A multi-population differential evolution algorithm based on cellular learning automata and evolutionary context information for optimization in dynamic environments

Reza Vafashoar, Mohammad Reza Meybodi

Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran.

Tel: +98-21-64545120; Fax: +98-21-66495521; e-mail: (vafashoar, mmeybodi)@aut.ac.ir

Abstract

This paper presents a multi-population differential evolution algorithm to address dynamic optimization problems. In the proposed approach, a cellular learning automaton adjusts the behavior of each subpopulation by adaptively controlling its updating schemes. As the environment changes over time, an evolving population may go through a quite number of state transitions. Each state demands specific characteristics from an optimizer; hence, an adapted evolutionary scheme for one state may be unsuitable for the upcoming ones. Additionally, a learning approach may have limited time to adapt to a newly encountered state due to the frequentness of the environmental changes. Hence, it is infeasible for a dynamic optimizer to unlearn its existing beliefs to accommodate the practices required to embrace newly encountered states. In order to address this issue, we introduce a context dependent learning approach, which can adapt the behavior of each subpopulation according to the contexts of its different states. The performance of the proposed approach is compared with several state-of-the-art dynamic optimizers over the GDBG benchmark set. Comparison results indicate that the proposed method can achieve statistically superior performances on a wide range of tested instances.

Keywords: Differential evolution, Dynamic optimization problems, Hyper-heuristics, Cellular learning automata, learning automata.

1. Introduction

Among various optimization approaches, nature-inspired methods such as evolutionary algorithms [1-2] have attracted considerable interest in recent years. Nature-inspired optimization methods have proven

to be quite successful in solving various complex optimization problems. Many of these problems often contain several uncertain and dynamic factors. Typical examples include traffic-aware routing in wireless networks and air traffic scheduling. Such problems are commonly referred to as dynamic optimization problems. Due to the time-varying characteristics of a dynamic optimization problem, its fitness landscape changes with time. Accordingly, the locations of its optima may change with time as well.

In a dynamic environment, convergence tendency imposes serious challenges on a conventional nature-inspired optimization method. The loss of diversity, due to convergence, prevents a population-based optimization algorithm from reacting and adapting to new environmental conditions. To address this limitation, population-based optimization approaches often utilize multiple populations and archival memories, along with techniques such as random immigrants, hypermutation, and adaptation [3].

Various scientific and engineering problems can be modeled as optimization tasks. Consequently, much effort has been put into the development of versatile nature-inspired optimization methods. It is well accepted that a general-purpose universal optimization algorithm is impossible: no strategy or scheme can outperform the others on all possible optimization problems [4]. Additionally, a single evolutionary scheme or operator may always follow similar trajectories. Therefore, it would be better to use diverse evolutionary operators to increase the chance of finding optima. Moreover, the parameter and strategy configuration of an optimization algorithm can significantly affect its performance. On a specific problem, proper configurations can result in high-quality solutions and high convergence speeds.

Considering the discussed issues, the utilization of multiple strategies or techniques along with a suitable adaptation mechanism can significantly enhance a nature-inspired optimization method. Indeed, some recently developed nature-inspired optimization methods incorporate a collection of strategies in some of their algorithmic elements such as mutation, crossover, or neighborhood structure. Such collections are referred to as ensemble components in some literature such as [4]. Approaches such as reinforcement learning [5], neural networks [6], and surrogate model [7] have been successfully utilized for the adaptive selection of proper strategies in nature-inspired optimization algorithms. Hyper-heuristics are also based on very similar ideas (i.e. the utilization of multiple strategies or techniques in one

algorithm) [8-9]. Hyper-heuristic is an approach intended to make a meta-heuristic less reliant on the problem to be solved. In this approach, an upper-level heuristic controls the application of some lower-level heuristics. On a specific problem, the high-level heuristic selects a low-level heuristic to be applied at any time instant. These low-level heuristics are problem dependent, and each one has its strengths and weaknesses. However, the high-level method has no knowledge of the solutions domain or the function of the low-level heuristics and only has access to the low-level heuristics domain.

One of the most effective and popular evolutionary algorithms is differential evolution (DE) [10]. The power of this method lies mainly in its incorporated mutation and crossover schemes. Numerous studies have been conducted on DE with respect to the novel mutation scheme design. It is demonstrated that a collection of well-designed mutation schemes along with a proper adaptation mechanism could guide a DE based method towards the more favorable trajectories [10]. However, the common adaptation mechanisms designed for these DE methods would be unsuitable for dynamic optimization problems. The dynamicity of the environment imposes several challenges on an adaptation mechanism. An appropriate adaptation mechanism should consider the state of the population, specifically the fitness and positional information of its individuals. Many introduced dynamic optimizers have some exclusion and re-initialization steps. These steps can drastically alter the positional information of individuals. Additionally, the dynamicity of the environment may result in severe fitness changes in the population elements. As these changes may occur with a high frequency, an adaptive mechanism has limited time to adjust itself with the encountered changes.

This paper introduces a multi-population differential evolution algorithm for optimization in dynamic environments. The proposed method uses a collection of three mutation schemes, each inducing different behavior on a subpopulation. This set of mutation schemes consists of DE/rand/1, which is commonly used in DE literature, and two mutation schemes specifically designed to promote the exploitation and exploration characteristics of the subpopulations. An adaptive mechanism based on cellular learning automaton (CLA) [11] is utilized to control the application of each mutation scheme. To address the mentioned challenges imposed by the environmental dynamics, we propose a context-aware learning

mechanism. The proposed method progressively learns the effectiveness of each scheme according to the context of the state of the subpopulations. In each state, a subpopulation exploits its previously learned knowledge for the state and improves this knowledge until a state transition occurs. This approach allows the adaptation mechanism to adjust each subpopulation to its residing states in a short time.

The rest of the paper is organized as follows: Section 2 briefly presents the related works concerning nature-inspired optimization algorithms. Section 3 presents some background on differential evolution algorithm, learning automata, and cellular learning automata. In section 4, we describe the proposed algorithm. Experimental results are discussed in Section 5. Finally, some brief concluding comments are provided in Section 6.

2. Related works

In this section, we briefly review some existing works related to CLA-based nature-inspired methods and dynamic optimization algorithms. All previous studies on the incorporation of CLA concepts in nature-inspired optimization algorithms are carried out in the domain of static optimization. Nonetheless, the concepts adopted in these works can be utilized in dynamic optimization as well. CLA-based nature-inspired optimization methods can be roughly classified into two categories. In some of these methods, the population or its elements are modeled using a CLA [12-14]. Each CLA cell resides a group of learning automata in CLA-DE [14], where each group is a probabilistic model of promising candidate solutions. This probabilistic model initially generates candidate solutions uniformly within the solution space. However, through repeated updating steps, the model learns to generate high accuracy near-optimum solutions. In the second category, CLA embeds the elements of population-based optimization algorithms and controls their evolutionary procedure [15-17]. This scheme has been utilized for adjusting the neighborhood structure of particles in particle swarm optimization (PSO) [16] and the updating distributions in bare bones PSO [15, 17].

There has been a growing interest in using nature-inspired or population-based methods for solving dynamic optimization problems. These methods deal with a population of candidate solutions, which can be spread over the entire solution space. Accordingly, they can efficiently keep track of multiple optima

or changing optima in a dynamic environment. Approaches like PSO [18], DE [19], clonal selection algorithm [20], artificial bee colony [21], ant colony optimization [22], firefly algorithm [23], and memetic algorithm [24] have been successfully employed for solving dynamic optimization problems. Several interesting schemes have been proposed to accommodate nature-inspired optimization methods with dynamic environments. In [3], these schemes are classified as change detection, change handling, diversity maintenance, external memory, prediction, self-adaptation, and multi-population. Most dynamic optimizers have algorithmic components for at least one of the mentioned schemes. For instance, the dynamic optimizer presented in [25] uses quantum particles as well as multiple populations to maintain diversity. It also uses two additional components, change detection and exclusion, to deal with environmental changes and track multiple optima. Using multiple populations is very common in dynamic optimizers. Usually, different populations are maintained in disjoint sub-areas of the fitness landscape, which can bring several benefits to a dynamic optimizer. First, the overall population diversity can be easily maintained at a high level by using multiple populations. Second, it helps in locating and tracking multiple optima. In what follows, we briefly review some multi-population dynamic optimizers.

Multi-population optimization methods can be categorized from different perspectives. A comprehensive survey of these methods and their classifications is provided in [26]. We can further classify multi-population dynamic optimization approaches according to their number of used populations (subpopulations) into three categories [27]: approaches that use a fixed number of populations, approaches that use a variable number of populations, and approaches that use an adaptive number of populations. jDE, as one the most successful dynamic optimizers, uses a fixed number of subpopulations [28]. It uses the "DE/rand/1/bin" scheme with a self-adapting parameter control mechanism to update its individuals. In contrast to jDE, DynPopDE uses an adaptive number of subpopulations according to the characteristics of its given problem [29]. The algorithm generates a new subpopulation of random individuals whenever it reaches a stagnation point. Additionally, DynPopDE removes its redundant subpopulations during the search procedure. Usually, different subpopulations of a multi-population approach have similar roles and are updated using a similar procedure [30-31]. However, in some of the

proposed methods, each subpopulation may be designated for a specific task [32-34]. In the following, we will refer to the former methods as homogenous models and the latter ones as heterogeneous models. These terms can be defined based on other criteria such as subpopulation sizes or the granted number of fitness evaluations for each subpopulation; however, we do not consider these criteria here.

Homogenous models: cluster-based dynamic differential evolution with external archive (CDDE_Ar) uses the k-means algorithm to partition its individuals into several clusters [35]. The algorithm starts with an initial number of clusters; however, it may increase or decrease the number of its clusters, during the search procedure, according to their performances. All individuals are updated in the same manner using the DE/best/1/bin scheme. It should be noted that different clusters may have different numbers of individuals (considering the utilized clustering algorithm). Cao et al. have proposed a multi-swarm optimization algorithm, which uses a collaboration-based strategy to update each swarm [36]. Each swarm has also an associated external archive and stores its best-visited positions in this archival memory. The archival memory of each swarm is used for its re-initialization upon detecting a change in the environment. Similar approaches have been introduced by Turky and Abdullah based on artificial bee colony, harmony search, and electromagnetic algorithm [21, 37-38]. In [37], the population is partitioned into several subpopulations, and these subpopulations are evolved using the electromagnetic algorithm. Upon detection of an environmental change, the algorithm reinitializes the subpopulations using a memory-based approach. Additionally, random immigrants are utilized to increase the diversity of subpopulations. Another algorithm following a similar paradigm was presented by Das et al. [31]. This algorithm evolves individuals according to three specifically designed strategies: neighborhood-driven double mutation, Brownian, and Quantum. Furthermore, aging and exclusion mechanisms were developed to spread subpopulations evenly over the entire solution space and to prevent stagnation.

Heterogeneous models: usually, one or more subpopulations are responsible for exploring the solution space and discovering new promising regions. Herein, these subpopulations will be called explorer subpopulations. Other subpopulations are responsible for exploiting the promising regions found by the explorer subpopulations. In the multi-swarm algorithm presented in [39], an explorer swarm is

responsible for the discovery of new peaks in the environment. Upon the discovery of a new peak, a tracker swarm is activated and assigned to the detected peak. Each tracker swarm has the responsibility of exploiting and tracking its assigned peak. Similarly, the algorithm proposed in [23] starts with a randomly initialized explorer subpopulation. Whenever the explorer finds a promising region; it is cloned, and the generated clone exploits the discovered region. Both explorer and tracker subpopulations use the firefly algorithm as their search procedures. However, the algorithm uses two different attractiveness factors for the tracker and explorer subpopulations to fine-tune their search procedures based on their assigned tasks.

