Swarm and Evolutionary Computation xxx (2018) 1–18

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

A novel framework for improving multi-population algorithms for dynamic optimization problems: A scheduling approach

Javidan Kazemi Kordestani [a,*], Amir Ehsan Ranginkaman [b], Mohammad Reza Meybodi [c], Pavel Novoa-Hernández [d,e]

[a] Department of Electrical, Computer and IT Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
[b] Department of Electrical, Computer and IT Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran
[c] Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran
[d] Faculty of Engineering Sciences, Technical State University of Quevedo, Km 1/2 Via Sto Domingo, Quevedo, Ecuador
[e] State University of Milagro, Spain

ARTICLE INFO

Keywords:
Dynamic optimization problems
Differential evolution
Moving peaks benchmark
Evolutionary computation
Scheduling
Learning automata

ABSTRACT

This paper presents a novel framework for improving the performance of multi-population algorithms in solving dynamic optimization problems (DOPs). The fundamental idea of the proposed framework is to incorporate the concept of scheduling into multi-population methods with the aim to allocate more function evaluations to the best performing sub-populations. Two methods are developed based on the proposed framework, each of which uses a different approach for scheduling the sub-populations. The first method combines the quality of sub-populations and the degree of diversity among them into a single feedback parameter for detecting the best performing sub-population. The second method uses the learning automata as the central unit for performing the scheduling operation. In order to validate the applicability of the proposed methods, they are incorporated into three well-known algorithms for DOPs. The experimental results show the efficiency of the scheduling approach for improving the multi-population methods on the moving peaks benchmark (MPB) and generalized dynamic benchmark generator.

1. Introduction

Several real-world optimization problems are dynamic, meaning that the optimal solution(s) may change over time. An example of such problems includes the dynamic vehicle routing problem [1], where there is an online arrival of customers. Another well-known example is dynamic shortest path routing problem [2] where the goal of optimization is to find the shortest path from a specific source to a specific destination in a changing network environment while minimizing the total cost associated with the path. In general, several processes of science and engineering involve the optimization of a set of complex problems, in which the objectives of the optimization, some restrictions or other elements of the problems may vary over time. In these cases, the optimum solution(s) to the problem are subject to change as well. As can be understood from above examples, unlike static optimization problems, the goal of optimization in dynamic problems is no longer locating the optimal solution(s), but tracking their trajectories over time with a high level of accuracy.

Due to their adaptive nature, evolutionary algorithms (EAs) and swarm intelligence methods (SIMs) have proven to be good optimizers for dynamic optimization problems (DOPs) [3,4]. Over the past years, various attempts have been done by researchers to improve the efficiency of traditional EAs and SIMs on DOPs. According to [4] the existing proposals can be grouped into the following approaches:

i. Increasing the diversity after detecting a change in the environment [5–8],
ii. Maintaining diversity during the optimization process [9,10].
iii. employing memory schemes to retrieve information about previously found solutions [11,12],
iv. predicting the location of the next optimal solution(s) after a change is detected [13,14].

* Corresponding author.
E-mail addresses: javidan.kazemi@gmail.com, jk.kordestani@srbiau.ac.ir (J.K. Kordestani), ranginkaman@qiau.ac.ir (A.E. Ranginkaman), mmeybodi@aut.ac.ir (M.R. Meybodi), pnovoa@uteq.edu.ec (P. Novoa-Hernández).

https://doi.org/10.1016/j.swevo.2018.09.002
Received 22 July 2017; Received in revised form 19 July 2018; Accepted 4 September 2018
Available online xxxx

v. Making use of the self-adaptive mechanisms of EAs, SIMs and other meta-heuristics [15,16],
vi. Using multiple sub-populations to handle separate areas of the search space concurrently [17–27].

Among the above-mentioned approaches, multi-population methods are very effective, especially for multimodal DOPs. Since they provide mechanisms to simultaneously explore the search space, the algorithm is able to find high-quality solutions very quickly, that is, before a change occurs in the environment. Despite the progress achieved in the past, we believe that there is still room for improving this approach. We focus on solving one important challenge related to efficiency: how to suitably exploit the function evaluations assigned to each sub-population.

In that sense, a framework for improving the performance of multi-population methods is proposed that adopts the idea of scheduling. Specifically, we have developed two scheduling methods. The first one combines the quality of sub-populations and the degree of diversity among them into a single feedback parameter for detecting the best performing sub-population. The second method uses the learning automata as the central unit for performing the scheduling operation. In order to validate the applicability of the proposed methods, they were incorporated into three well-known algorithms for DOPs. From several computational experiments, we show the efficiency of our proposals over many challenging instances of artificial DOPs.

The rest of the paper is structured as follows: Section 2 reviews some of the existing multi-population methods developed for DOPs. In Section 3, we explain our proposed framework and sub-population scheduling methods in detail. In Section 4, we describe how to apply our proposal in the algorithm DynDE. An experimental study for analyzing the impact of the proposed approach is presented in Section 5. Moreover, the applicability of the proposed framework on jDE and mSQDE-i is also investigated in Section 5. Finally, in Section 6 the conclusion and future work are outlined.

2. Multi-population approaches for DOPs

As mentioned before, one of the most efficient approaches to deal with existing challenges in DOPs is to employ several sub-populations. The idea behind this is to divide the individuals (candidate solutions) of the main population into several sub-populations. Thus, the algorithm is able to efficiently handle several issues arising in DOPs: (1) exploration, (2) optimum tracking, (3) change detection, and (4) premature convergence. In what follows, we review some of the most relevant works on multi-population methods for DOPs.

2.1. Multi-population methods with a fix number of sub-populations

The main idea of these methods is to establish a mutual repulsion among several fixed-size populations. Thus, they are placed over different promising areas of the search space. The pioneer work in this context was done by Ref. [28]. They proposed two multi-swarm algorithms based on the particle swarm optimization (PSO), namely mCPSO and mQSO. In mCPSO, each swarm is composed of *neutral* and *charged* particles. Neutral particles update their velocity and position according to the principles of pure PSO. On the other hand, charged particles move in the same way as neutral particles, but they are also mutually repelled from other charged particles residing in their own swarm. Therefore, charged particles help to maintain the diversity inside the swarm. In mQSO, instead of having charged particles, each swarm contains *quantum* particles. Quantum particles change their positions around the center of the best particle of the swarm according to a random uniform distribution with radius r_{cloud}. Consequently, they never converge and provide a suitable level of diversity to swarm in order to follow the shifting optimum. The authors also introduced the *exclusion* operator, which prevents populations from settling on the same peak. In another work [17], the same authors proposed a second operator referred to as *anti-convergence*, which is triggered after all swarms converge and reinitializes the worst swarm in the search space.

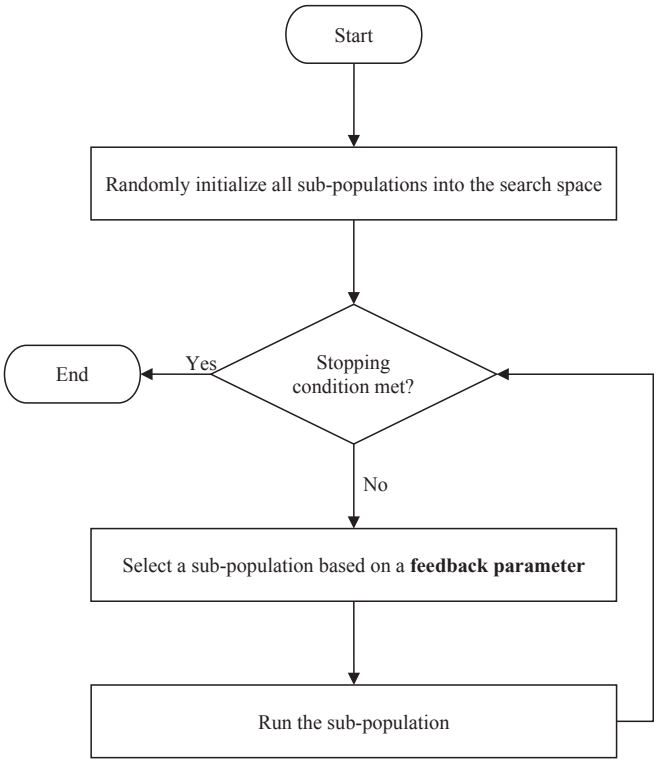

Fig. 1. Framework for multi-population methods with scheduling in dynamic environments.

Inspired by the concept of exclusion [29], introduced a multi-population differential evolution (DE) algorithm, called DynDE. In DynDE several populations are initialized in the search space and explore multiple peaks in the environment incorporating exclusion. DynDE also includes a method for *increasing diversity*, which enables the partially converged population to track the shifting optimum.

Some researchers combined desirable features of different optimization algorithms into a single method for DOPs. For example [30], introduced a hybrid collaborative method called collaborative evolutionary-swarm optimization (CESO). CESO has two equal-size populations: a main population to maintain a set of local and global optimum during the search process using crowding-based differential evolution, and a PSO population acting as a local search operator around solutions provided by the first population. During the search process, information is transmitted between both populations via collaboration mechanisms. In another work [31], a third population is incorporated to CESO which acts as a memory to recall some promising information from past generations of the algorithm. Inspired by CESO [32], proposed a bi-population hybrid algorithm. The first population, called QUEST, is evolved by crowding-based differential evolution principles to locate the promising regions of the search space. The second population, called TARGET, uses PSO to exploit useful information in the vicinity of the best position found by the QUEST. When the search process of the TARGET population around the best-found position of the QUEST becomes unproductive, the TARGET population is stopped and all genomes of the QUEST population are allowed to perform extra operation using hill-climbing local search. They also applied several mechanisms to conquer the existing challenges in the dynamic environments and to spend more function evaluations (FEs) around the best-found position in the search space.

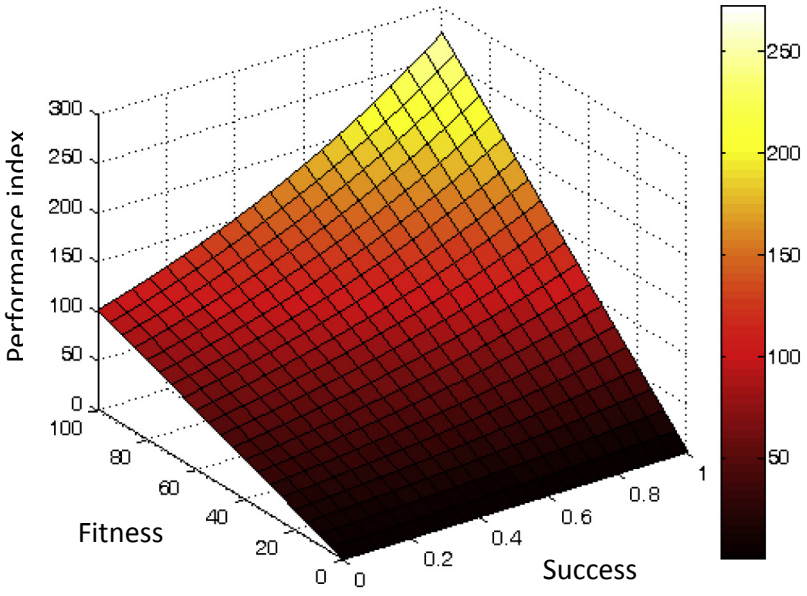

Fig. 2. Performance index values for different fitness values and success rates.

Table 1
Parameter settings for the moving peaks benchmark.

Parameter	Default values (Scenario 2)	Other tested values
Number of peaks (m)	10	1,2, 5, 7, 20, 30, 40, 50, 100, 200
Height severity	7.0	
Width severity	1.0	
Peak function	cone	Sphere
Number of dimensions (D)	5	10, 15, 20, 25
Height range (H)	$\in$ [30, 70]	
Width range (W)	$\in$ [1, 12]	
Standard height (I)	50.0	
Search space range (A)	$[0, 100]^D$	
Frequency of change (f)	5000	1000, 2000, 3000, 4000
Shift severity (s)	1.0	3.0, 5.0
Correlation coefficient (λ)	0.0	
Basic function	No	

Table 2
Default parameter settings for the proposed DynDE variants.

Parameter	Default value
Number of populations	10
Number of Brownian individuals	2
Number DE individuals	4
σ	0.2
Exclusion radius	31.5
F	0.5
CR	0.5
Reward parameter (a)	0.15
Penalty parameter (b)	0.05

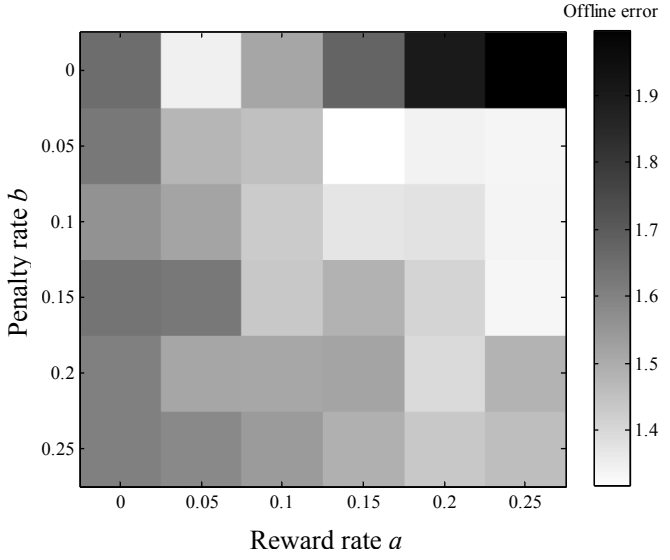

Fig. 3. Average offline error of DynDE + LA with different reward and penalty parameters in MPB's Scenario 2.

2.2. Methods with a variable number of populations

Another group of studies has investigated the multi-population schemes with a variable number of sub-populations. The first strategy to make a variable number of populations is to split off sub-populations from the main population. The major strategy in this approach is to divide the search space into different sub-regions, using a parent population, and carefully exploit each sub-region with a distinct child population. Such a method was first proposed by Ref. [33]. Borrowing the concept of forking, they proposed a multi-population genetic algorithm for DOPs called self-organizing scouts (SOS). In SOS, the optimization process begins with a large parent population exploring through the whole fitness landscape with the aim to find promising areas. When such areas are located by the parent, a child population is split off from the parent population and independently explores the respective sub-space, while the parent population continues to search in the remaining search space for locating new optimum. The search area of each child population is defined as a sphere with radius r and centered at the best individual. In order to determine the number of individuals in each

Table 3
Average offline error ±standard error of different DynDE variants in DOPs with *peak function* = cone, and varying the shift severity and change frequency. Results with asterisk symbol are not statistically significant according to the Wilcoxon's rank sum test.

