Computers, Materials & Continua
DOI: 10.32604/cmc.2023.027448
Article

Tech Science Press

Modified Differential Evolution Algorithm for Solving Dynamic Optimization with Existence of Infeasible Environments

Mohamed A. Meselhi*, Saber M. Elsayed, Daryl L. Essam and Ruhul A. Sarker

School of Engineering and Information Technology University of New South Wales Canberra, Australia
*Corresponding Author: Mohamed A. Meselhi. Email: m.radwan@unsw.edu.au
Received: 19 January 2022; Accepted: 06 June 2022

Abstract: Dynamic constrained optimization is a challenging research topic in which the objective function and/or constraints change over time. In such problems, it is commonly assumed that all problem instances are feasible. In reality some instances can be infeasible due to various practical issues, such as a sudden change in resource requirements or a big change in the availability of resources. Decision-makers have to determine whether a particular instance is feasible or not, as infeasible instances cannot be solved as there are no solutions to implement. In this case, locating the nearest feasible solution would be valuable information for the decision-makers. In this paper, a differential evolution algorithm is proposed for solving dynamic constrained problems that learns from past environments and transfers important knowledge from them to use in solving the current instance and includes a mechanism for suggesting a good feasible solution when an instance is infeasible. To judge the performance of the proposed algorithm, 13 well-known dynamic test problems were solved. The results indicate that the proposed algorithm outperforms existing recent algorithms with a margin of 79.40% over all the environments and it can also find a good, but infeasible solution, when an instance is infeasible.

Keywords: Dynamic optimization; constrained optimization; disruption; differential evolution

1 Introduction

Dynamic constrained optimization is necessary for solving many real-world problems in various fields [1,2]. In it, it is assumed that the objective function and/or constraint functions change over time. In particular changes can occur in one or more of the coefficients of the objective function, coefficients of the constraint functions, right-hand side of the constraints, and variable bounds. In some cases, the function's structure may be changed, and the addition and removal of constraints may occur. Here, we assume that each environment is represented by a problem instance. With the new environment, the problem instance may have a completely different or partially overlapped feasible space with an optimal solution probably located in a different place in the new feasible space. In such problems, the dynamics of the constraints impose more challenges that modify the shape and size of the feasible area [3]. In such a case, the objective of a solver is to find the best feasible solution in the current and subsequent environments. Examples of real-world dynamic problems include hydro-thermal scheduling [4] and ship scheduling [5].

Detecting changes in environments and tracking the changing optimum are two important tasks for solving dynamic optimization problems (DOPs). Many studies have been conducted to track the changing optimum [6–12]. Despite the large body of literature on solving dynamic problems, the majority of approaches focus mainly on unconstrained problems, while most real-world dynamic problems are dynamic constrained optimization problems (DCOPs) that have not yet been sufficiently explored. In DCOPs, handling constraints is an additional task dealt with using different methods [12,13]. However, solving DCOPs is computationally expensive [14,15] and their performances deteriorate if there are different types of changes in the environment [13–16].

To judge the performances of new algorithms, some synthetic constrained test problems, in which all instances are feasible, are proposed in the literature. However, in practice, some problem environments may not be feasible owing to various real-life issues; for example, their resource requirements can be significantly higher than expected due to defects in their supplied materials, shortages of resources in a given time period due to delivery disruptions, capacity reductions due to machine breakdowns and unexpected increases in market demand [17]. That means, it is possible that in some DCOPs, one or more problem instances can be infeasible, which does not suit decision-makers as infeasible solutions cannot be implemented. However, this does not mean that no activities will be performed in that time period. In this case, finding the closest feasible solution with suggestions for minimum changes in the problem instance would be beneficial for decision-makers. For example, it could be a suggestion to scale down activities for later continuation.

When solving optimization problems using EAs, it is necessary to define the search space, which is usually limited by variable bounds. The feasible space of a constrained problem is limited by both its constraint functions and variable bounds. For this reason, the feasible space is usually smaller than the search space, but they can be equal in some cases. To optimize a problem, we are interested in finding the best feasible solution, which is usually located on the boundary of the feasible space for most practical problems [18], but for some the optimum solution can be located inside this feasible space. If we can find the constraint limits that ensure feasibility, let us call this ‘limits of feasibility’, it is possible to suggest the nearest feasible solution for an infeasible problem instance.

Motivated by the above-mentioned facts, the aim of this paper is firstly, to develop a mechanism that determines the limits of feasibility, that can help to deal with possible infeasibility in problem instances. Secondly, a knowledge-transfer mechanism that will allow the retrieval and exploitation of solutions from past environments is proposed. Finally, to solve DCOPs with the existence of infeasible environments, a multi-operator-based differential evolution (DE) algorithm is developed by incorporating those two mechanisms. The performance of the proposed algorithm was tested by solving 13 single objective DCOPs with dynamic inequality constraints and compared its solutions with those of state-of-the-art dynamic constrained optimization algorithms. The experimental results revealed that the proposed approach can obtain high-quality solutions in a limited amount of time and is capable of dealing with infeasible instances, so it is suited for all environments.

The rest of this paper is organized as follows: in Section 2, EAs for solving DCOPs are reviewed; in Section 3, DCOPs are introduced and the proposed approach is explained in detail; in Section 4, the experimental results are discussed and the performance of the proposed method analyzed; and, finally, conclusions are drawn and future research directions suggested in Section 5.

2 Related Literature

As of the literature, evolutionary algorithms (EAs) are widely accepted as effective methods for solving static optimization problems. It is a common trend to extend EAs developed for static problems to solve dynamic ones. However, in this situation, they converge to an optimum by reducing the population's diversity, which requires additional mechanisms to adapt to the changed environment if the final population of a previous environment is used as an initial population of the current environment [19]. It is to be noted that many approaches begin with the final population from the previous environment when a change is detected. However, this method does not help to transfer an effective learning experience from one environment to the next. There are several approaches for tracking the changing optimum, that include introducing diversity [4–6], diversity-maintaining [7,8], memory-based [9], prediction-based [10,11] and multi-population approaches [12–20].

