective optimization problems [2], [17], [18].

Most EAs are designed to solve problems in static rather than dynamic environments. However, some have been adapted to solve DCOPs with either no or simple modifications [14], [19].

Singh *et al.* [19] adapted the infeasibility-driven EA algorithm (IDEA) for solving DCOPs by using a change detection strategy to handle changes. In this strategy, a random individual is selected as a change detector and evaluated at each generation. If a change is detected, the whole population is re-evaluated. Based on the results, it was shown to be competitive in comparison with other state-of-the-art algorithms for two test problems. However, randomly selecting the detector can affect its detection performance.

In order to introduce diversity when changes occur, Ameca-Alducin *et al.* [14] introduced a DE-based algorithm combined with two variants of differential evolution (DE) called the dynamic DE combined with variants (DDECV). It has also been used as a change detection method [20] with random immigrants (RI) [21] and with the superiority of feasible points approach [22] as a constraint handling technique (CHT). However, it cannot maintain sufficient diversity as infeasible solutions are considered inferior based on the feasibility rule.

Hasani-Shoreh *et al.* [23] introduced a framework for a dynamic benchmark generator with linear constraint changes. DE was adopted to solve these problems using a change detection mechanism and different CHTs.

In the existing literature, a memory-based approach is another way of using EAs to solve DCOPs; for instance, Richter [13] adapted the abstract memory strategy [3] by proposing two memory schemes, i.e., blending and censoring, for storing good individuals and their associated information, one for constrained and the other unconstrained problems. The algorithm was tested and the results showed that using these schemes with a repairing method rather than a penalty function as a CHT performed well. However, this method is only applicable to particular types of constrained dynamic problems (i.e., cyclic ones).

In another paper, Richter and Dietel [24] tested the performance of an EA using the above two memory schemes and another four different strategies on a special type of DCOPs. In these problems, the objective function and constraints could change in an asynchronous change pattern. The results showed that the memory-based methods performed better than the random ones.

The repairing strategy is another mechanism added to EAs to solve DCOPS. Pal *et al.* [25] introduced an offspring one within a gravitational search algorithm (GSA) [26], called GSA+ Repair, to improve the objective function. In this method, an infeasible solution moves towards its closest feasible solution selected from a reference set of the best ones found so far, with GSA+Repair outperforming the state-of-the-art algorithms. However, many feasibility checks are needed to produce a reference population if the feasible region is small or determine whether previously feasible individuals are still feasible every time a change occurs.

Ameca-Alducin *et al.* [11], [12], improved the DDECV algorithm by applying a mutant repair method (DDECV + Repair) to it as a CHT to produce better results without the need for a reference solution. However, this is computationally expensive when the feasible region is small.

In another paper, Ameca-Alducin *et al.* [27] evaluated each element of the DDECV + Repair by solving eight benchmark test problems [28]. Based on the results, this method was shown to be competitive in comparison with other state-of-the-art algorithms considering seven performance metrics.

Existing unconstrained dynamic optimization (DO) strategies with CHTs can be used to solve DCOPs. Lu *et al.* [29] proposed a speciation-based algorithm called speciated evolution with a local search (SELS). In it, a speciation method acts as an exploration promotion mechanism by helping an EA find multiple optima by comparing similar individuals. Also, a local search is used to accelerate convergence towards promising regions. SELS combines some existing DO strategies (i.e., a change detection mechanism and RI [30]) and a CHT (i.e., a simple feasibility rule) [31]

To demonstrate the effectiveness of using repairing methods as CHTs to solve DCOPs, a study of different ones [32] was conducted with DE. Three measures, called the offline error, success rate and average number of iterations required to repair infeasible solutions, were used to compare these approaches, with the gradient repairing method outperforming the others.

One of the mechanisms for overcoming the shortcomings of combining DO with a CHT to solve DCOPs is the lack of a method for tracking a moving feasible region. Bu *et al.* [5] solved DCOPs with multiple disconnected feasible regions by adopting the dynamic species-based particle swarm optimization (DSPSO) algorithm [33] along with an ensemble of different mechanisms to locate and track multiple feasible regions at the same time while dealing with different types of dynamics in the constraints. This algorithm always maintains good diversity during the search process.

In summary, as of the literature, the existing repair methods that require a feasible reference population are computationally expensive, especially when the feasible region is tiny. The memory-based approaches suffer from the curse of defining the optimal size of the memory and suitability for types of DCOPs. Although prediction approaches showed some advantages in solving DCOPs, the existing approaches failed in dealing with problems with large random changes and, similar to the memory-based approaches, handling problems with changing environments that did not occur in the past. Also, existing mechanisms suffer from handling problems with complex mathematical proprieties, i.e., non-linearity, dis-joint feasible region and rotation. Further, just combining static EAs with constraint handling technique to deal with DCOPs may not be able to track the movement of feasible regions and the extra fitness evaluations needed by the dynamic detectors [15], [23], [32].

III. PROPOSED ALGORITHM

In this section, the proposed framework and its components are discussed.

A. PROPOSED FRAMEWORK

As presented in Algorithm 1, the algorithm starts with an initial random population (X) of size NP generated within the search boundary, such as

$$x_{i,j} = L_j + \ rand \ \times \ \left(U_j - L_j\right), \ \ \forall \ j = 1, \ 2, \ldots, D, \quad (3)$$

where L_j and U_j are the lower and upper bounds respectively for decision variable $x_{i,j}$ and $rand$ a random number $\in [0, 1]$.

Then, X is evolved using a multi-operator DE (MODE) framework, as discussed in section III-C until a change occurs, i.e., $t = t+1$. Therefore, at the end of the evolutionary process in the current environment, the best solution ($\mathbf{x}_{best,t}$) is recorded.

Through analysis, we have found that some constraints might be more sensitive to a change than others, that is, not all changes in a constraint boundary can affect the optimal solution. As working on sensitive constraints can have more impact on the optimization process, we propose a sensitive-constraints detection mechanism being implemented once a dynamic change is detected (Algorithm 2). In it, the slack value ($Slack(g_k)$) for each constraint at $\mathbf{x}_{best,t-1}$ is calculated, as discussed in section III-B (equation (5)).

The g_k^{th} constraint is considered active (binding), if $Slack g_k = 0$, otherwise non-active (non-binding). Therefore, to determine which constraint is more sensitive than the others, all are ranked in ascending order based on their slack values. Note that this sensitive constraint detection mechanism is flexible and can be integrated with any other meta-heuristic.

Then, the new population in the new environment is considered the solutions obtained at the end of the evolutionary process at $t-1$, with the following changes:

- to avoid losing information about good solutions obtained at $t-1$, the worst Γ solutions are replaced with the best Γ ones.
- the best $P = (Percent\% \times NP) \geq \Gamma$ individuals are updated, as discussed in Algorithm 3, that is, P individuals are first divided into n sub-groups of equal size $(sub_{NP_1}, sub_{NP_2}, \ldots, sub_{NP_n})$, where n is the number of sensitive constraints detected above and $Percent$ a value $\in [0, 1]$. Subsequently, for each sensitive constraint, if there is a change in the right-hand side value, each individual in the corresponding sub-group is expected to move towards the new promising feasible area as:

$$x_{i,j,\zeta} = x_{i,j,\zeta} + \Delta_j \times |Slack(g_k)|, \quad \forall i = 1, \ldots, sub_{NP_\zeta}, \zeta = 1, \ldots, n, j = 1, \ldots, D \tag{4}$$

  where Δ_j is a random value ($\leq$ 1), and $Slack(g_k)$ is calculated based on equation (5).

Once the new population is created, the evolutionary optimizer evolves all solutions until a new change in the environment is detected. The same steps are carried out until all environments are optimized.