MPB Parameters		DynDE	DynDE + PI			DynDE + LA		
s	*f*	Offline Error ± Std Error	Offline Error ± Std Error	% Imp	p-val	Offline Error ± Std Error	% Imp	p-val
1	1000	3.35 ± 0.04	2.96 ± 0.05	11.64	0.00	2.83 ± 0.04	15.52	0.00
	2000	2.31 ± 0.05	2.05 ± 0.05	11.26	0.00	1.96 ± 0.04	15.15	0.00
	3000	1.89 ± 0.05	1.69 ± 0.06	10.58	0.00	1.65 ± 0.05	12.70	0.00
	4000	1.67 ± 0.05	1.56 ± 0.08	6.59	0.03	1.48 ± 0.06	11.38	0.00
	5000	1.50 ± 0.05	1.47 ± 0.08	2.00	0.34*	1.32 ± 0.06	12.00	0.00
3	1000	8.19 ± 0.06	6.44 ± 0.07	21.37	0.00	6.48 ± 0.05	20.88	0.00
	2000	5.13 ± 0.06	4.01 ± 0.07	21.83	0.00	4.13 ± 0.06	19.49	0.00
	3000	3.85 ± 0.07	3.05 ± 0.08	20.78	0.00	3.18 ± 0.08	17.40	0.00
	4000	3.14 ± 0.07	2.52 ± 0.08	19.75	0.00	2.63 ± 0.07	16.24	0.00
	5000	2.69 ± 0.07	2.24 ± 0.08	16.73	0.00	2.23 ± 0.06	17.10	0.00
5	1000	13.67 ± 0.10	10.13 ± 0.09	25.90	0.00	10.44 ± 0.08	23.63	0.00
	2000	8.66 ± 0.10	6.13 ± 0.10	29.21	0.00	6.57 ± 0.09	24.13	0.00
	3000	6.29 ± 0.09	4.45 ± 0.10	29.25	0.00	4.88 ± 0.09	22.42	0.00
	4000	4.98 ± 0.08	3.59 ± 0.09	27.91	0.00	3.87 ± 0.08	22.29	0.00
	5000	4.26 ± 0.10	3.17 ± 0.10	25.59	0.00	3.33 ± 0.09	21.83	0.00

Note: change severity (*s*), change frequency (*f*), percentage improvement over DynDE (%Imp), Wilcoxon rank sum test result (p-val).

Table 4
Average offline error ±standard error of different DynDE variants in DOPs with *peak function* = sphere, and varying the shift severity and change frequency. Results with asterisk symbol are not statistically significant according to the Wilcoxon's rank sum test.

MPB Parameters		DynDE	DynDE + PI			DynDE + LA		
s	*f*	Offline Error ± Std Error	Offline Error ± Std Error	% Imp	p-val	Offline Error ± Std Error	% Imp	p-val
1	1000	1.49 ± 0.08	1.25 ± 0.07	15.80	0.01	1.26 ± 0.07	15.32	0.01
	2000	0.95 ± 0.05	0.74 ± 0.05	22.16	0.00	0.82 ± 0.05	13.41	0.01
	3000	0.77 ± 0.04	0.61 ± 0.04	20.27	0.00	0.67 ± 0.04	13.57	0.01
	4000	0.70 ± 0.04	0.56 ± 0.04	19.98	0.00	0.59 ± 0.04	15.67	0.01
	5000	0.66 ± 0.04	0.53 ± 0.04	19.36	0.00	0.53 ± 0.03	19.00	0.00
3	1000	4.08 ± 0.06	3.30 ± 0.07	19.07	0.00	3.31 ± 0.06	18.89	0.00
	2000	2.39 ± 0.04	1.89 ± 0.06	20.89	0.00	1.90 ± 0.04	20.38	0.00
	3000	1.79 ± 0.04	1.39 ± 0.05	22.41	0.00	1.41 ± 0.04	21.32	0.00
	4000	1.47 ± 0.04	1.11 ± 0.05	24.30	0.00	1.16 ± 0.04	20.79	0.00
	5000	1.21 ± 0.03	0.95 ± 0.04	21.73	0.00	1.02 ± 0.04	15.51	0.00
5	1000	10.37 ± 0.06	8.30 ± 0.07	19.95	0.00	8.36 ± 0.04	15.51	0.00
	2000	5.99 ± 0.05	4.41 ± 0.08	26.43	0.00	4.51 ± 0.03	24.66	0.00
	3000	4.25 ± 0.03	3.03 ± 0.06	28.73	0.00	3.17 ± 0.03	25.48	0.00
	4000	3.32 ± 0.04	2.34 ± 0.06	29.64	0.00	2.49 ± 0.03	24.92	0.00
	5000	2.71 ± 0.03	1.85 ± 0.05	31.71	0.00	2.06 ± 0.04	24.15	0.00

Note: change severity (*s*), change frequency (*f*), percentage improvement over DynDE (%Imp), Wilcoxon rank sum test result (p-val).

population, including the parent, a quality measure is calculated for each population as defined in Ref. [33]. The higher the value of the quality measure for a population, the more will be the number of individuals are assigned to the population. The captured region by the child population is then isolated by re-initializing individuals in parent population that fall into the search range of the child population. In SOS, overlapping among child populations is usually accepted unless the best individual of a child population falls within the search radius of another population. In this case, the whole child population is removed.

Inspired by the SOS [26], proposed a multi-swarm algorithm, named FMSO. FMSO starts with a large parent swarm exploring the search space with the aim to locate promising regions. If the quality of the best particle in the parent swarm improves, it implies that a promising search area may be found. Therefore, a child swarm is split off from the parent swarm to exploit its own sub-space. The search territory of each child swarm is determined by a radius *r* which is relative to the range of the landscape and width of the peaks. If the best particle of one child swarm approaches to the area captured by another child swarm, the worse child swarm will be removed to prevent child populations to reside on the same peak. Besides, if a child swarm fails to improve its quality in a certain number of iterations, the best particle of the child swarm jumps to a new position according to a scaled Gaussian distribution. Another similar idea can be found in Ref. [21]. An interesting approach is the *hibernating* multi-swarm optimization algorithm proposed by Ref. [20], where *unproductive* child swarms are stopped by *hibernating* them. In turn, more FEs are available for *productive* child swarms.

[34] employed several mechanisms in a multi-swarm algorithm to tackle existing challenges in dynamic environments. In their proposal (FTMPSO), a randomly initialized *finder* swarm explores the search space for locating the position of the peaks. In order to enhance the exploitation of promising solutions, a *tracker* swarm is activated by transferring the fittest particles from the finder swarm into the newly created tracker swarm. Besides, the finder swarm is then reinitialized into the search space to capture other uncovered peaks. Afterward, the activated tracker swarm is responsible for finding the top of the peak using a local search along with following the respective peak upon detecting a change in the environment. In order to avoid tracker swarms exploiting the same peak, exclusion is applied between every pair of tracker swarms. Besides, if the finder swarm converges to a populated peak, it is reinitialized into the search space without generating any tracker swarm.

Recently [35], proposed a hybrid approach based on PSO and local search. In this approach, a swarm of particles is used to estimate the

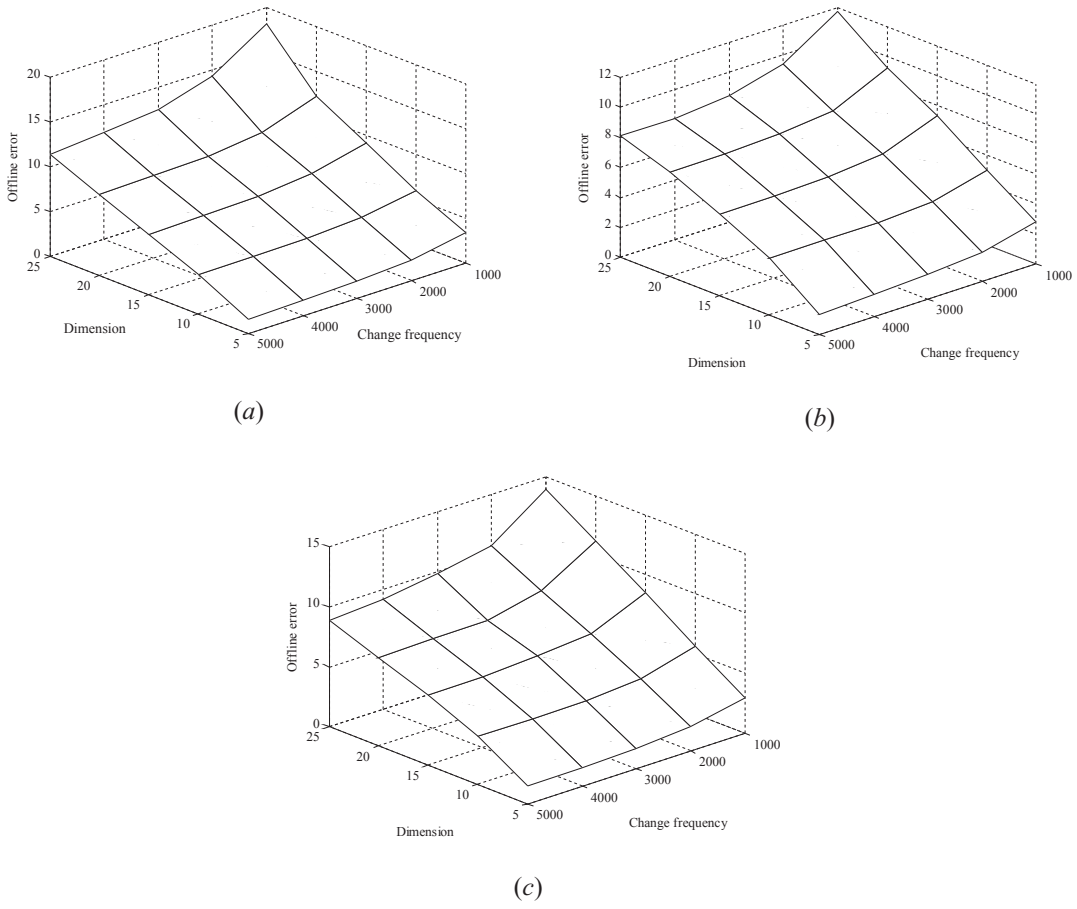

Fig. 4. Offline error of (*a*) DynDE, (*b*) DynDE + LA, and (*c*) DynDE + PI on various DOPs with conical peaks.

location of the peaks. Once the swarm has converged, a local search agent is created to exploit the respective region. Moreover, a density control mechanism is introduced to remove redundant local search agents. They also studied three adaptations to the basic approach. From the experimental results, they confirmed the benefits of the proposed approach.

Different from the above algorithms where the sub-populations are split off from the main population, another way to create multiple populations is to divide the main population of the algorithm into several clusters, i.e. sub-populations, via clustering methods. For example [36], proposed a speciation-based PSO (SPSO) for tracking multiple optimum in dynamic environments, which dynamically distributes particles of the main swarm over a variable number of so-called *species*. In Ref. [37] the performance of SPSO was improved by estimating the location of the peaks using a least squares regression method.

Similarly [27], proposed a clustering PSO (CPSO) for locating and tracking multiple optimums. CPSO employed a single linkage hierarchical clustering method to create a variable number of sub-swarms, and assign them to different promising sub-regions of the search space. Each created sub-swarm uses the PSO with a modified *gbest* model to exploit the respective region. In order to avoid different clusters from crowding, they applied a redundancy control mechanism. In this regard, when two sub-swarms are located on a single peak they are merged together to form a single sub-swarm. So, the worst performing individuals of the sub-swarm are removed until its size is equal to a predefined threshold. In another work [38], the fundamental idea of CPSO is extended by introducing a novel framework for covering undetectable dynamic environments.

In Ref. [39] a competitive clustering PSO for DOPs is introduced. It employs a multi-stage clustering procedure to split the particles of the main swarm. Then, the particles are assigned to a varying number of sub-swarms based on the particles' positions and their objective function values. In addition to the sub-swarms, there is also a group of free particles that is used to explore the environment to locate new emerging optimum or track the current optimum which are not followed by any sub-swarm. Recently [40], proposed a cluster-based differential evolution with external archive which uses *k-means* clustering method to create a variable number of populations.

Another approach for creating multiple sub-populations from the main population is to divide the search space into several partitions and keep the number of individuals in each partition less than a predefined threshold. In this group, we find the proposal of [19]. Here, cellular automata are incorporated into a PSO algorithm (cellular PSO). In cellular PSO, the search space is partitioned into some equally sized cells using cellular automata. Then, particles of the swarm are allocated to different cells according to their positions in the search space. Particles residing in each cell use their personal best positions and the best solution found in their neighborhood cells for searching an optimum. Moreover, whenever the number of particles within each cell exceeds a predefined threshold, randomly selected particles from the saturated cells are transferred to random cells within the search space. In addition, each cell has a memory that is used to keep track of the best position found within the boundary of the cell and its neighbors. In another work (Hashemi and Meybodi 2009a), they improved the performance of cellular PSO by changing the role of particles to quantum particles just at the moment when a change occurs.

Inspired by the cellular PSO [24], proposed an algorithm based on differential evolution, called CellularDE. It employs the *DE/rand-to--best/1/bin* scheme to provide local exploration capability for genomes residing in each cell. After detecting a change in the environment, the population performs a random local search for several upcoming iterations.

Following a similar idea [25], presented a two-phased cellular PSO to

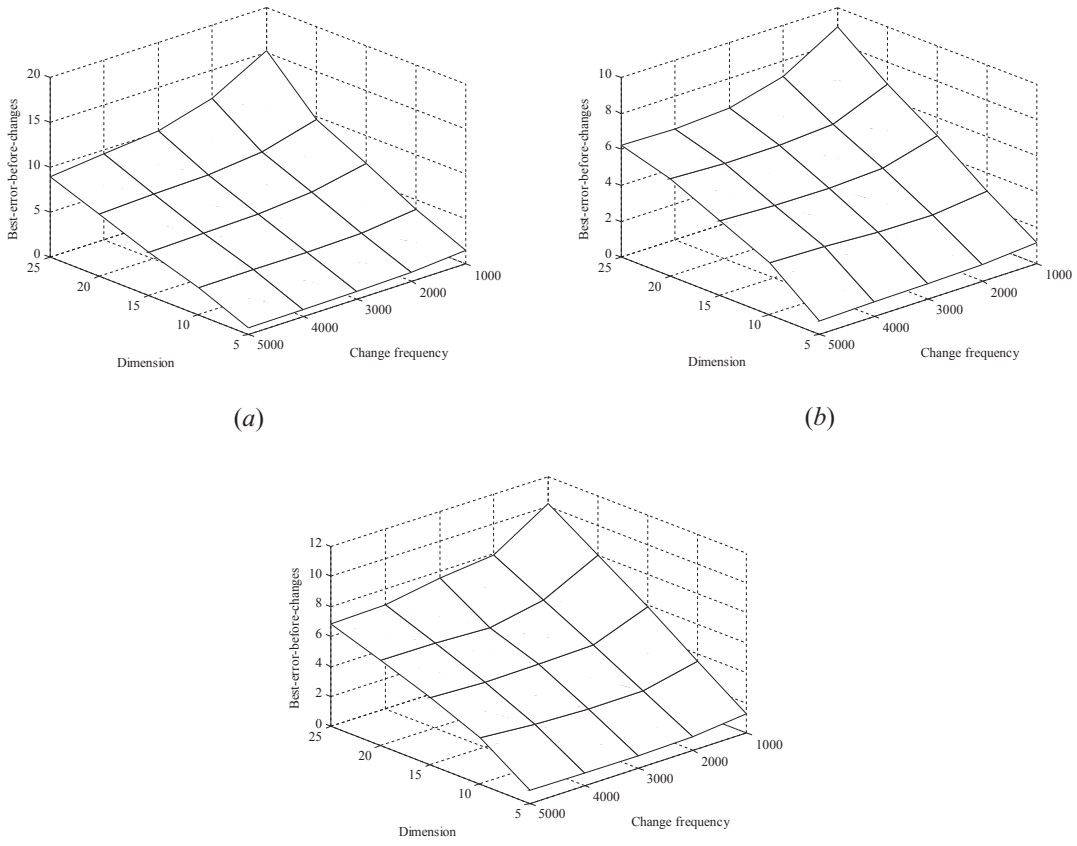

Fig. 5. Best-error-before-change of (*a*) DynDE, (*b*) DynDE + LA, and (*c*) DynDE + PI on various DOPs with conical peaks.