Compared to DOPs, not much work has been proposed in the literature for solving DCOPs. Thus, some EAs originally designed for static problems were adapted to solve dynamic constrained ones. Singh et al. [21] adjusted an infeasibility-driven EA (IDEA) to handle changes using a shift-detecting technique. This strategy's fundamental idea is that a random individual is chosen and tested in each generation as a change detector and when a change is observed, the entire population is re-assessed. The experimental results show the superior performance of this strategy to those of state-of-the-art algorithms for two test problems, although its detection capability may be affected if the detector is randomly selected.

A memory-based method is another EA strategy proposed for solving DCOPs. A study presented by Richter [16] focused on adapting an abstract memory strategy [22], with two memory schemes called censoring and blending suggested. In them, good individuals and their related data are stored for later use in constrained and unconstrained problems. From evaluating this proposed algorithm, the results reveal that using these two schemes while adjusting the strategy achieves better performance when compared to a penalty function. Its main drawback is that it works well on specific types of constrained dynamic problems, particularly cyclic ones.

In a study by Richter et al. [23], attention is paid to comparing the blending and censoring memory schemes with three other different strategies for a particular type of DCOP. The main idea in these problems is that both the objective function and constraints could vary in an asynchronous change pattern. From the experimental results obtained, it is observed that the memory-based schemes perform better than the random approaches. Generally, the former often suffer from solving problems with environmental changes that do not appear previously.

Ameca-Alducin et al. [24] proposed a diversity introduction mechanism by incorporating two popular DE variants and memory of the best-found solutions in the search. It detects changes in the objective function and/or constraints to elevate diversity in the population while following a new peak. Also, random immigrants and a local search operator are integrated to enable faster convergence. Nevertheless, it cannot support adequate diversity as an infeasible solution is always considered to be worse than a feasible one.

Also, some studies consider a repairing strategy for solving DCOPS. That by Pal et al. [25] attempts to improve solutions by proposing a GSA + Repair method within a gravitational search algorithm (GSA) [26]. Its key idea is that there is an infeasible solution that moves towards the nearest feasible one chosen from the best solutions available in the reference set. This method is evaluated, with the empirical results showing that it exhibits superior performances to those of other state-of-the-art algorithms. Two issues with this strategy are firstly, if the feasible space is small and, secondly, previously feasible individuals need to be re-checked to determine if they are still available when there is a change.

In another significant study proposed by Nguyen [27], DCOPs are solved using a set of algorithms and strategies. The algorithms consist of current dynamic optimization (DO) mechanisms (i.e., random-immigrant and hyper-mutation) and two constraint handling (CH) strategies which are repairing and stochastic ranking. However, there are some drawbacks of using DO and CH to solve DCOPs, which include tracking feasible moving regions, maintaining diversity, detecting changes, updating the feasibility and infeasibility balance, and searching outside the range.

Bu et al. [12] proposed the LTFR-DSPSO method to find and follow multiple disconnected feasible regions simultaneously using multiple sub-populations, each of which applies a gradient-based repair operator and local search to accelerate locating feasible areas and enhance exploiting them, respectively. Also, it employs a memory-based approach to track earlier found feasible areas by archiving previously found local optima and best solutions. Moreover, it involves a linear prediction method to predict the locations of future feasible areas.

Another strategy, called sensitive constraint detection, was introduced by Hamza et al. [13]. It attempts to offer meaningful information to help determine the movement of a solution in changing environments. One of its merits is that merging it with a search process accelerates the algorithm's convergence. Also, using it with a multi-operator EA can be helpful for constrained dynamic problems. However its performance deteriorates when solving problems with relatively large changes.

3 Materials and Methods

In this section, a framework for solving DCOPs is proposed. Firstly, a brief review of the optimization algorithm employed in this paper is provided. Then, an overview of the proposed framework is presented and then details of its components are discussed.

3.1 MODE Optimizer

The optimizer used in this method is based on multi-operator differential evolution (MODE) [28], where a population of size *NP* is randomly generated within the search space. Then, the solutions are randomly divided into two equal sub-populations, each of which is randomly assigned to one of two distinct mutation strategies, namely, *DE/current-to-p best* or *DE/rand-to-p best*, to generate new offspring. A selection mechanism is conducted between every offspring and its parent to decide the fittest solutions that survive to the following generation. Then, based on improvements calculated using the three criteria of the feasibility rate, population diversity and quality of solutions, the sizes of the sub-populations are updated in the next generation.

3.2 Proposed Approach

3.2.1 Dynamic Constrained Optimization

A DCOP is an optimization problem in which one or more of the functions and/or parameters change over time. The functions include the objective function and constraint functions (i.e., constraint left-hand side). In a given environment, it is considered a static problem instance which does not change for a given time period (i.e., the duration of the environment). In general, a DCOP can be expressed as

$$min/max\, f\,(x, t) \tag{1}$$

subject to $g_k(x,t) \leq 0, \forall k = 1, 2, \ldots, K$

$h_e(x, t) = 0, \forall e = 1, 2, \ldots, E$

$L_j(t) \leq x_{t,j} \leq U_j(t), \forall j = 1, 2, \ldots, D$

where $x \in S(t) \subset R^{D(t)}$, $x = [x_{t,1}, \ldots, x_{t,j}, \ldots, x_{t,D(t)}]^T$ is a solution vector of D variables that varies over time t, and f a single objective function. This problem has K inequality constraints ($g_k(x)$) and E equality ones ($h_e(x)$), and each decision variable ($x_{t,j}$) has lower and upper bounds ($L_j(t)$ and $U_j(t)$, respectively). The feasible space ($F(t)$) $\subseteq$ search space ($S(t)$), where a feasible solution for a time t is a point that fulfils all the constraints. The aim is to find a feasible solution in each environment which minimizes or maximizes $f(x, t)$, with all feasible points described as: a feasible solution for a time t that meets all the constraints and satisfies the boundary limits.

DCOPs can be classified as one of three types based on the dynamics of its objective function and constraints as: 1) the objective function is dynamic and the constraints static [29]; 2) the objective function is static while the constraints are dynamic [13]; and 3) both the objective function and constraints are dynamic [30]. For all these categories, global solutions' movement and appearance after each change is affected by the infeasible regions.