Algorithm 1 Proposed Algorithm

```
t ← 1 (current environment);
t_max ← total number of environments;
FES ← 0 current fitness evaluation;
FES_max ← total number of fitness evaluations;
initialization while t ≤ t_max do
    while FES ≤ FES_max do
        run MODE;
    end
    record x_best,t;
    if change detected then
        t ← t + 1; // next environment
        Detect sensitive constraints (Algorithm 2);
        Update selected individuals (Algorithm 3);
        FES ← 0
    end
end
```

B. CHANGE DETECTION

In DO, it is crucial to detect when and by how much a change occurs. In this paper, to detect if a change occurs in each RHS_{g_k}, the best individual is used to calculate the total value of each constraint at t. Any $g_k(\mathbf{x}_t) \neq g_k(\mathbf{x}_{t-1}), \forall i = 1, 2, \ldots, K$ (also known as, $RHS_{g_k,t} - RHS_{g_k,t-1}$) means a change occurred. Subsequently, the amount of change for any g_k, also called slack, is calculated as

$$Slack(g_k) = g_k(\mathbf{x}_{best,t}) - g_k(\mathbf{x}_{best,t-1}), \quad \forall k = 1, 2, \ldots, K, \tag{5}$$

This change detection makes the proposed algorithm sufficiently flexible to deal with DCOPs in which the number and frequency of changes are not fixed.

Algorithm 2 Sensitive-Constraints Detection Mechanism

```
x_best,t ← best individual at t − 1;
g_k ← the k^th inequality constraint;
s ← sensitive constraint;
n ← number of sensitive constraints;
foreach g_k do
    Calculate Slack(g_k) (equation 5);
end
Rank constraints in ascending order;
foreach g_k do
    if rank (g_k) == 1 then
        s_n ← g_k;
        n = n + 1;
    end
end
```

Algorithm 3 Proposed Move

```
n ← number of sensitive constraints
s ← sensitive constraint;
if n ≥ 1 then
    Select P = percent × NP individuals;
    Divide P into equal n subgroups of equal size,
    sub_NP_1 ... sub_NP_n;
    foreach s_n do
        if change in RHS_g_k then
            foreach x_i,j ∈ sub_NP_1 do
                Update x_i,j using equation (4);
            end
        end
    end
end
```

C. OPTIMIZER: MULTI-OPERATOR DE

The optimization algorithm used in this paper is based on the MODE one (hereafter called MODE) presented in [34]. In it, a random set of populations of size NP is generated within the search boundaries and then randomly divided into nps sub-populations of equal size, each of which is evolved using a different DE mutation operator, namely, rand-to-p best or current-to-p best.

A binomial crossover and selection mechanism are used to finalize the set of solutions that should survive to the next generation. Then, an improvement measure index, which

reflects the capability of each operator to generate better solutions, is calculated and measured based on two criteria: (1) the quality of solutions; and (2) diversity of solutions/the population, with the size of each sub-population updated in the following generation.

MODE also uses:

- an information-sharing mechanism in which, at the end of every generation, solutions in all *nps* sub-populations are collected and randomly re-divided into *nps* sub-populations, with their sizes dynamically updated based on the abovementioned improvement measure index;
- a self-adaptive technique for updating the F and Cr parameters which considers the improvement rates of the raw fitness values and/or total constraint violations; and
- three rules for selecting the solutions that should survive to the next generation, that is, (1) for 2 feasible candidates, the one with a lower fitness value, (2) a feasible solution is always considered better than an infeasible one and (3) for two infeasible points, the one with a smaller sum of constraint violations.

For more details about MODE, readers are referred to [34]. To sum up, the difference between the current MODE and that proposed in [34] is the introduction of the sensitive constraint detection mechanism as well as the way individuals are updated after every change.

D. TIME COMPLEXITY

This sub-section aims to discuss the time complexity of the proposed framework.

- Time complexity for the sensitive constraint detection mechanism: the calculation of the slack is $O(K)$ where K is the number of constraints. Ranking(sorting) constraints will be $O(Klog(K))$, given the merge sort used. The last part of this framework is $O(K)$. Consequently, the big-oh will be $2O(K) + O(Klog(K))$. By removing constants and focusing only on the dominant term, the time complexity of this detection mechanism will be $Klog(K)$.
- Time complexity for the proposed move, assume the worst scenario where all constraints are sensitive, the time complexity will be $O(P * D)$.
  As the second part is equal to the time complexity of generating new random solutions using any other initialization method, the extra complexity added to the proposed framework is that of the detection mechanism, that is, $O(Klog(K))$. This, in turn, makes the time complexity of the overall framework equal to the time complexity of the optimizer itself plus $O(Klog(K))$. Recall that the proposed framework is flexible and can be adapted with any optimizer.

IV. EXPERIMENTAL RESULTS

In this section, the proposed algorithm is tested and analyzed by solving the set of benchmark problems introduced in the CEC2006 special session on COPs [35]. These problems possess different mathematical properties, such as the objective function and/or constraints being either linear or non-linear, the constraints either equality or inequality ones, the objective function either unimodal, such as G_{02} or multi-modal and the feasible space possibly being very tiny with the search one.

Although these are static problems, they can be transformed into dynamic ones by applying dynamic rules to their parameters, as discussed in section section IV-A. More details of this concept can be found in [36].

13 test problems with only inequality constraints were adopted, i.e., 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}.

A. CHANGE CONSTRAINT BOUNDARY

A random type of change was proposed and applied on the constraint boundary of the inequality constraints as follows.

Given the general static form of the problem

$$\begin{aligned} &\min f(\mathbf{x}) \\ &\text{subject to } g_k(\mathbf{x}) \leq b_k, \quad \forall k = 1, 2, \ldots, K \\ &\qquad h_e(\mathbf{x}) = 0 \end{aligned} \tag{6}$$

where b_k is the boundary of the inequality constraint g_k, ($b_k = 0$). This problem was modified to one with a fixed objective function and dynamic constraints by adapting the boundary of each constraint as:

$$g_k(\mathbf{x_t}) \leq b_k(t), \quad \forall k = 1, 2, \ldots, K$$

with the random change

$$b_k(t) = rand, \quad \forall k = 1, 2, \ldots, K \tag{7}$$

where *rand* is a random number $\in [-1, 1]$ [37]. Based on this, the constraint boundary was represented by a matrix of size (k, t) which determined how the dynamic constraint for each benchmark problem changed over time.

B. EXPERIMENTAL SETTINGS

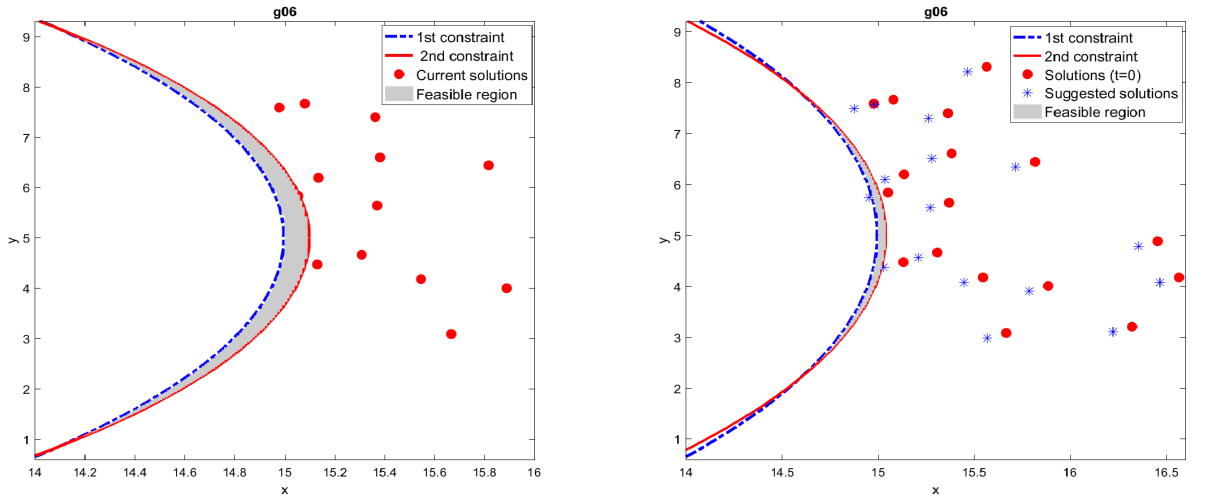
For each benchmark problem, 25 independent runs were performed. In each, there were seven time periods (t_{max}) with the frequency of change (FE_{change}) set to 500, 1000 and 1500 fitness evaluations (FEs). All the algorithms were coded using Matlab.