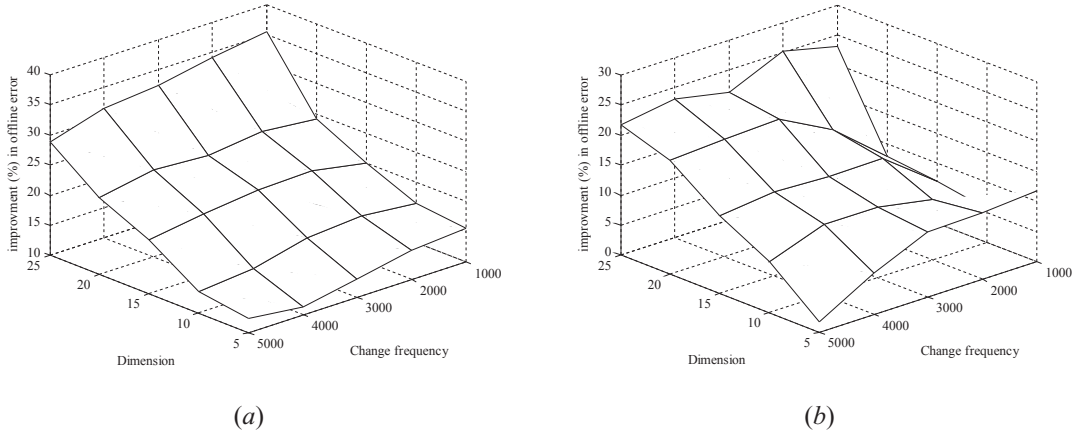

Fig. 6. Percentage of improvement in offline error of (*a*) DynDE + LA, and (*b*) DynDE + PI over DynDE on various DOPs with conical peaks.

address the shortcomings of cellular PSO. In their approach, each cell has two phases: an *exploration* phase, in which the search is performed by a modified PSO algorithm, and an *exploitation* phase, in which a directed local search is conducted. Initially, all cells are in exploration phase, but upon converging to local optimum they go to the exploitation phase. Moreover, they eliminated the need for defining a threshold for each cell.

2.3. Methods with an adaptive number of populations

The last group of studies includes methods that control the search progress and adapt the number of populations based on one or more feedback parameters.

The critical drawback of the multi-swarm algorithm proposed in Ref. [17] is that the number of swarms should be defined before the optimization process. This is a clear limitation in real-world problems where information about the environment might be not available. The very first attempt to adapt the number of populations in dynamic environments was made by Ref. [41]. He developed an adaptive mQSO (AmQSO) which adaptively determines the number of swarms either by spawning new swarms into the search space, or by destroying redundant swarms [41]. In this algorithm, swarms are categorized into two groups: (1) *free* swarms, whose expansion, i.e. the maximum distance between

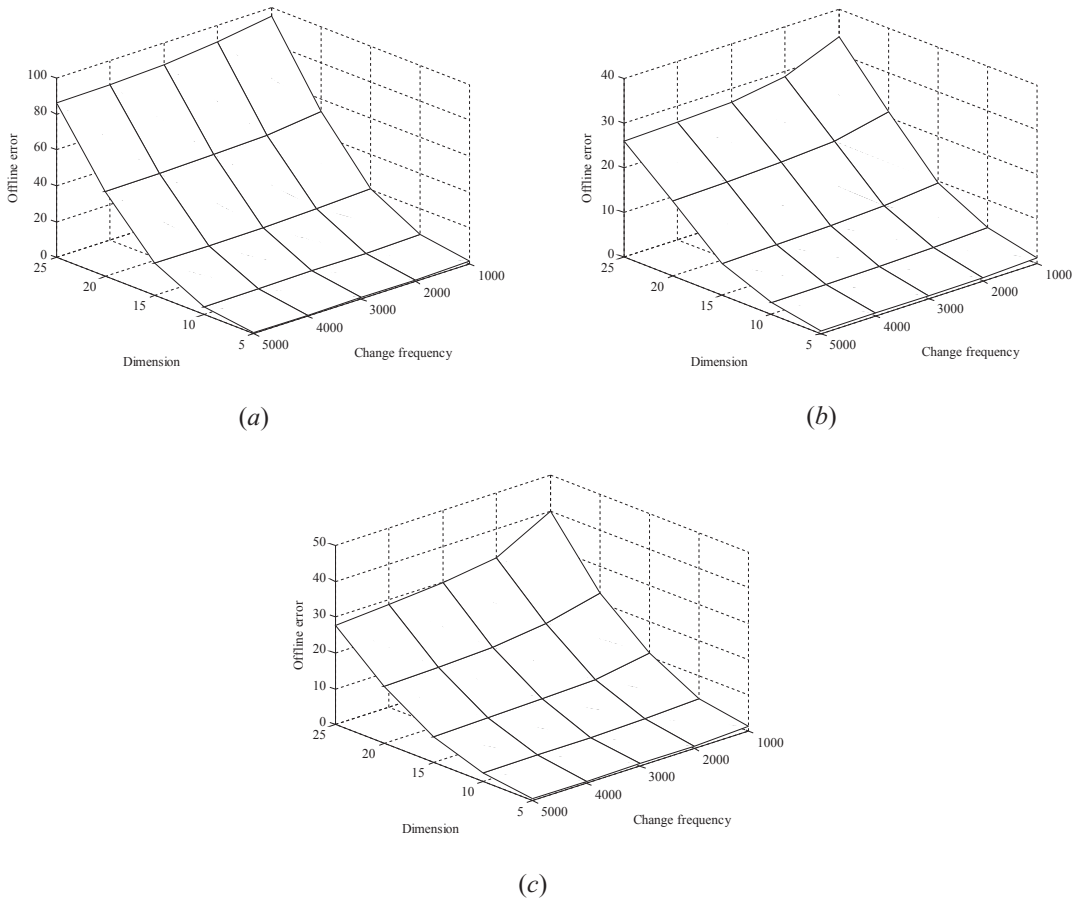

Fig. 7. Offline error of (*a*) DynDE, (*b*) DynDE + LA, and (*c*) DynDE + PI on various DOPs with spherical peaks.

any two particles in the swarm in all dimensions, is larger than a predefined radius r_{conv}, and (2) *converged* swarms. Once the expansion of a free swarm becomes smaller than a radius r_{conv}, it is converted to the converged swarm. In AmQSO, when the number of free swarms (M_{free}) is dropped to zero, a free swarm is initialized in the search space for capturing undetected peaks. On the other hand, free swarms are removed from the search space if M_{free} is higher than a threshold n_{excess}.

[42] proposed a dynamic modified multi-population artificial fish swarm algorithm based on the general principles of [41]. In this approach, the authors modified the basic parameters, behaviors and general procedure of the standard artificial fish swarm algorithm to fulfill the requirements for optimization in dynamic environments.

In Ref. [43] the authors proposed the dynamic population differential evolution (DynPopDE), in which the populations are adaptively spawned and removed based on their performance. In this approach, when all of the current populations fail to improve their fitness, DynPopDE produces a new population of random individuals in the search space. They defined a function $\Upsilon(t)$, which indicates if all existing populations in the search space have been stagnated, as follows:

$$\Upsilon(t) = \begin{cases} true & if\ (\Delta f_k(t) = 0) \forall k \in \kappa \\ false & otherwise \end{cases} \quad (1)$$

where κ is the set of current populations, and $\Delta f_k(t)$ is the amount of improvement in the quality of the kth population at time t:

$$\Delta f_k(t) = |f_k(t) - f_k(t-1)| \quad (2)$$

DynPopDE starts with a single population and gradually adapt to an appropriate number of populations. On the other hand, when the number of populations surpass the number of peaks in the landscape, redundant populations can be detected as those which are frequently reinitialized by the exclusion operator. Therefore, a population k will be removed from the search space when it is marked for restart due to exclusion, and it has not improved since its last FEs ($\Delta f_k(t) \neq 0$).

[44] adapted the ideas from Ref. [41] to a cuckoo search algorithm for DOPs. Their approach uses a modified cuckoo search algorithm to increase the convergence rate of populations for finding the peaks, and exclusion to prevent populations from converging to the same areas of the search space.

Recently [45], proposed an adaptive multi-population framework to identify the correct number of populations. Their framework has three major procedures: clustering, tracking and adapting. Moreover, they have employed various components to further enhance the overall performance of the proposed approach, including a hibernation scheme, a peak hiding scheme, and two movement schemes for the best individuals. Their method was empirically shown to be effective in comparison with a set of algorithms.

Several attempts have been done to enhance the performance of multi-population methods in DOPs. Some of them can be categorized as follows:

i. Changing the number and distribution of special individuals, e.g. quantum particles, Brownian individuals [46–49].
ii. Using different memory schemes to enhance the performance of the multi-population methods [40,50]
iii. Enhancing the local search ability of each sub-population [51,52]
iv. Modifying the exclusion operator [45,53,54]
v. Managing the FEs [55–58]

The present study falls into the last category where the main objective is to allocate more FEs to the most promising areas of the search space.

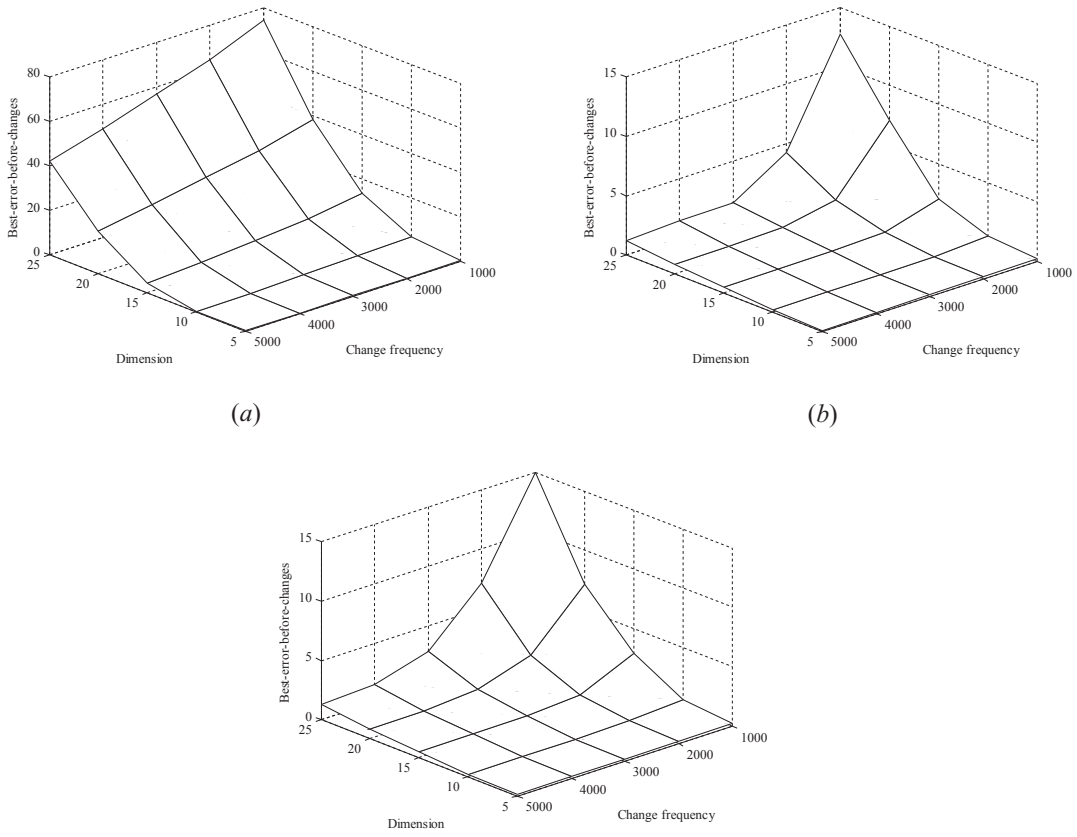

Fig. 8. Best-error-before-change of (*a*) DynDE, (*b*) DynDE + LA, and (*c*) DynDE + PI on various DOPs with spherical peaks.

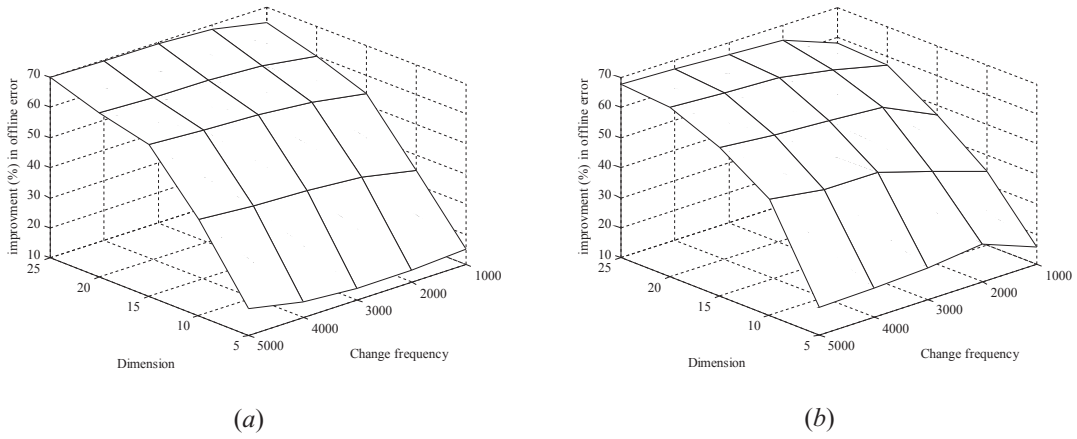

Fig. 9. Percentage of improvement in offline error of (*a*) DynDE + LA, and (*b*) DynDE + PI over DynDE on various DOPs with spherical peaks.