The structure of the proposed framework is shown in Algorithm 1. It starts by randomly generating an initial population (P) of size NP $\left(\text{i.e.,} P = \left\{\vec{X}_1, \vec{X}_2, \ldots, \vec{X}_{\text{NP}}\right\}\right)$, with each individual evaluated and the number of fitness evaluations (FEs) updated. Then, the limits of feasibility are determined using the limit of feasibility (LOF) algorithm introduced in Section 3.2.2.

```
Algorithm 1: Dynamic Constrained Optimization
   $t$ : environment number
   $FEs$ : number of fitness evaluations
   $FEs_{max}$ : maximum number of fitness evaluations
   $solArc$ : previously found best solutions archive
   $chVal$ : the magnitude of the change archive
1  $t \leftarrow 0$;
2  $FEs \leftarrow 0$;
3  $solArc \leftarrow \phi$;
4  randomly generate initial population $X$ of size $NP$;
5  calculate the fitness value and constraint violations for each
     $X_i, \forall i = 1, 2, ..., NP$;
6  update $FEs$;
7  calculate the limit of feasibility ($LOF$) (see Section 3.2.2);
8  while $FEs \leq FEs_{max}$ do
9      if an environmental change is detected then
10         $t \leftarrow t + 1$;
11         $solArc_{t+1} \leftarrow [solArc_t, bestSol]$;
12         determine the $chVal$ (Algorithm 2);
13         Change Handling (Algorithm 3);
14     end
15     run multi-operator DE (see Section 3.1);
16     update $FEs$;
17     Record $xbest$;
18 end
```

Next, the algorithm enters a loop (Algorithm 1, lines 8–18) which has two main phases: 1) a search and 2) a change-handling technique. In the first, the multi-operator DE algorithm [28] discussed in Section 3.1 is used to optimize the solution vector (X) until a change occurs. Then, the best solution is recorded before proceeding to the next phase which deals with the environmental changes and updates the population for the new environment. Although the proposed framework can adapt to any optimizer, in this paper, we selected MODE due to its good performance in solving optimization problems [13–31].

As many real-world DCOPs have cyclical changes [12,26], a previously found feasible solution may become a solution again after some time, and so tracking solutions in previous environments might be efficient. In this paper, a new strategy which tracks both the location information of the previous best feasible solutions and the change value, to handle environmental changes, is proposed. In it, previous solutions are retrieved based on the similarity in their change values with the current environment. Two archives are used, i.e., $solArc(t)$ for the previously found best solutions in environment t and $chngArc(t)$ for the magnitude of the environmental change.

It is essential to identify when an environmental change occurs and its amount in many real-world DOPs [32]. A detector (i.e., certain individual) is employed to evaluate each constraint in the procedure for detecting change. A change occurs if the value of the detector in one environment is different from that in the previous one. Consequently, the change value is calculated as the difference between the evaluations in both environments, as described in Algorithm 2.

```
**Algorithm 2:** Change Detection
$g$ : number of constrains
$i$ : iteration
$chng$ : change detection
$chng \leftarrow zeros[1 \times g]$;
$chngVal \leftarrow zeros[1 \times g]$;
for $i \leftarrow 1 : g$ do
    if $g_k(x_t) \neq g_k(x_{t-1})$ then
        $chng[g] \leftarrow 1$ ;                /* A change is detected */
        $chngVal[g] \leftarrow g_k(x_t) - g_k(x_{t-1})$
    end
end
Return $chng, chngVal$;
```

In changing environments, the constraints may alter in a way that a solution in the search space that was feasible in the previous environment may become infeasible. This may happen due to changes in the right-hand side values of the constraints (e.g., the availability of certain resources) and also changes in either the constraint coefficients or right-hand side values, or both. We assume that the variable bounds will be unchanged (although they can be changed), but the constraint right-hand-side values can be changed. No feasible solution means that no action can be taken and there is discontinuity in the actions between the environments, which is not acceptable in practice. To maintain continuity in actions, we believe it will be beneficial to find the nearest feasible solution so that the practitioner can make adjustments to the resources required. In other words, this will generate a feasible solution with minimum changes in the values on the right-hand side (Algorithm 3, lines 2–4). Since the new environment may be near the previous one, some solutions from the latter may be adopted. Therefore, to transfer the knowledge of promising solutions from the previous environment to the new one, k number of best solutions in the former will replace the worst individuals in the latter

(Algorithm 3, lines 7–8). This number negatively correlates with the level of change, that is, *k* is small when *chngVal* is large and vice versa.

```
Algorithm 3: Change Handling
   $idx$ : environment index
1  $chngVal \leftarrow zeros[1 \times g]$;
2  if $chVal < LOF$ then
3      $chVal \leftarrow LOF$;                    /* adjust the limits */
4  end
5  $chngArc_t \leftarrow chVal$;
6  $chngArc_{t+1} \leftarrow [chngArc_t, chVal]$;
7  select $k$ best solutions $\propto \frac{1}{chVal}$;
8  $x(NP-k+1:NP)_{t+1} \leftarrow bestSol_1, ..., bestSol_k$;
9  $idx =$ index $(chVal$ in $chngArc)$;
10 if $idx == 0$ then
11     $dist \leftarrow abs(chVal - chngArc)$; $minVal \leftarrow min(dist)$;
        $minIdx \leftarrow find(dist == minVal)$;
12     if $minIdx > 1$ then
13         update $x_{t+1}$ with $solArc_{minIdx(1)}$& $solArc_{minIdx(2)}$;
14     else
15         update $x_{t+1}$ with $solArc_{minIdx(1)}$;
16     end
17 else if $idx > 1$ then
18     update $x_{t+1}$ with $solArc_{Idx(length(Idx))}$;
19 else
20     $x_{t+1} \leftarrow bestSol_{Idx}$; update $x_{t+1}$ with $solArc_{Idx}$;
21 end
```

Also, as environments with the same change may have similar optimal solutions, solutions from previous environments are beneficial for use in a similar environment. The search process can start where other environments end (Algorithm 3, lines 9–21), which assists obtaining high-quality solutions in a limited amount of time. Firstly, the magnitude of change (*chngVal*) is used to measure the similarities between the current and all previous environments. Then, if a previous environment has the same *chngVal* as the current one, its best solution will be retained in the new one.