3. Preliminaries

In this section, we briefly review the fundamental concepts of learning automata, cellular learning automata, and differential evolution algorithms.

3.1. Learning automaton

A variable structure learning automaton (LA) can be represented by a sextuple like $\{\Phi, \alpha, \beta, A, G, P\}$ [40], where Φ is the set of internal states; α is the set of outputs or actions of the LA; β is the set of inputs or environmental responses; A is the learning algorithm; $G(.):\Phi\rightarrow\alpha$ is the function that maps the current state into the current output; and P is the probability vector that determines the selection probability of a state at each step. Usually, there is a one to one correspondence between Φ and α; as a result, the two terms can be used interchangeably.

The objective of an LA is to identify the best action which maximizes the expected received payoff from the environment [41]. To attain this goal, the LA selects an action according to its action probability distribution and applies this action to the environment. The environment measures the favorability of the received action, and responds the LA with a noisy reinforcement signal β. The LA uses this response to adjust its internal action probabilities via its learning algorithm. We assume that the environmental response to a selected action like a_i at step k is $\beta\in\{0,1\}$, where 0 and 1 are used to represent pleasant and unpleasant responses, respectively. The internal action probabilities can be updated according to Eq. (1) when the response is pleasant and according to Eq. (2) when it is not.

$$P_j(k+1) = \begin{cases} P_j(k) + \lambda(1 - P_j(k)) & \text{if selected action} = a_j \\ P_j(k)(1-\lambda) & \text{if selected action} \neq a_j \end{cases}, \ \forall j = 1, ..., r \ . \quad (1)$$

$$P_j(k+1) = \begin{cases} P_j(k)(1-\rho) & \text{if selected action} = a_j \\ \frac{\rho}{r-1} + P_j(k)(1-\rho) & \text{if selected action} \neq a_j \end{cases}, \ \forall j = 1, ..., r \ . \quad (2)$$

In these two equations, r is the number of actions of the LA. λ and ρ are called reward and penalty parameters, respectively. For $\rho=\lambda$ the learning algorithm is called L_{R-P}; when $\rho << \lambda$, it is called $L_{R\varepsilon P}$; and when b is zero, it is called L_{R-I}.

3.2. Cellular learning automaton

A cellular learning automaton is a combination of a cellular automaton (CA) [42] with learning automata [11]. In this model, each cell of a CA contains one or more LAs. The LA or LAs residing in a particular cell define the state of the cell. Like CA, a local rule controls the behavior of each cell. At each time step, the local rule defines the reinforcement signal for a particular LA based on its selected action and the actions chosen by its neighboring LAs. Consequently, the neighborhood of each LA is considered to be its local environment. A formal definition of CLA is provided by Beigy and Meybodi [11, 43]. The authors have also investigated the asymptotic behavior of the model and provided some theoretical analysis on its convergence.

The structure of a CLA can be represented by a graph, where each vertex denotes a CLA cell [44]. Each edge of this graph defines a neighborhood relation between its two incident nodes. A CLA starts from some initial state (it is the internal state of every cell). At each stage, each automaton selects an action according to its probability vector and performs it in its local environment. Next, the rule of the CLA determines the reinforcement signals (the responses) to the LAs residing in its cells. Based on the received signals, each LA updates its internal probability vector. This procedure continues until a termination condition is satisfied.

3.3. Differential evolution algorithm

DE evolves a population of *NP* candidate solutions, called individuals, over some generations. During each generation, the operations mutation, crossover, and selection are performed on each individual. At generation k, the position of each individual X_i, $\forall i \in \{1,\ldots,\mathrm{NP}\}$, in a D dimensional search space is represented by $X_i(k)=[X_{i1}(k),\ldots, X_{iD}(k)]$. Individuals are initially generated randomly using a uniform distribution within the search space:

$$X_{ij}(0) = L_j^{\min} + \mathrm{rand}(0,1).(L_j^{\max} - L_j^{\min}),\ \forall i \in \{1,...,NP\}, \forall j \in \{1,...,D\}, \qquad (3)$$

where L_j^{min} and L_j^{max} are, respectively, the lower and the upper bounds of the j^{th} dimension; and rand(0, 1) is a uniformly distributed random number in (0, 1). At each generation k, a mutant vector $V_i(k)$ is generated for each individual X_i using a mutation scheme. Several DE mutation schemes have been developed during the recent decades. Three of the most common ones are as follows:

$DE/rand/1$:

$$V_i(k) = X_{r1}(k) + F.\left(X_{r2}(k) - X_{r3}(k)\right), \qquad (4)$$

$DE/best/1$:

$$V_i(k) = X_{best}(k) + F.\left(X_{r1}(k) - X_{r2}(k)\right) \qquad (5)$$

$DE/rand-to-best/1$:

$$V_i(k) = X_i(k) + F.\left(X_{best}(k) - X_i(k)\right) + F.\left(X_{r1}(k) - X_{r2}(k)\right) \qquad (6)$$

where r_1, r_2, and r_3 are three mutually exclusive integers randomly chosen from the set $\{1,\ldots,NP\}$. F is a control parameter which lies in the range [0,2] and scales the differential vectors. $X_{best}(k)$ represents the fittest individual of the k^{th} generation. After obtaining the mutant vector, a crossover operator is applied to generate a trail vector $U_i(k)=[U_{i1}(k),\ldots,U_{iD}(k)]$. DE algorithms mainly use two kinds of crossover methods: exponential (or two-point modulo) and binomial (or uniform). The binomial crossover, which is commonly adopted in the DE literature, is defined as follows:

$$U_{ij}(k) = \begin{cases} V_{ij}(k) & \text{if } \mathrm{rand}(0,1) < CR \text{ or } j = irand \\ X_{ij}(k) & \text{otherwise} \end{cases} \quad j = 1,...,D, \qquad (7)$$

where $CR \in [0,1]$ (called crossover rate) is a constant parameter. *irand* is a random index selected from the set {1,…,D} and ensures that the generated trail vector differs from its corresponding target vector.

Each generated trial vector is evaluated using an objective function. Then, either of the parent vector (also called target vector) or its corresponding trial vector is selected for the next generation by using a greedy selection scheme:

$$X_i(k+1) = \begin{cases} U_i(k) & \text{if } f\left(U_i(k)\right) < f\left(X_i(k)\right) \\ X_i(k) & \text{otherwise} \end{cases}, \qquad (8)$$

where $f(y)$ represents the objective function value of vector y.

4. A CLA based multi-population method for optimization in dynamic environments (CLA-MPD)

This section introduces a CLA based multi-population method for optimization in dynamic environments. CLA-MPD incorporates a cellular learning automaton to adaptively control the behavior of its subpopulations according to their context information.

4.1. Motivation and contribution

Many studies have shown the importance of utilizing a set of different DE schemes in one algorithm. Even for a particular problem, the most appropriate schemes may be different at various stages of the evolutionary process. Several introduced approaches for adaptively controlling the evolutionary operators have demonstrated effective performances in static environments. However, such approaches cannot be employed directly in a dynamic environment due to its time-varying characteristics. In a dynamic environment, the developed adaptive control approaches suffer from the following shortcomings:

- They mostly utilize only the fitness information of the individuals, which can lead to the loss of diversity in the population.
- They follow a gradual adaptation procedure which does not consider drastic changes in the population conditions. Accordingly, they cannot accommodate to the drastic changing conditions in a proper time.

- Common dynamic optimizers often utilize some exclusion and re-initialization modules for maintaining diversity. These modules have a severe impact on the gradual adaption mechanisms.

To address these issues, we introduce a context-dependent adaptive approach in the proposed dynamic optimizer. The population is divided into several subpopulations for preserving diversity. In order to monitor different population conditions, we define a state for each subpopulation. The state of a subpopulation is obtained using its fitness and distribution information.

The proposed approach uses a CLA for adaptively learning and choosing appropriate updating schemes for its subpopulations based on their states (i.e. their fitness and distribution information). Each cell of the CLA contains a set of learning automata and a subpopulation. Subpopulations move along different trajectories and spread in different areas of the fitness landscape; accordingly, each should be controlled in a specific manner for robust evolution. Each learning automaton of a cell is adapted in a way to specifically control the behavior of its peer subpopulation in a particular context state. A learning automaton is adapted using its received reinforcement signals. CLA provides a structure and proper mechanisms for appropriate adaptation of learning automata. Each selected action is evaluated and a reinforcement signal is generated for the LA which has chosen the action using a set of local rules and the selected actions in the neighborhood.

Whenever the state of a subpopulation changes during the evolution, the learning mechanism utilizes the previously acquired knowledge for the new state of the subpopulation. Consequently, the proposed method can adapt to different conditions in a short time. Moreover, to reach higher performance, it is crucial to make sure that the constituent DE operators are powerful while having different capabilities so that they can support each other during the evolutionary process.

Based on our empirical studies, we suggest three operators with different characteristics suitable for dynamic optimization problems.

4.2. Population structure

The formation of niches is a common phenomenon in biology. Because different niches evolve in isolation from each other, the overall diversity of the population tends to be high [45]. Based on this idea, in the proposed model, the entire population of individuals is partitioned into several subpopulations. Each subpopulation resides in one cell of a CLA. These cells are connected in a ring topology (Fig. 1). The CLA has M cells, each embedding a particular subpopulation and a set of six learning automata. Each LA is associated with a specific context state (which will be introduced in the following sections) and is activated according to the context state of its related subpopulation.

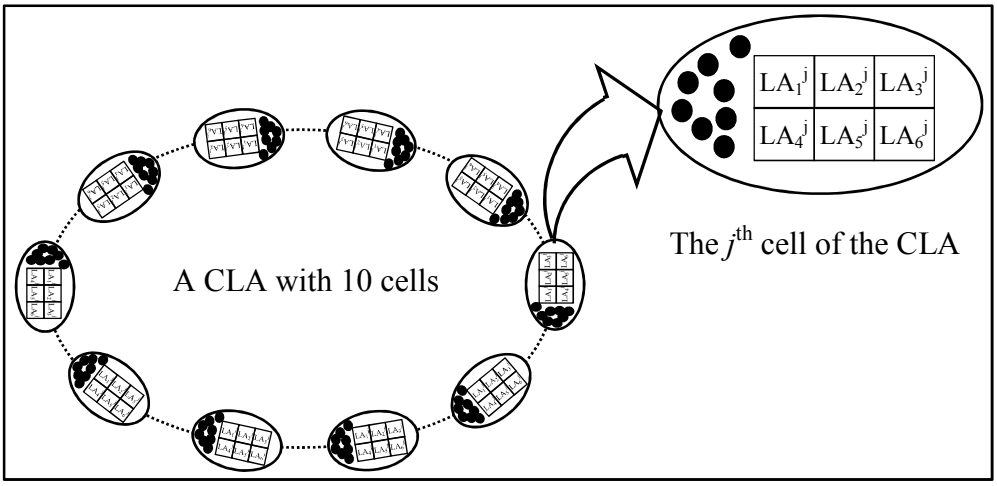


Figure 1. Population structure in CLA-MPD

4.3. General framework

Fig. 2 shows the building blocks of the proposed method. This section briefly introduces these building blocks, and their detailed descriptions are provided in the subsequent sections. CLA-MPD uses an adaptive mechanism to evolve each subpopulation based on its context information. This context information is retained in a structure called context vector. The context vector of a subpopulation

embodies its relative fitness along with some specific information about the distribution of its individuals. In the proposed method, the whole context vector space is partitioned into six different classes. The class of the context vector of a subpopulation will be called the context state of the subpopulation. CLA-MPD calculates the context vectors after population initialization and updates them during the evolutionary procedure.

