AbCD: A Component-wise Adjustable Framework for Dynamic Optimization Problems

Alexandre Mascarenhas
University of Tsukuba

Claus Aranha
University of Tsukuba

Yuri Lavinas
University of Tsukuba

ABSTRACT

Dynamic Optimization Problems (DOPs) are characterized by changes in the fitness landscape that can occur at any time and are common in real world applications. The main issues to be considered include detecting the change in the fitness landscape and reacting in accord. Over the years, several evolutionary algorithms have been proposed to take into account this characteristic during the optimization process. However, the number of available tools or open source codebases for these approaches is limited, making reproducibility and extensive experimentation difficult. To solve this, we developed a component-oriented framework for DOPs called Adjustable Components for Dynamic Problems (AbCD), inspired by similar works in the Multiobjective static domain. Using this framework, we investigate components that were proposed in several popular DOP algorithms. Our experiments show that the performance of these components depends on the problem and the selected components used in a configuration, which differs from the results reported in the literature. Using irace, we demonstrate how this framework can automatically generate DOP algorithm configurations that take into account the characteristics of the problem to be solved. Our results highlight existing problems in the DOP field that need to be addressed in the future development of algorithms and components.

CCS CONCEPTS

• **Computing methodologies → Bio-inspired approaches**.

KEYWORDS

Evolutionary computation, Evolutionary Dynamic Optimization, Dynamic Optimization Problems, Component Design

ACM Reference Format:
Alexandre Mascarenhas, Claus Aranha, and Yuri Lavinas. 2023. AbCD: A Component-wise Adjustable Framework for Dynamic Optimization Problems. In *Lisbon '23: GECCO 2023, July 15–19, 2023, Lisbon, PT*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Dynamic Optimization Problems (DOPs) are problems where the fitness landscape changes over time. Because of this, a solution can have multiple fitness values as the search progresses. DOPs include vehicle routing, which changes depending on traffic conditions, and team scheduling, which changes as task requests arrive. Many real-world problems have dynamic characteristics [21], making the study of DOPs crucial.

Different from static problems, evolutionary algorithms (EAs) applied to DOPs have to adapt to changes in the fitness functions in the middle of the optimization process. So the EA needs to detect the change to the fitness function and find the new optimum quickly.

Several EAs and variations have been proposed in the DOP literature in recent years [4, 28–30, 32]. However, there is a lack of analysis of individual algorithm components and their interactions. One of the main reasons for this is the lack of source codes [31]. This causes problems with reproducibility and diminishes our progress and ability to build upon previous knowledge.

Moreover, the algorithms proposed in these works have been presented and analyzed as a single entities. On the other hand, a component-wise perspective of this class of EAs, such as what has been done in the Multi-Objective EA domain recently [1, 9, 18], could lead us to a better understanding of the reasons behind variations in the performance of different algorithms.

In this work, we propose a component-wise framework for DOPs, which we call the *AbCD*: an Adjustable Component framework for Dynamic Problems. Our goals with this framework are three-fold: 1) to make available an open-source tool that facilitate the reproducible analysis of several DOP algorithms and experiments; 2) to examine individual components from the literature, with their individual impact and interaction with other component and problem classes; and 3) to automatically design new DOP algorithm configuration, both manually and by automatically searching from the available components and their parameters.

We validate the proposed framework by applying it on the Moving Peaks DOP Benchmark (MPB) [5]. We design two algorithm configurations, one manually by the analysis individual components, and another automatically by using Iterated Racing [20]. We compare both configurations, as well as some algorithms from the literature, using standard DOP metrics. The individual analysis of the components also shows interesting insights about existing DOP EAs, and point to promising directions of how to improve the state of the art. [1]

2 RELATED WORKS

Many algorithms have been proposed for DOPs, many of them based on PSO, such as FTmPSO [32], RPSO [14], CPSO [29], DSPSO [23], and mQSO [4]. Their approach is to add dynamic components to PSO that improve its performance in dynamic environments. In particular, mQSO was one of the first to use dynamic components to deal with fitness landscape changes over the optimization time,

Lisbon '23, July 15–19, 2023, Lisbon, PT
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

[1] For reproducibility purposes, all the code and experimental scripts are available in the supplementary materials.

while the other algorithms were strongly influenced by its ideas. However, in spite of the large number of proposed algorithms, there was little focus on creating and sharing tools to enable their use and study, a problem that was well stated in 2013 [25], and still placed as one of the main issues in DOPs by 2021 [31].

A very small number of works use a component-oriented approach to DOPs. For example, Xiong-Wen et al. show a framework using six different strategies to make PSO and DE perform better on DOPs [19]. Haluk et al. apply Hyper-heuristics to the Memory/Search algorithm to improve characteristics that are important for DOPs, such as population diversity [26]. Daniel et al. defines three strategies (Cooperative Evolutionary, Tracking multiple moving optima, and Resource allocation) that are applied to several optimizers such as CMA-ES, jDE, DynDE and PSO [33]. Even in these cases, the problems due to the absence of source code remain, such as reproducibility and the difficulty of analyzing components individually as well as their interaction effects, although the last work in the previous paragraph discusses in detail the differences between using or not certain strategies on the different algorithms.

3 PRELIMINARIES

3.1 Dynamic Optimization Terms

A *environment change* happens when the parameters of the objective function change in a dynamic problem. Let $NE \in \mathbb{N}$ be the total number of evaluations in one run of an algorithm, an environment change can occur at any evaluation $n \in [1, NE] \subset \mathbb{N}$. Let $CE \in \mathbb{N}$ be the total number of changes that can occur ($CE < NE$), each $e \in [1, CE] \subset \mathbb{N}$ represents an environment in the run. $N(e) \in \mathbb{N}$ is the total number of fitness evaluations on the e^{th} environment.

Subpopulations are often used in algorithms for dynamic optimization. Let $pop \in \mathbb{N}$ the total number of individuals in the population, and $N_{subpops} \in \mathbb{N}$ the number of subpopulations, and therefore, $N_{subpops} < pop$, and each $subpop \in [1, N_{subpops}] \subset \mathbb{N}$ represents a subpopulation.