Another scenario may be where several previous environments have the same *chngVal*. In this situation, the algorithm selects solutions from the most recent one which includes higher-quality solutions. In contrast, if *chngVal* does not exist in *chngArc*, this environment did not previously appear. Therefore, the algorithm retains the best solutions from the nearest environment (i.e., has a *chngVal* closest to the current one) in the next environment. If there is more than one close environment, their best solutions will remain in the next one.

3.2.2 Feasibility Measures

In this section, in a case of infeasibility, the idea of finding the nearest feasible solution is discussed. Firstly, an analytical example is provided and then a genetic algorithm-based approach presented.

a) Analytical Example

Let us consider a problem (G24 from [33]) as an example instance for a given environment, as

$$min f(x) = -x_1 - x_2 \tag{2}$$

subject to $g_1(x) = -2x_1^4 + 8x_1^3 - 8x_1^2 + x_2 - 2 \leq 0$

$g_2(x) = -4x_1^4 + 32x_1^3 - 88x_1^2 + 96x_1 + x_2 - 36 \leq 0$

where $0 \leq x_1 \leq 3$ *and* $0 \leq x_2 \leq 4$

As previously discussed, to solve a problem using EAs, the search space considered is usually larger than the feasible space of the problem and is defined by the variable bounds, as shown in the mathematical model and Fig. 1. For different corner points of the search space, the constraints' slacks and violations are presented in Tab. 1. In this problem, out of four corner points ((0, 0), (3, 0), (0, 4) and (3, 4)), two of them ((0, 4) and (3, 4)) are infeasible, which means that they are outside the feasible space, which is acceptable for our definitions of feasible and search spaces. However, at (0, 4), the violation is 2 units for $g_1(x)$ and, at (3, 4), 4 units for $g_2(x)$. The other two points are feasible as they are inside the constraints' boundary. However, point (0, 0) has only 2 units slack for $g_1(x)$ and point (3, 0) is on the boundary of $g_2(x)$. The maximum slack is 20 for $g_1(x)$ at (3, 0) and 36 for $g_2(x)$ at (0, 0).

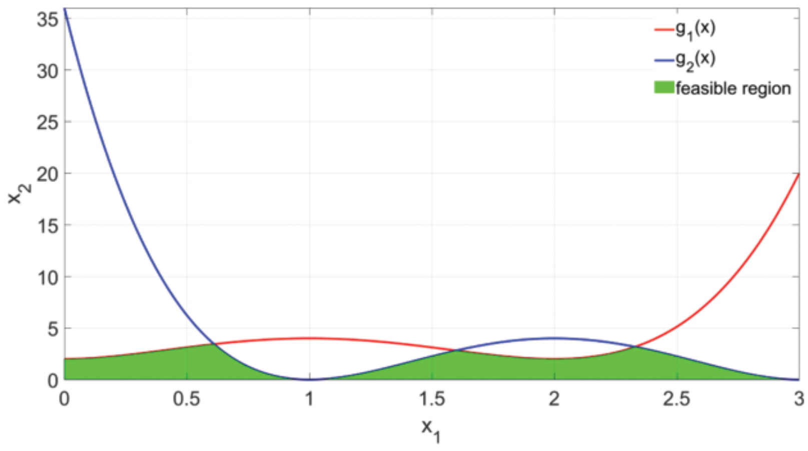

Figure 1: An analytical example of the G24 function that shows the slacks and violations of the corner points of the search space determined by the two inequality constraints $g_1(x)$ and $g_2(x)$

Table 1: Results obtained from numerical example of analytical method

x_1	x_2	$g_1(x)$	$g_2(x)$
0	0	−2	−36
3	0	−20	0
0	4	2	−32
3	4	−16	4

We assume that the search space is fixed but the feasible one may change due to a change in either the constraint function or right-hand side of the constraints or both. For any such variation, the optimal solution obtained in the previous environment may move to a new location in the new one and in some cases, can even be infeasible. In some situations, there may be no feasible solution if there is a large change in one or more constraints. In COPs, it is well known that the optimal solution often lies on the boundary of the feasible search space [18]. As the infeasible solution is not useful, we suggest a feasible one with minimum possible changes in the constraints. An analysis of the corner points of the search space, as discussed above, can be helpful for understanding the feasibility gaps between two consecutive environments. Therefore, in the analytical method shown in Algorithm 4, all the decision variables of a certain constraint (g_k) are set to all possible combinations of its lower and upper bounds and then the corresponding constraint slacks or violations calculated. For less-than-equality constraints, the smallest value obtained is chosen to be the boundary limits of feasibility. This process is repeated until all the boundary limits are calculated. It is to be noted that the greater-than-equality and equality constraints can be converted to less-than-equality ones mathematically.

```
Algorithm 4: Steps of the analytical method
     $n$ : number of decision variables
     $L$ : lower bounds
     $H$ : upper bounds
     $C$ : possible combinations
1  $L \leftarrow \{l_1, ..., l_n\}$;
2  $H \leftarrow \{h_1, ..., h_n\}$;
3  $C \leftarrow zeros[1 \times 2^n]$;
4  $Limit \leftarrow zeros[1 \times g]$;
5  for $i \leftarrow 1 : g$ do
6      for $j \leftarrow 1 : 2^n$ do
7          $X_j = \{x_1, x_2, ..., x_n\}$;
8          where $x_1 \leftarrow [l_1, h_1]$, $x_2 \leftarrow [l_2, h_2]$,..., $x_n \leftarrow [l_n, h_n]$
9          $C(j) \leftarrow g_i(X_j)$;
10     end
11     $Limit(i) \leftarrow min(C)$;
12 end
13 Return $Limit$;
```

b) Genetic-based Method

In the genetic-based method, to determine the limits of feasibility for each g_k constraint, we use this constraint as an objective function that is evolved to determine its best slack value while satisfying all the remaining functional constraints and variable bounds. This value is then considered the boundary limits of feasibility for this g_k constraint. This method is capable of handling cases in which the optimal solution lies either on the boundary of the feasible region or inside it. The same procedure is carried out until all the constraints are optimized. The pseudo-code of this method is presented in Algorithm 5.