CLA-MPD utilizes a set of specifically designed DE schemes, each inducing a different behavior on a subpopulation. The learning automata associated with each subpopulation learn the effectiveness of each scheme in different context states of their related subpopulation. Each LA in a CLA cell handles this learning task for a particular context state of its related subpopulation. Given the calculated context state of a subpopulation, the corresponding LA for that context state is activated in the action selection phase. Then, the activated LA selects a DE scheme, according to its probability vector, for its related subpopulation.

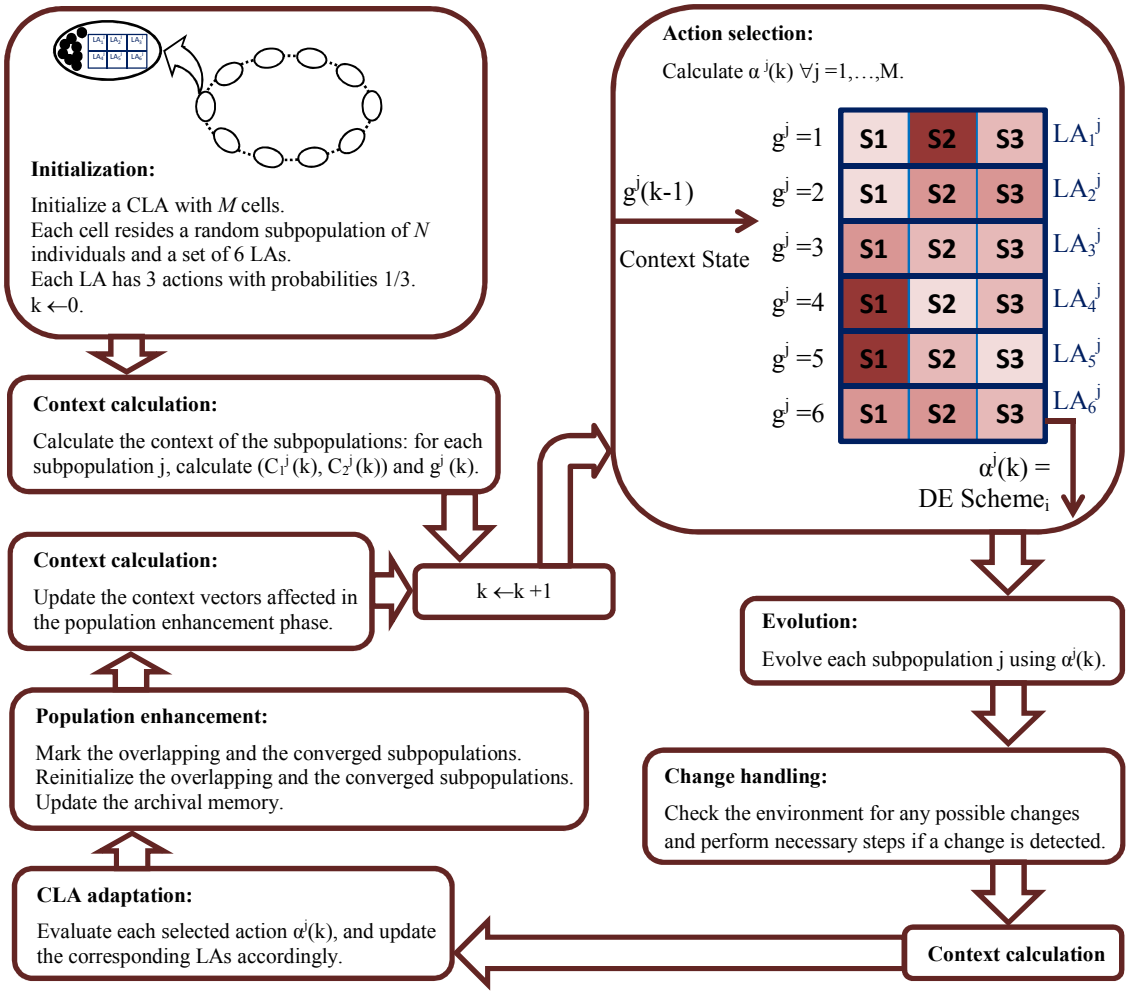


Figure 2. General framework of CLA-MPD

After the action selection phase, using the chosen DE schemes, the subpopulations evolve in the evolution phase. Then, change detection is performed. Following environmental change detection, some necessary steps like re-initialization and reevaluation of individuals can take place. As the subpopulations change during the latter two phases, their corresponding context vectors and context states are recalculated.

Table 1. The summary of the notations that are used in the proposed method.

Notation	Description	Notation	Description
L^{min}	Lower bound of the search space	X_B^j	The fittest individual of the j^{th} subpopulation
L^{max}	Upper bound of the search space	NB(j)	The index of the best subpopulation in the neighborhood of subpopulation j
M	Number of cells (subpopulations)	LA_s^j	The s^{th} LA of the j^{th} cell
N	Number of individuals within a subpopulation	$P_s^j(k)$	The action probability vector of LA_s^j at iteration k

NE(j,h)	The index of the h^{th} neighboring cell (subpopulation) to cell (subpopulation) j	$P_{s,l}^j(k)$	The selection probability of the l^{th} action for LA_s^j
NE(j,1)	The immediate left cell connected to cell j (considering ring topology, Fig. 1)	$\alpha^j(k)$	The action chosen by the active LA of the j^{th} cell
NE(j,1)	The immediate right cell connected to cell j	(C_1^j,C_2^j)	The context vector of the j^{th} subpopulation
X_i^j	The i^{th} individual within the j^{th} subpopulation	$g^j(k)$	Context state of the j^{th} subpopulation
BI(j)	Index of the fittest individual within subpopulation j	$f(y)$	f stores the objective function value of vector y

Context Calculation
Input j: index of a subpopulation
1. If $f(X_B^j(k))\leq f(X_B^{NE(j,1)}(k))$ And $f(X_B^j(k))\leq f(X_B^{NE(j,2)}(k))$, then $C_1^j(k)\leftarrow 0$.
2. Else if $f(X_B^j(k))>f(X_B^{NE(j,1)}(k))$ And $f(X_B^j(k))>f(X_B^{NE(j,2)}(k))$, then $C_1^j(k)\leftarrow 2$.
3. Else $C_1^j(k)\leftarrow 1$.
4. For each individual X_i^j do: Calculated $d_i^j(k)$ according to Eq. 9.
5. Calculate $\tilde{d}^j(k)$ according to Eq. 10.
6. Calculate $C_2^j(k)$ using Eq. 11.
7. Obtain the context state of the j^{th} subpopulation, $g^j(k)$, based on Eq. 12.

Figure 3. Context calculation procedure for the j^{th} subpopulation

As stated previously, each LA is responsible for selecting an appropriate DE scheme for its associated subpopulation in a specific context state. The LA achieves this through repeated cycles of scheme selection (action selection) and scheme adaptation. During the CLA adaptation phase, the effectiveness of the selected DE schemes is evaluated, and the LAs are updated accordingly. After CLA adaption phase, CLA-MPD reinitializes the overlapping and the converged subpopulations to maintain diversity. If a subpopulation changes in this phase, the affected context vectors will be recalculated. Table 1 summarizes the notations that are used in the following sections.

4.3.1. Context calculation

The context vector of each subpopulation j during generation k is a tuple like $(C_1^j(k),C_2^j(k))$. C_1^j represents the number of fitter nearby subpopulations, where the fitness of a subpopulation is defined as the fitness of its best individual. Considering the employed neighborhood structure in CLA-MPD, $C_1^j\in\{0,1,2\}$. C_1^j will be called the rank of the j^{th} subpopulation. C_2^j gives some information about the distribution of individuals in the subpopulation and is obtained as follows. First, the average distance of each individual in the subpopulation to all other individuals of the same subpopulation is obtained:

$$d_i^j(k)=\frac{1}{N-1}\sum_{l=1}^{N}\left|X_i^j(k)-X_l^j(k)\right| \qquad (9)$$

where |.| is the norm operator. After obtaining the average distances within the subpopulation, the average distance of the subpopulation to the nearest best individual is obtained according to the following equation:

$$\tilde{d}^{j}(k)=\begin{cases} d^{j}_{BI(j)}(k) \text{ if } C_1^j(k)=0 \\ D^{j}(k) \quad \text{if } C_1^j(k)>0 \end{cases}$$

$$\text{where } D^{j}(k)=\begin{cases} \frac{1}{N}\sum_{l=1}^{N}\left|X_B^{\mathrm{NB}(j)}(k)-X_l^j(k)\right| & \text{if } C_1^j(k)=1 \\ \min\left(\frac{1}{N}\sum_{l=1}^{N}\left|X_B^{\mathrm{NE}(j,1)}(k)-X_l^j(k)\right|,\frac{1}{N}\sum_{l=1}^{N}\left|X_B^{\mathrm{NE}(j,2)}(k)-X_l^j(k)\right|\right) & \text{if } C_1^j(k)=2 \end{cases} \quad (10)$$

Based on the Eq. 9 and Eq. 10, $C_2^j(k)$ is calculated as follows:

$$C_2^j(\mathrm{k})=\frac{\tilde{d}^{j}(k)}{\sqrt{median_{l=1..N}(d_l^j(k))}} \quad (11)$$

To utilize the described context information in the proposed learning approach, we partition the context vector space into a manageable number of classes. The class of the context vector of a subpopulation will be called the context state of the subpopulation. Based on our empirical studies, we use six different classes to represent the context state of each subpopulation j as follows:

$$g^{j}(k)=\begin{cases} 1 \text{ if } C_1^j(k)>0 \text{ And } 0\le C_2^j(k)\le 1 \\ 2 \text{ if } C_1^j(k)>0 \text{ And } 1< C_2^j(k)\le 2 \\ 3 \text{ if } C_1^j(k)>0 \text{ And } 2< C_2^j(k) \\ 4 \text{ if } C_1^j(k)=0 \text{ And } 0\le C_2^j(k)\le 1 \\ 5 \text{ if } C_1^j(k)=0 \text{ And } 1< C_2^j(k)\le 2 \\ 6 \text{ if } C_1^j(k)=0 \text{ And } 2< C_2^j(k) \end{cases} \quad (12)$$

where $g^j(k)$ is the context state of the j^{th} subpopulation at generation k. Fig. 3 represents the context calculation procedure for a typical subpopulation. For each subpopulation j, its context state and context vector values can be interpreted as follows:

Case $C_1^j(k)>0$. After (re)initialization of a subpopulation, its individuals gradually cluster around a peak of the landscape as the algorithm proceeds. Whenever this cluster gets closer to a peak already

covered by a fitter neighboring subpopulation, $C_2^j(k)$ becomes smaller ($g^j(k)=1$). Whenever the cluster draws closer to an uncovered peak, $C_2^j(k)$ becomes larger ($g^j(k)=3$). $g^j(k)=2$ denotes a transition state for the j^{th} subpopulation.

Case $C_1^j(k)=0$. $C_2^j(k)$ merely shows the convergence level of the j^{th} subpopulation: individuals tend to be distributed evenly around the best individual of their subpopulations in the convergence conditions.