C. PARAMETER SETTINGS

The parameters were set to a population size (NP) of 100, $Percent = 0.5$, that is $P = 50$ individuals, $\Gamma = 5$ and $\Delta \in [0, 0.1]$. For MODE, the settings were the same as those reported in [34].

D. PERFORMANCE MEASURES

We used the average fitness value and average feasibility ratio (FR) obtained from all the runs in each environment as performance measures. Note that, if all the compared algorithms could not obtain feasible solutions, the average sum of the constraint violations was considered a measure of the quality



(a) solutions at the end of $t = 0$

(b) solutions at the start of $t = 1$

FIGURE 1. Movements of solutions using proposed algorithm.

TABLE 1. Summary of comparisons MODE(proposed), MODE and MODE(re-initialization).

FE_{change}	MODE(proposed) *vs.* MODE					MODE(proposed) *vs.* MODE(re-initialization)				
	Better	Equal	Worse	p	Decision	Better	Equal	Worse	p	Decision
500	**66**	15	10	0.00	+	**77**	13	1	0.00	+
1000	**63**	15	13	0.00	+	**77**	12	2	0.00	+
1500	**64**	15	12	0.00	+	**78**	13	0	0.00	+

of solutions. A non-parametric statistical significance test, namely, the Wilcoxon Signed Rank Test [38], was also used. With a 5% significance level, a null hypothesis assumed that there was no significant difference between the results for two samples while the alternative hypothesis assumed that there was. Based on the test results/rankings, we assigned one of three signs ($+$, $-$, and $\approx$) to the comparison of any two algorithms (a significant difference in the average fitness values is shown in the 'Decision' column), where '$+$' means that the first algorithm was significantly better than the second, '$-$' means that it was significantly worse, and '$\approx$' sign that there was no significant difference between them.

E. ANALYSIS

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

1) BENEFITS OF PROPOSED APPROACH

This subsection aims to show the benefits of the proposed framework. To do this, three variants of the MODE algorithm were run, that is: (1) using the proposed approach; (2) considering the entire population before a change as an initial set of solutions in the new environment: and (3) re-initialize the entire population after every change. Detailed results are shown in the supplementary documents in Appendices A-1 to A-3, with summaries of comparisons in Table 1.

This table clearly demonstrates the superiority of MODE using the proposed approach over the other two versions given its capability to obtain better solutions for the majority of dynamic environments and statistically significantly outperforms them.

The main reason for this good behavior is the detection of the sensitive constraints that helps to identify the new feasible space for concentrating the search process. We have demonstrated in our experimental study that the proposed approach is able to obtain better results more quickly in the majority of tested environments, as illustrated in Fig. 1 using G_{06} as an example. Firstly, at time $t = 0$, it evolved the solutions until the end of the first time period and, based on the best solutions found so far, the second constraint was defined as the sensitive one. Secondly, at time $t = 1$, a change in the boundary of the second constraint was applied which affected the size of the feasible region. Subsequently, the proposed approach was implemented which, in turn, could help current solutions move in the direction of feasibility.

Regarding the average FRs, Table 2 shows that the proposed approach could obtain higher rates of feasible solutions at different change frequencies ($FE_{change} = 500, 1000, 1500$) than the other two versions.

Also, the effects of the proposed approach at different change frequencies ($FE_{change} = 500, 1000, 1500$) are presented in Fig. 2. Firstly, when $FE_{change} = 500$, the proposed approach helped MODE to converge towards feasibility faster and obtain a better solution than MODE alone. While re-initializing the population could not obtain any feasible solution. Secondly, when $FE_{change} = 1000$,

TABLE 2. Average feasibility rate (FR%) for MODE(proposed), MODE and MODE(re-initialization).

FE_{change}	MODE(proposed)	MODE	MODE(re-initialization)
500	**58.15%**	57.36%	42.15%
1000	**65.98%**	65.19%	53.36%
1500	**67.69%**	67.47%	56.79%
Average FR	**63.94%**	63.34%	50.77%

although the proposed approach and MODE began to converge towards feasibility in the same environment, the former started with a better solution with no feasible solution for MODE
(re-initialization), no feasible solution was found. Finally, when $FE_{change} = 1500$, MODE was able to obtain a feasible solution in the first environment. Once there was a change to the second environment, the proposed approach was able to obtain better solutions than MODE. Moreover, although the re-initialized population approach was able to converge to feasibility, this was at a later stage in the evolutionary process but still before the next change.

To conclude, due to its ability to move to the changed feasible space quickly and concentrate search in that space for the optimal solution, the proposed approach was able to outcompete other approaches, especially in quickly changing environments.

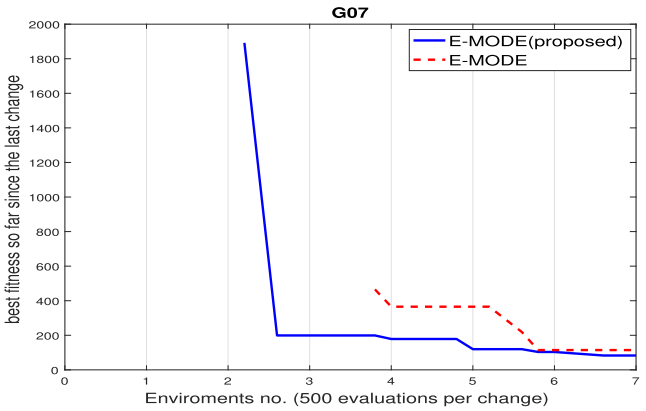
2) EFFECT OF STEP SIZE (△)

As previously discussed, Δ is a small real-value that controls how far each decision variable may move towards a new change and was set to a random value in the range $\in [0, 0.1]$ in the above mentioned algorithm. The reason for applying different values to each decision variable was to maintain diversity. However, to analyze this effect, a MODE (proposed), as analyzed above, was compared against a MODE (proposed) with a fixed value of Δ, i.e., $\Delta_j = 0.1 \forall j = 1, 2, \ldots, D$. The detailed results are shown in the supplementary documents in Appendices B-1 to B-2.

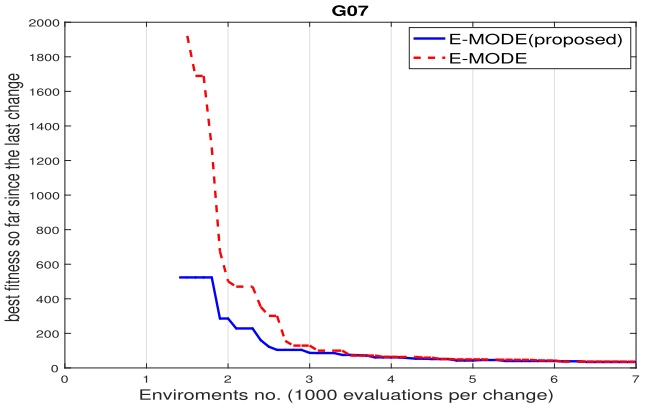
The summary of comparisons shown in Table 3 indicates that using $\Delta_j \in [0, 0.1]$ was better than setting it as $\Delta_j = 0.1$ when $FE_{change} = 500$ or 1000 FEs whereas it was quite similar when $FE_{change} = 1500$. Statistically, $\Delta_j \in [0, 0.1]$ is better than $\Delta_j = 0.1$ in the case when the frequency of change is 1000 FEs.