So, the term $I_{i,subpop}(e, n)$ is the individual[2] i of the subpopulation $subpop$ at an environment e at the evaluation n; The term $I_{best_{i,subpop}}(e, n)$ is used to refer to the position with the highest fitness an individual $I_{i,subpop}$ has had at the environment e up to evaluation n; The term $S_{best_{subpop}}(e, n)$ is the best individual of the subpopulation $subpop$ at the environment e up to evaluation n; and the term $P_{best}(e, n)$ is used to refer to the best individual among the entire population pop at the environment e up to evaluation n.

3.2 Performance measures

The metrics implemented in the AbCD Framework are the most frequently used in the field of dynamic optimization: the Offline error (E_o) [7] and the best error before a change (E_b) [22, 27].

Offline error. This metric is calculated as the average of the best results found up to a given number of evaluations. It can be used both to generate an error curve along the evaluations and as a single value at the end of all evaluations and runs. This measure has been used since the early research in DOPs and is still one of the most commonly used measures in the literature.

$$E_o = \frac{1}{NE'} \sum_{n=1}^{NE'} |f(G_{Op}(n)) - f(P_{best}(n))| \tag{1}$$

where $f(G_{Op}(n))$ is the fitness value of the global optimum in the current environment at the nth fitness evaluation, $f(P_{best}(n))$ is the fitness value of the best individual at the nth fitness evaluation and NE' is the number of fitness evaluations so far. During the execution of the algorithm NE' is less than NE, and at the end of the execution, we have $NE' = NE$.

Best error before a change. Proposed in [27], but first used under this name in [22], this metric is calculated by the simple average of the best individuals found in each environment e.

$$E_b = \frac{1}{CE} \sum_{e=1}^{CE} |f(G_{Op}(e)) - f(P_{best}(e))| \tag{2}$$

where $f(G_{Op}(e))$ is the fitness value of the global optimum in the eth environment, $f(P_{best}(e))$ is the fitness value of the best individual just before the eth change happens and CE is the total number of changes of environment.

4 AbCD: ADJUSTABLE COMPONENTS FOR DYNAMIC PROBLEMS

In this work, we create the Adjustable Component framework for Dynamic Problems based on commonly used components of dynamic evolutionary algorithms. The general functioning of the framework can be seen in Algorithm 1. We designed AbCD following the component-wise framework, similar to the protocols used in the multiobjective domain [2, 9, 18].

Table 1: Dynamic components in the AbCD framework

Component Level	Component	Parameter(s)
Global component	Multipopulation	$N_{subpops}$
	Exclusion	r_{excl}
Local component	Reevaluation S_{best}	None
	Anti-convergence	r_{conv}
	Local Search	r_{ls}, $etry$
Optimizer	ES	r_{cloud}
	PSO	ϕ_1, ϕ_2
	Hybrid(PSO+ES)	$\phi_1, \phi_2, \%ES_{ind}, r_{cloud}$

The framework components are categorized according to how they act on individuals in the population. The categories are Local (L), when it involves only one subpopulation or Global (G) when it involves more than one subpopulation[3]. The list of all components in AbCD is shown in Table 1. Each component can be classified given their functions: optimizer; change detection; convergence detection; diversity control; and population division and management [30].

[2] To make the notation cleaner, sometimes the index $subpop$ of the subpopulation and the parameter e of the environment can be omitted. In these cases, it should be understood as the subpopulation to which the individual belongs and the current environment respectively.

[3] The letters (L) and (G) next to the component name will be used to identify the scope in which the component operates. With the exception of optimizers because it is not a dynamic component.

Algorithm 1 AbCD framework

```
1: Get a configuration
2: Initialize and evaluate the initial population $pop$
3: Initialize the control list $restart$ with $False$ and size equal to
   the number of subpopulations $N_{subpops}$.
4: Initialize the individual type list $ES_{ind}$ with percentage of $ES$
   individuals in $pop$
5: $P_{best} \leftarrow best$ of $pop$
6: update($pop$)
7: if Multipopulation is True then
8:     $pop$ = Multipopulation($pop$)                              ▷ (G)
9: end if
10: while Number of fitness evaluations is not reached do
11:     if Multipopulation is True then
12:         if Exclusion is True then
13:             $restart$ = exclusion($pop$)                       ▷ (G)
14:         end if
15:         if Anti-Convergence is True then
16:             $restart$ = antiConvergence($pop$)                 ▷ (L)
17:         end if
18:     end if
19:     if Local-Search is True then
20:         $P_{best}$ = localSearch($P_{best}$)                   ▷ (G)
21:     end if
22:     for each subpopulation $subpop$ in $pop$ do
23:         $restart$ = reevaluate($subpop$)          ▷ Change detection
24:         if $subpop$ in $restart$ then                ▷ (Lines 13, 16)
25:             randomize($subpop$)
26:         else                ▷ sub.pop not in $restart$ (Lines 3 or 23)
27:             for each individual $i$ in $subpop$ do
28:                 if $i$ in $ES_{ind}$ then
29:                     $i$ = ES($i$, $subpop$)
30:                 else
31:                     $i$ = PSO($i$, $subpop$)
32:                 end if
33:                 update($pop$)
34:             end for
35:         end if
36:     end for
37: end while
```

4.1 Components

Below we give a description of each component and parameter available in the framework.

4.1.1 Optimizers.

Particle swarm optimization (PSO). An established population-based optimizer, largely used in static optimization [17]. Given $\vec{x}_i = (x_i^1, x_i^2, ..., x_i^D)$ the position vector and $\vec{v}_i = (v_i^1, v_i^2, ..., v_i^D)$ the velocity vector of the individual I_i, in a $D \in \mathbb{N}$ dimensional space, the update equations that govern the movement of individuals are equations 3 and 4:

$$v_i(n+1) = \chi\{v_i(n) + \phi_1 R1_i[x_{S_{best}}(n) - x_i(n)] + \phi_2 R2_i[x_{i_{best}}(n) - x_i(n)]\} \quad (3)$$

$$\vec{x}_i(n+1) = \vec{x}_i(n) + \vec{v}_i(n+1) \quad (4)$$

where $v_i(n)$ is dth dimension of the ith individual velocity at the evaluation n, $x_{S_{best}}(n)$ is the dth dimension of the position of the best individual in its subpopulation at the evaluation n, $x_{i_{best}}$ is the dth dimension of the best position of the individual ith at the evaluation n, $R1_i$ and $R2_i$ are uniformly generated random numbers in the range of [0,1], χ is the constriction factor to works like friction for the individual's velocity and ϕ_1 and ϕ_2 are the social and individual acceleration coefficients respectively. For more information, see [15, 16].