4.3.2. Action selection

A CLA controls the behavior of subpopulations in their contexts by adaptively controlling their updating rules. The CLA has M cells, and each cell resides a subpopulation and a group of six learning automata. Each learning automaton is associated with a particular context state of its related subpopulation and can choose an action from the set A={ 'DE scheme 1', 'DE scheme 2', 'DE scheme 3'}. These actions correspond to the updating rules employed by CLA-MPD and are described in the following section.

During generation k, each cell j activates the learning automaton associated with $g^j(k\text{-}1)$. Each activated learning automaton selects a DE scheme (an action) according to its action probability vector. The chosen DE scheme of the active LA within the j^{th} cell during iteration k is denoted by $\alpha^j(k)$ (Fig. 4). Initially, the selection probability of each scheme for each LA is 1/3 (i.e. $P_{i,l}^j(0)=1/3$, $\forall i \in \{1,\ldots,6\}$, $\forall j \in \{1,\ldots,M\}$, $\forall l \in \{1,2,3\}$). As the algorithm proceeds, the learning automata adaptively adjust their action probabilities according to their performances (which will be described in CLA adaptation phase).

Action selection
1. For each cell (subpopulation) j do:
   a. $i \leftarrow g^j(k\text{-}1)$.
   b. $r \leftarrow$ rand(0,1). // *r is a uniformly distributed random number in (0, 1).*
   c. If $r < P_{i,1}^j$, then $\alpha^j(k) \leftarrow$ 'DE scheme 1'.
   d. Else if $r < P_{i,2}^j$, then $\alpha^j(k) \leftarrow$ 'DE scheme 2'.
   e. Else $\alpha^j(k) \leftarrow$ 'DE scheme 3'.

Figure 4. Scheme (action) selection phase of CLA-MPD

Evolution

1. For each subpopulation j do:
   a. For i=1 to N do:
      i. Obtain the control parameters as follows:
         - Calculate $\widehat{F}_i^j(k)$ according to Eq. 13.
         - Calculate $\widehat{Cr}_i^j(k)$ according to Eq. 14.
      ii. Obtain the mutant vector $V_i^j(k)$ as follows:
         - If $\alpha_i^j(k)$= 'DE scheme 1' use scheme 1 (Eq. 15).
         - If $\alpha_i^j(k)$= 'DE scheme 2' use scheme 2 (Eq. 16).
         - If $\alpha_i^j(k)$= 'DE scheme 3' use scheme 3 (Eq. 17).
      iii. Perform binomial crossover to generate a trial vector $U_i^j(k)$ according to Eq. 7.
      iv. Evaluate $U_i^j(k)$ using objective function and obtain $X_i^j(k)$:
      v. $X_i^j(k) = \begin{cases} X_i^j(k-1) \text{ if} f(X_i^j(k-1)) < f(U_i^j(k)) \\ U_i^j(k) \quad \text{otherwise} \end{cases}$
      vi. Update the control parameters using Eq. 18 and Eq. 19.

Figure 5. The evolution procedure of subpopulations.

4.3.3. Evolution of subpopulations

During the evolution phase, each subpopulation j is updated using its selected action, $\alpha^j(k)$ (Fig. 5). As stated previously, $\alpha^j(k) \in$ {'DE scheme 1', 'DE scheme 2', 'DE scheme 3'}, where each scheme defines a mutation scheme. Like jDE, the proposed method utilizes an adaptive approach to control the scaling factor and crossover rate of target vectors. Each individual has its own independent parameters for each mutation scheme, and these parameters are adjusted by means of evolution. During generation k, the control parameters are generated for the selected schemes according to the following rules:

$$\widehat{F}_i^j(k) = \begin{cases} F_l + r_1.F_u & \text{with probability } 0.1 \\ F_i^j\left(k-1, \alpha^j(k)\right) & \text{otherwise} \end{cases} \quad (13)$$

$$\widehat{Cr}_i^j(k) = \begin{cases} U(0,1) & \text{with probability } 0.1 \\ Cr_i^j\left(k-1, \alpha^j(k)\right) & \text{otherwise} \end{cases} \quad (14)$$

where F_l and F_u define, respectively, the lower and upper bounds for the scale factor (Their values are adopted form [28]). F_i^j(k, DE scheme s) is the archived scale factor of X_i^j for DE scheme s. Similarly, Cr_i^j(k, DE scheme s) is the archived crossover rate of X_i^j for DE scheme s. After calculating its control parameters, each individual X_i^j is mutated according to one of the following schemes.

Mutation Scheme 1:

$$V_i^j(k) = X_i^j(k-1) + \widehat{F}_i^j(k).\left(X_{br}^j(k-1) - X_i^j(k-1)\right) + \widehat{F}_i^j(k).\left(X_{r1}^j(k-1) - X_{r2}^j(k-1)\right)$$
$$br = \arg\min_l \left\{ f\left(X_l^j(k-1)\right) \mid \; |X_i^j(k-1) - X_B^n(k-1)| < |X_l^j(k-1) - X_B^n(k-1)| \right\} \qquad (15)$$
$$\text{with } n = \begin{cases} \text{NE}(j,1) \text{ with probablity 0.5} \\ \text{NE}(j,2) \text{ otherwise} \end{cases}$$

where X_{br}^j is the fittest individual such that the distance between X_{br}^j and X_B^n is greater than the distance between X_i^j and X_B^n. Whenever all of the individuals of the j^{th} subpopulation are closer than X_i^j to X_B^n, br is selected randomly from the set $\{1,\ldots,N\}$-$\{i\}$. r_1,r_2 are two mutually exclusive indices randomly chosen from the set $\{1,\ldots,N\}$ such that $r_1,r_2 \neq i,br$.

Mutation Scheme 2:

$$V_i^j(k) = X_i^j(k-1) + \widehat{F}_i^j(k).\left(X_{bl}^j(k-1) - X_i^j(k-1)\right) + \widehat{F}_i^j(k).\left(X_{r1}^j(k-1) - X_{r2}^j(k-1)\right)$$
$$bl = \arg\min_l \left\{ |X_l^j(k-1) - X_i^j(k-1)| \;\; | f\left(X_l^j(k-1)\right) < f\left(X_i^j(k-1)\right) \right\} \qquad (16)$$

where X_{bl}^j is the closest individual of the j^{th} subpopulation to X_i^j which is fitter than $X_i^j(k)$. Whenever X_i^j is the fittest individual of its subpopulation, bl is selected randomly from the set $\{1,\ldots,N\}$-$\{i\}$.

Mutation Scheme 3 ('DE/rand/1'):

$$V_i^j(k) = X_{r1}^j(k) + \widehat{F}_i^j(k).\left(X_{r2}^j(k) - X_{r3}^j(k)\right) \qquad (17)$$

After mutation, the crossover operation is performed between the target vector $X_i^j(k\text{-}1)$ and the mutant vector $V_i^j(k)$ to produce a trial vector according to Eq. 7. Once the trial vector is produced, the selection operator compares the target vector $X_i^j(k\text{-}1)$ and the trial vector $U_i^j(k)$ and selects the fitter one for the next generation. Additionally, promising control parameters are propagated to the next generation according to the following equations:

$$F_i^j\left(k, \alpha^j(k)\right) = \begin{cases} \widehat{F}_i^j(k) & \text{if } f\left(U_i^j(k)\right) < f\left(X_i^j(k-1)\right) \\ F_i^j\left(k-1, \alpha^j(k)\right) & \text{otherwise} \end{cases} \qquad (18)$$

$$Cr_i^j\left(k, \alpha^j(k)\right) = \begin{cases} \widehat{Cr}_i^j(k) & \text{if } f\left(U_i^j(k)\right) < f\left(X_i^j(k-1)\right) \\ Cr_i^j\left(k-1, \alpha^j(k)\right) & \text{otherwise} \end{cases} \qquad (19)$$

Mutation Scheme 1 encourages an individual to search promising areas farther from its neighboring subpopulations. Accordingly, this strategy can improve the overall population diversity and can increase the chance of detecting multiple optima. By using the Mutation Scheme 2, an individual is encouraged to search its nearby promising areas. Mutation scheme 2 is a useful scheme for exploiting promising areas that are covered by a subpopulation. The third Scheme is one of the most popular mutation schemes in the DE literature. Mutation Schemes 1 and 2, although very helpful in dynamic environments, would produce limited and specific variations on a population. Hence, the use of a more general mutation scheme like DE/rand/1 can help the algorithm to overcome this drawback.

4.3.4. Change handling

A change detection module is called at the change handling phase to detect potential environmental changes. Upon the detection of an environmental change, CLA-MPD reinitializes each subpopulation j in the following manner. First, it finds the median of the stored objective function values of the subpopulation individuals. Then, it randomly reinitializes the individuals which have larger objective function values than the obtained median. Whenever more than one individual has an objective function value equal to the calculated median, CLA-MPD randomly retains one of them and reinitializes the rest. After the re-initialization step, all individuals are reevaluated. Finally, CLA-MPD archives the best individual of each subpopulation along with its fitness value for the next call of change detection module. The pseudocode of this phase is given in Fig. 6.

Change handling phase
1. Call change detection module.
2. If a change is detected
   a. For each subpopulation j do:
      i. $f_{mid}^{j} \leftarrow \text{median}\{f(X_1^j(k)),\ldots,f(X_N^j(k))\}$.
      ii. $A \leftarrow \{i | f(X_i^j(k)) = f_{mid}^{j}\}$.
      iii. If $A \neq \varnothing$, then $r \leftarrow$ a random element form A; else $r \leftarrow N+1$.
      iv. For i=1 to N do:
         - If $f(X_i^j(k)) \geq f_{mid}^{j}$ And $i \neq r$, then randomly initialize X_i^j according to Eq. 3.
      v. For each X_i^j do: $f(X_i^j(k)) \leftarrow objective\_fun(X_i^j(k))$.
3. For each subpopulation j do: $(X_{Arch}^{j}, f_{Arch}(X_{Arch}^{j})) = (X_B^j(k), f(X_B^j(k)))$

Figure 6. Change handling procedure of CLA-MPD

4.3.5. CLA adaptation

The learning automata are updated in the CLA adaptation phase (Fig. 7) to select effective DE schemes at the future generations. To do this, CLA evaluates the effectiveness of each performed action (DE scheme); then, it generates a reinforcement signal to the LA which emitted the action. Finally, using the received reinforcement, the LA updates its probability vector. We can evaluate the effectiveness of a utilized scheme by a subpopulation from two perspectives. 1- Its impact on population diversity (i.e. how the distribution part of the context vector of the subpopulation changes after the utilization of the scheme). 2- Its impact on the fitness improvement of the subpopulation. In order to evaluate the amount of fitness improvement, the following three metrics will be employed:

$$\Lambda^j(k) = f\left(X_B^j(k-1)\right) - f\left(X_B^j(k)\right) \qquad (20)$$

$$\Gamma^j(k) = \frac{1}{N}\sum_{i=1}^{N}\left(f\left(X_i^j(k-1)\right) - f\left(X_i^j(k)\right)\right) \qquad (21)$$

$$\Delta^j(k) = C_1^j(k-1) - C_1^j(k) \qquad (22)$$

Here, Λ^j represents the amount of improvement in the fitness of the j^{th} subpopulation, while Γ^j represents the average improvement of its individuals. Finally, Δ^j represents the amount of improvement in the rank of the j^{th} subpopulation (first component of its context vector). Based on these three metrics, the selected action (scheme) for the j^{th} subpopulation is rewarded (according to Eq. 1) if one of the following conditions is satisfied; otherwise it is penalized (according to Eq. 2).