For further consideration different values of Δ_j were analyzed, that is, the algorithm was run with: (1) $\Delta_j \in [0, 0.25]$; (2) $\Delta_j \in [0, 0.5]$; (3) $\Delta_j \in [0, 0.75]$; and (4) $\Delta_j \in [0, 1.0]$. All versions were compared against the variant with $\Delta_j \in [0, 0.1]$, the results presented in Table 3.

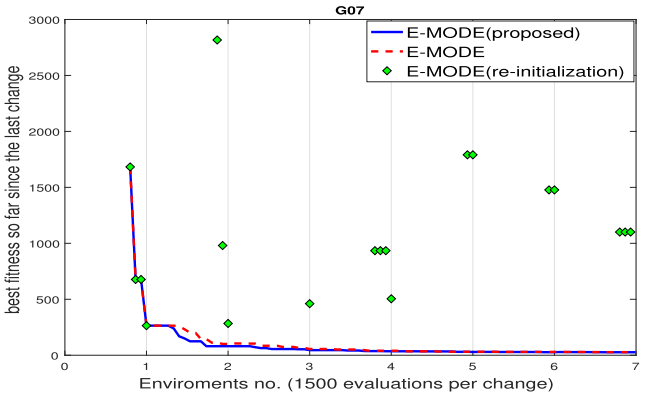
For $FE_{change} = 500$ and 1000, no single value of Δ_j was the best, that is, each variant performed well in some problems and worse in others while the variant with $\Delta_j \in [0, 0.25]$ did better than that with $\Delta_j \in [0, 0.1]$. For $FE_{change} = 1500$, it is noted that the variant with $\Delta_j \in [0, 0.1]$ was better than the others with different values of Δ_j.



(a) $FE_{change} = 500$



(b) $FE_{change} = 1000$



(c) $FE_{change} = 1500$

FIGURE 2. Convergence plots for MODE(proposed), MODE and MODE(re-initialization).

To clarify the results, we used a Friedman test to rank all the variants at different FE_{change} values, as summarized in Table 4. It is clear that using $\Delta_j \in [0, 0.1]$ was the best when changes were more frequent while $\Delta_j \in [0, 0.25]$ was the best when changes occurred every 1000 and 1500 FEs.

In summary, the version with $\Delta_j \in [0, 0.25]$ was selected as it had the best overall ranking. Remember that using a

TABLE 3. MODE with proposed approach and different Δ.

MODE(proposed)	FE_{change}	Better	Equal	Worse	p	Decision
$\Delta_j \in [0, 0.1]$ vs. $\Delta_j = 0.1$	500	**42**	20	29	0.83	≈
	1000	**45**	21	25	0.01	+
	1500	**36**	21	34	0.70	≈
$\Delta_j \in [0, 0.1]$ vs. $\Delta_j \in [0, 0.25]$	500	35	20	**63**	0.83	≈
	1000	33	21	**37**	0.68	≈
	1500	**38**	21	32	0.18	≈
$\Delta_j \in [0, 0.1]$ vs. $\Delta_j \in [0, 0.5]$	500	**44**	19	28	0.23	≈
	1000	**42**	21	28	0.10	≈
	1500	**46**	21	24	0.04	+
$\Delta_j \in [0, 0.1]$ vs. $\Delta_j \in [0, 0.75]$	500	**45**	20	26	0.36	≈
	1000	**44**	21	26	0.10	≈
	1500	**54**	21	16	0.00	+
$\Delta_j \in [0, 0.1]$ vs. $\Delta_j \in [0, 1.0]$	500	**42**	20	29	0.50	≈
	1000	**51**	21	19	0.00	+
	1500	**55**	21	15	0.00	+

TABLE 4. Friedman rankings for proposed algorithm with different values of Δ.

FE_{change}	Ranking over Δ				
	[0.0, 0.1]	[0.0, 0.25]	[0.0, 0.5]	[0.0, 0.75]	[0.0, 1.0]
500	**2.74**	2.79	3.13	3.09	3.25
1000	2.67	**2.54**	3.14	3.04	3.60
1500	2.42	**2.48**	2.87	3.52	3.71
Average ranking	2.61	**2.60**	3.05	3.22	3.52

TABLE 5. Average feasiblity rates (FR%) for proposed algorithm with different values of Δ.

FE_{change}	[0.0,0.1]	[0.0,0.25]	[0.0,0.5]	[0.0,0.75]	[0.0,1.0]
500	**58.15%**	**58.51%**	58.02%	58.07%	58.07%
1000	65.98%	66.29%	**66.37%**	66.11%	66.33%
1500	67.69%	67.78%	**67.87%**	67.65%	67.82%
Average FR	63.94%	**64.19%**	64.09%	63.94%	64.07%

different value for each j enhances the performance due to the diversity it adds to the algorithm.

Table 5 shows that the variants with $\Delta_j \in [0, 0.1]$ and $\Delta_j \in [0, 0.25]$ had high rates of feasible solutions for $FE_{change} = 500$. However, Δ_j values of [0, 0.5] were the best for $FE_{change} = 1000$ and, 1500, respectively. Based on the overall FR, the variant with $\Delta_j \in [0, 0.25]$ was selected. The effects of the proposed approach with different values of Δ_j are presented in Fig. 3 in which it is clear that the performance of $\Delta_j \in [0, 0.25]$ was the best of all the values.

3) EFFECT OF *P*

As previously discussed, *P* was initially set to 0.5*NP*. We analyzed its effect by changing its value to 0.25*NP*, 0.75*NP* and *NP* with all these versions using $\Delta_j \in [0, 0.25]$ which was found to be the best. Subsequently, all these variants were compared with that with the setting of 0.5*NP*. Details of their fitness values are provided in Appendices D-1 to D-3 in the supplementary material and a summary of comparisons in Table 6. These results show that there was no single value of *NP* that was best for all test problems which is consistent with the No Free Lunch Theorem. Overall, it was found that there was a bias towards setting *P* to 0.5*NP* when $FE_{change} = 1000$ and 1500 while setting it to 0.25*NP* and *NP* was slightly more preferable when FE_{change} 500.

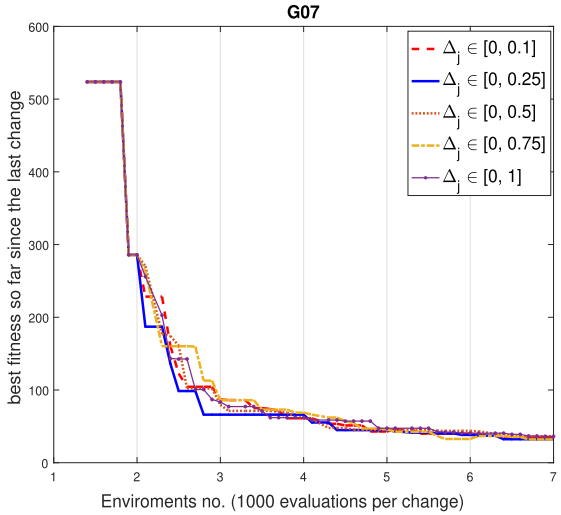


FIGURE 3. Convergence plots for Proposed algorithm with different values of Δ.

TABLE 6. Effect of *P*.