Algorithm 5: Steps of the genetic-based method

```
1 $FEs \leftarrow 0$;
2 $Limit \leftarrow zeros[1 \times g]$;
3 for $i \leftarrow 1 : g$ do
4     while $FEs \leq FEs_{max}$ do
5         $Limit(i) \leftarrow$ optimize $g(i)$ subject to the remaining $(g-1)$
            constrains
6         update $FEs$;
7     end
8 end
9 Return $Limit$;
```

4 Experimental Study

In this section, the performance of the proposed approach is analyzed by solving a set of benchmark functions introduced in the CEC2006 special competition sessions on COPs [33]. These test problems have different mathematical properties for the objective function and constraint(s). There are different types of objective functions, such as linear, nonlinear, polynomial, quadratic and cubic, while the constraints can be either equality or inequality ones. The proposed approach is tested on only the CEC2006 problems with inequality constraints. A dynamic version of these basic static functions is proposed by using time-dependent expressions for their parameters [13,34], including 13 benchmark problems with inequality constraints, namely, $f_1, f_2, f_4, f_{6-}, f_{10}, f_{12}, f_{16}, f_{18}, f_{19}$ and f_{24}.

For each benchmark problem, 25 independent runs were executed, each over 30 time periods with a change frequency of 5000 fitness evaluations (*FEs*). To find the limit of feasibility in the genetic approach, the optimizer is run 25 times for each constraint for up to $FEsmax = 200{,}000$. For MODE, the settings are the same as those reported in [28]. All the algorithms used in this manuscript were coded in MATLAB R2019b and experiments conducted on a PC with Intel Core i7 at 3.6 GHz and 16.0 GB of DDR3 main memory with a Windows 10 operating system.

4.1 Performance Measures

The average feasibility ratios and fitness values produced from all the experimental runs in each environment are used as performance measures. If feasible solutions were not obtained from the compared algorithms, the average sum of the constraint violations is considered the solutions' quality measure. A statistical comparison is conducted using a Wilcoxon signed-rank test [35] with a 0.05 significance level. If the p-value is less than or equal to 5%, the null hypothesis is rejected; otherwise, it is accepted. The null hypothesis is there are no significant difference between the two algorithms. Whereas the alternative hypothesis is that there is a significant difference in the two algorithms' values. The three signs $+$, - and $\approx$ designate that the first algorithm is either significantly better, worse than or not different from the second, respectively.

4.2 Experimental Results

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

4.2.1 Finding Limits of Feasibility

In this section, the results obtained from both the analytical and genetic methods are presented and discussed. Tab. 2 show the results obtained from both the analytical and genetic methods. It can be seen that the limits of feasibility are equal in both methods for some constraints, especially G01 and G24, as all the corresponding constraints have the same limit.

To evaluate the performances of the two different methods, these feasibility limits are applied on the dynamic variant of MODE [13] as a change in the right-hand side of each constraint. Therefore, this test environment contains the second type of combination of the objective function and constraints, with the former static and latter dynamic, and is formulated as

$$min f(x) \tag{3}$$

subject to $g_k(x, t) \leq \; boundary_k$

where the change in constraints' boundaries from 0 to $boundary_k$ determines how the dynamic constraints change over time. Therefore, there are two time periods with their change frequencies set to 2.00E + 05 FEs.

The experimental results obtained for each test problem from the analytical and genetic methods are illustrated in Tab. 3. Based on these results, it is obvious that the genetic and analytical approaches can find feasible solutions for 98 and 41 instances, respectively. Although many feasible solutions are obtained from the analytical method for several test problems, such as G09 and G19, their limits of feasibility may not be accurate, as those obtained from the genetic method are both smaller and feasible.

4.2.2 Benefits of Proposed Algorithm

In this subsection, the merits of the proposed framework are discussed based on comparisons of four different variants of the MODE algorithm. These include:

- the proposed approach;
- dynamic MODE using a sensitive constraint-detection mechanism [13];
- default MODE initializing the population in each new environment with the population before a change occurs [28]; and
- re-initialization MODE whereby the entire population is randomly re-initialized after every change [28].

Detailed results are reported in the supplementary material in Appendices A and B, with summaries of the comparisons presented in Tab. 4. It can be observed that the performance of the proposed algorithm is superior to those of the other three variants as it obtains higher-quality solutions after the majority of environmental changes. Of the 1170 cases, it is superior, inferior and similar to the others for 929, 134 and 107, respectively. Therefore, it can be deduced that it performs better than the others for 79.40% of all the environments and is only outperformed for 11.45%.

Table 2: Feasibility limits obtained from both the analytical and genetic methods