$$\text{Condition 1: } \begin{array}{l} C_1^j(k) = 0 \text{ and} \\ \Lambda^j(k) > \min\left(\Lambda^{N(j,1)}(k), \Lambda^{N(j,2)}(k)\right) \text{ or } \Gamma^j(k) > \min\left(\Gamma^{N(j,1)}(k), \Gamma^{N(j,2)}(k)\right) \end{array} \qquad (23)$$

$$\text{Condition 2: } \begin{array}{l} C_1^j(k) > 0 \text{ and} \\ C_2^j(k) > 2 \text{ and } \Delta^j(k) > 0 \end{array} \qquad (24)$$

$$\text{Condition 3: } \begin{array}{l} C_2^j(k) > 2 \text{ and} \\ \Lambda^j(k) > \max\left(\Lambda^{N(j,1)}(k), \Lambda^{N(j,2)}(k)\right) \text{ or } \Gamma^j(k) > \max\left(\Gamma^{N(j,1)}(k), \Gamma^{N(j,2)}(k)\right) \end{array} \qquad (25)$$

Condition 4: $C_2^j(k) > C_2^j(k-1)$ and $\Delta^j(k) \geq 0$ and $\Lambda^j(k) > \min\left(\Lambda^{N(j,1)}(k), \Lambda^{N(j,2)}(k)\right)$ or $\Gamma^j(k) > \min\left(\Gamma^{N(j,1)}(k), \Gamma^{N(j,2)}(k)\right)$ (26)

The defined conditions are the formulation of some general concepts, which can express the effectiveness of the employed schemes. Whenever a subpopulation with rank 0 experiences an acceptable amount of fitness improvement, its utilized scheme is rewarded (Condition 1). As there is not any clear way of determining this acceptable amount of improvement, we defined it relatively in comparison with the neighboring subpopulations. Similarly, whenever a subpopulation experiences an improvement in its diversity without a decline in its rank, we can assume that its utilized scheme was effective (Condition 4). We reward a utilized scheme which moves a subpopulation toward an undetected peak. If a subpopulation has a good distance from other fitter subpopulations and enjoys a great amount of fitness improvement, it is likely that the subpopulation is moving toward an uncovered optimum (Condition 3). In the same way, we reward the schemes that cause an improvement in the rank of a subpopulation while maintaining its diversity (Condition 2).

CLA adaptation phase

1. For each subpopulation j do:
   a. Calculate $\Lambda^j(k)$, $\Gamma^j(k)$, and $\Delta^j(k)$ according to Eq. 20, Eq. 21, and Eq. 22, respectively.
   b. If one of the conditions in Eq. 23-26 is satisfied, $\beta^j(k) \leftarrow 0$.
   c. Else $\beta^j(k) \leftarrow 1$.
   d. $h \leftarrow g^j(k\text{-}1)$. // *context state of the subpopulation, which also indicates the active LA*
   e. If $\beta^j(k)=0$ // *Reward the selected action*

$$p_{h,l}^j(k+1) = \begin{cases} p_{h,l}^j(k) + a\left(1 - p_{h,l}^j(k)\right) & \text{if } l = \alpha^j(k) \\ p_{h,l}^j(k) + (1-a) & \text{if } l \neq \alpha^j(k) \end{cases}$$

   f. Else // *penalize the selected action*

$$p_{h,l}^j(k+1) = \begin{cases} p_{h,l}^j(k)(1-b) & \text{if } l = \alpha^j(k) \\ \dfrac{b}{r-1} + p_{h,l}^j(k)(1-b) & \text{if } l \neq \alpha^j(k) \end{cases}$$

Figure 7. The pseudo code of the CLA adaptation phase

4.3.6. Population enhancement

An anti-convergence mechanism can improve an optimizer's ability in detecting multiple optima. To this end, CLA-MPD is incorporated with an anti-convergence mechanism, that reinitializes a converged

or an overlapping subpopulation. This re-initialization is performed with the aid of the external memory of promising positions. Fig. 8 gives the pseudocode of the population enhancement phase.

Population Enhancement
1. $A \leftarrow \varnothing$.
2. For each subpopulation j do:
   a. For each individual X_i^j do: calculate $\tilde{d}_i^j(k)$ (Eq. 27).
   b. If $\min_{i=1:N}\left(\hat{d}_i^j(k)\right) < (0.1)^{2-C_1^j(k)}\varepsilon_1$, then $converged^j \leftarrow$True.
   c. If $\exists l \neq j$($f(X_B^j(k)) < f(X_B^l(k))$ And $|X_B^j(k)-X_B^l(k)| < \varepsilon_2$), then $overlapping^j \leftarrow$True.
   d. If $converged^j$=True And $C_1^j(k)$=0, then $A \leftarrow A \cup X_B^j(k)$.
3. For each subpopulation j do:
   a. If $converged^j$=True Or $overlapping^j$=True, then reinitialize the subpopulation:
      i. With probability 0.5:
         Re-initialize the individuals of the subpopulation randomly within the search space (Eq. 3).
      ii. Else: Re-initialize the subpopulation using the external memory
4. For each $X \in A$ do:
   a. If $|M_E| < MN$, then $M_E \leftarrow M_E \cup X$.
   b. Else let Y be a memory element such that:
      $\exists Z \in (M_E \cup X)$ ($Z \neq Y$ And $\forall W,U \in (M_E \cup X)$ ($W \neq Y$ and $U \neq Y \Rightarrow |X-Z| \leq |U-W|$)).
   c. $M_E \leftarrow M_E - Y$. $M_E \leftarrow M_E \cup X$.

Figure 8. Pseudocode of the population enhancement phase

Anti-convergence

Whenever the individuals of a subpopulation are converged well to a small Euclidean neighborhood, they cannot contribute much to the optimization process. In this case, the individuals are pretty close to each other. Accordingly, they cannot improve further by means of mutation and crossover. CLA-MPD reinitializes the converged subpopulations so that they can start to search for new optima. The proposed method considers each subpopulation j as a converged subpopulation if its radius becomes smaller than a threshold value:

$$converged^j = \min_{i=1:N}\left(\hat{d}_i^j(k)\right) < (0.1)^{2-C_1^j(k)}\varepsilon_1$$
$$\text{where } \hat{d}_i^j(k) = \text{median}_{l=1:N}\left(\left|X_i^j - X_l^j\right|\right) \qquad (27)$$

Here, $(0.1)^{2-C_l^j(k)}\varepsilon_1$ is the convergence threshold. The defined convergence threshold gives more evolution budget to the promising peaks. Accordingly, CLA-MPD can achieve a high degree of accuracy on detected global optima.

CLA-MPD uses an exclusion rule to ensure that different subpopulations are located around different basins of attraction. The exclusion rule marks each pair of subpopulations as overlapping if the distance between their best individuals is less than a predefined threshold, ε_2, called exclusion radius. Once a pair of subpopulations is marked as overlapping, the population with the highest fitness is kept, and the other is reinitialized.

Memorization and re-initialization

Like many dynamic optimizers, CLA-MPD uses an external memory, M_E, to recall useful information from the past. Especially, this idea would to be very useful when the changing optimum repeatedly returns to the previous positions. The size of the external memory is identical to the population size (i.e. MN). Whenever, a subpopulation like j marked as a converged subpopulation has a better fitness in comparison to both its neighbors, its best individual is added to the external memory. If the memory is already full, CLA-MPD finds two closest elements from the set $M_E \cup X_B^j(k)$ and replaces one of them with $X_B^j(k)$.

As stated previously, a subpopulation marked as a converged or overlapping subpopulation is reinitialized to discover new optima. Such a subpopulation is initialized either randomly within the search space (step 3.a.i in Fig. 8) or by using an external memory element (Fig. 9). In order to initialize a subpopulation j by using a memory position, an external memory element like X_m is selected randomly. Then, the position of each individual X_i^j is randomly generated within the search space or around the chosen memory element. In the latter case, each component $X_{i,l}^j$ of X_i^j is drawn from Gaussian distribution with mean $X_{m,l}$ and standard deviation σ_l. $\sigma=[\sigma_1,\ldots,\sigma_D]$ is generated in the following manner. First, each σ_l is randomly sampled from a uniform distribution within the interval $[0, 0.5|L_l^{Min}-L_l^{Max}|]$. Then, σ is decreased in a random manner so that we can produce various distributions around X_m.

The idea behind this initialization procedure is as follows. Random initialization (step 3.a.i in Fig. 8) provides good exploration of the landscape, while initialization around a memory element is useful for tracking the previously detected optima. The position of a peak may change drastically after a sequence of environmental changes. It is also possible that some landscape peaks will be diminished or eliminated after a period of time. To address these issues, we initialize almost half of a tracking subpopulation using uniform distribution. The remaining individuals are sampled in different distance from the memory element using different Gaussian distributions.

Initialization using the external memory
Input: j // subpopulation index
1. Select a random element like X_m from M_e.
2. For each individual X_i^j do:
   a. With probability 0.5, redistribute the individual randomly within the search space.
   b. Otherwise:
      i. $R \leftarrow 0.5|L^{Min} - L^{Max}|$
      ii. Generate σ randomly within the search range: $\sigma_l \sim U[0, R_l]\ \forall l \in \{1,\ldots,D\}$.
      iii. Repeat:
         - Generate σ' randomly within the search range: $\sigma'_l \sim [0, R_l]\ \forall l \in \{1,\ldots,D\}$.
         - Set $\sigma_l \leftarrow \min(\sigma_l, \sigma'_l)\ \forall l$.
      iv. Until rand(0,1)<0.5.
      v. $X_{il} \leftarrow \sigma_l N(0,1) + X_{ml}\ \forall l$.

Figure 9. Subpopulation redistribution using a memory element

4.4. Change detection module

Changes in the environment are detected by reevaluating the fitness of the best individual of each subpopulation (Fig. 10). At the end of each change handling phase, we archive the best individual of each subpopulation along with its fitness value. Then, at the beginning of the next change detection phase, the archived individuals are reevaluated. Next, the newly obtained fitness values are compared with the archived ones. Clearly, if there is a mismatch in the compared values, it can be inferred that an environmental change must have taken place.

Change detection Module
For each subpopulation j do:
   a. $Tem \leftarrow objective\_fun(X_{Arch}^j)$.
   b. If $Tem \neq f_{Arch}(X_{Arch}^j)$, then signal change.

Figure 10. Change detection module

5. Experimental studies

In this section, we evaluate the performance of the proposed method using IEEE CEC 2009 benchmark problems for dynamic optimization (generated by using the GDBG system) [46]. GDBG presents greater challenges to dynamic optimizers in comparison to the other commonly used benchmarks such as moving peaks problems due to its incorporated rotation methods, higher dimensionalities, and larger number of local optima [31]. There are seven change types in this test bed, which are small step (T_1), large step (T_2), random (T_3), chaotic (T_4), recurrent (T_5), recurrent with noise (T_6), and dimensional change (T_7). The benchmark set consists of six test functions, which are rotation peak function (F_1), composition of sphere functions (F_2), composition of Rastrigin's functions (F_3), composition of Griewank's functions (F_4), composition of Ackley's functions (F_5), and hybrid composition functions (F_6). F_1 is a maximization problem. For this problem, two different instances are considered, one using 10 peaks and the other using 50 peaks. The remaining problems are to be minimized. We follow the experimental settings described in IEEE CEC 2009 [46]. The summary of these settings is as follows. The number of dimensions is 10 (D=10) in scenarios T_1-T_6 and is changing between 5 and 15 in scenario T_7. The number of changes per run is set to 60, and the change frequency is set to 10000D fitness evaluations. Simulation results are obtained over 20 independent runs on each benchmark. The average mean error criterion is used as a performance measure, which is defined as follows:

$$Avg\_mean = \frac{1}{runs \times num\_change} \sum_{i=1}^{run\_num} \sum_{j=1}^{num\_change} E_{i,j}^{last}(t) \qquad (28)$$

where E^{last} is the absolute function error recorded just before each change in the environment.