P	FE_{change}	Better	Equal	Worse	p	Decision
$0.5NP$ vs $0.1NP$	500	**38**	20	33	0.95	≈
	1000	33	21	**37**	0.68	≈
	1500	**38**	21	32	0.18	≈
$0.5NP$ vs $0.25NP$	500	35	20	**36**	0.38	≈
	1000	**42**	21	28	0.10	≈
	1500	**46**	21	24	0.03	+
$0.5NP$ vs $0.75NP$	500	**44**	20	27	0.57	≈
	1000	**44**	21	26	0.10	≈
	1500	**54**	21	16	0.00	+
$0.5NP$ vs NP	500	34	20	**37**	0.04	−
	1000	**51**	21	19	0.00	+
	1500	**55**	21	15	0.00	+

The Friedman rankings shown in Table 7 demonstrate that the variant with *P* set to a value of *NP* was the best when $FE_{change} = 500$, while that with a *P* value of 0.5*NP* was the best for $FE_{change} = 1000$. However, when $FE_{change} = 1500$, setting *P* to a value of 0.25*NP* was the best.

Table 8 shows the value of $P = 0.5NP$ had the highest FR for $FE_{change} < 1500$ while using fewer solutions to which to apply the move,i.e., 0.1*NP* for $FE_{change} = 1500$ was better.

TABLE 7. Friedman ranking for proposed algorithm with different values of *P*.

FE_{change}	$0.1NP$	$0.25NP$	$0.5NP$	$0.75NP$	NP
500	3.14	2.97	2.89	3.30	**2.70**
1000	2.95	2.93	**2.63**	3.20	3.30
1500	2.76	**2.60**	2.88	3.34	3.42
Average ranking	2.95	2.84	**2.80**	3.28	3.14

TABLE 8. Average feasiblity rates (FR%) for proposed algorithm with different values of *P*.

FE_{change}	$0.1NP$	$0.25NP$	$0.5NP$	$0.75NP$	NP
500	58.24%	57.73%	**58.51%**	57.73%	57.73%
1000	66.11%	65.82%	**66.29%**	66.04%	65.87%
1500	**67.82%**	67.42%	67.78%	67.29%	67.29%
Average FR	64.06%	63.66%	**64.19%**	63.69%	63.63%

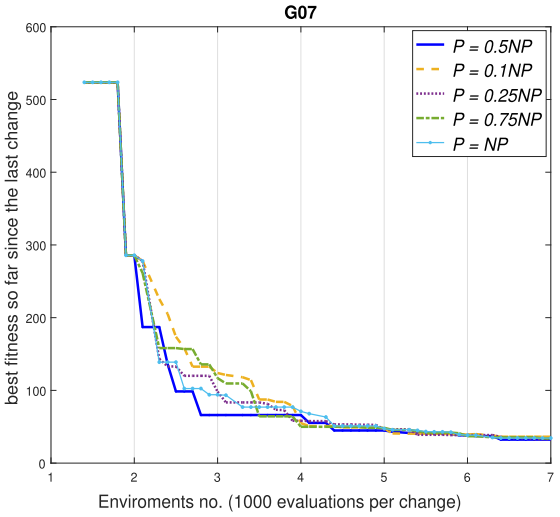


FIGURE 4. Convergence plots for proposed algorithm with different values of *P*.

Also, it is clear in Fig. 4 that the value of $P = 0.5NP$ was better than all the others in terms of convergence.

In summary, given the superiority of using $0.5NP$ to obtain higher FRs, which is a major aspect of constrained optimization and its good Friedman ranking, this P value was considered the best. Remember that if the environment changes slowly, updating a smaller number of solutions may also work fine, as the optimization algorithm will have enough time to improve its solutions.

4) EFFECT OF Γ

Given the values of $P = 0.5NP$ and $NP = 100$, Γ was analyzed by changing its value to 1, 5, 10, 25 and 50, noting that $\Gamma \leq P$. Detailed fitness results are provided in the supplementary documents in Appendices E-1 to E-3.

The summary of comparisons shown in Table 9 indicates that setting the value of Γ to 5 rather than 1 was better for all frequencies of change but setting it to > 5 was preferable as, statistically, all these variants outperformed that with $\Gamma = 5$ for all frequencies of change.

TABLE 9. Effect of Γ.

Γ	FE_{change}	Better	Equal	Worse	p	Decision
$\Gamma = 5$ vs $\Gamma = 1$	500	**67**	15	9	0.000	+
	1000	**62**	15	14	0.000	+
	1500	**60**	15	16	0.000	+
$\Gamma = 5$ vs$\Gamma = 10$	500	19	15	**57**	0.000	-
	1000	21	15	**55**	0.001	-
	1500	18	15	**58**	0.000	-
$\Gamma = 5$ vs$\Gamma = 25$	500	11	15	**65**	0.000	-
	1000	12	15	**64**	0.000	-
	1500	12	15	**64**	0.000	-
$\Gamma = 5$ vs$\Gamma = 50$	500	13	15	**63**	0.000	-
	1000	12	15	**64**	0.000	-
	1500	15	15	**61**	0.000	-

TABLE 10. Friedman rankings for proposed algorithm with different values of Γ.

FE_{change}	$\Gamma = 1$	$\Gamma = 5$	$\Gamma = 10$	$\Gamma = 25$	$\Gamma = 50$
500	3.46	4.35	2.90	2.34	**1.96**
1000	3.49	4.21	2.95	2.35	**2.00**
1500	3.52	4.07	2.97	2.27	**2.18**
Average ranking	3.49	4.21	2.94	2.32	**2.05**

TABLE 11. Average feasiblity rates (FR%) for proposed algorithm with different values of Γ.

FE_{change}	$\Gamma = 1$	$\Gamma = 5$	$\Gamma = 10$	$\Gamma = 25$	$\Gamma = 50$
500	57.98%	58.51%	58.55%	58.27%	**58.89%**
1000	65.89%	66.29%	**66.37%**	65.29%	65.29%
1500	**68.31%**	67.78%	67.56%	67.11%	67.11%
Average FR	64.06%	**64.19%**	64.16%	63.56%	63.76%

TABLE 12. Effect of selecting *P*.

P	FE_{change}	Better	Equal	Worse	p	Decision
Best P vs Random P	500	37	15	**39**	0.675	$\approx$
	1000	**42**	15	34	0.649	$\approx$
	1500	**39**	15	37	0.428	$\approx$

Based on the overall Friedman rankings shown in Table 10 setting Γ at a value of 50 was the best variant for all frequencies of change.

Also the average FRs presented in Table 11 show that the variant with $\Gamma = 50$ had the highest rate of feasible solutions for $FE_{change} = 500$ while setting the value of Γ to 10 and 1 was the best when $FE_{change} = 1000$ and 1500, respectively. Overall, as no single value of Γ was the best, it was considered that there was a bias towards setting it to 10 based on the average results and overall FRs.

5) EFFECT OF SELECTING *P*

Given the value of $\Gamma = 10$ found above, the way of selecting P individuals was analyzed by choosing them randomly. Detailed fitness values are reported in Appendices F-1 to F-3 in the supplementary material, with a summary of comparisons shown in Table 12. These results indicate that selecting P randomly was better than selecting it based on its fitness when there were more frequent changes (i.e., $FE_{change} = 500$) while selecting the best one was preferable when $FE_{change} = 1000$ and 1500. However, statistically, no significant difference was found.

TABLE 13. Average feasibility rates (FR%) for proposed algorithm with different selections of P.

FE_{change}	Best P	Random P
500	58.55%	**58.73%**
1000	**66.37%**	66.11%
1500	**67.56%**	67.47%
Average FR	**64.16%**	64.10%

TABLE 14. Average running time, in seconds, for MODE variants in seconds with different FE_{change}.

Algorithms	FE_{change}		
	500	1000	1500
MODE	0.17	0.40	0.59
MODE(re-initialization)	0.17	0.40	0.57
MODE(proposed)	0.17	0.40	0.57

Also, the average FRs presented in Table 13 show that selecting P individuals randomly enabled a high rate of feasibility when $FE_{change} = 500$, while selecting the best individuals brought more benefits to overall FR when $FE_{change} = 1000$ and 1500.