Evolutionary strategy (ES). A powerful metaheuristic, concerning the fitness function [24]. Given $\vec{x}_i = (x_i^1, x_i^2, ..., x_i^D)$ the position vector of the individual I_i in a $D \in \mathbb{N}$ dimensional space, the update equation that controls the sampling of new individuals is 5:

$$x_i(n+1) = x_{S_{best}}(n) + R_i r_{cloud} \quad (5)$$

where $x_{S_{best}}(n)$ is the dth dimension of the position of the best individual of the subpopulation in which I_i belongs, R_i is a uniformly generated random number in the range of [0,1] and r_{cloud} is the maximum radius for sampling around $x_{S_{best}}(n)$.

4.1.2 Change detection.

Reevaluate S_{best} (L). Is the method used in the change detection component the reevaluation of the S_{best} [4]. The S_{best} of each subpopulation is reevaluated every generation, and if the new value is different from the previous one, we consider that there has been a change where this subpopulation is and all individuals in this subpopulation are re-evaluated. The pseudo-code of the Change detection component is shown on Algorithm 2.

Algorithm 2 Reevaluate S_{best}

```
Require: subpopulation $subpop$ and control list $restart$
1: $S_{best} \leftarrow$ best of $subpop$
2: sensor $\leftarrow$ evaluate($S_{best}$)
3: if $sensor$ != $f(S_{best})$ then    ▷ If different, a change occurred
4:     evaluate($subpop$)      ▷ Reevaluate the entire subpopulation
5:     $restart[subpop] \leftarrow False$
6: end if
7: Return $restart$
```

4.1.3 Diversity control.

Exclusion (G). This method is used to maintain diversity in the search space at a global level [4]. The method is applied to every pair of subpopulations, and the best individual of each subpopulation S_{best} is used to represent them. When the Euclidean distance of the representative individuals is smaller than a defined radius, r_{excl}, all individuals from the subpopulation with the individual with the worse fitness are restarted randomly. The pseudo-code of the Exclusion component is shown on Algorithm 3.

Algorithm 3 Exclusion

```
Require: population $pop$ and control list $restart$
1: for each pair of subpopulations $subpop_1$, $subpop_2$ in $pop$ do
2:     $distance \leftarrow$ euclideanDistance($S_{best_{subpop_1}}$, $S_{best_{subpop_2}}$)
3:     if $distance < r_{excl}$ then
4:         if f($S_{best_{subpop_1}}$) < f($S_{best_{subpop_2}}$) then
5:             $restart[subpop_1] \leftarrow$ True
6:         else
7:             $restart[subpop_2] \leftarrow$ True
8:         end if
9:     end if
10: end for
11: Return $restart$
```

Algorithm 4 Local Search

```
Require: individual $P_{best}$
1: for 1 to $etry$ do
2:     $P'_{best} \leftarrow P_{best}$
3:     for each dimension $d$ in $P'_{best}$ do
4:         $x'^{d}_{P_{best}}$ = Equation6($P'_{best}$)
5:     end for
6:     if $f(P_{best}) < f(P'_{best})$ then
7:         $f(P_{best}) \leftarrow f(P'_{best})$
8:     end if
9: end for
10: Return $P_{best}$
```

Local search (G). This component is used to intensify the search around P_{best} [32]. A number of $etry$ new individuals at an r_{ls} radius around the P_{best} are visited. The individual with the best fitness, including P_{best}, becomes the new P_{best}. Given $\vec{x}_{P_{best}} = (x^1_{P_{best}}, x^2_{P_{best}}, ..., x^D_{P_{best}})$ the position of the individual P_{best} in a $D \in \mathbb{N}$ dimensional space, the update equation that controls the sampling of new individuals is 6:

$$x_{P_{best}}(n+1) = x_{P_{best}}(n) + R_i r_{ls} \tag{6}$$

where $x_{P_{best}}(n)$ is the dth dimension of the position of the best individual of the entire population, R_i is a uniformly generated random number in the range of [0,1] and r_{ls} is the maximum radius at which the sample will be taken around the $x_{P_{best}}(n)$. The pseudo-code of the LS component is shown on Algorithm 4.

Anti-convergence (L). This method deals with the convergence of subpopulations [4]. If the distance between two individuals, for all dimensions, is less than r_{conv}, the subpopulation has converged. If all subpopulations have converged, the one where the S_{best} individual has the lowest fitness, is restarted. The pseudo-code of the Anti-Convergence component is shown on Algorithm 5.

4.1.4 Population division and management.

Multi-population (G). Inspired by multi-population approaches [3, 4, 6], we select this component to increase diversity in the search space by dividing the population pop into $N_{subpops}$ subpopulations.

Algorithm 5 Anti-convergence

```
Require: population $pop$ and control list $restart$
1: $converged \leftarrow True$
2: $S_{best_{worst.sub}} \leftarrow Inf$   ▷ Highest fitness value for an individual
3: for each subpopulation $subpop$ in $pop$ do
4:     $S_{best} \leftarrow$ best of $subpop$
5:     for each pair of individuals in $subpop$ do
6:         $distance$ = Euclidean distance per dimension
7:         $max.dist$ = Get the maximum value of $distance$
8:         if $max.dist > r_{conv}$ then
9:             $converged \leftarrow False$   ▷ The $subpop$ didn't converge
10:        end if
11:    end for
12:    if $f(S_{best_{worst.sub}}) < f(S_{best_{subpop}})$ then
13:        $S_{best_{worst.sub}} \leftarrow S_{best_{subpop}}$
14:    end if
15: end for
16: if $converged$ is $True$ then   ▷ All $subpops$ converged
17:     $restart[S_{best_{worst.sub}}] \leftarrow True$   ▷ Restart the worst
18: end if
19: Return $restart$
```

Individuals from each subpopulation do not interact with individuals from another subpopulation directly.

5 DESIGN OF DYNAMIC EVOLUTIONARY ALGORITHMS

Here we show how to use AbCD to create new configurations of evolutionary algorithms for DOPs. We follow two different approaches: a manual design, where we select the components one by one, and an automatic design, where we use irace to find a well-performing configuration of the components altogether.