As mentioned above, the idea of managing FEs for improving the performance of the multi-population approaches has been previously addressed by other researchers. For example [59], proposed an extension to the DynDE referred to as favored populations DE. In their approach, FEs between two successive changes in the environment are divided into three phases: (1) all populations are evolved according to normal DynDE for ζ_1 generations in order to locate peaks, (2) the weaker populations are frozen and stronger populations are executed for another ζ_2 generations, and (3) frozen populations are back to search process and all populations are evolved for ζ_3 generations in a normal DynDE manner. This strategy adds three parameters ζ_1, ζ_2 and ζ_3 to DynDE, which must be tuned manually. In another work [53], they equipped DynDE with a new mechanism, namely competitive population evaluation (CPE). The main motivation behind CPE is to manage resources (i.e. FEs) in a way that enables the optimization algorithm to reach the lowest error faster. In CPE, populations compete with each other to obtain FEs, which are allocated to populations according to their performance. The performance of each population is measured based on the current fitness of the best individual in the population and the improvement amount of the best individual during the previous evaluation of the population. Hence, the best-performing population will thus be the population with the highest fitness and improvement values. At each iteration, the best-performing population takes the FEs and evolves itself until its performance drops below that of another population, where the other

Table 5
Average offline error ±standard error of different DynDE variants in DOPs with *peak function* = cone, and varying the number of peaks. Results with asterisk symbol are not statistically significant according to the Wilcoxon's rank sum test.

m	DynDE	DynDE + PI			DynDE + LA		
	Offline Error ± Std Error	Offline Error ± Std Error	% Imp	p-val	Offline Error ± Std Error	% Imp	p-val
1	3.80 ± 0.17	2.70 ± 0.11	28.89	0.00	3.07 ± 0.12	19.27	0.00
2	2.41 ± 0.17	1.63 ± 0.13	32.60	0.00	1.88 ± 0.14	22.10	0.00
5	1.64 ± 0.09	1.32 ± 0.09	19.74	0.00	1.41 ± 0.08	14.30	0.01
7	1.63 ± 0.08	1.50 ± 0.09	7.94	0.05*	1.32 ± 0.05	18.66	0.00
10	1.50 ± 0.05	1.47 ± 0.08	1.71	0.34*	1.32 ± 0.06	12.32	0.00
20	2.74 ± 0.07	2.46 ± 0.08	10.40	0.00	2.60 ± 0.07	5.24	0.10*
30	3.22 ± 0.10	2.91 ± 0.11	9.66	0.01	3.05 ± 0.10	5.15	0.18*
40	3.46 ± 0.08	3.28 ± 0.09	5.19	0.15*	3.34 ± 0.07	3.31	0.49*
50	3.81 ± 0.10	3.38 ± 0.10	11.29	0.00	3.56 ± 0.09	6.51	0.06*
100	4.21 ± 0.12	3.78 ± 0.11	10.21	0.01	3.88 ± 0.11	7.68	0.07*
200	3.99 ± 0.12	3.62 ± 0.09	9.18	0.05*	3.71 ± 0.09	6.99	0.15*

Note: number of peaks (m), percentage improvement over DynDE (%Imp), and Wilcoxon rank sum test result (p-val).

Table 6
Average offline error ±standard error of different DynDE variants in DOPs with *peak function* = sphere, and varying the number of peaks. Results with asterisk symbol are not statistically significant according to the Wilcoxon's rank sum test.

m	DynDE	DynDE + PI			DynDE + LA		
	Offline Error ± Std Error	Offline Error ± Std Error	% Imp	p-val	Offline Error ± Std Error	% Imp	p-val
1	19.04 ± 0.59	15.78 ± 0.37	17.12	0.00	15.36 ± 0.41	19.37	0.00
2	10.05 ± 0.38	7.70 ± 0.25	23.35	0.00	7.34 ± 0.17	26.99	0.00
5	3.10 ± 0.18	2.04 ± 0.10	34.32	0.00	2.26 ± 0.12	27.03	0.00
7	1.53 ± 0.08	1.04 ± 0.04	32.37	0.00	1.17 ± 0.07	23.56	0.00
10	0.66 ± 0.04	0.53 ± 0.04	19.21	0.00	0.53 ± 0.03	18.86	0.00
20	2.14 ± 0.08	2.08 ± 0.09	2.93	0.50*	2.07 ± 0.07	3.38	0.60*
30	2.83 ± 0.08	2.47 ± 0.11	12.68	0.00	2.52 ± 0.11	10.91	0.00
40	2.96 ± 0.10	2.75 ± 0.09	6.99	0.14*	2.93 ± 0.09	0.78	0.99*
50	3.17 ± 0.09	3.00 ± 0.08	5.53	0.34*	3.11 ± 0.10	2.13	0.71*
100	3.40 ± 0.09	3.31 ± 0.09	2.78	0.80*	3.38 ± 0.08	0.74	0.92*
200	3.77 ± 0.10	3.59 ± 0.09	4.86	0.23*	3.76 ± 0.10	0.45	0.94*

Note: number of peaks (m), percentage improvement over DynDE (%Imp), and Wilcoxon rank sum test result (p-val).

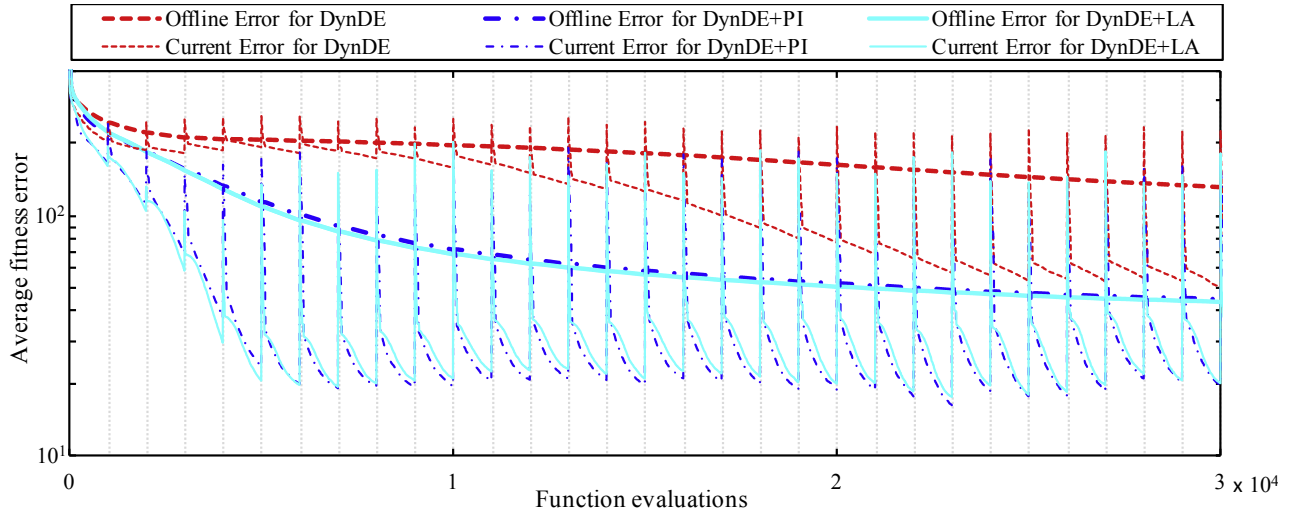

Fig. 10. Evolution of the offline error and current error for the DynDE-based algorithms in a problem with $m = 10$, $f = 1000$, $s = 5$, $d = 25$, and *peak function* = cone. The vertical dotted lines indicate the beginning of a new environment.

population takes the resource, and this process continues during the run. This adaptation scheme allows FEs to be mostly spent on higher peaks.

3. Proposed framework

This section presents the proposed framework for improving the performance of the multi-population methods for DOPs. One frequent feature in most of the multi-population methods for DOPs is that the FEs are equally distributed among the sub-populations. It implies that the same quota of the FEs is allocated to all sub-populations, regardless to the quality of the positions where they stand on. However, for several reasons, the equal distribution of FEs among sub-populations is not a good policy:

i. The main goal in dynamic optimization is to locate the global optimum in a minimum amount of time and track its movements

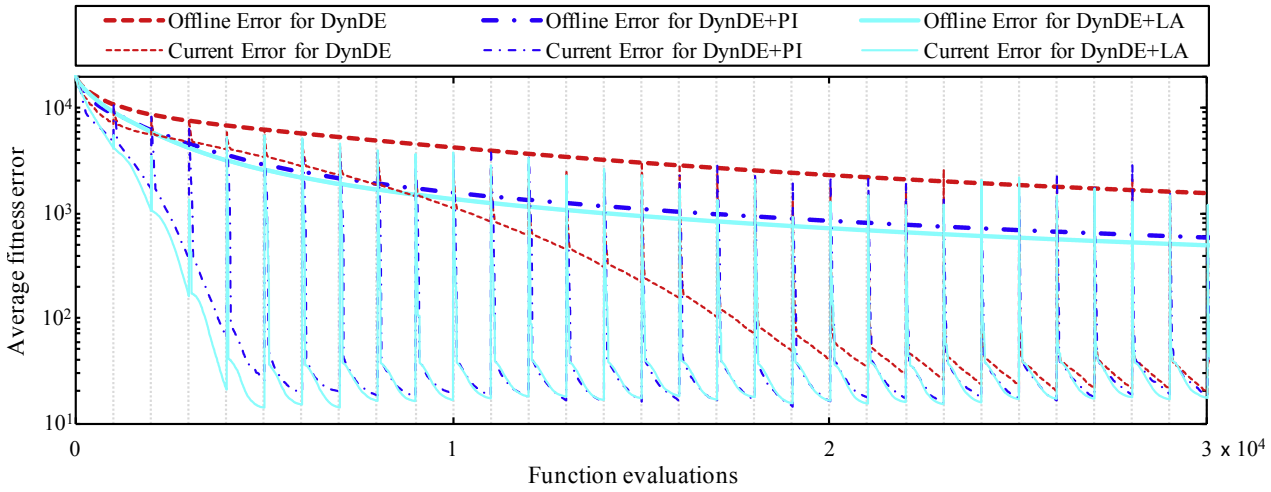

Fig. 11. Evolution of the offline error and current error for the DynDE-based algorithms in a problem with $m = 10$, $f = 1000$, $s = 5$, $d = 25$, and *peak function* = sphere. The vertical dotted lines indicate the beginning of a new environment.

Table 7
Average offline error ±standard error of different multi-population algorithms on MPB's instances with different number of peaks.

Method	*m*								
	1	5	10	20	30	40	50	100	200
RPSO	0.56 ± 0.04	12.22 ± 0.76	12.98 ± 0.48	12.79 ± 0.54	12.35 ± 0.62	11.37 ± 0.41	11.34 ± 0.29	9.73 ± 0.28	8.90 ± 0.19
mCPSO[a]	4.93 ± 0.17	2.07 ± 0.08	2.05 ± 0.07	2.95 ± 0.08	3.38 ± 0.11	3.69 ± 0.11	3.68 ± 0.11	4.07 ± 0.09	3.97 ± 0.08
mQSO[a]	5.07 ± 0.17	1.81 ± 0.07	1.75 ± 0.07	2.74 ± 0.07	3.27 ± 0.11	3.60 ± 0.08	3.65 ± 0.11	3.93 ± 0.08	3.86 ± 0.07
SPSO	2.64 ± 0.10	2.15 ± 0.07	2.51 ± 0.09	3.21 ± 0.07	3.64 ± 0.07	3.85 ± 0.08	3.86 ± 0.08	4.01 ± 0.07	3.82 ± 0.05
MEPSO	**0.53 ± 0.15**	4.68 ± 1.01	4.02 ± 0.56	4.19 ± 0.57	4.27 ± 0.83	–	4.20 ± 0.47	–	–
RVDEA/Mem	1.23	–	4.88	5.68	5.86	5.65	5.21	4.98	4.92
MDUMDA	4.45 ± 0.69	1.53 ± 0.47	**1.14 ± 0.39**	2.99 ± 0.50	3.98 ± 0.74	4.05 ± 0.46	4.66 ± 0.54	–	–
DHPSO	1.64 ± 0.01	1.62 ± 0.01	2.73 ± 0.03	2.82 ± 0.02	3.97 ± 0.03	4.67 ± 0.03	5.13 ± 0.03	–	–
DynDE + LA	3.07 ± 0.12	1.41 ± 0.08	1.32 ± 0.06	2.60 ± 0.07	3.05 ± 0.10	3.34 ± 0.07	3.56 ± 0.09	3.88 ± 0.11	3.71 ± 0.09
DynDE + PI	2.70 ± 0.11	**1.32 ± 0.09**	1.47 ± 0.08	**2.46 ± 0.08**	**2.91 ± 0.11**	**3.28 ± 0.09**	**3.38 ± 0.10**	**3.78 ± 0.11**	**3.62 ± 0.09**

[a] In order to have a fair comparison, the algorithms without anti-convergence operator have been considered.

Table 8
Average offline error ±standard error of DynDE with different function evaluation management scheme in DOPs with *peak function* = sphere, and varying the number of peaks.

m	DynDE + CPE	DynDE + PI	DynDE + LA
	Offline Error ± Std Error	Offline Error ± Std Error	Offline Error ± Std Error
1	2.79 ± 0.16	**2.70 ± 0.11**	3.07 ± 0.12
2	1.93 ± 0.15	**1.63 ± 0.13**	1.88 ± 0.14
5	1.55 ± 0.08	**1.32 ± 0.09**	1.41 ± 0.08
7	1.59 ± 0.06	1.50 ± 0.09	**1.32 ± 0.05**
10	1.49 ± 0.05	1.47 ± 0.08	**1.32 ± 0.06**
20	2.66 ± 0.07	**2.46 ± 0.08**	2.60 ± 0.07
30	3.25 ± 0.11	**2.91 ± 0.11**	3.05 ± 0.10
40	3.53 ± 0.09	**3.28 ± 0.09**	3.34 ± 0.07
50	3.77 ± 0.09	**3.38 ± 0.10**	3.56 ± 0.09
100	4.15 ± 0.12	**3.78 ± 0.11**	3.88 ± 0.11
200	3.98 ± 0.11	**3.62 ± 0.09**	3.71 ± 0.09

in the solution space. Therefore, strategies for quickly locating the global optimum are preferable.

ii. A dynamic optimization problem may contain several peaks (local optimum). However, as the heights of the peaks are different, they do not have the same importance from the optimality point of view. Therefore, spending an equal number of FEs on all of them postpones the process of reaching a lower error, and

iii. Many real-world optimization problems are large-scale in nature. For these problems, the equal distribution of FEs among sub-populations would be detrimental to the performance of the algorithms.

To address the above issues, in this paper we propose an approach based on the idea of scheduling. The general idea is depicted in Fig. 1. It shows that the most important component of the proposed framework is the feedback parameter for determining which sub-population will be executed. As we discuss below several options exist for defining such a parameter.