In real-world applications, environmental changes can occur in random times. Therefore, the adaptability of a dynamic optimizer is of great importance. The average error of an algorithm over an entire run can represent how fast and accurate a dynamic optimizer can react to environmental changes. The adaptability metric can be expressed as [31]:

$$Ada = \frac{1}{num\_change} \sum_{i=1}^{num\_change} \left(\frac{1}{\tau} \sum_{j=1}^{\tau} error(i,j) \right) \qquad (29)$$

where τ is number of generations between two consecutive environmental changes, and *error*(*i*,*j*) is the fitness error between the best obtained individual and the global optima of the environment at the j^{th} generation.

5.1. Parameter settings of CLA-MPD

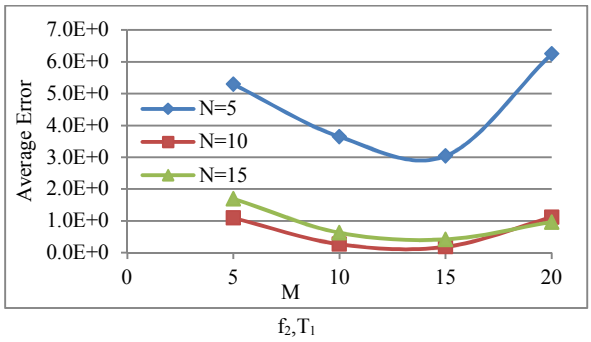
The parameter settings of CLA-MPD are summarized in Table 2. We use the same settings suggested in [16] for the LA parameters *a* and *b* (Eq. 1-2 and Fig. 7). In the following sections, we experimentally analyze the population size and the convergence threshold. The other parameters can be tuned experimentally in the same manner.

Table 2. Parameter settings of CLA-MPD

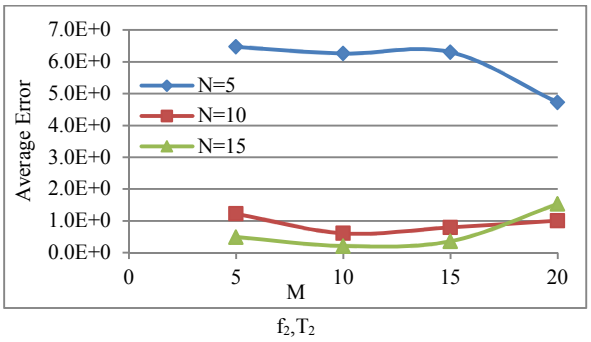
Parameter	values
Number of subpopulations (*M*)	10
Number of individuals within each subpopulation (*N*)	10
Learning rate (*a*,*b*)	(0.1,0.01)
convergence threshold (ε_1)	1E-3
exclusion radius (ε_2)	5E-2

5.1.1. Number of subpopulations and population size

In this section, we experimentally investigate the effect of different settings of *M* and *N* on the performance of the proposed method. For the sake of space economy, the experiments of this section are conducted on a small set of test instances. The performance of CLA-MPD under different settings of *M* and *N* is illustrated in Fig. 11.



f_2, T_1



f_2, T_2

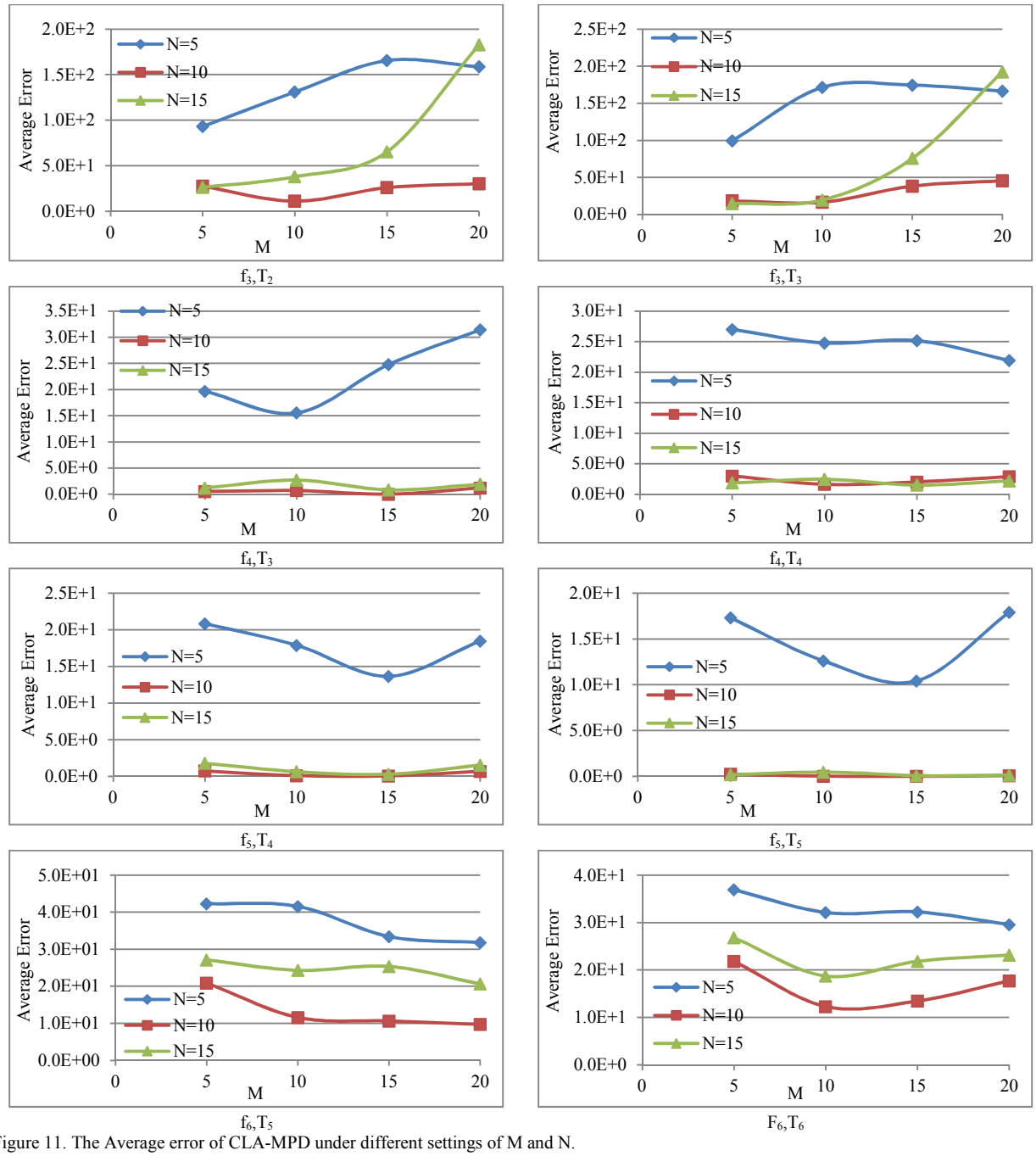


Figure 11. The Average error of CLA-MPD under different settings of M and N.

Considering Fig. 11, CLA-MPD has relatively poor performance when it uses subpopulations of size five. A subpopulation with an insufficient number of individuals loses its diversity after a few generations and can easily get trapped in a local optimum. Moreover, using a large number of individuals may be inappropriate as well. The number of generations between two consecutive environmental changes decreases with increasing subpopulation size. Accordingly, the proposed algorithm obtains relatively poor

results on some test problems like f_3 when it uses a large number of individuals. CLA-MPD demonstrates better performance with N=10 or N=15. The obtained results of the algorithm with N=10 and N=15 are very close on most of the tested instances. CLA-MPD exhibits somehow better performance with N=10 on f_6 and f_3. Also, among different settings of M with N=10, M=10 seems to be an appropriate choice.

5.1.2. Convergence threshold (ε_1)

This section investigates the sensitivity of CLA-MPD to its convergence threshold. To this end, the influence of different settings of ε_1 on the performance of CLA-MPD is experimentally studied on the test instances from the previous section. Fig. 12 gives the average errors of the obtained results.

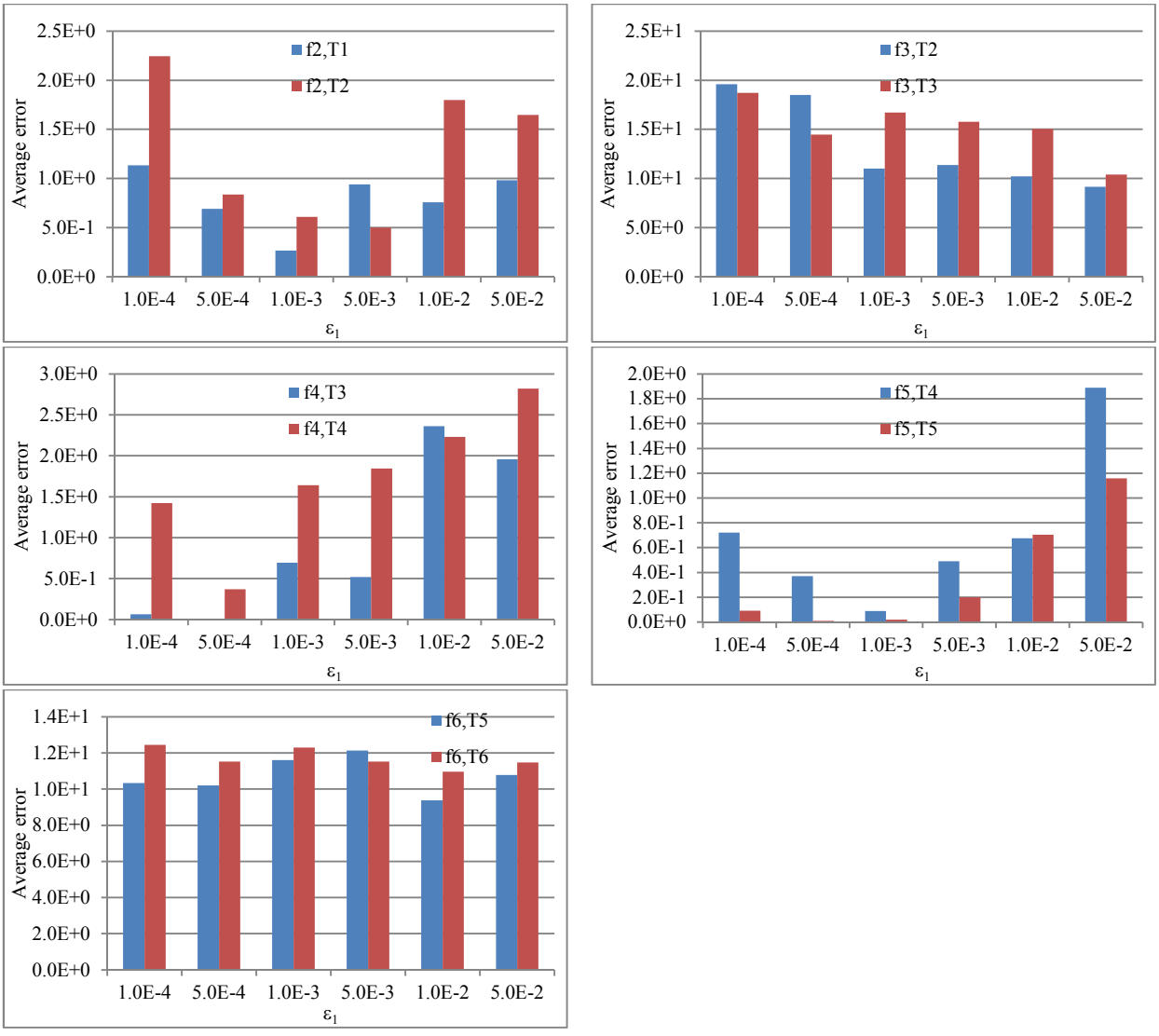


Figure 12. The effect of different settings of ε_1 on the performance of CLA-MPD

It is evident that the most appropriate value of ε_1 differs between various test instances. Using a large convergence threshold for a landscape which contains many close high fitness optima is unsuitable: a subpopulation close to a global optimum may be removed (reinitialized) before reaching the optimum. A similar argument can be made in the case of landscapes containing sharp peaks. However, fitness evaluations would be consumed unnecessarily when the algorithm employs a small convergence threshold. Considering the obtained results, the performance of the algorithm on f_3 is poor when ε_1 is very small (∼1E-4). The performance gets better with the increment of ε_1. In contrast, CLA-MPD exhibits very poor performances on f_4 and f_5 when ε_1 is very large (∼5E-2). It can be seen that CLA-MPD achieves better average errors on most of the tested instances when $\varepsilon_1 \in$[5E-4, 5E-3]. Following this observation, ε_1=1E-3 will be used for the rest of the paper.