5.1 Manual Design

Assuming independence among the components, we do the manual configuration to select a configuration of the components and their parameterization that performances well in the problems, for each optimizer. First, we search for the best parameter in a wide range of values for each component independently. We start with the multipopulation, since some of the other components require more than one sub-population. Then we select one component and we search for its best parameterization. We repeat this process for each component. Finally, we combine these best configuration of components to create the $AbcD_{man}$. To evaluate performance, we use the offline error (E_o).

The range of parameters was decided given the most used values used in the literature. For χ, ϕ_1 and ϕ_2, the parameters of PSO, as well for the exclusion, anti-convergence, and local search, we follow the suggestions from Blackwell and Branke [4, 32]. The range of the r_{cloud} parameter of ES was selected to be in the same interval as the radii of the exclusion and anti-convergence components.

The nomenclature of the configuration generated by the framework is $AbCD_{ES\%}$, where % indicates the percentage of individuals in the population that uses ES as the optimizer, and the rest of the individuals use PSO. Thus, the $AbCD_{ES100}$ configuration indicates that 100% of the individuals use the ES optimizer and the $AbCD_{ES0}$ configuration indicates that no individual use ES, that is, all are optimized by PSO. For all configuration explored, see Table 2.

Table 2: AbCD optimizers and the percentage of solutions in a population following PSO or ES.

Optimizers	PSO Individuals	ES Individuals
$PSO(AbCD_{ES0})$	100%	0%
$AbCD_{ES25}$	75%	25%
$AbCD_{ES50}$	50%	50%
$AbCD_{ES75}$	25%	75%
$ES(AbCD_{ES100})$	0%	100%

To evaluate the components as well as test the generated algorithms, the Moving Peaks Benchmark (MPB) was used, as it is available from the Python library DEAP [13]. This benchmark has several parameters configurations, for example, the number of peaks, frequency of change, and severity of the change, among others. Among the predefined scenarios,we set MPB to the scenario 2, as it is frequently used [4, 21, 29, 32, 34].

5.2 Automatic Design

Assuming dependence among the components, we use irace [20] to automatically design an $AbCD_{auto}$ configuration based on the components available in our framework. We run irace with its default settings, except for the number of elite configurations tested, which we increase from 1 to 7, following Campelo et al. work [9]. We run irace with a budget of 30000 runs. In a preliminary run of irace, we found that we were facing a heterogeneous scenario, and thus adjusted the configuration and increased the number of instances executed before doing a statistical test to 10, the number of instances evaluated between elimination tests to 5, and increase the number of new instances executed in each iteration to 5 as recommended in irace's user guide[4] and by Souza et al. [11]. Here, we use the offline error (E_o) as the training metric for evaluating the instances generated automatically by irace. We chose to fix the parameter $\chi = 1$ to reduce the number of parameters of PSO and the range of ϕ_1 and ϕ_2 was based on the work of Camacho-Villalón et al.[8]. Finally, each configuration runs for 500000 evaluations.

We use the same set of the MPB as in the manual design. Since we aim to find good configurations to perform well in many problems, we use more instances than those used in the manual design. These instances show vary not only the positions of the optimal points change but also the dimensions and the number of peaks.

6 EXPERIMENTS AND RESULTS

For all the experiments in this section, the metrics presented are calculated over the results of 50 runs, each one with 500000 fitness evaluation. Only the change detection component is always enabled during all experiments, as it is necessary in any DOP. The metrics used to evaluate the configurations are the offline error (E_o) and the best error before a change (E_b). We only use the E_b as a tiebreaker.

[4]https://mlopez-ibanez.github.io/irace/irace-package.pdf, accessed 17th January 2023

6.1 Baseline

As a baseline, we configure PSO and ES in the AbCD framework. The PSO parameters values used are $\chi = 0.729$, $\phi_1 = 2.05$, and $\phi_2 = 2.05$, these are commonly used in the literature, and more details about these values can be seen in [10, 12].

Now, for the ES, and hybrid variations, we set the parameter value r_{cloud} experimentally. We looked at the performance of $AbCD_{ES25}$, $AbCD_{ES50}$, $AbCD_{ES75}$ and ES in terms of variable values of r_{cloud} in the MPB set, as can be seen in the Figure 1. From this Figure, we can see that there is a drop in the offline error when r_{cloud} increases from zero to very small values, followed by an increase in the metric values from values above 0.1.

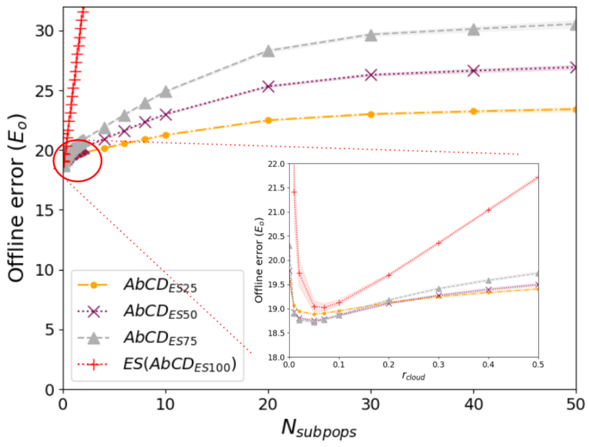


Figure 1: Offline error of the $AbCD_{ES25}$, $ES50$, $ES75$, ES **optimizers along the variation of the radius** r_{cloud}.

6.2 Manual Design

To design the Manual Designed AbCD configuration, $AbCD_{man}$, we use the settings of MPB shown in Table 3, and the range of values for each component are shown in Table 4. For the design of the $AbCD_{man}$, each component is evaluated individually, as the only addition to the main base algorithm, for each optimizers. The number of individuals in a population is fixed in $pop = 100$, a commonly used value from the literature. The value $etry = 20$ for the Local search component was chosen based on the experiments in [32].