In summary, based on the average results and overall FRs, selecting the best P individuals was considered optimum.

F. RUNNING TIME

In this section, we compare the running time of MODE, MODE with re-initialization and MODE with the proposed components. The algorithms were run on a PC with a 3.10 GHz Core (TM) i7 processor, 72GB RAM and Windows 10. As the optimal solutions of the majority of the environments are not known, the algorithms were run up to the maximum number of fitness evaluations ($FE_{change} = 500$, 1000 and 1500).

Given the calculation of the the big-Oh discussed in section III-D, we did not expect significant extra running time, which was consistent with the time (in seconds) reported in Table 14. From that Table, it is clear that the proposed components were computationally efficient, as no significant additional time was required for all variants over all FE_{change}. Note that the algorithm that produces better quality solutions with similar running time is also considered as the better performed approach.

V. TESTING PROPOSED APPROACH WITH OTHER EAs

The performance of the proposed approach using 4 algorithms, (1) DE1 (rand-to-p best) (2) DE2 (current-to-p best) (3) multi-parent crossover GA (MPC-GA) [39] and (4) success-history based parameter adaptation DE (SHADE) [40] were compared. DE1 and DE2 used the same parameter settings as in [34] and SHADE and MPC-GA the same ones as those reported in corresponding papers, with all the algorithms having a NP of 100 solutions.

Three variants of each algorithm were run, that is, with and without the proposed approach, and re-initializing the population after every change. Similar to the analysis discussed in the previous sections, all the versions were evaluated with $FE_{change} = 500$, 1000 and 1500 FEs. Detailed results are shown in Appendices G-1 to J-3 and a summary of comparisons in Tables 15 to 17.

From the tables, it is clear that all the algorithms using the proposed approach dominated both those without it and those with a re-initialization mechanism for all FE_{change} values. This was confirmed by the statistical tests carried out, that is, all the algorithms using the proposed approach were statistically superior to the other versions.

The FR presented in Table 18 reveal that using the proposed approach enabled the algorithms to obtain a higher rates of feasible solutions compared with those of the others.

VI. COMPARISONS WITH OTHER EXISTING ALGORITHMS FOR DCOPs

The proposed algorithm was compared with two existing algorithms, known as dynamic constrained optimization DE (DyCODE) [41] and clustering particle swarm optimizer (CPSO) [42] [1]. It was run with the same parameters settings as those reported in its corresponding paper with $NP = 100$ solutions and $FE_{change} = 500$, 1000 and 1500. Detailed results are shown in Appendices K-1 to K-3. To be consistent with the comparison technique used in [41], all algorithms were compared based on (1) the modified offline error for DCOPs (offline error for short), and (2) the normalized score. The former was calculated as:

$$e = \frac{1}{t_{max}} \sum_{t=1}^{t_{max}} (\mathbf{x}_t^* - \mathbf{x}_{best,t})$$

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

while the normalized score was calculated as:

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

where G is the number of test problems, N is the number of compared algorithms, $e(I,j)$ the modified offline error of algorithm I in problem j, and $e_{max}(j)$ and $e_{min}(j)$ are 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 having the best performance.

The average offline error results presented in Table 19 show that the proposed algorithm performed better than DyCODE for all the test problems with a quick change frequency (i.e., 500 FFEs). This was also true for the other change frequencies, except for only 1 and 2 problems at $FE_{change} = 1000$ and $FE_{change} = 1500$, respectively.

The comparison between the proposed algorithm and CPSO revealed that the proposed approach outperformed

[1] the source code of both algorithms was provided by the authors of [41]

TABLE 15. summary of comparisons between different variants of DE1, DE2, MPC-GA and SHADE with $FE_{change} = 500$.

Algorithms	Better	Equal	Worse	p	Decision
DE1(proposed) *vs.* DE1	**64**	15	12	0.00	+
DE1(proposed) *vs.* DE1(re-initialization)	**77**	13	1	0.00	+
DE2(proposed) *vs.* DE2	**64**	15	12	0.00	+
DE2(proposed) *vs.* DE2 (re-initialization)	**77**	13	1	0.00	+
MPC-GA(proposed) *vs.* MPC-GA	**66**	13	12	0.00	+
MPC-GA(proposed) *vs.* GA (re-initialization)	**74**	13	4	0.00	+
SHADE(proposed) *vs.* SHADE	**73**	2	16	0.00	+
SHADE(proposed) *vs.* SHADE (re-initialization)	**77**	13	1	0.00	+

TABLE 16. summary of comparisons between different variants of DE1, DE2, MPC-GA and SHADE with $FE_{change} = 1000$.

Algorithms	Better	Equal	Worse	p	Decision
DE1(proposed) vs. DE1	**69**	15	7	0.00	+
DE1(proposed) vs. DE1(re-initialization)	**78**	13	0	0.00	+
DE2(proposed) vs. DE2	**69**	15	7	0.00	+
DE2(proposed) vs. DE2 (re-initialization)	**77**	13	1	0.00	+
MPC-GA(proposed) vs. MPC-GA	**64**	13	14	0.00	+
MPC-GA(proposed) vs. GA (re-initialization)	**66**	13	12	0.00	+
SHADE(proposed) vs. SHADE	**69**	15	7	0.00	+
SHADE(proposed) vs. SHADE (re-initialization)	**77**	13	1	0.00	+

TABLE 17. Summary of comparisons summary between different variants of DE1, DE2, MPC-GA and SHADE with $FE_{change} = 1500$.

Algorithms	Better	Equal	Worse	p	Decision
DE1(proposed) vs. DE1	**60**	15	16	0.00	+
DE1(proposed) vs. DE1(re-initialization)	**77**	13	1	0.00	+
DE2(proposed) vs. DE2	**66**	15	10	0.00	+
DE2(proposed) vs. DE2 (re-initialization)	**78**	13	0	0.00	+
MPC-GA(proposed) vs. MPC-GA	**52**	15	24	0.15	+
MPC-GA(proposed) vs. GA (re-initialization)	**65**	15	11	0.00	+
SHADE(proposed) vs. SHADE	**66**	15	10	0.00	+
SHADE(proposed) vs. SHADE (re-initialization)	**78**	13	0	0.00	+

TABLE 18. Average feasiblity rate (FR%) for different variants of DE1, DE2, MPC-GA and SHADE.

FE_{change}	500	1000	1500	Average FR
DE1(proposed)	**59.03%**	**66.24%**	**67.56%**	**64.28%**
DE1	58.46%	66.02%	67.16%	63.88%
DE1(re-initialization)	42.77%	46.55%	57.45%	48.92%
DE2(proposed)	**58.51%**	**65.93%**	**68.40%**	**64.28%**
DE2	56.79%	65.49%	68.44%	63.58%
DE2(re-initialization)	42.20%	45.63%	56.09%	47.97%
MPC-GA(proposed)	**61.41%**	**65.10%**	**65.54%**	**64.01%**
MPC-GA	60.66%	62.15%	60.00%	60.94%
GA (re-initialization)	49.32%	67.08%	69.45%	61.95%
SHADE(proposed)	**58.51%**	**65.93%**	**68.40%**	**64.28%**
SHADE	57.27%	65.49%	68.44%	63.74%
SHADE (re-initialization)	42.20%	45.63%	56.09%	47.97%

CPSO for all the problems with $FE_{change} = 500$, except $G01$ and $G02$. We think that the diversity mechanism and the local search of CPSO played a role in its superiority for those two problems only, especially when multi-modality exists ($G02$). For all the problems with $FE_{change} = 1000$ and 1500, the proposed algorithm dominates CPSO.