Con.	*Met.*	*G01*	*G02*	*G04*	*G06*	*G07*	*G08*	*G09*	*G10*	*G12*	*G16*	*G18*	*G19*	*G24*
g_1	A.	−1.00E+01	−1.00E+20	−2.96E+00	−1.80E+04	−3.15E+02	−9.00E+00	3.04E+04	−9.50E−01	2.94E+00	−6.56E+02	1.99E+02	−1.80E+03	−2.00E+01
	G.	−1.00E+01	−3.17E+17	−2.96E+00	−1.94E+00	−2.15E+02	−2.12E+00	−1.82E+02	−9.50E−01	−6.25E−02	−1.21E+02	−1.00E+00	−2.32E+03	−2.00E+01
g_2	A.	−1.00E+01	−1.50E+02	−9.70E+01	−8.81E+00	−3.70E+02	7.00E+00	5.98E+02	−3.45E+00		−4.33E+02	−1.00E+00	−2.76E+03	−3.60E+01
	G.	−1.00E+01	−1.30E+02	−9.20E+01	−1.81E+00	−2.25E+02	−7.52E−01	−3.39E+02	−1.33E+00		−4.86E+01	−1.00E+00	−3.50E+03	−3.60E+01
g_3	A.	−1.00E+01		−1.38E+01		−1.82E+02		1.94E+02	−1.09E+01		−2.33E+00	1.99E+02	−2.56E+03	
	G.	−1.00E+01		−1.28E+01		−3.20E+01		−4.11E+02	−7.18E−01		−2.33E+00	−1.00E+00	−3.18E+03	
g_4	A.	−8.00E+01		−2.31E+01		3.98E+02		2.40E+02	−9.08E+06		−5.37E+02	1.99E+02	−2.06E+03	
	G.	−8.00E+01		−1.88E+01		−1.88E+02		−1.44E+02	−2.98E+06		−1.31E+02	−1.00E+00	−2.81E+03	
g_5	A.	−8.00E+01		−8.24E+00		3.76E+02			−9.90E+06		−2.52E+02	−1.00E+00	−1.01E+03	
	G.	−8.00E+01		−5.00E+00		−4.63E+01			−3.23E+06		−4.24E+01	−1.00E+00	−1.57E+03	
g_6	A.	−8.00E+01		−8.45E+00		−1.72E+02			−8.68E+06		−2.95E+02	−1.00E+00		
	G.	−8.00E+01		−5.00E+00		−1.07E+02			−7.18E+05		−1.92E+02	−1.00E+00		
g_7	A.	−3.00E+01				3.34E+02					−1.04E+03	−1.00E+00		
	G.	−3.00E+01				−2.45E+01					−2.17E+00	−1.00E+00		
g_8	A.	−3.00E+01				−1.12E+02					−1.04E+03	−1.00E+00		
	G.	−3.00E+01				−7.93E+01					−1.04E+03	−1.00E+00		
g_9	A.	−3.00E+01									−2.38E+01	1.99E+02		
	G.	−3.00E+01									−2.60E+00	−1.00E+00		
g_{10}	A.										−2.38E+01	−2.00E+02		
	G.										−2.37E+01	−1.00E+00		
g_{11}	A.										−4.51E+02	−2.00E+02		
	G.										−4.94E+01	−1.00E+00		
g_{12}	A.										−4.51E+02	−2.00E+02		
	G.										−4.50E+02	−1.00E+00		
g_{13}	A.										−5.85E+02	−2.00E+02		
	G.										−4.60E+02	−1.00E+00		
G16	Met.	g_{14}	g_{15}	g_{16}	g_{17}	g_{18}	g_{19}	g_{20}	g_{21}	g_{22}	g_{23}	g_{24}	g_{25}	g_{26}
	A.	−5.77E+02	−2.65E+02	−2.65E+02	−5.43E+00	−5.63E+00	−7.76E−02	−7.52E−02	−2.15E+02	−2.16E+02	−3.63E+02	−3.63E+02	−5.17E+02	−5.17E+02
	G.	−2.40E+02	−2.65E+02	−1.34E+02	−1.74E+00	−4.44E+00	−4.51E−02	−7.51E−02	−4.68E+01	−1.53E+02	−3.63E+02	−3.17E+02	−5.17E+02	−3.30E+02
	Met.	g_{27}	g_{28}	g_{29}	g_{30}	g_{31}	g_{32}	g_{33}	g_{34}	g_{35}	g_{36}	g_{37}	g_{38}	
	A.	−5.19E+02	−5.18E+02	−2.17E+03	−2.17E+03	−2.17E+04	−1.79E+04	−2.02E−01	−3.33E−01	−1.37E+05	−1.13E+04	−9.33E+06	−1.01E+07	
	G.	−5.53E+00	−5.18E+02	−2.17E+03	−3.16E+02	−8.57E+03	−1.20E+04	−2.27E−01	−1.40E−01	−6.89E+04	−1.57E+04	−9.28E+06	−6.57E+05	

Table 3: Results obtained from dynamic MODE using analytical and genetic-based method

Con.	*Met.*	*G01*	*G02*	*G04*	*G06*	*G07*	*G08*	*G09*	*G10*	*G12*	*G16*	*G18*	*G19*	*G24*
g_{1}	A.	**0.00E+00**	5.00E+01	**0.00E+00**	1.78E+04	9.98E+01	6.50E+00	**0.00E+00**	**0.00E+00**	**0.00E+00**	3.38E+02	**0.00E+00**	**0.00E+00**	**0.00E+00**
	G.	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
g_{2}	A.	**0.00E+00**	7.50E−01	5.04E+00	7.00E+00	1.41E+02	**0.00E+00**	**0.00E+00**	2.01E+00		3.08E+02	**0.00E+00**	**0.00E+00**	**0.00E+00**
	G.	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**		**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
g_{3}	A.	**0.00E+00**		9.96E−01		1.50E+02		**0.00E+00**	1.72E+00		**0.00E+00**	**0.00E+00**	**0.00E+00**	
	G.	**0.00E+00**		**0.00E+00**		**0.00E+00**		**0.00E+00**	**0.00E+00**		**0.00E+00**	**0.00E+00**	**0.00E+00**	
g_{4}	A.	**0.00E+00**		3.70E+00		**0.00E+00**		**0.00E+00**	1.52E+00		2.22E+02	**0.00E+00**	**0.00E+00**	
	G.	**0.00E+00**		**0.00E+00**		**0.00E+00**		**0.00E+00**	**0.00E+00**		**0.00E+00**	**0.00E+00**	**0.00E+00**	
g_{5}	A.	**0.00E+00**		3.24E+00		**0.00E+00**			1.73E+00		2.08E+02	**0.00E+00**	**0.00E+00**	
	G.	**0.00E+00**		**0.00E+00**		**0.00E+00**			**0.00E+00**		**0.00E+00**	**0.00E+00**	**0.00E+00**	
g_{6}	A.	**0.00E+00**		3.45E+00		6.51E+00			8.90E+00		1.03E+02	**0.00E+00**		
	G.	**0.00E+00**		**0.00E+00**		**0.00E+00**			**0.00E+00**		**0.00E+00**	**0.00E+00**		
g_{7}	A.	**0.00E+00**				**0.00E+00**					1.03E+03	**0.00E+00**		
	G.	**0.00E+00**				**0.00E+00**					**0.00E+00**	**0.00E+00**		
g_{8}	A.	**0.00E+00**				3.27E+1					3.91E−02	**0.00E+00**		
	G.	**0.00E+00**				**0.00E+00**					**0.00E+00**	**0.00E+00**		
g_{9}	A.	**0.00E+00**									2.12E+01	**0.00E+00**		
	G.	**0.00E+00**									**0.00E+00**	**0.00E+00**		
g_{10}	A.										8.39E−02	1.99E+02		
	G.										**0.00E+00**	**0.00E+00**		
g_{11}	A.										2.67E+02	1.99E+02		
	G.										**0.00E+00**	**2.22E−16**		
g_{12}	A.										1.59E+00	1.99E+02		
	G.										**0.00E+00**	**2.22E−16**		
g_{13}	A.										1.08E+02	1.99E+02		
	G.										**0.00E+00**	**0.00E+00**		
	Met.	g_{14}	g_{15}	g_{16}	g_{17}	g_{18}	g_{19}	g_{20}	g_{21}	g_{22}	g_{23}	g_{24}	g_{25}	g_{26}
G16	A.	**2.53E+02**	**0.00E+00**	**6.99E+01**	**3.70E+00**	**1.20E+00**	**3.25E−02**	**1.13E−04**	**1.64E+02**	**6.19E+01**	**3.78E−05**	**2.67E+01**	**0.00E+00**	**1.70E+02**
	G.	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Met.	g_{27}	g_{28}	g_{29}	g_{30}	g_{31}	g_{32}	g_{33}	g_{34}	g_{35}	g_{36}	g_{37}	g_{38}	
	A.	**5.13E+02**	**1.19E−01**	**2.41E−02**	**1.29E+03**	**9.18E+03**	**1.39E+02**	**0.00E+00**	**1.92E−01**	**6.78E+04**	**0.00E+00**	**3.96E+04**	**7.76E+05**	
	G.	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	