Table 3: MPB settings for baseline and manual design

Parameter	Setting
Peak function	$cone()$[13]
Number of peaks	10
Number of dimensions	10
Peak heights	$\in [30, 70]$
Peak widths	$\in [1, 12]$
Change frequency	Every 5000 evaluations
Change severity s	1
Correlation coefficient λ	0

Table 4: Components search space for manual design

Components	Domain
Optimizer	$PSO, AbCD_{ES25}, _{ES50}, _{ES75}, ES$
Multipopulation	$N_{subpops} = 1, 5, 10, 25, 50, 100$
Exclusion	$r_{excl} \in [0, 50]$
Anti-convergence	$r_{conv} \in [0, 50]$
Local Search	$r_{ls} \in [0, 50]$ and $etry = 20$

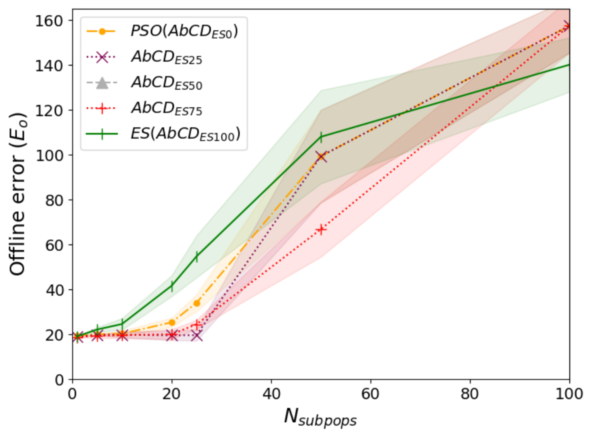


Figure 2: Offline error of the PSO, $_{ES25}$, $_{ES50}$, $_{ES75}$, ES optimizers given different numbers of subpopulation $N_{subpops}$.

We first analyze the multipopulation component. Figure 2 shows the results of different sub-polulations. Increasing the sub-population size deteriorates the performance of the AbCD variants. This result is confirmed in Table 5, and that for the pair instances and configurations tested, having $N_{subpops} = 1$, or just a single population, leads to the best results in terms of offline error.

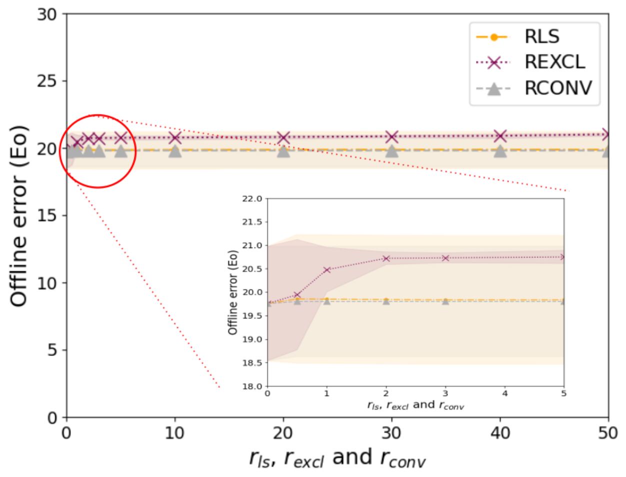
That said, most of the dynamic components (with the exception of Local search) depend on the configurations using multi-populations. Therefore, for the setting of the parameters of such components, we follow the suggested value of the number of sub-populations being equal to the number of peaks in the problem, $N_{subpops} = 10$, [4] .

Figure 3 show how the performance of different AbCD optimizers is affected by the parameters of the different dynamic components (we chose these examples because they represent well the behavior of the configurations we studied here). Interestingly, we can only observe small differences for low values of radii while higher values lead to higher error values. Given the limited positive impact of the dynamic components, we select the $AbCD_{man}$ as the $AbCD_{ES75}$ without dynamic components as can be seen in Table 8.

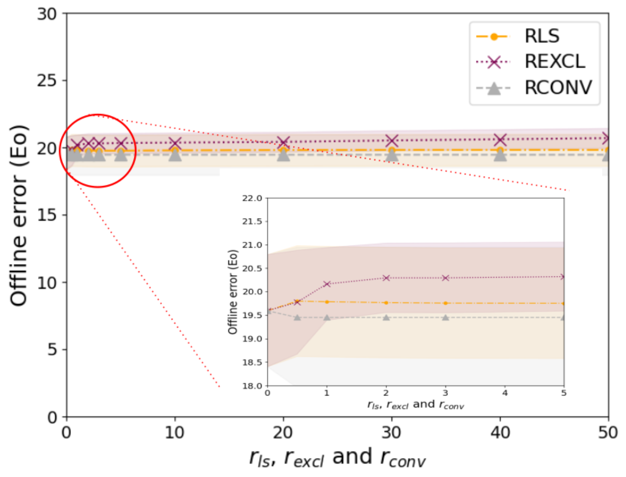
6.3 Automatic Design

To design the automatic AbCD configuration ($AbCD_{auto}$) we use different settings of MPB for training and testing, as shown in Table 6. The range of values for each component are shown in Table 7.

Figure 4 shows the frequency of the different choices of components and parameters after the tuning is performed. There is a consensus over the components and parameters in terms of elite



(a) $AbCD_{ES25}$



(b) $AbCD_{ES50}$

Figure 3: Offline error of different $AbCD$ optimizers with $N_{subpops=10}$ and $r_{cloud} = 0.05$ given variations of radii values of the components: exclusion r_{excl}, anti-convergence r_{conv} and local-Search r_{ls}.

configurations returned by irace. This suggests that these configurations have at least an adequate overall performance in the instances used during the configuration process.

Figure 5 shows the final elite configurations (last iteration), where each line in the plot represents one configuration. We can see from the distribution of the parameter values, that most of the final elite configurations are similar, and differ only on the number of sub-populations and the use of anti-convergence.

We were surprised by the $AbCD_{auto}$ configuration in terms of the number of sub-populations and the Local search values. Having a high sub-population size seems to benefit $AbCD_{auto}$ by reducing the interaction among the individuals in the population, probably allowing $AbCD_{auto}$ to explore more peaks. Also, this configuration

Table 5: Offline error (and standard deviation) of optimizers varying the number of subpopulations $N_{subpops}$, with $r_{cloud} = 0.05$

$N_{subpops}$	PSO	$AbCD_{ES25}$	$AbCD_{ES50}$	$AbCD_{ES75}$	ES
1	19.4187(0.0248)	18.8832(0.0186)	18.7498(0.0232)	18.7176(0.0320)	19.0403(0.1172)
5	20.0468(0.0287)	19.6459(0.0236)	19.4912(0.281)	19.2591(0.9935)	22.06689(1.0232)
10	20.1826(1.3809)	19.7556(1.2184)	19.5932(1.1950)	19.5737(1.3541)	24.5242(2.5880)
20	25.2869(2.0204)	19.5041(2.1427)	19.5563(2.1798)	19.9342(2.1658)	41.5702(4.6395)
25	33.9557(3.7422)	19.5660(2.4690)	19.8819(2.4329)	24.4162(1.9479)	54.6549(9.3672)