Also, it was noted that CPSO works better than DyCODE for problems with $FE_{change} = 500$, but was worse for $FE_{change} = 1000$ and 1500.

As expected, due to its ability to obtain better results, the normalized scores, shown in Table 20, clearly demonstrate the superiority of the proposed approach for all change frequencies.

In addition to the above-mentioned comparison criteria, the average FRs obtained by each algorithm are reported in Table 21. The results clearly demonstrate that the proposed algorithm could attain high rates of feasibility for all change frequencies.

TABLE 19. Average offline errors obtained by proposed approach, DyCODE and CPSO with different dynamic change rates.

FE_{change}	500			1000			1500		
prob	DyCODE	CPSO	proposed	DyCODE	CPSO	proposed	DyCODE	CPSO	proposed
G01	1.75E+01	**8.68E+00**	1.28E+01	9.81E+00	8.43E+00	**5.96E+00**	6.87E+00	7.27E+00	**4.76E+00**
G02	6.10E-01	**4.83E-01**	5.04E-01	5.06E-01	4.28E-01	**4.26E-01**	4.20E-01	3.97E-01	**3.71E-01**
G04	8.96E+02	8.78E+02	**5.14E+02**	5.02E+02	6.55E+02	**3.00E+02**	2.99E+02	5.34E+02	**2.14E+02**
G06	2.62E+03	3.00E+03	**1.33E+03**	1.24E+03	1.84E+03	**7.16E+02**	9.13E+02	1.50E+03	**3.75E+02**
G07	1.01E+03	7.94E+02	**6.21E+02**	9.06E+02	3.72E+02	**1.87E+02**	2.18E+02	2.62E+02	**9.61E+01**
G08	5.39E-02	4.06E-02	**8.27E-03**	4.71E-03	1.45E-02	**1.02E-03**	7.22E-04	6.17E-03	**1.85E-04**
G09	2.38E+04	7.97E+03	**1.89E+02**	2.29E+02	1.87E+02	**5.80E+01**	4.87E+01	8.49E+01	**2.72E+01**
G10	5.64E+03	5.64E+03	**6.67E+03**	5.64E+03	5.64E+03	**1.60E+03**	1.08E+03	5.64E+03	**3.91E+01**
G12	7.92E-03	5.33E-03	**3.33E-03**	2.08E-03	1.77E-03	**1.34E-03**	**4.96E-04**	1.08E-03	6.20E-04
G16	5.26E-01	4.97E-01	**3.99E-01**	1.92E-02	2.54E-01	**6.71E-02**	**9.43E-02**	2.09E-01	1.09E-01
G18	2.24E+01	2.29E+01	**1.91E+01**	8.73E+00	1.44E+01	**5.53E+00**	4.20E+00	6.33E+00	**2.13E+00**
G19	3.94E+03	1.65E+03	**8.57E+02**	1.37E+03	1.26E+03	**4.48E+02**	5.98E+02	9.23E+02	**2.65E+02**
G24	2.85E-01	1.66E-01	**1.04E-01**	6.53E-02	6.67E-02	**2.53E-02**	1.96E-02	3.95E-02	**1.15E-02**

TABLE 20. normalized score of proposed algorithm, DyCODE and CPSO at different change frequencies.

	DyCODE	CPSO	MODE(proposed)
500	1.06E-01	5.19E-01	**8.74E-01**
1000	2.59E-01	2.20E-01	**9.84E-01**
1500	5.95E-01	3.55E-02	**9.74E-01**

TABLE 21. Average feasibility rate (FR%) obtained by proposed approach, DyCODE and CPSO with different dynamic change rates.

Alg.	FE_{change}		
	500	1000	1500
DyCODE	48.44%	62.42%	67.56%
CPSO	52.1%	54.7%	60.1%
proposed	**58.55%**	**66.37%**	**66.46%**

VII. CONCLUSION AND FUTURE WORK

This paper introduced a new sensitivity-based direction of movement approach for solving DCOPs, with their sensitivity values calculated based on the slack value of each constraint. By utilizing this information, in every new environment, the solutions obtained by an optimization algorithm in the previous environment were divided into a number of sub-groups depending on the number of sensitive constraints. Subsequently, the solutions in each sub-group were updated to move towards a new promising search area based on the slack value of its corresponding constraint.

The algorithm was incorporated in several optimization algorithms and tested on 13 simulated dynamic constrained problems, each of which had 7 dynamic changes which occurred every 500, 1000 and 1500 FEs. The results revealed that EAs with the proposed approach were superior to both those without it and re-initialization methods. Also, the proposed algorithm significantly outperformed another state-of-the-art algorithm.

Different components (P, Γ and Δ) of the algorithm were analyzed. It was concluded that: (1) applying a proposed move on the best 50% of the population led to better results; (2) the distance each decision variable may move was recommended to be up to 25% of the change in the constraint on the right-hand side; and (3) keeping information from previous environments helped the algorithm obtain better solutions.

It is worthwhile to mention here that we noticed that if a change in the environment happens quickly (i.e., if sufficient fitness evaluations are not permitted), the detection of a sensitive constraint may not be accurate. A possible general remedy for this drawback, may be to develop another mechanism that would be able to detect the correct sensitive constraints quickly or decide the search based on the evaluation of a subset of the constraints.

For possible future work, the proposed algorithm should be tested for different types of changes including bigger changes. Also, including dynamic changes in the range of variable as well as objective function would be worth investigating. Another important future task is to incorporate changes in the objective function and then test the algorithm. In that case, information about the quality of the move (in both the constraint and objective function spaces) could possibly be used in the proposed method.

ACKNOWLEDGMENT

The authors would like to thank Prof. Yong Wang for kindly providing the source code of DyCODE and CPSO [41].

REFERENCES

[1] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 3, 1999, pp. 1875–1882.

[2] R. Azzouz, S. Bechikh, L. B. Said, and W. Trabelsi, "Handling time-varying constraints and objectives in dynamic evolutionary multi-objective optimization," *Swarm Evol. Comput.*, vol. 39, pp. 222–248, Apr. 2018.

[3] H. Richter and S. Yang, "Memory based on abstraction for dynamic fitness functions," in *Proc. Workshops Appl. Evol. Comput.* Berlin, Germany: Springer, 2008, pp. 596–605.

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

[5] C. Bu, W. Luo, and L. Yue, "Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies," *IEEE Trans. Evol. Comput.*, vol. 21, no. 1, pp. 14–33, Feb. 2017.

[6] A. Ahrari, S. Elsayed, R. Sarker, and D. Essam, "A new prediction approach for dynamic multiobjective optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 2269–2276.

[7] Y. Diao, C. Li, S. Zeng, M. Mavrovouniotis, and S. Yang, "Memory-based multi-population genetic learning for dynamic shortest path problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 2277–2284.

[8] D. Gong, B. Xu, Y. Zhang, Y. Guo, and S. Yang, "A similarity-based cooperative co-evolutionary algorithm for dynamic interval multiobjective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 24, no. 1, pp. 142–156, Feb. 2020.

[9] M. Rong, D. Gong, Y. Zhang, Y. Jin, and W. Pedrycz, "Multidirectional prediction approach for dynamic multiobjective optimization problems," *IEEE Trans. Cybern.*, vol. 49, no. 9, pp. 3362–3374, Sep. 2019.

[10] B. Xu, Y. Zhang, D. Gong, Y. Guo, and M. Rong, "Environment sensitivity-based cooperative co-evolutionary algorithms for dynamic multi-objective optimization," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 15, no. 6, pp. 1877–1890, Nov. 2018.

[11] M.-Y. Ameca-Alducin, E. Mezura-Montes, and N. Cruz-Ramírez, "A repair method for differential evolution with combined variants to solve dynamic constrained optimization problems," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, 2015, pp. 241–248.