3.1. Scheduling multi-population method based on a performance index

The quality of the best solution found by the sub-populations and the degree of diversity among the individuals are two critical factors in determining the performance of sub-populations. Various studies exist in the literature that have used these factors to improve the performance of the multi-population methods for DOPs. For example [20], considered the diversity of the particles for improving the performance of mPSO. Their proposed hibernation mechanism makes more FEs available for the parent swarm to explore the search space, and for other child swarms to exploit their respective areas in the landscape [55,56]. proposed a resource management mechanism, called *swarm control mechanism*, to improve the performance of the mQSO. The swarm control mechanism is activated on the swarms with low diversity and bad fitness, and stops them from consuming FEs. All mentioned modified algorithms reported improvements over the basic methods.

In this study, we combine the *success rate* and the quality of the best solution found by the sub-populations in a single criterion called

Table 9
Mean (standard deviation) for the algorithms on GDBG instances with change frequency = 5000 and dimension = 10.

Method	F	Type of changes						
		T1	T2	T3	T4	T5	T6	T7
jDE	F1-10	6.07 (5.09)	12.29 (10.18)	12.61 (10.72)	14.28 (11.50)	11.72 (7.99)	25.98 (16.98)	11.82 (10.08)
	F1-50	8.42 (5.67)	13.76 (9.29)	12.89 (9.39)	15.58 (12.01)	9.99 (6.32)	30.80 (18.51)	13.40 (9.33)
	F2	68.48 (51.59)	303.03 (151.35)	292.69 (155.27)	83.26 (82.31)	346.47 (109.18)	193.50 (141.84)	221.45 (130.75)
	F3	517.00 (256.43)	983.81 (90.01)	992.29 (76.17)	617.21 (459.84)	955.40 (75.62)	960.84 (308.36)	929.86 (121.83)
	F4	90.20 (72.79)	380.75 (181.99)	382.93 (180.47)	106.22 (112.51)	429.84 (125.63)	244.62 (178.69)	307.96 (154.95)
	F5	151.73 (133.58)	324.35 (196.65)	328.05 (189.23)	143.07 (157.33)	377.67 (211.30)	278.64 (191.64)	301.94 (180.60)
	F6	110.47 (85.57)	390.00 (224.26)	384.30 (220.05)	141.66 (162.92)	459.48 (209.67)	276.52 (196.10)	306.96 (181.87)
jDE + LA	F1-10	5.13 (6.17)	9.59 (10.68)	10.25 (10.73)	12.36 (12.19)	8.28 (7.66)	22.05 (18.90)	10.06 (10.99)
	F1-50	8.31 (6.92)	11.79 (10.11)	11.44 (9.65)	12.91 (11.70)	6.82 (5.72)	25.99 (20.60)	11.29 (9.78)
	F2	40.85 (35.39)	240.90 (186.69)	224.33 (185.04)	54.63 (53.08)	294.95 (154.70)	109.82 (118.04)	152.36 (140.06)
	F3	515.88 (268.95)	985.81 (85.30)	988.05 (76.73)	605.55 (470.12)	937.48 (76.27)	967.46 (301.97)	910.95 (145.65)
	F4	64.62 (71.73)	320.47 (220.09)	317.10 (223.44)	88.52 (112.25)	401.90 (165.46)	159.07 (166.79)	229.26 (178.26)
	F5	72.37 (105.19)	164.70 (182.36)	169.50 (182.89)	83.77 (117.49)	174.34 (201.00)	138.60 (164.21)	146.22 (161.88)
	F6	64.84 (69.01)	295.59 (262.60)	260.83 (243.41)	81.49 (116.88)	348.06 (265.78)	178.39 (193.12)	197.88 (183.95)
mSQDE-i	F1-10	15.05 (4.02)	19.99 (7.95)	19.23 (7.90)	28.04 (9.69)	26.59 (7.32)	34.60 (9.91)	20.92 (8.90)
	F1-50	17.46 (5.61)	18.39 (6.63)	18.03 (6.69)	30.26 (9.46)	21.02 (5.05)	33.11 (9.15)	19.95 (7.44)
	F2	118.92 (27.16)	302.50 (97.18)	299.53 (92.96)	163.16 (72.69)	364.23 (69.92)	247.07 (102.20)	278.34 (88.88)
	F3	490.50 (209.74)	1064.12 (63.55)	1058.64 (67.85)	873.24 (376.36)	1033.59 (69.95)	1132.58 (309.98)	1020.70 (125.12)
	F4	155.06 (39.18)	393.37 (109.61)	399.68 (107.69)	220.84 (105.32)	463.75 (68.00)	330.06 (131.72)	387.46 (89.82)
	F5	390.24 (91.85)	623.59 (121.53)	631.04 (117.78)	450.58 (173.76)	973.36 (202.50)	692.01 (244.85)	686.92 (184.15)
	F6	213.81 (52.61)	508.16 (153.00)	500.70 (151.17)	304.07 (157.70)	702.55 (136.62)	466.59 (208.07)	519.07 (157.63)
mSQDE-i + LA	F1-10	9.91 (5.36)	14.54 (9.89)	13.46 (9.04)	20.58 (9.71)	14.58 (8.37)	25.48 (12.44)	13.38 (9.60)
	F1-50	12.79 (6.13)	14.21 (7.96)	13.88 (8.23)	22.33 (10.19)	12.47 (6.21)	24.72 (11.38)	14.66 (8.55)
	F2	88.14 (29.95)	238.26 (141.49)	229.29 (135.23)	107.56 (50.97)	281.72 (134.81)	140.53 (72.05)	156.80 (84.27)
	F3	391.14 (255.84)	1046.59 (71.52)	1040.69 (68.38)	826.86 (418.10)	1007.92 (73.82)	1103.33 (312.23)	1005.37 (131.79)
	F4	111.60 (38.51)	318.56 (172.10)	318.49 (169.42)	137.52 (72.58)	401.68 (141.78)	181.05 (106.04)	252.72 (142.04)
	F5	260.07 (94.46)	367.60 (142.85)	361.85 (134.35)	264.03 (113.27)	417.18 (177.23)	327.44 (162.21)	332.89 (143.42)
	F6	138.20 (50.74)	312.36 (199.53)	314.03 (201.73)	168.81 (101.64)	542.49 (261.49)	262.08 (209.53)	296.46 (200.15)

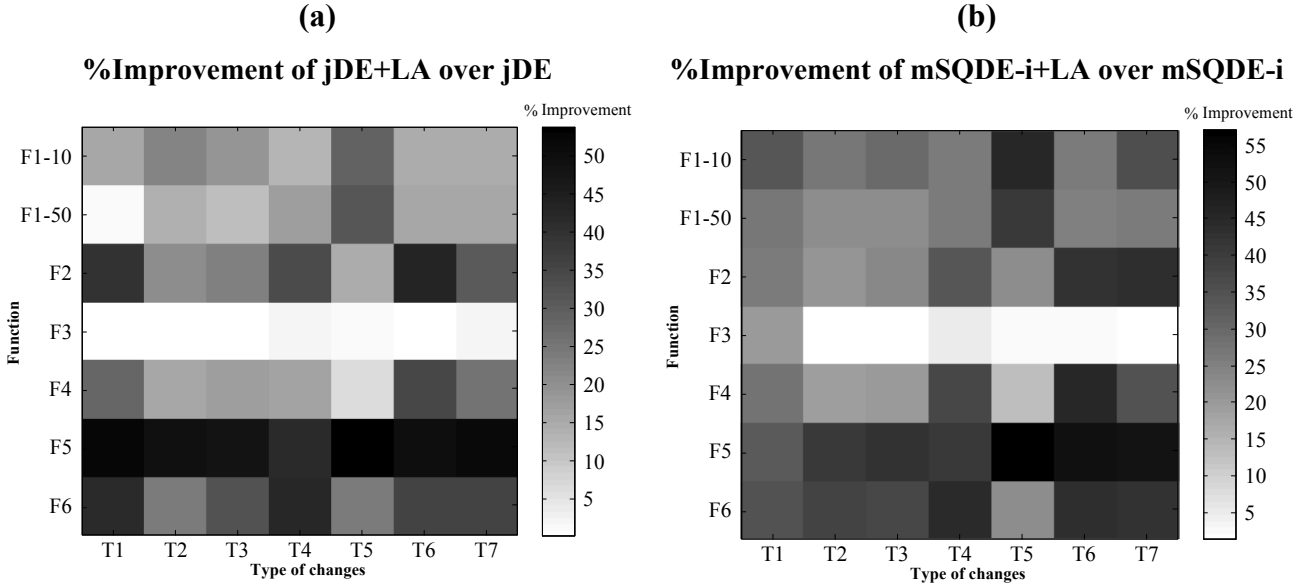

Fig. 12. Percentage of improvement of (*a*) jDE + LA, and (*b*) mSQDE-i + LA over jDE and mSQDE-i on different instances of GDBG, respectively.

performance index. The success rate was previously used by Ref. [60] for adaptive adjustment of the inertia weight in PSO. The reported results by the authors in Ref. [60] proved that the success rate is a good criterion for monitoring the progress of the search process. The success rate determines the portion of improvement in the individuals of a population which is defined as bellow:

$$SR = \frac{number\ of\ improved\ individuals}{total\ number\ of\ individuals} \tag{3}$$

For function maximization, the performance index PI_i for each sub-population i is computed as:

$$PI_i = exp(SR_i) \cdot f_i \tag{4}$$

where SR_i is the success rate of the sub-population i, and f_i represents the fitness value of the best solution found by i.

For function minimization, we must convert the function minimization into a fitness maximization in which a fitness value $f_i = \overline{f}_{\max} - \overline{f}_i$ is used. Here, $\overline{f}_{\max}$ is the maximum value of the fitness function, while $\overline{f}_i$ is the fitness value of the best-found position by sub-population i.

The FEs at each stage are then allocated to the sub-population with the highest performance index, which is selected as follows:

$$selected\ population := argmax\{PI_i\}, \quad i = 1, 2, \ldots, N \tag{5}$$

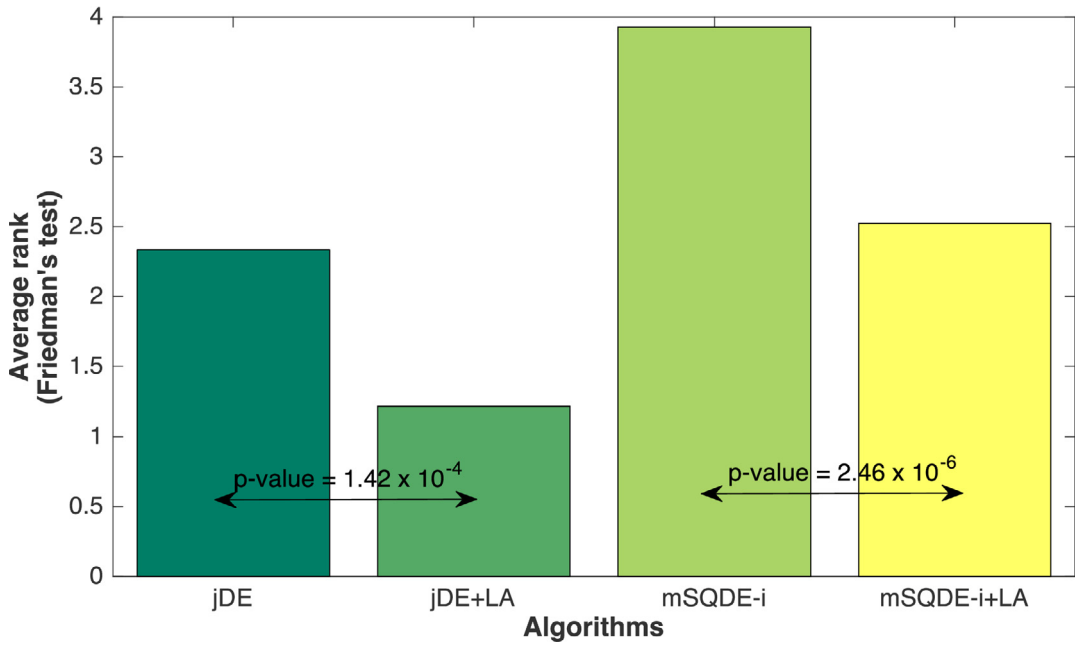

Fig. 13. Statistical results for the algorithms in the GDBG. The average ranks of the algorithms are obtained from the Friedman's test. The p-values of the pairwise comparisons correspond to the Holm's test ($\alpha = 0.05$).

To give an idea of the distribution followed by the performance index, in Fig. 2 we have plotted it according to different combinations of fitness values and success rates. In this illustrative example, we assumed that the fitness function can take values in the range [0, 100].

It is easy to see that the best-performing population will be the population with the highest fitness and success rate. The pseudo code for the proposed scheduling algorithm based on the performance index is shown in Algorithm 1.

Algorithm 1
Scheduling algorithm based on the performance index.

1. Setting the parameters *N*, *pop_size*;/*Set the multi-population cardinality and population size*/
2. Randomly initialize *N* populations of individuals in the search space;/*Initialize populations into the search space*/
3. **while** *termination criterion is not met* **do**
4. **if** a change in the environment is detected **then**
5. Re-evaluate all sub-populations;
6. **end-if**
7. Calculate success rate for each sub-population according to Eq. (3);
8. Update the performance index for each sub-population according to Eq. (4);
9. Select the best-performing sub-population with respect to Eq. (5) and evolve it using the main algorithm;/*e.g. DE, PSO, etc.*/
10. **end-while**

3.2. Scheduling multi-population methods using learning automata

Learning automata (LA) are stochastic learning tools for learning the optimal policy through iterative interaction with an unknown random environment [61,62]. The learning process of the automaton can be divided into three steps:

i. The automaton selects an action from a set of finite actions and applies it to the environment,
ii. The environment evaluates the effectiveness of the selected action and generates a response for the automaton, and
iii. The automaton uses the feedback of the environment to update its internal action probabilities.

By repeating this three-step procedure, the LA will be gradually guided toward selecting the optimal action.

In this work, we rely on a variable-structure learning automata (VSLA) for choosing the sub-population that should be executed at each time step. VSLA can be represented with a quadruple $\{\alpha, \beta, p, T\}$, where $\alpha = \{\alpha_1, \ldots, \alpha_r\}$ is the set of actions, $\beta = \{\beta_1, \ldots, \beta_m\}$ is the set of inputs, $p = \{p_1, \ldots, p_r\}$ is the probability vector which determines the selection probability of each action, and T is the learning algorithm which is used to govern the reinforcement scheme, i.e. $p(n+1) = T\ [\alpha(n), \beta(n), p(n)]$. The framework for the learning process of a VSLA in a stationary environment is shown in Algorithm 2.

Algorithm 2
The framework for the VSLA in a random unknown environment.