5.2. Investigating the effectiveness of the proposed method.

In this section, CLA-MPD is compared with a set of state-of-the-art nature-inspired dynamic optimizers. The comparative results are expressed using the defined measures in Eq. 28 and Eq. 29.

5.2.1. Evaluation based on the average error

In this section, CLA-MPD is compared experimentally with seven dynamic optimization approaches given in Table 3. All of the peer methods in this section have already been tested on the GDBG benchmark set and exhibited effective performances. jDE, EDESAC, and mSQDE-i are adaptive multi-population DE algorithms. Accordingly, they have been chosen to evaluate the effectiveness of the proposed multi-population adaptive DE algorithm. Moreover, the building blocks of jDE are much similar to those of CLA-MPD. The main difference between the two algorithms lies in the context-aware learning scheme of CLA-MPD and its incorporated updating strategies. To compare CLA-MPD with approaches which are not based on DE, four algorithms including ShrBA, PLBA, bCCEA, and DASA are selected.

The comparative results in terms of average mean errors and standard deviations are provided in Table 4 and Table 5. A two-tailed t-test at a 0.05 significance level is employed to compare the results of CLA-MPD against those of the other peer approaches. In addition, Table 6 summarizes the overall comparative results. Each entry in Table 6 has the form "*w*/*t*/*l*" which means that CLA-MPD is better than, equal to and worse than the corresponding compared method on *w*, *t*, and *l* test instances, respectively.

Table 3. A brief description of the comparison methods

ShrBA [47]:	Shrinking-based bees algorithm (BA) is an improved variant of basic BA and includes an additional step called neighborhood shrinking, which is implemented in all of the patches simultaneously using a global memory.
PLBA [47]:	A BA in which the population initialization, local search, and global search stages are performed according to the patch concept and Levy motion.
bCCEA [48]:	A cooperative coevolutionary approach for dynamic optimization which uses two types of individuals: random immigrants and elitist individuals.
EDESAC [49]:	A multi-population variant of DE that incorporates an ensemble of adaptive mutation strategies alongside different parameter settings. It also uses an archive of the past promising solutions with an adaptive clearing scheme to enhance its population diversity.
mSQDE-i [50]:	A multi-population differential evolution algorithm with a self-adaptive strategy. Additionally, it uses an interaction mechanism between quantum and conventional individuals.
DASA [51]:	An ant colony based approach that uses pheromone as a means of communication between ants and uses the so-called differential graph representation of the search space.
jDE [28]:	Multi-population self-adaptive DE, which uses mechanisms such as aging, archival memory, and overlapping control to cope with the dynamicity of environments.

Table 4. The average errors and the standard deviations of the peer methods on the functions f_1-f_3. † indicates that CLA-MPD performs better than the compared algorithm by t-test. ‡ means that the corresponding algorithm is better than CLA-MPD.

		CLA-MPD	EDESAC	bCCEA	ShrBA	PLBA	mSQDE-i	DASA	jDE
F1-10	T1	**6.68E-03**	1.06E-01	1.30E+01	4.38E+00	8.07E+00	1.10E-01	1.86E-01	3.42E-02
		8.04E-02	7.89E-01†	1.79E+01†	8.22E+00†	1.30E+01†	7.80E-01†	1.33E+00†	4.25E-01†
	T2	2.78E+00	3.55E+00	1.87E+01	1.20E+01	1.31E+01	**1.06E+00**	4.20E+00	3.60E+00
		6.33E+00	8.14E+00†	2.10E+01†	1.68E+01†	1.52E+01†	**4.12E+00‡**	7.28E+00†	7.92E+00†
	T3	3.77E+00	6.86E+00	2.06E+01	1.13E+01	6.10E+00	**9.10E-01**	6.41E+00	3.11E+00
		7.99E+00	1.10E+01†	1.91E+01†	1.41E+01†	1.07E+01†	**3.52E+00‡**	9.34E+00†	8.26E+00‡
	T4	3.49E-01	1.77E+00	5.56E+00	1.10E+01	1.49E+01	1.10E-01	5.01E-01	**2.14E-02**
		8.74E-01	2.90E+00†	5.80E+00†	1.74E+01†	1.69E+01†	5.10E-01‡	1.74E+00†	**6.28E-01‡**
	T5	**9.05E-04**	4.12E-01	2.93E+01	5.68E+00	2.99E+00	5.80E-01	1.22E+00	2.33E+00
		3.84E-04	2.04E+00†	2.49E+01†	6.57E+00†	5.31E+00†	2.07E+00†	3.09E+00†	5.38E+00†
	T6	**2.16E-02**	8.48E-01	3.49E+01	1.18E+01	1.45E+01	1.40E-01	2.86E+00	1.24E+00
		2.70E-01	4.45E+00†	2.71E+01†	1.83E+01†	1.99E+01†	6.30E-01†	8.21E+00†	5.54E+00†
F1-50	T1	2.02E-01	4.57E-01	1.37E+01	9.24E+00	1.16E+01	2.78E+00	4.37E-01	**1.80E-01**
		1.46E+00	1.32E+00†	1.77E+01†	9.75E+00†	1.15E+01†	4.21E+00†	1.86E+00†	**4.56E-01‡**
	T2	3.93E+00	5.47E+00	1.92E+01	1.10E+01	1.20E+01	**2.31E+00**	5.03E+00	4.09E+00
		7.67E+00	7.63E+00†	1.97E+01†	1.19E+01†	1.35E+01†	**3.90E+00‡**	8.56E+00†	6.46E+00
	T3	4.24E+00	6.69E+00	2.61E+01	9.48E+00	8.92E+00	**2.03E+00**	8.77E+00	4.29E+00
		6.78E+00	8.16E+00†	1.79E+01†	1.06E+01†	7.99E+00†	**3.65E+00‡**	1.07E+01†	6.45E+00
	T4	5.64E-01	1.47E+00	4.39E+00	8.22E+00	1.67E+01	3.39E+00	5.39E-01	**8.77E-02**
		9.95E-01	2.54E+00†	4.92E+00†	1.24E+01†	1.89E+01†	6.13E+00†	1.68E+00	**2.46E-01‡**
	T5	**4.43E-01**	6.25E-01	3.06E+01	2.29E+00	3.56E+00	6.10E-01	9.82E-01	9.48E-01
		1.22E+00	9.65E-01†	2.45E+01†	3.82E+00†	4.43E+00†	1.28E+00†	3.69E+00†	1.77E+00†
	T6	**7.29E-01**	1.71E+00	4.44E+01	1.94E+01	2.00E+01	1.27E+00	2.16E+00	1.77E+00
		1.52E+00	5.64E+00†	2.72E+01†	2.11E+01†	2.43E+01†	2.23E+00†	4.86E+00†	5.83E+00†
F2	T1	**2.65E-01**	3.92E+00	4.32E+01	1.18E+01	1.37E+01	1.40E+00	2.15E+00	9.28E-01
		1.59E+00	6.71E+00†	4.80E+01†	1.48E+01†	1.22E+01†	4.64E+00†	7.65E+00†	2.75E+00†

	T2	**6.10E-01**	4.77E+00	5.36E+01	8.89E+01	7.71E+01	4.82E+00	2.43E+01	4.41E+01
		2.42E+00	8.15E+00†	5.87E+01†	1.64E+02†	1.50E+02†	1.52E+01†	8.85E+01†	1.13E+02†
	T3	**2.54E-01**	3.63E+00	4.10E+01	9.77E+01	7.27E+01	5.93E+00	1.89E+01	5.26E+01
		1.83E+00	6.36E+00†	5.82E+01†	1.66E+02†	1.43E+02†	2.19E+01†	6.71E+01†	1.21E+02†
	T4	2.05E+00	5.04E+00	2.10E+02	3.16E+01	2.87E+01	**5.90E-01**	1.36E+00	7.36E-01
		6.35E+00	8.10E+00†	1.37E+02†	3.14E+01†	2.57E+01†	**1.36E+00‡**	4.73E+00‡	3.07E+00‡
	T5	**1.20E-02**	2.19E+00	5.44E+01	1.81E+02	1.24E+02	1.02E+01	4.69E+01	6.29E+01
		5.44E-02	5.43E+00†	5.68E+01†	1.90E+02†	1.72E+02†	3.01E+01†	1.11E+02†	1.36E+02†
	T6	**4.67E-03**	1.89E+00	6.59E+01	1.99E+01	1.29E+01	9.00E-01	2.76E+00	3.97E+00
		1.28E-03	5.03E+00†	7.81E+01†	3.48E+01†	1.77E+01†	1.74E+00†	4.95E+00†	1.47E+01†
F3	T1	1.38E+01	3.08E+01	**7.10E+00**	6.56E+02	5.97E+01	1.98E+01	1.53E+01	1.14E+01
		2.08E+01	7.05E+01†	**9.34E+00‡**	1.50E+02†	1.60E+02†	1.12E+02	6.60E+01	6.18E+01
	T2	1.10E+01	3.16E+01	**7.28E+00**	9.13E+02	7.90E+02	9.86E+02	8.22E+02	5.63E+02
		2.08E+01	6.79E+01†	**1.03E+01‡**	1.97E+02†	2.48E+02†	8.01E+01†	1.97E+02†	3.27E+02†
	T3	1.67E+01	3.05E+01	**6.38E+00**	8.91E+02	7.10E+02	9.79E+02	6.91E+02	5.87E+02
		5.80E+01	7.32E+01†	**8.05E+00‡**	1.01E+02†	2.90E+02†	8.73E+01†	3.13E+02†	3.75E+02†
	T4	7.13E+01	7.53E+01	**2.08E+00**	7.94E+02	4.77E+02	6.01E+02	4.37E+02	6.61E+01
		1.32E+02	1.70E+02	**4.22E+00‡**	3.05E+02†	4.50E+02†	5.36E+02†	4.69E+02†	2.11E+02
	T5	2.82E+01	2.47E+01	**6.01E+00**	9.01E+02	7.24E+02	9.58E+02	6.99E+02	4.76E+02
		7.13E+01	5.35E+01	**7.80E+00‡**	6.07E+01†	1.93E+02†	9.48E+01†	3.26E+02†	3.27E+02†
	T6	1.38E+01	2.29E+01	**5.81E+00**	9.15E+02	5.34E+02	8.60E+02	6.33E+02	2.63E+02
		5.03E+01	5.18E+01†	**7.89E+00‡**	2.99E+02†	4.36E+02†	4.72E+02†	4.54E+02†	3.93E+02†

Table 5. The average errors and the standard deviations of the peer methods on the functions f_4-f_6. † indicates that CLA-MPD performs better than the compared algorithm by t-test. ‡ means that the corresponding algorithm is better than CLA-MPD.