Table 4: Summary of results obtained by the proposed algorithm, default, dynamic and re-initialization for CEC'2006 inequality problems

Algorithms	Better	Equal	Worse	p-value	Decision
Proposed *vs.* default	287	42	61	0	+
Proposed *vs.* dynamic	287	44	59	0	+
Proposed *vs.* re-initialization	355	21	14	0	+

Furthermore, Tab. 5 shows the results obtained from the Friedman ranking test, that ranks all the variants according to the obtained mean results. From this table, it is demonstrated that the proposed algorithm ranks first (has the smallest value), as shown in bold, followed by MODE (dynamic), MODE (default) and MODE (re-initialization), respectively.

Table 5: Results obtained from Friedman rank test

Algorithm	Mean rank
Proposed	**1.48**
Default	2.66
Dynamic	2.56
Re-initialization	3.30

The proposed algorithm has superior performance because it tracks the useful location information of previous environments that helps to handle possible environmental changes. Also, it can deal with large changes effectively, which accelerates convergence towards feasibility and provides better solutions. However, the high violations of the constraints found in other variants lead to no feasible solutions. The experimental study shows the efficiency of the proposed method over the others, based on the results for most of the tested environments which, in turn, facilitates the movement of current solutions towards the feasible region.

Also, in order to better evaluate the proposed algorithm, the results obtained using the limits of feasibility for the other compared algorithms are shown in Tab. 6, with detailed results presented in supplementary documents in Appendices C and D. The results show the superiority of the proposed algorithm over MODE (default), MODE (dynamic) and MODE (re-initialization) in 259, 263 and 351 of 390 environments, respectively. This reveals the effectiveness of the proposed algorithm, even without using the limits of feasibility.

Table 6: Summary of results obtained using the limits of feasibility by the proposed algorithm, default, dynamic and re-initialization for CEC'2006 inequality problems

Algorithms	Better	Equal	Worse	p-value	Decision
Proposed *vs.* default	259	48	83	0	+
Proposed *vs.* dynamic	263	49	78	0	+

(Continued)

Table 6: Continued

Algorithms	Better	Equal	Worse	p-value	Decision
Proposed *vs.* re-initialization	351	21	18	0	+

5 Conclusions

In this paper, a method for handling DCOPs under resource disruptions is introduced. In it, a mechanism that helps to locate useful information from previous environments when addressing DCOPs is developed. The solutions in previous environments are exploited in a new one based on their similarities. Also, when the constraints change so much that a solution lies in the infeasible area, the right-hand side of the original problem is modified to a limit that provides a feasible solution. Therefore, analytical and genetic methods for specifying the limits of feasibility, that can help address DCOPs under resource disruptions, are proposed.

The proposed algorithm is combined with an optimization one and used to test 13 DCOPs in each of which 30 dynamic changes occur with a frequency of 5000 FEs. The experimental results show that this proposed method outperforms other approaches and can handle big changes effectively, which accelerates convergence and achieves better solutions. The results also demonstrate that the genetic-based approach outperforms the analytical one for finding the limit of feasibility, as it is capable of providing more accurate results for such problems.

For possible future research, the proposed method could include a constraint-handling approach for bringing solutions back to the feasible region quickly when a disruption occurs. Also, as each mechanism has its advantages, a possible avenue of investigation is to choose the most suitable (i.e., either analytical or genetic) based on a problem's characteristics. In addition, this implementation could be extended to consider dynamism in the objective function, variables' boundaries and/or active variables.

Funding Statement: This research is supported by the Australian Research Council Discovery Project (Grant Nos. DP210102939).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] Z. Zhang, S. Yue, M. Liao and F. Long, "Danger theory based artificial immune system solving dynamic constrained single-objective optimization," *Soft Computing*, vol. 18, no. 1, pp. 185–206, 2014.
[2] T. Zhu, W. Luo, C. Bu and L. Yue, "Accelerate population-based stochastic search algorithms with memory for optima tracking on dynamic power systems," *IEEE Transactions on Power Systems*, vol. 31, no. 1, pp. 268–277, 2015.
[3] T. T. Nguyen and X. Yao, "Continuous dynamic constrained optimization—The challenges," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 6, pp. 769–786, 2012.
[4] K. Deb, U. B. Rao N and S. Karthik, "Dynamic multi-objective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling," in *Proc. 4th Int. Conf. on Evolutionary Multi-Criterion Optimization (EMO)*, Matsushima, Japan, pp. 803–817, 2007.