[12] M.-Y. Ameca-Alducin, E. Mezura-Montes, and N. Cruz-Ramírez, "Differential evolution with a repair method to solve dynamic constrained optimization problems," in *Proc. Companion Publication Genet. Evol. Comput. Conf. (GECCO Companion)*, 2015, pp. 1169–1172.

[13] H. Richter, "Memory design for constrained dynamic optimization problems," in *Proc. Eur. Conf. Appl. Evol. Comput.* Berlin, Germany: Springer, 2010, pp. 552–561.

[14] M.-Y. Ameca-Alducin, E. Mezura-Montes, and N. Cruz-Ramirez, "Differential evolution with combined variants for dynamic constrained optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2014, pp. 975–982.

[15] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 6, pp. 1–24, Oct. 2012.

[16] C. Li, T. T. Nguyen, M. Yang, M. Mavrovouniotis, and S. Yang, "An adaptive multipopulation framework for locating and tracking multiple optima," *IEEE Trans. Evol. Comput.*, vol. 20, no. 4, pp. 590–605, Aug. 2016.

[17] S. Jiang and S. Yang, "Evolutionary dynamic multiobjective optimization: Benchmarks and algorithm comparisons," *IEEE Trans. Cybern.*, vol. 47, no. 1, pp. 198–211, Jan. 2017.

[18] S. B. Gee, K. C. Tan, and H. A. Abbass, "A benchmark test suite for dynamic evolutionary multiobjective optimization," *IEEE Trans. Cybern.*, vol. 47, no. 2, pp. 461–472, Feb. 2017.

[19] H. K. Singh, A. Isaacs, T. T. Nguyen, T. Ray, and X. Yao, "Performance of infeasibility driven evolutionary algorithm (IDEA) on constrained dynamic single objective optimization problems," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 3127–3134.

[20] H. Richter, "Detecting change in dynamic fitness landscapes," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 1613–1620.
[21] S. Yang, "Memory-based immigrants for genetic algorithms in dynamic environments," in *Proc. Conf. Genet. Evol. Comput. (GECCO)*, 2005, pp. 1115–1122.
[22] K. Deb, "An efficient constraint handling method for genetic algorithms," *Comput. Methods Appl. Mech. Eng.*, vol. 186, nos. 2–4, pp. 311–338, Jun. 2000.
[23] M. Hasani-Shoreh, M.-Y. Ameca-Alducin, W. Blaikie, F. Neumann, and M. Schoenauer, "On the behaviour of differential evolution for problems with dynamic linear constraints," 2019, *arXiv:1905.04099*. [Online]. Available: http://arxiv.org/abs/1905.04099
[24] H. Richter and F. Dietel, "Solving dynamic constrained optimization problems with asynchronous change pattern," in *Proc. Eur. Conf. Appl. Evol. Comput.* Berlin, Germany: Springer, 2011, pp. 334–343.
[25] K. Pal, C. Saha, S. Das, and C. A. C. Coello, "Dynamic constrained optimization with offspring repair based gravitational search algorithm," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 2414–2421.
[26] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "GSA: A gravitational search algorithm," *J. Inf. Sci.*, vol. 179, no. 13, pp. 2232–2248, 2009.
[27] M.-Y. Ameca-Alducin, E. Mezura-Montes, and N. Cruz-Ramírez, "Dynamic differential evolution with combined variants and a repair method to solve dynamic constrained optimization problems: An empirical study," *Soft Comput.*, vol. 22, no. 2, pp. 541–570, 2018.
[28] T. T. Nguyen and X. Yao, "Continuous dynamic constrained optimization—The challenges," *IEEE Trans. Evol. Comput.*, vol. 16, no. 6, pp. 769–786, Dec. 2012.
[29] X. Lu, K. Tang, and X. Yao, "Speciated evolutionary algorithm for dynamic constrained optimisation," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Cham, Switzerland: Springer, 2016, pp. 203–213.
[30] J. J. Grefenstette, "Genetic algorithms for changing environments," in *Proc. PPSN*, vol. 2, 1992, pp. 137–144.
[31] E. Mezura-Montes, C. A. C. Coello, and E. I. Tun-Morales, "Simple feasibility rules and differential evolution for constrained optimization," in *Proc. Mexican Int. Conf. Artif. Intell.* Berlin, Germany: Springer, 2004, pp. 707–716.
[32] M.-Y. Ameca-Alducin, M. Hasani-Shoreh, and F. Neumann, "On the use of repair methods in differential evolution for dynamic constrained optimization," in *Proc. Int. Conf. Appl. Evol. Comput.* Cham, Switzerland: Springer, 2018, pp. 832–847.
[33] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Trans. Evol. Comput.*, vol. 10, no. 4, pp. 440–458, Aug. 2006.
[34] S. Elsayed, R. Sarker, and C. A. C. Coello, "Enhanced multi-operator differential evolution for constrained optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2016, pp. 4191–4198.
[35] J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. C. Coello, and K. Deb, "Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization," *J. Appl. Mech.*, vol. 41, no. 8, pp. 8–31, 2006.
[36] T. T. Nguyen, "A proposed real-valued dynamic constrained benchmark set," School Comput. Sci., Univ. Birmingham, Tech. Rep., 2008.
[37] C. Li, S. Yang, T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H. Beyer, and P. Suganthan, "Benchmark generator for CEC 2009 competition on dynamic optimization," Univ. Leicester, Leicester, U.K., Univ. Birmingham, Birmingham, U.K., Nanyang Technol. Univ., Singapore, Tech. Rep., 2008.
[38] G. Corder and D. Foreman, "Nonparametric statistics: An introduction," in *Nonparametric Statistics for Non-Statisticians: A Step-By-Step Approach*. Hoboken, NJ, USA: Wiley, 2009, pp. 101–111.
[39] S. M. Elsayed, R. A. Sarker, and D. L. Essam, "GA with a new multi-parent crossover for solving IEEE-CEC2011 competition problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2011, pp. 1034–1040.
[40] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 71–78.
[41] Y. Wang, J. Yu, S. Yang, S. Jiang, and S. Zhao, "Evolutionary dynamic constrained optimization: Test suite construction and algorithm comparisons," *Swarm Evol. Comput.*, vol. 50, Nov. 2019, Art. no. 100559.
[42] S. Yang and C. Li, "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 959–974, Dec. 2010.

NOHA HAMZA received the M.Sc. and Ph.D. degrees in computer science from the University of New South Wales (UNSW) at Canberra, Australia, in 2012 and 2016, respectively. She is currently a Research Associate with the School of Engineering and Information Technology (SEIT), UNSW at Canberra. Her research interest includes the areas of evolutionary algorithms and constrained optimization.

RUHUL SARKER (Member, IEEE) received the Ph.D. degree from Dalhousie University, Halifax, Canada, in 1992. He is currently a Professor with the School of Engineering and Information Technology and the Director of the Faculty Postgraduate Research, University of New South Wales at Canberra, ACT, Australia. He is the lead author of the book *Optimization Modelling: A Practical Approach* (CRC Press, 2007) and published over 300 refereed paper in international journals, edited books, and conference proceedings. His current research interests include evolutionary optimization and applied operations research. He is currently an Associate Editor of *Memetic Computing* journal, the *Journal of Industrial and Management Optimization*, and *Flexible Service and Manufacturing Journal*.

DARYL ESSAM received the B.Sc. degree from the University of New England, Portland, ME, in 1990, and the Ph.D. degree from the University of New South Wales at Canberra, Australia, in 2000. He was an Associate Lecturer with the University of New England, Portland, ME, in 1991 and has been with the University of New South Wales at Canberra, since 1994, where he is currently a Senior Lecturer. His research interest includes genetic algorithms, with a focus on both genetic programming and multiobjective optimization. He was a Finance Chair of the IEEE CEC 2003 and a Proceedings Co-Chair for WCCI 2012.