1. begin
2. Let $a = \{\alpha_1, \ldots, \alpha_r\}$ be the set of finite actions, where r is the number of actions;
3. **Let** $p = [p_1, \ldots, p_r]$ be the action probability vector of the automaton;
4. **repeat**
5. The learning automaton chooses one of its actions based on the probability distribution P;
6. The environment evaluates the selected action and calculates a reinforcement signal β;
7. The environment sends the reinforcement signal β to the learning automaton;
8. The automaton updates its probability vector p according to learning algorithm T;
9. **until** (the automaton converges to one of its actions)
10. end

The proposed scheduling approach is modeled by an *N*-action learning automaton, where its actions corresponds to different sub-populations, i.e. $n_1, n_2, \ldots, n_N$. In order to select a new sub-population to be executed, the learning automaton $LA_{sub-pop}$ selects one of its actions, e.g. α_i. Then, depending on the selected action, the corresponding sub-population is executed. Next, the learning automaton updates its probability vector using the feedback received from the evolved sub-

population. The structure of the proposed scheduling method is given in Algorithm 3.

Algorithm 3
The proposed learning automata based scheduling method.

1. Set the parameters N, pop_size;/*Multi-population cardinality and population size*/
2. Randomly initialize N populations of individuals in the search space;/*Initialize populations into the search space*/
3. Initialize the $LA_{sub-pop}$ with action set $\alpha = \{1,2, \ldots, N\}$ and action probability vector $p = [(1/N), (1/N), \ldots, (1/N)]$.
4. Set $a = 0.15$ and $b = 0.05$;/* Parameters for updating probability vector in step 12*/
5. **while** *termination criterion not met* **do**
6. **if** a change in the environment is detected **then**
7. Re-evaluate all sub-populations;
8. Reset the action probability vector of $LA_{sub-pop}$;
9. **end**
10. Select a sub-population i using probability vector p and evolve it according to the inner optimizer;/*e.g. DE, PSO, etc.*/
11. Compute the reinforcement signal β;
12. *updatingProbabilityVector*(i, β);
13. **end**

3.2.1. Detection and response to the changes

Several methods have been suggested for detecting changes in dynamic environments. See for example [4] for a good survey on the topic. The most used method consists in reevaluate memories of the algorithm for detecting inconsistencies in their corresponding fitness values. In this work, we considered the best solution of each sub-population. At the beginning of each iteration, the algorithm reevaluates these solutions and compare their current fitness values against those from the previous iteration. Thus, if at least one inconsistency is found from these comparisons, then we will assume that the environment has changed.

Once a change is detected, all sub-populations are re-evaluated and the action probability vector of the automaton is restarted to its initial value.

3.2.2. Updating the probability vector of the automaton

After execution of the selected sub-population, the automaton updates its probability vector based on the reinforcement signal received in response to the selected action. Based on the received reinforcement signal, the automaton determines whether its action was right or wrong, updating its probability vector accordingly. In this work, the reinforcement signal is generated as follows:

$$\text{Reinforcement signal } \beta = \begin{cases} 0 & \text{if } f\left(\vec{X}_{g,t}\right) < f\left(\vec{X}_{g,t+1}\right) \\ 1 & \text{otherwise} \end{cases} \quad (6)$$

where f is the fitness function, $\vec{X}_{g,t}$ is the global best individual of the algorithm at time step t. Eq. (6) implies that if the quality of the best-found solution improves, then a favorable signal is received by the automaton $LA_{sub-pop}$. Afterward, the probability vector of the $LA_{sub-pop}$ is modified based on the learning algorithm of the automaton. See Algorithm 4 for more details.

Algorithm 4
Pseudo-code for updating the probability vector of the automaton. *updatingProbabilityVector* (selected action i, reinforcement signal β).

1. for each action $j \varepsilon [1, 2, \ldots, r]$ of $LA_{sub-pop}$ **do**
2. if ($\beta = 0$) **then**//favorable response
3. $$p_j(n+1) = \begin{cases} p_j(n) + a \bullet (1 - p_j(n)) & \text{if } i = j \\ p_j(n) \bullet (1 - a) & \text{if } i \neq j \end{cases} \quad (7)$$
4. **else if** ($\beta = 1$)//unfavorable response
5. $$p_j(n+1) = \begin{cases} p_j(n) \bullet (1 - b) & \text{if } i = j \\ \frac{b}{r-1} + (1 - b) \bullet p_j(n) & \text{if } i \neq j \end{cases} \quad (8)$$
6. **end-if**
7. **end-for**

In Algorithm 4, a and b are *learning parameters* associated with the reward and penalty, respectively. They take values in the range of [0.0, 1.0]. Three linear reinforcement schemes can be obtained by selecting different values for the reward and penalty parameters, namely L_{R-P}, L_{R-I} and $L_{R-\varepsilon P}$. In L_{R-P} (Linear Reward-Penalty), the value of the parameters a and b are equal ($a = b$). In L_{R-I} (Linear Reward-Inaction) the learning parameter ($b = 0$). Finally, in $L_{R\varepsilon P}$ (Linear Reward-epsilon-Penalty) the learning parameter b is much smaller than a ($a \gg b$).

4. Application of the proposed framework

In what follows we describe how the proposed framework can be incorporated into an existing algorithm for DOPs. Although several options exist, we have considered DynDE, the algorithm proposed in Ref. [29]. DynDE has been extensively studied in the past, showing very good results in challenging DOPs. However, we believe that it is necessary to describe the Differential Evolution paradigm first, which is the underlying optimization method in DynDE. Later, we give a brief overview of DynDE.

4.1. Differential evolution

DE is one of the most powerful stochastic real-parameter optimization algorithms, which was originally proposed by Refs. [63,64]. Over the past decade, DE becomes very popular because of its simplicity, effectiveness, and robustness.

The main idea of DE is to use spatial difference among the population of vectors to guide the search process toward the optimum solution. In short, almost all DE variants work according to the following steps:

i. *Initialization*: A number of NP points are randomly sampled from the D-dimensional search space to form the initial population.
ii. Repeat steps (iii), (iv) and (v) for each vector $\vec{x}_i \bullet (i \varepsilon \{1, 2, \ldots, NP\})$ of the current population.
iii. *Mutation*: A mutant vector $\vec{v}_i$ is generated for $\vec{x}_i$ according to a specified mutation strategy.
iv. *Repair*: if the mutant vector $\vec{v}_i$ is out of the feasible region, a repair operator is utilized to make it feasible.
v. *Crossover*: A crossover operator is applied to combine the information from $\vec{x}_i$ and $\vec{v}_i$, and form a trial vector $\vec{u}_i$.
vi. *Selection*: The vector with best fitness among $\vec{x}_i$ and $\vec{u}_i$ is transferred to the next generation.
vii. If the termination condition is not met, go to step (ii).

Different extensions of DE can be specified using the general convention *DE/x/y/z*, where DE stands for "Differential Evolution", x represents a string denoting the base vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z stands for the type of crossover being used, i.e. *exponential* or *binomial*, [65].

DE has several advantages that make it a powerful tool for optimization tasks. Specifically, (1) DE has a simple structure and is easy to implement; (2) despite its simplicity, DE exhibits a high performance; (3) the number of control parameters in the canonical DE [63] are very few (i.e. NP, F and CR); (4) due to its low space complexity, DE is suitable for handling large scale problems.

4.2. DynDE

DynDE starts with a predefined number of sub-populations, exploring the entire search space to locate the optimum. Each sub-population is composed of *normal* and *Brownian* individuals. Normal individuals follow the principles of the DE/best/2/bin scheme, where the following

mutation strategy is used:

$$\vec{v}_i = \vec{x}_{best} + F \cdot (\vec{x}_1 + \vec{x}_2 - \vec{x}_3 - \vec{x}_4) \quad (9)$$

here, $\vec{x}_1 \neq \vec{x}_2 \neq \vec{x}_3 \neq \vec{x}_4$ are four randomly selected vectors from the sub-population.

If the generated mutant vector is out of the search boundary, a repair operator is used to bring $\vec{v}_i$ back to the feasible region. Different strategies have been proposed to repair the out of bound individuals. In this article, if the j^{th} component of the i^{th} mutant vector, i.e. v_{ij}, is out of the search region $[lb_j, ub_j]$, then it is repaired as follows:

$$v_{ij} = \begin{cases} lb_j & if\ v_{ij} < lb_j \\ ub_j & if\ v_{ij} > ub_j \end{cases} \quad (10)$$

Afterward, the binomial crossover is applied to form the trial vector $\vec{u}_i$ as follows:

$$u_{ij} = \begin{cases} v_{ij} & if\ rand_{ij}[0,1] \leq CR\ or\ j = jrand \\ x_{ij} & otherwise \end{cases} \quad (11)$$

where $rand_{ij}[0,1]$ is a random number drawn from a uniform distribution between 0 and 1, $CR \in (0,1)$ is the crossover probability which is used to control the approximate number of components that are transferred to the trial vector. *jrand* is a random index in the range $[1, D]$, which ensures the transmission of at least one component from the donor vector to the trial vector.

On the other hand, Brownian individuals are not generated according to the DE/best/2/bin scheme. They are created around the center of the best individual of the sub-population according to the following equation:

$$\vec{x}_{Brownian} = \vec{x}_{best} + \vec{N}(0,\sigma) \quad (12)$$

where $\vec{N}(0,\sigma)$ is a Gaussian distribution with zero mean and standard deviation σ. As shown in Ref. [29], $\sigma = 0.2$ is a suitable value for this parameter.

They also applied an exclusion operator [17] to prevent different sub-populations from settling on the same peak. This operator triggers when the distance between the best individuals of any two sub-populations becomes less than the threshold r_{excl}, and reinitializes the worst performing sub-population in the search space. The parameter r_{excl} is calculated as follows:

$$r_{excl} = \frac{X}{2p^{1/d}} \quad (13)$$

where X is the range of the search space for each dimension, and p is the number of existing peaks in the landscape.

4.3. Proposed DynDE variants

Bearing in mind the inner working of DynDE, it will be easy to understand how our proposals can be included in it. The first DynDE extension, that we have called DynDE + PI, incorporates the scheduling mechanism based on the performance index. The second extension, called DynDE + LA, employs the learning automata scheduling.

In short, the DynDE + PI performs as follows:

i. At the beginning of the run, randomly initialize sub-populations into the search space.
ii. If a change in the environment is detected, re-evaluate all sub-populations.
iii. Calculate the performance index PI for each sub-population according to Eq. (4).
iv. Evolve the sub-population with the highest PI using the principles of DynDE.
v. Apply exclusion among the sub-populations.
vi. Return to step (ii).

The DynDE + LA is summarized as follows:

i. At the beginning of the run, randomly initialize sub-populations into the search space.
ii. If a change in the environment is detected, re-evaluate all sub-populations and reset the action probability vector of $LA_{sub-pop}$.
iii. Select a sub-population using $LA_{sub-pop}$.
iv. Evolve the selected sub-population with the principles of DynDE.
v. Calculate the reinforcement signal β according to Eq. (6).
vi. Updating the probability vector of the automaton $LA_{sub-pop}$ using Algorithm 4.
vii. Apply exclusion among the sub-populations.
viii. Return to step (ii).

5. Experimental study

In order to suitable analyze the impact of our proposal, in this section we described the results obtained from several computational experiments. First, we introduce the technical aspects of the artificial DOP used for testing the algorithms. The performance measures employed for assessing the algorithms are described later. Finally, we present and discuss the results we obtained in the experiments.

5.1. Dynamic test functions

One of the most widely used dynamic test bed in the literature is the *Moving Peaks Benchmark* (MPB) [11]. MPB is a real-valued dynamic environment with a D-dimensional landscape consisting of m peaks, where the height, width and position of each peak are changed slightly every time a change occurs in the environment [66]. Different landscapes can be defined by specifying the shape of the peaks and some other parameters. A typical peak shape is conical which is defined as follows:

$$f(\vec{x}, t) = \max_{i=1,\ldots,m} H_t(i) - W_t(i)\sqrt{\sum_{j=1}^{D} (x_t(j) - X_t(i,j))^2} \quad (14)$$

where $H_t(i)$ and $W_t(i)$ are the height and width of peak i at time t, respectively. The coordinates of each dimension $j \in [1, D]$ of peak i at time t are expressed by $X_t(i,j)$, while D is the problem dimensionality. A typical change of a single peak can be modeled as follows:

$$H_{t+1}(i) = H_t(i) + height_{severity} \cdot \sigma_h \quad (15)$$

$$W_{t+1}(i) = W_t(i) + width_{severity} \cdot \sigma_w \quad (16)$$

$$\vec{X}_{t+1}(i) = \vec{X}_t(i) + \vec{v}_{t+1}(i) \quad (17)$$

$$\vec{v}_{t+1}(i) = \frac{s}{|\vec{r} + \vec{v}_t(i)|}((1-\lambda)\vec{r} + \lambda\vec{v}_t(i)) \quad (18)$$

where σ_h and σ_w are two random Gaussian numbers with zero mean and standard deviation one. Moreover, the shift vector $\vec{v}_{t+1}(i)$ is a combination of a random vector $\vec{r}$, which is created by drawing random numbers in [-0.5, 0.5] for each dimension, and the current shift vector $\vec{v}_t(i)$, and normalized to the length s. Parameter $\lambda \in [0.0,\ 1.0]$ specifies the correlation of each peak's changes to the previous one. This parameter determines the trajectory of changes, where $\lambda = 0$ means that the peaks are shifted in completely random directions and $\lambda = 1$ means that the peaks always follow the same direction, until they hit the boundaries where they bounce off.

In this paper, we consider the so-called *Scenario 2* which is the most widely used configuration of MPB. Unless stated otherwise, the MPB's parameters are set according to the values listed in Table 1 (*default*

values). In addition, to investigate the effect of the environment parameters (i.e. the change severity, change period, number of peaks, number of dimensions, and peak shape) on the performance of the proposed approach, various experiments were carried out with different combinations of other tested values listed in Table 1.

Although MPB has been widely used by many researchers and has served as a valuable tool for evaluating the performance of different methods, it suffers some disadvantages:

i. It provides a limited number of peak functions [58],
ii. It supports random and linearly correlated change types only [58],
iii. It has unequal challenge per change for algorithms when the position of a peak bounces back from the search boundary [67].

Therefore, we have also used the Generalized Dynamic Benchmark Generator (GDBG) [68] to validate the effectiveness of our proposed function evaluation management schemes (Sec. 5.3.8).

5.2. Performance measures

For measuring the efficiency of the tested algorithms, we considered the *offline error,* which is the most well-known metric for dynamic environments [3]. There are two measures in the literature which have been termed as "offline error". The first one is the performance measure suggested by Ref. [69] which is defined as the average of the smallest error found by the algorithm in every time step:

$$E_{off} = \frac{1}{T}\sum_{t=1}^{T} e_t^* \tag{19}$$

where T is the maximum number of FEs so far and e_t^* is the minimum error obtained by the algorithm at the time step t. We assume that a function evaluation made by the algorithm corresponds to a single time step.