Table 6: MPB settings for automatic training and testing

Parameter	training	testing
Peak function	$cone()$	$cone()$
Peaks	8, 10	9, 11
Dimensions	8, 10	7, 9
Peak heights	$\in [30, 70]$	$\in [30, 70]$
Peak widths	$\in [1, 12]$	$\in [1, 12]$
Change frequency	Every 5000 evals	Every 5000 evals
Change severity s	1, 2	1.5, 2.5
Correlation λ	0	0

Table 7: Components search space for automatic design

Components	Domain
Population size	100, 150, 200, 300
Optimizer	$PSO \rightarrow \phi_1 \in [0, 2.50], \phi_2 \in [0, 2.50]$ $ES \rightarrow r_{cloud} \in [0, 5]$
	$AbCD_{ES\%} \rightarrow \begin{cases} \% = 25, 50, 75 \\ \phi_1 \in [0, 2.50], \phi_2 \in [0, 2.50] \\ r_{cloud} \in [0, 5] \end{cases}$
Multipopulation	True, $N_{subpops} = 10, 25, 50, 100$ False, not used
Exclusion	True, $r_{excl} \in [0, 80]$ False, not used
Anti-convergence	True, $r_{conv} \in [0, 80]$ False, not used
Local Search	True, $r_{ls} \in [0, 80], etry \in [1, 50]$ False, not used

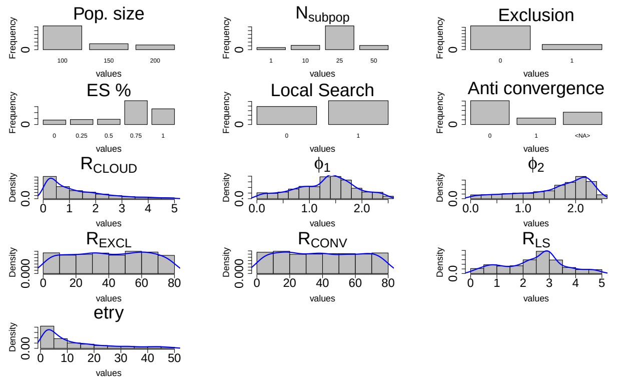


Figure 4: irace output with the frequency of the different choice of components and their parameters.

has Local search with a high r_{ls} value and a small number of iterations ($etry = 2$), which suggests that Local search is used to explore a wider area near the best individual.

When we compare the results on $AbCD_{auto}$ with $AbCD_{man}$ we see some consensus for most of the components. For example, both

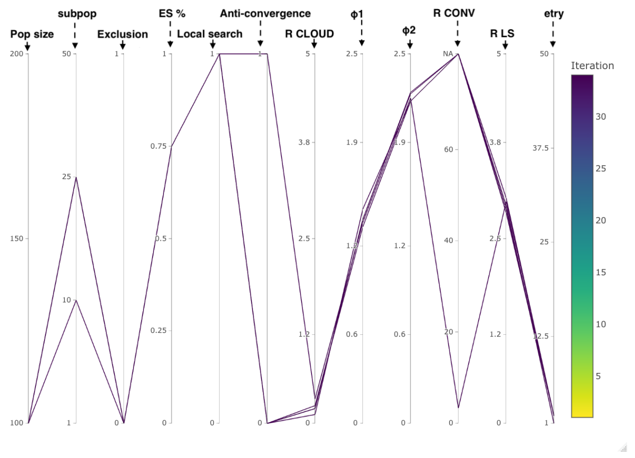


Figure 5: Elite configurations from the irace output. Each line in the plot represents one configuration.

configurations use 75% ES individuals (with 25% being PSO individuals). Also, they don't use the dynamic components exclusion and anti-convergence, as can be seen in Table 8. The PSO and ES parameters are slightly different. Moreover, the most divergent choices of components are the number of sub-populations, in different extremes, and the presence of local search in $AbCD_{auto}$. Although we can see a difference in the use of the local search component, irace selected a small number of tries, which suggests that the impact of this component is likely to be small. On the other hand, the reasons behind the contrary number of sub-populations needs more work.

6.4 Results

Here compare PSO, ES, mQSO, $AbCD_{man}$, and $AbCD_{auto}$using the offline error (E_o) and the error before a change (E_b). The configurations can be seen in Table 8. We evaluate them in five instances, generated under the MPB scenario 2 setting. We refer to them first by their dimension, then by the number of peaks, and finally by the severity. For example, the 5D-10P-1s instance with 5 dimensions, 10 peaks, and severity 1. We show the results in Table 9.

For lower dimensions, 5D-10P-1s instance, we can see that the performance of the configurations can be divided into two groups, those with a good performance value, with mQSO and $AbCD_{auto}$, and the second group with the other configurations. The performance of both mQSO and $AbCD_{auto}$ deteriorates as the number of dimensions increases, although $AbCD_{auto}$ keeps a good overall performance.

Table 8: Configurations generated with AbCD framework

Optimizer	Multipopulation	Exclusion	Local search	Anti-Convergence
PSO $\rightarrow \{\chi = 0.729, \phi_1 = 2.05, \phi_2 = 2.05$	Disabled	Disabled	Disabled	Disabled
ES $\rightarrow \{r_{cloud} = 0.05$	Disabled	Disabled	Disabled	Disabled
mQSO [4]$\rightarrow \begin{cases}\chi = 0.729, \phi_1 = 2.05, \phi_2 = 2.05 \\ r_{cloud} = 1\end{cases}$	$N_{subpop} = 10$	$r_{excl} = 22.9$	Disabled	$r_{conv} = 39.7$
$AbCD_{man} \rightarrow \begin{cases}\% = 75, r_{cloud} = 0.05 \\ \chi = 0.729, \phi_1 = 2.05, \phi_2 = 2.05\end{cases}$	Disabled	Disabled	Disabled	Disabled
$AbCD_{auto} \rightarrow \begin{cases}\% = 75, r_{cloud} = 0.2 \\ \chi = 1, \phi_1 = 1.38, \phi_2 = 2.24\end{cases}$	$N_{subpop} = 25$	Disabled	$\begin{cases} r_{ls} = 2.9 \\ etry = 2\end{cases}$	Disabled