[5] K. Mertens, T. Holvoet and Y. Berbers, "The DynCOAA algorithm for dynamic constraint optimization problems," in *Proc. 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, Hakodate, Japan, pp. 1421–1423, 2006.

[6] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 440–458, 2006.

[7] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 459–472, 2006.

[8] S. Das, A. Mandal and R. Mukherjee, "An adaptive differential evolution algorithm for global optimization in dynamic environments," *IEEE Transactions on Cybernetics*, vol. 44, no. 6, pp. 966–978, 2013.

[9] W. Luo, J. Sun, C. Bu and H. Liang, "Species-based particle swarm optimizer enhanced by memory for dynamic optimization," *Applied Soft Computing*, vol. 47, pp. 130–140, 2016.

[10] A. Ahrari, S. Elsayed, R. Sarker, D. Essam and C. A. C. Coello, "Adaptive multilevel prediction method for dynamic multimodal optimization," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 463–477, 2021.

[11] M. Jiang, Z. Huang, L. Qiu, W. Huang and G. G. Yen, "Transfer learning-based dynamic multiobjective optimization algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 4, pp. 501–514, 2017.

[12] C. Bu, W. Luo and L. Yue, "Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 1, pp. 14–33, 2017.

[13] N. Hamza, R. Sarker and D. Essam, "Sensitivity-based change detection for dynamic constrained optimization," *IEEE Access*, vol. 8, pp. 103900–103912, 2020.

[14] 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. of Genetic and Evolutionary Computation Conf. (GECCO)*, New York, NY, USA, pp. 1169–1172, 2015.

[15] 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. of Genetic and Evolutionary Computation Conf. (GECCO)*, New York, NY, USA, pp. 241–248, 2015.

[16] H. Richter, "Memory design for constrained dynamic optimization problems," in *European Conf. on the Applications of Evolutionary Computation*, Istanbul, Turkey, pp. 552–561, 2010.

[17] F. Zaman, S. Elsayed, R. Sarker, D. Essam and C. A. C. Coello, "Pro-reactive approach for project scheduling under unpredictable disruptions," *IEEE Transactions on Cybernetics*, pp. 1–14, 2021. https://doi.org.10.1109/TCYB.2021.3097312.

[18] M. R. Bonyadi and Z. Michalewicz, "On the edge of feasibility: A case study of the particle swarm optimizer," in *Proc. IEEE Congress on Evolutionary Computation (CEC)*, Beijing, China, pp. 3059–3066, 2014.

[19] J. Branke, "Evolutionary optimization in dynamic environments," in *Genetic Algorithms and Evolutionary Computation*, vol. 3. New York, NY, USA: Springer, 2012.

[20] D. Yazdani, M. N. Omidvar, J. Branke, T. T. Nguyen and X. Yao, "Scaling up dynamic optimization problems: A divide-and-conquer approach," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 1–15, 2019.

[21] 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 Congress on Evolutionary Computation (CEC)*, Trondheim, Norway, pp. 3127–3134, 2009.

[22] H. Richter and S. Yang, "Memory based on abstraction for dynamic fitness functions," in *Workshops on Applications of Evolutionary Computation*, Naples, Italy, pp. 596–605, 2008.

[23] H. Richter and F. Dietel, "Solving dynamic constrained optimization problems with asynchronous change pattern," in *Proc. European Conf. on the Applications of Evolutionary Computation*, Torino, Italy, pp. 334–343, 2011.

[24] M. -Y. Ameca-Alducin, E. Mezura-Montes and N. Cruz-Ramirez, “Differential evolution with combined variants for dynamic constrained optimization,” in *Proc. IEEE Congress on Evolutionary Computation (CEC)*, Beijing, China, pp. 975–982, 2014.

[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 Congress on Evolutionary Computation (CEC)*, Cancun, Mexico, pp. 2414–2421, 2013.

[26] E. Rashedi, H. Nezamabadi-Pour and S. Saryazdi, “GSA: A gravitational search algorithm,” *Information Sciences*, vol. 179, no. 13, pp. 2232–2248, 2009.

[27] T. T. Nguyen, “Continuous dynamic optimisation using evolutionary algorithms,” Ph.D. dissertation, University of Birmingham, England, 2011.

[28] S. Elsayed, R. Sarker and C. C. Coello, “Enhanced multi-operator differential evolution for constrained optimization,” in *Proc. IEEE Congress on Evolutionary Computation (CEC)*, Vancouver, BC, Canada, pp. 4191–4198, 2016.

[29] L. Araujo and J. J. Merelo, “A genetic algorithm for dynamic modelling and prediction of activity in document streams,” in *Proc. of Genetic and Evolutionary Computation Conf. (GECCO)*, London, United Kingdom, pp. 1896–1903, 2007.

[30] Y. Wang and M. Wineberg, “Estimation of evolvability genetic algorithm and dynamic environments,” *Genetic Programming and Evolvable Machines*, vol. 7, no. 4, pp. 355–382, 2006.

[31] S. Elsayed, R. Sarker, N. Hamza, C. A. C. Coello and E. Mezura-Montes, “Enhancing evolutionary algorithms by efficient population initialization for constrained problems,” in *Proc. IEEE Congress on Evolutionary Computation (CEC)*, Glasgow, United Kingdom, pp. 1–8, 2020.

[32] D. Yazdani, R. Cheng, D. Yazdani, J. Branke, Y. Jin *et al.,* “A survey of evolutionary continuous dynamic optimization over two decades–part b,” *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 609–629, 2021.

[33] J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan *et al.,* “Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization,” *Journal of Applied Mechanics*, vol. 41, no. 8, pp. 8–31, 2006.

[34] T. T. Nguyen, “A proposed real-valued dynamic constrained benchmark set,” *School of Computer Science, Univesity of Birmingham, Tech. Rep.*, pp. 1–54, 2008.

[35] G. Corder and D. Foreman, “Nonparametric statistics: An introduction,” in *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, Hoboken, NJ, USA: Wiley, pp. 101–111, 2009.