The other measure, which was first proposed in Ref. [70] as *accuracy* and later named as *best error before the change* by Ref. [4], is computed as the average of the minimum fitness error achieved by the algorithm at the end of each environment, that is, just before a new change:

$$E_{bbc} = \frac{1}{K}\sum_{k=1}^{K} (h_k - f_k) \tag{20}$$

where f_k is the fitness value of the best solution obtained by the algorithm before the kth change occurs, h_k is the optimum value of the kth environment and K is the total number of environments. In this study, we use both measures to evaluate the performance of the proposed approach. For more detail on the difference between these two measures, interested readers are referred to [71].

5.3. Experiment results

The computational experiments were organized in order to study:

1. different parameter settings for the LA mechanism;
2. The effect of varying the change frequency, shift severity, and peak function;
3. The effect of varying the search space dimension;
4. The effect of varying the number of peaks;
5. The convergence behavior of the algorithms;
6. The behavior of DynDE's extensions against other methods;
7. The comparison with other FE management schemes;
8. The benefits of applying the proposed framework to other multi-population methods;
9. The computational complexity of the proposed framework.

In all experiments, we performed 50 independent runs with different random seeds. Specifically, each run finished when the algorithm reaches 500000 FEs. Unless otherwise stated, the results are reported in terms of average offline error and standard error.

In addition, we used the improvement rate (%Imp), which is computed as follows:

$$\%Imp = 100 \times \left(1 - \frac{e_M}{e_B}\right) \tag{21}$$

where e_M and e_B are the error values of the proposed modified DynDE and the basic DynDE, respectively. These values were averaged over 50 runs.

In order to show the significant statistical difference between our modifications and DynDE, the Wilcoxon rank sum test is used for independent samples at the 0.05 significance level. The outcomes of the Wilcoxon rank-sum test revealing that no significant differences exist between the algorithms, are highlighted with an asterisk symbol.

The parameters setting of the DynDE, the reward and penalty factors for the proposed approach are as shown in Table 2.

In order to draw valid conclusions regarding the effect of including the proposed framework in existing multi-population algorithms, we have employed the same parameter settings reported by the authors of those algorithms.

5.3.1. Parameter sensitivity of the LA method

As was noted before, the mechanism based on the performance index (PI) does not involve any parameters. So, in this experiment we will focused on the learning automata which adds two extra parameters to the DynDE + LA algorithm, i.e. reward and penalty. Fig. 3 illustrates the effect of different combination of values for a and b parameters on the performance of DynDE + LA.

As stated in Section 3.2.2, the values of a and b, govern the linear reinforcement scheme of the LA. So, as can be observed in Fig. 3, a and b has an important effect of the performance of the proposed approach. See for example that the worst result is obtained when $a = 0.25$ and $b = 0.00$. The reason can be attributed to the fact that when $b = 0.00$ and a has large value, i.e. 0.20 and 0.25, the proposed approach quickly converges toward the non-optimal actions. Similarly, it is also possible to note that best result is obtained with $a = 0.15$ and $b = 0.05$.

5.3.2. Effect of varying the change frequency, shift severity and peak function

In this experiment, the effect of the change frequency, change severity, and peak function on the performance of the proposed DynDE variants is examined. The various combinations of (f, s) and peak function resulted in 30 different problem instances. The numerical results of different DynDE variants are reported in Tables 3 and 4.

From Tables 3 and 4, it is clearly observed that our scheduling methods significantly improve the performance of DynDE. As it is inferred form Tables 3 and 4, DynDE + PI and DynDE + LA are able to improve the performance of DynDE by 20.76% and 18.69% on average. The obtained results confirmed that our approach of allocating more FEs in promising areas of the search space gives significant benefits to DynDE.

5.3.3. Effect of increasing the search space dimension

Many real-world problems consist of optimizing a large number of components. Therefore, this experiment aims to investigate the effect of increasing the problem dimensionality in the performance of the proposed approach. The results are illustrated in Figs. 4–9.

As can be seen in Figs. 4–9, the proposed methods are able to effectively improve the performance of DynDE. Specifically, as the dimension of the problem increases, the improvement percentages of the proposed variants over the basic algorithm also increase.

5.3.4. Effects of varying the number of peaks

In this experiment, we studied how the proposed framework is

affected by DOPs with different number of peaks. In this case, we considered the following settings $m \varepsilon \{1, 2, 5, 7, 10, 20, 30, 40, 50, 100, 200\}$. The other environmental parameters are set according to the default MPB settings shown in Table 1. The numerical results of different DynDE variants are reported in Tables 5 and 6.

From Tables 5 and 6, it can be observed that the entire obtained improvement rates (%Imp) are positive. It confirms the benefits of the scheduling framework in improving the performance of DynDE.

5.3.5. Convergence behavior of the proposed approach

This experiment aims to provide a visual view of the convergence behavior of the proposed approach. Fig. 10 and Fig. 11 illustrate a graphical representation of the convergence of different DynDE-variants in two different DOPs generated by MPB.

From Figs. 10 and 11, the first thing that stands out is the poor performance of DynDE during the whole search process. From the curves, it is noticeable that the proposed framework (included in DynDE + PI and DynDE + LA) improves the performance of the basic algorithm. The reason is obvious: DynDE + LA and DynDE + PI spend more FEs around promising areas of the search space.

5.3.6. Comparison with other methods

In this sub-section, we study the performance of the proposed methods in comparison with several approaches from the literature. The involved algorithms were: RPSO [6], mQSO [17], mCPSO [17], SPSO [36], MEPSO [57], RVDEA/Mem [72], MDUMDA [73], and DHPSO [74].

The results in terms of the offline error and standard error are shown in Table 7. Notice that since the results of literature algorithms come from the references given above, certain algorithms like MEPSO and RVDEA/Mem, have missing values. It means that these algorithms were not tested on the corresponding DOPs. In addition, see that we have highlighted the best results in boldface.

From these results, it is easy to observe that the proposed methods (e.g. DynDE + LA and DynDE + PI) show an overall superiority over the other methods. Only in two cases literature algorithms are better than our methods. For example, when the environment is unimodal (1 peak), MEPSO is the best-performing algorithm. However, a different pattern arises when the number of peaks is increased: our algorithms reach lower error than the rest of the group.

5.3.7. Comparison with other function evaluation management schemes

The objective of this experiment is to investigate the effectiveness of the proposed function evaluation management schemes. For this purpose, the CPE algorithm was re-implemented as described in Ref. [53]. The main reason is because CPE implements an alternative mechanism to the proposed strategies. So, it will be a good reference for measuring the impact of our proposal.

Table 8 indicates how different function evaluation management schemes can enhance the performance of DynDE in DOPs with varying number of peaks.

Regarding Table 8, it can be observed that the proposed schemes have a better capability in managing the FEs.

5.3.8. Applicability of the proposed framework

In order to further investigate the applicability of the proposed framework we incorporated it into two existing algorithms for DOPs: jDE [75] and mSQDE-i [76]. We have selected them since: 1) both algorithms resulted very effective in challenging DOPs, and 2) their corresponding source code is available thanks to their authors. In that sense, with aims of improving our comparison analysis and to avoid *reproducibility* issues regarding the results of these algorithms, we relied only on the results obtained in our own simulations. Besides, our goal is not to beat any state-of-art algorithm. In this section, we are more interested on studying how our proposals impact the algorithm performance when solving DOPs. Obtaining more competitive algorithms using our approaches is out of the scope of this work, but it will be subject of our near future research. For instance, algorithms like the proposed in Refs. [77,78] are interesting optimizers to include in our future study. For the sake of simplicity, in this experiment we have considered only the learning automata mechanism with $a = 0.15$ and $b = 0.05$. Thus, the extended versions of jDE and mSQDE-i will be denoted as jDE + LA and mSQDE-i + LA. Several problem instances from the GDBG were employed in this study. As it was mentioned before, the GDBG provides dynamic scenarios more difficult to solve than those from MPB. This is because they have been derived from the combination of challenging change types and peaks function. Following the parameter settings proposed in Ref. [67] the considered change types were: *small step* (T1), *large step* (T2), *random* (T3), *recurrent* (T4), *chaotic* (T5), *recurrent with noise* (T6), *random change with changed dimension* (T7). Regarding the peaks functions the following were studied: *Rotation peak function* with 10 peaks (F1-10), *Rotation peak function* with 50 peaks (F1-50), *Composition of Sphere's function* (F2), *Composition of Rastrigin's function* (F3), *Composition of Griewank's function* (F4), *Composition of Ackley's function* (F5), and *Hybrid Composition function* (F6). For more details about the parameter settings used in the GDBG, the reader is referred to [67] and [79].

The performance measure used to assess the algorithms was the average error of the best solution found by the algorithm before the change [79]. The numerical results in terms of the mean and standard deviation of such a performance measure are shown in Table 9.

From these results, we observe that our extensions jDE + LA and mSQDE-i + LA are much better than their corresponding basic versions (e.g. jDE and mSQDE-i). This assertion can be easily verified in Fig. 12, where the percentage of improvement of our extensions is represented through color maps.

Finally, with aim to formalize the above conclusions we performed a non-parametric statistical analysis. Based on the suggestions made in Ref. [80], we applied the Friedman's test to verify whether significant differences exist at group level, that is, among all algorithms. We confirmed such differences since the obtained p-value was $7.42 \times 10^{-11} < 0.05$ (lower than the significance level). Another important result provided by the Friedman's test is the average rank of the algorithms, where the lower is the rank the better the algorithm is. Fig. 13 shows this information along with the pairwise comparisons between our extensions and the basic algorithms. These comparisons were carried out using the Holm's test ($\alpha = 0.05$).

As we expected from the previous analysis, the algorithms including our proposal are significantly better than their basic versions. Undoubtedly, this is an important evidence about the benefits of the proposed framework, which really improves the performance of existing multi-population algorithms for DOPs.

5.3.9. Computational complexity of the proposed framework

One important question arising when developing new mechanisms for enhancing the performance of a certain algorithm, is how they affect the computational complexity of the algorithm. In what follows, we analyze the involved computational complexity of our proposals, showing that they do not introduce any significant, computational overhead to the base algorithm.

First at all, we will define two types of iterative processes, which are common in most multi-population algorithms for evolutionary dynamic optimization:

Multi-population cycle (MPC): It is a cycle over the set of all populations, which has a size of M. Several MPC can be used for different task. For example, to evolve all subpopulations, to detect environment change, to update the memories.

Population cycle (PC): It loops over all individuals of a given population (with size N). Like the MPC, several PC can be employed for different task inside the algorithm. For instance, the evolutionary process for each individual according to inner algorithm.

From Fig. 1, we may observe that the proposed framework has two main operations: 1) Select a sub-population based on a feedback

parameter, and 2) run the sub-population. While operation 2 depends on the inner optimizer (e.g. PSO, DE, etc.), the first one depends on how the computation of such feedback parameter is performed for each subpopulation. It is clear that our analysis must be oriented to characterize the first operation, since running the subpopulation depends on the selected inner optimizer.

Let consider the PI approach. In this case, every subpopulation requires the computation of a PI. So, we need to iterate over the set of subpopulations, that is, using a MPC. The PI depends on the number of individuals. In practical terms, we need to define a variable for counting those individuals with fitness improvements. Remember that an improvement occurred when the current fitness is better than the previous one. So, we need to rely on a PC for this counting operation. Suppose that computing each individual improvement has a cost C_1, while computing the PI for each subpopulation costs C_2. It is easy to verify that the cost of computing the PIs for all subpopulations is:

$$C_{PI} = M \cdot (N \cdot C_1 + C_2) = M \cdot N \cdot C_1 + M \cdot C_2 \tag{22}$$

In other words, the PI approach involves a linear complexity $(O(n))$ depending on the number of the candidate solutions the algorithm has, which is given by $n = M \cdot N$. In general, it shows almost the same computational complexity than existing literature algorithms like mQSO or DynDE. These algorithms require a nested combination of MPC and PC for performing the evolution process. However, while mQSO and DynDE use this nested combination for evolving the subpopulations, our approach uses it for computing the performance index.

Regarding the LA approach, a similar analysis shows that it has even a lower complexity compared to the PI approach. According to Algorithm 3, LA approach involves four major steps: 1) Selecting a subpopulation using the probability vector; 2) evolve the selected subpopulation, 3) computing the β parameter, and 4) update the probability vector. Again, the subpopulation evolution (step 2) will be excluded from our analysis.

For executing step 1, we require a MPC over the probability vector, which has a size of M. In the case of step 3 and step 4, it is clear from the explanation given in Sec. 3.2, that they are single operations related to the selected subpopulation. Thus, these steps do not rely on any loop. Suppose that each comparison involved in step 1 (for finding the maximum probability) has a cost C_1; while steps 3 and 4 has the costs C_2 and C_3, respectively. Then, the LA approach has a linear complexity, given as:

$$C_{LA} = M \cdot C_1 + C_2 + C_3 \tag{23}$$

This is indeed a lower complexity than the PI approach (see Eq. (22)), since no PC is required. In summary, our instantiations of the proposed framework introduce only a marginal complexity to the base algorithm.

6. Conclusion and future works

A framework for improving the performance of multi-population algorithms in dynamic environments was presented in this paper. The main motivation behind our proposal is to enhance the multi-population algorithms by allocating more function evaluations to the best performing sub-populations. For this purpose, the proposed framework involves two adaptations based on the idea of scheduling:

- the first one aims to allocate more FEs to the best performing subpopulation taking into account a performance index,
- the second adaptation uses a learning automata approach to determine the sup-population being executed, based on the feedback received from the search progress of the sub-populations.

In order to validate the effectiveness of the proposed framework, it has been incorporated into DynDE, a well-known algorithm for solving DOPs. A wide range of experiments were carried out to investigate the effectiveness of the proposed methods on MPB and GDBG. The experimental results confirm that the proposed framework has a significant impact on the performance of multi-population approaches in dynamic environments.

Several research directions can be planned from the obtained results. First, the proposed scheduling methods can be used to improve the performance of other multi-population methods. Second, other scheduling mechanisms will be designed to enhance the effectiveness of the multi-population algorithms.

Finally, it is important to highlight that the source code of the proposed approach is freely available for research purposes as a GitHub repository: https://github.com/javidankazemi.

Acknowledgements

The authors would like to thank the anonymous reviewers for their suggestions and valuable comments, which have greatly helped to improve the quality of the paper. The authors are very grateful to Professor J. Brest for letting us use the source code of jDE.