Table 9: E_o and E_b results of the different configurations

Algorithm	5D-10P	8D-10P	10D-10P
PSO	$E_o = 15.12(1.62)$	$E_o = 13.01(2.43)$	$E_o = 19.42(0.03)$
ES	$E_o = 14.63(1.71)$	$E_o = 13.17(2.71)$	$E_o = 19.04(0.12)$
mQSO	$E_o = 7.97(1.21)$	$E_o = 13.64(0.66)$	$E_o = 23.36(0.89)$
$AbCD_{man}$	$E_o = 14.78(1.71)$	$E_o = 12.80(2.53)$	$E_o = 18.72(0.03)$
$AbCD_{auto}$	$E_o = 6.37(0.71)$	$E_o = 11.10(1.23)$	$E_o = 19.90(2.49)$

This good performance of $AbCD_{auto}$ comes without a surprise since irace searched for a configuration that would perform well in most of the instances. Interestingly, $AbCD_{auto}$ was designed for instances with high dimensions, but it could extrapolate its good performance to an easier problem. We ask ourselves if the same would be true for more challenging instances.

A more interesting result is that dimensionality has a big impact on the performance of the configurations. For example, mQSO is among the best in the problem with the lowest dimension and the worse as the number of dimensions increases. This goes in agreement with the results of the manual and automatic design, which found that using the main components of mQSO, exclusion, and anti-convergence, lead to a reduction in the performance of the configurations. We believe one of the reasons for this loss in performance in high dimensions could be given that these two components use the Euclidean distance and a more suited metric for distances in many dimensions should be considered in future works.

Another remarkable result is that a recombination of commonly used evolutionary algorithms in static problems can lead to good dynamic configurations, as shown by the increments in performance by the configurations $AbCD_{man}$ and $AbCD_{auto}$. This suggests that incorporating even more effective evolutionary algorithms into the AbCD framework would benefit the dynamic community.

7 CONCLUSION

The aim of this work was to introduce a new component-wise based on commonly used operators in DOPs, the Adjustable Component framework for Dynamic Problems. The main goals were to make it available as a framework for extensive experimentation of dynamic EAs and the analysis of new dynamic components and to facilitate the design of ones as well as a tool to simplify the manual and automatic design of algorithms. All the code and experimental scripts are available online at https://zenodo.com/ZZZ/ZZZ and the most current version of the framework can be found in GitHub https://github.com/ZZZ/ZZZ[5].

This analysis allowed us to verify that, contrary to our expectations, the dynamic components were less influential in leading to increments in the performance of the algorithm configurations studied here. It seems that the effect can only be observed in problems with low dimensionality. We understand that this suggests that more work should be done to improve their influence in more dimensions. We also found that the dimensionality of the problem seems to have a higher impact on the search ability of the EAs studied in this work.

We conducted an analysis on the choices of components in the $AbCD_{man}$ and the $AbCD_{auto}$, and we observed that there is an agreement in the choice of the percentage of the number of individuals that are optimized by ES or PSO and in the choice of the dynamic components, not selected in both configurations. However, it seems that there is some interaction among some components since the choice of Local search and the number of sub-population is clearly different between these two configurations. More focus should be directed towards clarifying these interactions.

We studied PSO, ES, mQSO, $AbCD_{man}$, and $AbCD_{auto}$. The first two algorithms are commonly used metaheuristics, mQSO is a representative of EAs for dynamic problems, and the Man-AbDC and $AbCD_{auto}$ are algorithm configurations generated using the components available in our framework. The results have shown that their relative efficiency depends on the problem's difficulty: mQSO and $AbCD_{auto}$ are the best in lower dimensional problems while PSO, ES, and $AbCD_{man}$ achieve a higher performance as the number of dimensions increase. These results show the power of the AbCD framework, since we can develop established and new EAs and study their performance in dynamic problems.

This study strengthens the view using a component-based perspective can lead us to a better understanding of the reasons behind variations in the performance of different algorithms. One limitation of our work is that the number of components and optimizers

[5]The source code for the framework, as well as experimental data, is available in the supplementary materials. It will also be on an online repository by the final submission

in the AbCD framework is limited and future works should focus on finding good components candidates, mainly from effective dynamic algorithms from the literature, as well as introducing new components with insights from our results.

REFERENCES

[1] Leonardo C. T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle. 2015. To DE or Not to DE? Multi-objective Differential Evolution Revisited from a Component-Wise Perspective. In *Evolutionary Multi-Criterion Optimization*, António Gaspar-Cunha, Carlos Henggeler Antunes, and Carlos Coello Coello (Eds.). Springer International Publishing, Cham, 48–63.

[2] Leonardo C. T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle. 2016. Automatic Component-Wise Design of Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation* 20, 3 (2016), 403–417. https://doi.org/10.1109/TEVC.2015.2474158

[3] Tim Blackwell and Jürgen Branke. 2004. Multi-swarm Optimization in Dynamic Environments. In *Applications of Evolutionary Computing*, Günther R. Raidl, Stefano Cagnoni, Jürgen Branke, David Wolfe Corne, Rolf Drechsler, Yaochu Jin, Colin G. Johnson, Penousal Machado, Elena Marchiori, Franz Rothlauf, George D. Smith, and Giovanni Squillero (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 489–500.

[4] T. Blackwell and J. Branke. 2006. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation* 10, 4 (2006), 459–472. https://doi.org/10.1109/TEVC.2005.857074

[5] J. Branke. 1999. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, Vol. 3. 1875–1882 Vol. 3. https://doi.org/10.1109/CEC.1999.785502

[6] Juergen Branke. 2000. *Evolutionary Optimization in Dynamic Environments*. Ph. D. Dissertation. https://doi.org/10.1007/978-1-4615-0911-0

[7] Jürgen Branke. 2002. *Empirical Evaluation*. Springer US, Boston, MA, 67–98. https://doi.org/10.1007/978-1-4615-0911-0_5

[8] Christian L. Camacho-Villalón, Marco Dorigo, and Thomas Stützle. 2022. PSO-X: A Component-Based Framework for the Automatic Design of Particle Swarm Optimization Algorithms. *IEEE Transactions on Evolutionary Computation* 26, 3 (2022), 402–416. https://doi.org/10.1109/TEVC.2021.3102863

[9] Felipe Campelo, Lucas Batista, and Claus Aranha. 2020. The MOEADr Package: A Component-Based Framework for Multiobjective Evolutionary Algorithms Based on Decomposition. *Journal of Statistical Software* (2020). In press. Available from: https://arxiv.org/abs/1807.06731.