References

[1] V. Pillac, M. Gendreau, C. Guéret, A.L. Medaglia, A review of dynamic vehicle routing problems, Eur. J. Oper. Res. 225 (2013) 1–11.
[2] S. Yang, H. Cheng, F. Wang, Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks, IEEE Trans. Syst. Man Cybern. C Appl. Rev. 40 (2010) 52–63.
[3] C. Cruz, J.R. González, D.A. Pelta, Optimization in dynamic environments: a survey on problems, methods and measures, Soft Comput. 15 (2011) 1427–1448.
[4] T.T. Nguyen, S. Yang, J. Branke, Evolutionary dynamic optimization: a survey of the state of the art, Swarm Evol. Comput. 6 (2012) 1–24.
[5] H.G. Cobb, An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-dependent Nonstationary Environments, DTIC Document, NAVAL RESEARCH LAB, Washington USA, 1990.
[6] X. Hu, R.C. Eberhart, Adaptive particle swarm optimization: detection and response to dynamic systems, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2002, pp. 1666–1670.
[7] F. Vavak, K.A. Jukes, T.C. Fogarty, Performance of a genetic algorithm with variable local search range relative to frequency of the environmental changes, Genet. Progr. (1998) 22–25.
[8] F. Vavak, K. Jukes, T.C. Fogarty, Learning the local search range for genetic optimisation in nonstationary environments, in: Evolutionary Computation, 1997., IEEE International Conference on, IEEE, 1997, pp. 355–360.
[9] S. Janson, M. Middendorf, A hierarchical particle swarm optimizer for noisy and dynamic environments, Genet. Program. Evolvable Mach. 7 (2006) 329–354.
[10] N. Mori, H. Kita, Y. Nishikawa, Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm, Trans. Inst. Syst. Contr. Inf. Eng. 14 (2001) 33–41.
[11] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, in: P.J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, A. Zalzala (Eds.), Proceedings of the Congress on Evolutionary Computation, IEEE Press, Mayflower Hotel, Washington D.C., USA, 1999, pp. 1875–1882.
[12] H. Wang, D. Wang, S. Yang, Triggered memory-based swarm optimization in dynamic environments, in: Applications of Evolutionary Computing, Springer, Berlin, Heidelberg, 2007, pp. 637–646.
[13] I. Hatzakis, D. Wallace, Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, Seattle, Washington, USA, 2006, pp. 1201–1208.
[14] C. Rossi, M. Abderrahim, J.C. Díaz, Tracking moving optima using kalman-based predictions, Evol. Comput. 16 (2008) 1–30.
[15] J.J. Grefenstette, Evolvability in dynamic fitness landscapes: a genetic algorithm approach, in: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 3, IEEE, 1999, pp. 1–2038.
[16] P. Novoa-Hernández, C.C. Corona, D.A. Pelta, Self-adaptation in dynamic environments - a survey and open issues, Int. J. Bio-Inspired Comput. 8 (2016) 1–13.
[17] T.M. Blackwell, J. Branke, Multiswarms, exclusion, and anti-convergence in dynamic environments, IEEE Trans. Evol. Comput. 10 (2006) 459–472.
[18] A.B. Hashemi, M.R. Meybodi, A multi-role cellular PSO for dynamic environments, in: Proceedings of the 14th International CSI Computer Conference, IEEE, 2009a, pp. 412–417 (a).
[19] A.B. Hashemi, M.R. Meybodi, Cellular PSO: a PSO for dynamic environments, in: The 4th International Symposium on Intelligence Computation and Applications, Springer, Berlin, Heidelberg, 2009b, pp. 422–433 (b).
[20] M. Kamosi, A.B. Hashemi, M.R. Meybodi, A hibernating multi-swarm optimization algorithm for dynamic environments, in: Proceedings of the Second World Congress on Nature and Biologically Inspired Computing, IEEE, 2010a, pp. 363–369.
[21] M. Kamosi, A.B. Hashemi, M.R. Meybodi, A new particle swarm optimization algorithm for dynamic environments, in: Proceedings of the First International

Conference on Swarm, Evolutionary, and Memetic Computing, Springer, Berlin, Heidelberg, 2010b, pp. 129–138.
[22] S. Nabizadeh, A. Rezvanian, M.R. Meybodi, A multi-swarm cellular PSO based on clonal selection algorithm in dynamic environments, in: Proceedings of the International Conference on Informatics, Electronics & Vision, IEEE, 2012, pp. 482–486.
[23] V. Noroozi, A.B. Hashemi, M.R. Meybodi, Alpinist CellularDE: a cellular based optimization algorithm for dynamic environments, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, Philadelphia, Pennsylvania, USA, 2012, pp. 1519–1520.
[24] V. Noroozi, A.B. Hashemi, M.R. Meybodi, CellularDE: a cellular based differential evolution for dynamic optimization problems, in: Proceedings of the 10th International Conference on Adaptive and Natural Computing Algorithms, Springer, Berlin, Heidelberg, 2011, pp. 340–349.
[25] A. Sharifi, V. Noroozi, M. Bashiri, A.B. Hashemi, M.R. Meybodi, Two phased cellular PSO: a new collaborative cellular algorithm for optimization in dynamic environments, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2012, pp. 1–8.
[26] S. Yang, C. Li, Fast multi-swarm optimization for dynamic optimization problems, in: Proceeding of the Fourth International Conference on Natural Computation, IEEE, 2008, pp. 624–628.
[27] S. Yang, C. Li, A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments, IEEE Trans. Evol. Comput. 14 (2010) 959–974.
[28] T. Blackwell, J. Branke, Multi-swarm optimization in dynamic environments, in: Workshops on Applications of Evolutionary Computation, Springer, 2004, pp. 489–500.
[29] R. Mendes, A.S. Mohais, DynDE: a differential evolution for dynamic optimization problems, in: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, 2005, pp. 2808–2815.
[30] R.I. Lung, D. Dumitrescu, A collaborative model for tracking optima in dynamic environments, in: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, 2007, pp. 564–567.
[31] R.I. Lung, D. Dumitrescu, Evolutionary swarm cooperative optimization in dynamic environments, Nat. Comput. 9 (2009) 83–94.
[32] J.K. Kordestani, A. Rezvanian, M.R. Meybodi, CDEPSO: a bi-population hybrid approach for dynamic optimization problems, Appl. Intell. 40 (2014) 682–694.
[33] J. Branke, T. Kaussler, C. Smidt, H. Schmeck, A multi-population approach to dynamic optimization problems, in: I.C. Parmee (Ed.), Evolutionary Design and Manufacture, Springer, London, 2000, pp. 299–307.
[34] D. Yazdani, B. Nasiri, A. Sepas-Moghaddam, M.R. Meybodi, A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization, Appl. Soft Comput. 13 (2013) 2144–2158.
[35] A. Sharifi, J.K. Kordestani, M. Mahdaviani, M.R. Meybodi, A novel hybrid adaptive collaborative approach based on particle swarm optimization and local search for dynamic optimization problems, Appl. Soft Comput. 32 (2015) 432–448.
[36] D. Parrott, X. Li, Locating and tracking multiple dynamic optima by a particle swarm model using speciation, IEEE Trans. Evol. Comput. 10 (2006) 440–458.
[37] S. Bird, X. Li, Using regression to improve local convergence, in: 2007 IEEE Congress on Evolutionary Computation, 2007, pp. 592–599.
[38] C. Li, S. Yang, A general framework of multipopulation methods with clustering in undetectable dynamic environments, IEEE Trans. Evol. Comput. 16 (2012) 556–577.
[39] A. Nickabadi, M.M. Ebadzadeh, R. Safabakhsh, A competitive clustering particle swarm optimizer for dynamic optimization problems, Swarm Intell. 6 (2012) 177–206.
[40] U. Halder, S. Das, D. Maity, A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments, IEEE Trans. Cybern. 43 (2013) 881–897.
[41] T. Blackwell, Particle swarm optimization in dynamic environments, in: Evolutionary Computation in Dynamic and Uncertain Environments, Springer, Berlin, Heidelberg, 2007, pp. 29–49.
[42] D. Yazdani, M.R. Akbarzadeh-Totonchi, B. Nasiri, M.R. Meybodi, A new artificial fish swarm algorithm for dynamic optimization problems, in: IEEE Congress on Evolutionary Computation, 2012, pp. 1–87.
[43] M.C. du Plessis, A.P. Engelbrecht, Differential evolution for dynamic environments with unknown numbers of optima, J. Global Optim. 55 (2013) 73–99.
[44] N. Fouladgar, S. Lotfi, A novel approach for optimization in dynamic environments based on modified cuckoo search algorithm, Soft Comput. (2015) 1–15.
[45] C. Li, T.T. Nguyen, M. Yang, M. Mavrovouniotis, S. Yang, An adaptive multipopulation framework for locating and tracking multiple optima, IEEE Trans. Evol. Comput. 20 (2016) 590–605.
[46] I.G. del Amo, D.A. Pelta, J.R. González, P. Novoa, An analysis of particle properties on a multi-swarm pso for dynamic optimization problems, in: Conference of the Spanish Association for Artificial Intelligence, Springer, 2009, pp. 32–41.
[47] M.C. du Plessis, A.P. Engelbrecht, A. Calitz, Self-adapting the brownian radius in a differential evolution algorithm for dynamic environments, in: Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII, ACM, 2015, pp. 114–128.
[48] K. Trojanowski, Adaptive non-uniform distribution of quantum particles in mqso, in: Asia-Pacific Conference on Simulated Evolution and Learning, Springer, 2008, pp. 91–100.
[49] K. Trojanowski, Properties of quantum particles in multi-swarms for dynamic optimization, Fundam. Inf. 95 (2009) 349–380.
[50] A.M. Turky, S. Abdullah, A multi-population harmony search algorithm with external archive for dynamic optimization problems, Inf. Sci. 272 (2014) 84–95.
[51] P. Novoa, D.A. Pelta, C. Cruz, I.G. del Amo, Controlling particle trajectories in a multi-swarm approach for dynamic optimization problems, in: International Work-conference on the Interplay between Natural and Artificial Computation, Springer, 2009, pp. 285–294.
[52] A. Sepas-Moghaddam, A. Arabshahi, D. Yazdani, M.M. Dehshibi, A novel hybrid algorithm for optimization in multimodal Dynamic environments, in: Hybrid Intelligent Systems (HIS), 2012 12th International Conference on, IEEE, 2012, pp. 143–148.
[53] M.C. du Plessis, A.P. Engelbrecht, Using Competitive Population Evaluation in a differential evolution algorithm for dynamic environments, Eur. J. Oper. Res. 218 (2012) 7–20.
[54] X. Zuo, L. Xiao, A DE and PSO based hybrid algorithm for dynamic optimization problems, Soft Comput. 18 (2014) 1405–1424.
[55] P. Novoa-Hernández, C.C. Corona, D.A. Pelta, Efficient multi-swarm PSO algorithms for dynamic environments, Memetic Comput. 3 (2011) 163–174.
[56] P. Novoa-Hernández, D.A. Pelta, C.C. Corona, Improvement strategies for multi-swarm PSO in dynamic environments, in: J. González, D. Pelta, C. Cruz, G. Terrazas, N. Krasnogor (Eds.), Studies in Computational Intelligence, Springer, Berlin/Heidelberg, 2010, pp. 371–383.
[57] W. Du, B. Li, Multi-strategy ensemble particle swarm optimization for dynamic optimization, Inf. Sci. 178 (2008) 3096–3109.
[58] M.C. du Plessis, Adaptive Multi-population Differential Evolution for Dynamic Environments, University of Pretoria, 2012.
[59] M.C. du Plessis, Improved differential evolution for dynamic optimization problems, in: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), 2008, pp. 229–234.
[60] A. Nickabadi, M.M. Ebadzadeh, R. Safabakhsh, A novel particle swarm optimization algorithm with adaptive inertia weight, Appl. Soft Comput. 11 (2011) 3658–3670.
[61] K.S. Narendra, M.A.L. Thathachar, Learning automata - a survey, IEEE Trans. Syst. Man. Cybern. 4 (1974) 323–334.
[62] M.A.L. Thathachar, P.S. Sastry, Varieties of learning automata: an overview, IEEE Trans. Syst. Man, Cybern. Part B 32 (2002) 711–722.
[63] R. Storn, K. Price, Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (1997) 341–359.
[64] R. Storn, K. Price, Differential Evolution-a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces, International Computer Science Institute, Berkeley, 1995.
[65] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (2011) 4–31.
[66] J. Branke, Evolutionary Optimization in Dynamic Environments., Boston, MA, 2002.
[67] C. Li, S. Yang, A generalized approach to construct benchmark problems for dynamic optimization, in: Asia-Pacific Conference on Simulated Evolution and Learning, Springer, 2008, pp. 391–400.
[68] C. Li, S. Yang, T.T. Nguyen, E.L. Yu, X. Yao, Y. Jin, H.G. Beyer, P.N. Suganthan, Benchmark Generator for CEC 2009 Competition on Dynamic Optimization, University of Leicester, University of Birmingham, Nanyang Technological University, 2008b.
[69] J. Branke, H. Schmeck, Designing evolutionary algorithms for dynamic optimization problems, in: Advances in Evolutionary Computing: Theory and Applications, Springer, Berlin, Heidelberg, 2003, pp. 239–262.
[70] K. Trojanowski, Z. Michalewicz, Searching for optima in non-stationary environments, in: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, 1999, p. 2348.
[71] A.E. Ranginkaman, J. Kazemi Kordestani, A. Rezvanian, M.R. Meybodi, A note on the paper 'A multi-population harmony search algorithm with external archive for dynamic optimization problems' by Turky and Abdullah, Inf. Sci. 288 (2014) 12–14.
[72] Y.G. Woldesenbet, G.G. Yen, Dynamic evolutionary algorithm with variable relocation, IEEE Trans. Evol. Comput. 13 (2009) 500–513.
[73] V. Wu, Y. Wang, X. Liu, J. Ye, Multi-population and diffusion UMDA for dynamic multimodal problems, J. Syst. Eng. Electron. 21 (2010) 777–783.
[74] J. Karimi, H. Nobahari, S.H. Pourtakdoust, A new hybrid approach for dynamic continuous optimization problems, Appl. Soft Comput. 12 (2012) 1158–1167.
[75] J. Brest, A. Zamuda, B. Bošković, M.S. Maučec, V. Žumer, Dynamic optimization using self-adaptive differential evolution, in: CEC'09: Proceedings of the Eleventh Conference on Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA, 2009, pp. 415–422.
[76] P. Novoa-Hernández, C.C. Corona, D.A. Pelta, Self-adaptive, multipopulation differential evolution in dynamic environments, Soft Comput. 17 (2013) 1861–1881.
[77] J. Brest, P. Korošec, J. Šilc, A. Zamuda, B. Bošković, M.S. Maučec, Differential evolution and differential ant-stigmergy on dynamic optimisation problems, Int. J. Syst. Sci. 44 (2013) 663–679.
[78] S. Das, A. Mandal, R. Mukherjee, An adaptive differential evolution algorithm for global optimization in dynamic environments, IEEE Trans. Cybern. 44 (2014) 966–978.
[79] C. Li, S. Yang, T.T. Nguyen, E.L. Yu, X. Yao, Y. Jin, H.-G. Beyer, P.N. Suganthan, Benchmark Generator for CEC'2009 Competition on Dynamic Optimization, University of Leicester, University of Birmingham, Nanyang Technological University, 2008a.
[80] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization, J. Heuristics 15 (2009) 617–644.