[10] M. Clerc and J. Kennedy. 2002. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation* 6, 1 (2002), 58–73. https://doi.org/10.1109/4235.985692

[11] Marcelo de Souza, Marcus Ritt, Manuel López-Ibáñez, and Leslie Pérez Cáceres. 2021. ACVIZ: A tool for the visual analysis of the configuration of algorithms with irace. *Operations Research Perspectives* 8 (2021), 100186. https://doi.org/10.1016/j.orp.2021.100186

[12] R.C. Eberhart and Y. Shi. 2000. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, Vol. 1. 84–88 vol.1. https://doi.org/10.1109/CEC.2000.870279

[13] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.

[14] Xiaohui Hu and R.C. Eberhart. 2002. Adaptive particle swarm optimization: detection and response to dynamic systems. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 2. 1666–1670 vol.2. https://doi.org/10.1109/CEC.2002.1004492

[15] Xiaohui Hu and R.C. Eberhart. 2002. Adaptive particle swarm optimization: detection and response to dynamic systems. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 2. 1666–1670 vol.2. https://doi.org/10.1109/CEC.2002.1004492

[16] Keisuke KAMEYAMA. 2009. Particle Swarm Optimization–A Survey. (2009).

[17] J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, Vol. 4. 1942–1948 vol.4. https://doi.org/10.1109/ICNN.1995.488968

[18] Yuri Lavinas, Marcelo Ladeira, Gabriela Ochoa, and Claus Aranha. 2022. Component-Wise Analysis of Automatically Designed Multiobjective Algorithms on Constrained Problems. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 538–546. https://doi.org/10.1145/3512290.3528719

[19] Xiong-Wen Luo, Zi-Jia Wang, Ren-Chu Guan, Zhi-Hui Zhan, and Ying Gao. 2019. A Distributed Multiple Populations Framework for Evolutionary Algorithm in Solving Dynamic Optimization Problems. *IEEE Access* 7 (2019), 44372–44390. https://doi.org/10.1109/ACCESS.2019.2906121

[20] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43–58. https://doi.org/10.1016/j.orp.2016.09.002

[21] Trung Nguyen. 2011. *Continuous dynamic optimisation using evolutionary algorithms*. Ph. D. Dissertation.

[22] Trung Thanh Nguyen, Shengxiang Yang, and Juergen Branke. 2012. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation* 6 (2012), 1–24. https://doi.org/10.1016/j.swevo.2012.05.001

[23] D. Parrott and Xiaodong Li. 2006. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation* 10, 4 (2006), 440–458. https://doi.org/10.1109/TEVC.2005.859468

[24] Ingo Rechenberg. 1973. Evolutionsstrategie. *Optimierung technischer Systeme nach Prinzipien derbiologischen Evolution* (1973).

[25] Philipp Rohlfshagen and Xin Yao. 2013. Evolutionary Dynamic Optimization: Challenges and Perspectives. In *Evolutionary Computation for Dynamic Optimization Problems*, Shengxiang Yang and Xin Yao (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 65–84.

[26] Haluk Rahmi Topcuoglu, Abdulvahid Ucar, and Lokman Altin. 2014. A hyper-heuristic based framework for dynamic optimization problems. *Applied Soft Computing* 19 (2014), 236–251. https://doi.org/10.1016/j.asoc.2014.01.037

[27] K. Trojanowski and Z. Michalewicz. 1999. Searching for optima in non-stationary environments. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, Vol. 3. 1843–1850 Vol. 3. https://doi.org/10.1109/CEC.1999.785498

[28] Yonas G. Woldesenbet and Gary G. Yen. 2009. Dynamic Evolutionary Algorithm With Variable Relocation. *IEEE Transactions on Evolutionary Computation* 13, 3 (2009), 500–513. https://doi.org/10.1109/TEVC.2008.2009031

[29] Shengxiang Yang and Changhe Li. 2010. A Clustering Particle Swarm Optimizer for Locating and Tracking Multiple Optima in Dynamic Environments. *IEEE Transactions on Evolutionary Computation* 14, 6 (2010), 959–974. https://doi.org/10.1109/TEVC.2010.2046667

[30] Danial Yazdani, Ran Cheng, Donya Yazdani, Jürgen Branke, Yaochu Jin, and Xin Yao. 2021. A Survey of Evolutionary Continuous Dynamic Optimization Over Two Decades—Part A. *IEEE Transactions on Evolutionary Computation* 25, 4 (2021), 609–629. https://doi.org/10.1109/TEVC.2021.3060014

[31] Danial Yazdani, Ran Cheng, Donya Yazdani, Jürgen Branke, Yaochu Jin, and Xin Yao. 2021. A Survey of Evolutionary Continuous Dynamic Optimization Over Two Decades—Part B. *IEEE Transactions on Evolutionary Computation* 25, 4 (2021), 630–650. https://doi.org/10.1109/TEVC.2021.3060012

[32] Danial Yazdani, Babak Nasiri, Alireza Sepas-Moghaddam, and Mohammad Reza Meybodi. 2013. A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization. *Applied Soft Computing* 13, 4 (2013), 2144–2158. https://doi.org/10.1016/j.asoc.2012.12.020

[33] Danial Yazdani, Mohammmad Nabi Omidvar, Jurgen Branke, Trung Nguyen, and Xin Yao. 2019. Scaling Up Dynamic Optimization Problems: A Divide-and-Conquer Approach. *IEEE Transactions on Evolutionary Computation* PP (03 2019), 1–1. https://doi.org/10.1109/TEVC.2019.2902626

[34] Danial Yazdani, Mohammad Nabi Omidvar, Ran Cheng, Jürgen Branke, Trung Thanh Nguyen, and Xin Yao. 2022. Benchmarking Continuous Dynamic Optimization: Survey and Generalized Test Suite. *IEEE Transactions on Cybernetics* 52, 5 (2022), 3380–3393. https://doi.org/10.1109/TCYB.2020.3011828