		CLA-MPD	**EDESAC**	**bCCEA**	**ShrBA**	**PLBA**	**mSQDE-i**	**DASA**	**jDE**
F4	T1	**5.06E-01**	6.55E+00	5.25E+01	8.99E+00	2.03E+01	9.40E-01	5.48E+00	1.57E+00
		2.19E+00	9.66E+00†	7.83E+01†	1.12E+01†	1.91E+01†	3.13E+00†	2.87E+01†	4.88E+00†
	T2	**5.37E-01**	7.24E+00	6.84E+01	1.19E+02	2.90E+02	6.83E+00	6.60E+01	4.98E+01
		2.53E+00	1.08E+01†	9.29E+01†	2.12E+02†	2.67E+02†	1.98E+01†	1.46E+02†	1.33E+02†
	T3	**6.96E-01**	5.01E+00	5.94E+01	1.50E+02	1.49E+02	6.84E+00	5.30E+01	5.21E+01
		3.00E+00	8.26E+00†	8.35E+01†	2.29E+02†	2.30E+02†	1.93E+01†	1.48E+02†	1.57E+02†
	T4	1.64E+00	8.88E+00	3.18E+02	2.90E+01	1.64E+01	**6.00E-01**	3.64E+00	1.67E+00
		4.14E+00	1.55E+01†	1.18E+02†	3.03E+01†	2.09E+01†	**1.30E+00‡**	6.68E+00†	5.03E+00
	T5	**1.35E+00**	4.99E+00	9.12E+01	2.57E+02	2.50E+02	1.75E+01	1.21E+02	6.69E+01
		4.85E+00	8.48E+00†	1.15E+02†	2.30E+02†	2.37E+02†	5.42E+01†	1.89E+02†	1.49E+02†
	T6	**5.56E-03**	4.59E+00	7.75E+01	2.72E+01	2.47E+01	1.21E+00	2.35E+00	2.56E+00
		2.26E-03	7.86E+00†	1.02E+02†	8.76E+01†	6.62E+01†	3.20E+00†	8.54E+00†	5.85E+00†
F5	T1	**6.62E-02**	4.92E+00	7.55E+01	2.86E+01	1.04E+01	1.32E+00	9.75E-01	1.60E-01
		3.25E-01	7.95E+00†	1.88E+02†	2.09E+01†	1.19E+01†	2.56E+00†	3.42E+00†	1.24E+00†
	T2	**2.26E-02**	5.55E+00	7.76E+01	1.39E+01	1.49E+01	2.97E+00	2.11E+00	3.46E-01
		2.84E-02	8.74E+00†	1.82E+02†	1.84E+01†	1.92E+01†	5.30E+00†	4.75E+00†	1.39E+00†
	T3	7.01E-01	5.56E+00	7.13E+01	7.43E+00	1.28E+01	2.76E+00	9.38E-01	**3.61E-01**
		2.53E+00	8.56E+00†	1.65E+02†	1.03E+01†	1.50E+01†	4.53E+00†	3.27E+00†	**2.08E+00‡**
	T4	**8.88E-02**	5.21E+00	3.28E+01	2.76E+01	2.43E+01	1.28E+00	4.10E-01	1.08E-01
		2.85E-01	8.71E+00†	2.02E+01†	2.75E+01†	2.78E+01†	2.25E+00†	1.71E+00†	9.18E-01
	T5	**1.88E-02**	4.52E+00	1.00E+02	1.52E+01	1.56E+01	2.13E+01	2.43E+00	4.34E-01
		6.71E-03	7.85E+00†	2.31E+02†	1.60E+01†	1.71E+01†	1.30E+02†	5.65E+00†	1.58E+00†
	T6	**1.80E-02**	3.63E+00	7.13E+01	1.13E+01	1.06E+01	4.75E+00	4.61E-01	3.75E-01
		6.90E-03	7.05E+00†	1.81E+02†	1.47E+01†	1.51E+01†	1.81E+01†	1.70E+00†	9.87E-01†
F6	T1	3.53E+00	9.80E+00	1.70E+02	3.02E+01	9.34E+00	**3.13E+00**	6.33E+00	6.34E+00
		5.12E+00	1.29E+01†	2.62E+02†	1.76E+01†	1.38E+01†	**5.46E+00**	1.64E+01†	1.07E+01†
	T2	**6.42E+00**	1.16E+01	1.87E+02	4.74E+01	3.46E+01	1.95E+01	2.62E+01	1.05E+01
		1.10E+01	1.38E+01†	2.71E+02†	9.83E+01†	6.85E+01†	6.39E+01†	1.21E+02†	1.38E+01†
	T3	**6.74E+00**	8.98E+00	1.78E+02	6.97E+01	4.83E+01	2.67E+01	1.86E+01	1.04E+01
		6.96E+00	1.07E+01†	2.69E+02†	1.78E+02†	3.68E+02†	8.90E+01†	6.93E+01†	2.58E+01†
	T4	**5.13E+00**	9.31E+00	1.38E+02	3.33E+01	1.84E+01	6.90E+00	7.46E+00	6.24E+00
		7.87E+00	1.36E+01†	2.40E+02†	6.28E+01†	2.37E+01†	9.65E+00†	2.37E+01†	1.33E+01†
	T5	**1.16E+01**	1.93E+01	2.17E+02	1.95E+02	1.10E+02	6.03E+01	3.68E+01	1.50E+01
		1.77E+01	2.81E+01†	2.88E+02†	3.06E+02†	2.26E+02†	1.64E+02†	1.22E+02†	4.43E+01†
	T6	1.23E+01	2.09E+01	1.87E+02	2.87E+01	3.20E+01	1.30E+01	1.37E+01	**7.65E+00**
		1.82E+01	2.46E+01†	2.70E+02†	6.64E+01†	7.95E+01†	5.59E+01	5.47E+01	**1.08E+01‡**

Table 6. Summary of comparison results of CLA-MPD against other peer approaches.

f	EDESAC	bCCEA	ShrBA	PLBA	mSQDE-i	DASA	jDE
1-10	6/0/0	6/0/0	6/0/0	6/0/0	3/0/3	6/0/0	4/0/2
1-50	6/0/0	6/0/0	6/0/0	6/0/0	4/0/2	6/0/0	4/0/2
2	6/0/0	6/0/0	6/0/0	6/0/0	5/0/1	5/0/1	5/0/1
3	4/2/0	0/0/6	6/0/0	6/0/0	5/1/0	5/1/0	4/2/0
4	6/0/0	6/0/0	6/0/0	6/0/0	5/0/1	6/0/0	5/1/0
5	6/0/0	6/0/0	6/0/0	6/0/0	6/0/0	6/0/0	4/1/1
6	6/0/0	6/0/0	6/0/0	6/0/0	4/2/0	5/1/0	5/0/1
All	40/2/0	36/0/6	42/0/0	42/0/0	32/3/7	39/2/1	31/4/7

Considering the obtained results, it is evident that CLA-MPD is superior to the other compared methods. It outperforms ShrBA and PLBA in a statistically significant fashion over all tested instances. Also, CLA-MPD obtains better results on most of the tested problems in comparison to EDESAC and DASA. CLA-MPD performs weaker than bCCEA on f_3; however, it outperforms bCCEA over the remaining problems. It should be noted that CLA-MPD is the second best performer or is statistically indistinguishable from the second best performers on f_3. Considering the poor performance of bCCEA on problems other than f_3, it can be observed that CLA-MPD is more robust compared to bCCEA. When compared to jDE, CLA-MPD is statically superior on 31 test instances and obtains some distinguishable results on some benchmarks like f_2, f_3, f_4. Accordingly, it can be inferred that the proposed context-aware learning scheme and the adopted strategies are beneficial in dynamic environments. The comparison with the other multi-population adaptive DE method, mSQDE-i, also demonstrates the effectiveness of the presented adaptive DE approach. CLA-MPD obtains better results than mSQDE-i on 32 test instances, while its results are weaker on only 7 test cases. The consistently good performance over the majority of the benchmark instances indicates that the introduced schemes in CLA-MPD for tackling dynamically changing fitness landscapes do have an edge over the compared dynamic optimizers.

5.2.2. Evaluation based on the adaptability criterion

In this section, we compare the results of the proposed algorithm with those reported in [19]. Consequently, CLA-MPD is compared with the following algorithms: DASA [52], jDE [28], DynDE [52], doptaiNET [53], CPSO [54], and MDELiGO [19]. Table 7 and Table 8 give the average adaptability of the compared methods. Additionally, the average Freidman rank of each compared algorithm on each

function is provided in Table 9. Also, the convergence curves of CLA-MPD on some test instances are given in Fig. 12 and Fig. 13 to illustrate the convergence behavior of the proposed method.

Except for f_5, CLA-MPD obtains the best average Freidman ranks compared to the other methods. It obtains the best results on 28 test instances. On some test cases like f_3 with T_2, T_3, T_5, and T_6, CLA-MPD outperforms the other approaches with a large margin. f_3 is a composition of Rastrigin's functions, which is subjected to rotation. The rotated Rastrigin's function is considered as one of the hardest optimization benchmarks in the static optimization literature. Typically, approaches based on an ensemble of DE strategies with appropriate adaptive mechanisms can achieve good performances on this type of benchmarks (see CEC special sessions and competitions on real-parameter optimization for some instances [55]).

The low average error of CLA-MPD suggests that it can track and discover changing optima faster than the other compared methods. The appropriate convergence rate of CLA-MPD is also demonstrated in the convergence curves given in Fig. 13 and Fig. 14. Various factors of CLA-MPD contribute to this suitable convergence rates. The major factor responsible for this behavior is the proposed context-aware strategy selection mechanism. This mechanism enables the proposed method to adjust its behavior in a short time after encountering an environmental change. Additionally, the proposed ensemble of strategies enables the algorithm to follow various appropriate trajectories in complex environments. This, in turn, increases the speed and the accuracy of finding optima.

Table 7. Adaptability of each compared method on the problems f_1-f_3

		CLA-MPD	MDELiGO	DASA	jDE	DynDE	doptaiNET	CPSO
F1-10	T1	**2.5321e+00**	1.3955e+01	2.0224e+01	1.5415e+01	1.9888e+01	1.9981e+01	1.7463e+01
	T2	**7.9296e+00**	2.2616e+01	3.0788e+01	2.6483e+01	2.7175e+01	3.1010e+01	2.9666e+01
	T3	1.0122e+01	**4.1378e+00**	2.4145e+01	8.8520e+00	1.5555e+01	3.2060e+01	1.9615e+01
	T4	**4.1905e+00**	1.2698e+01	2.1991e+01	1.4331e+01	1.8552e+01	3.1092e+01	1.8120e+01
	T5	**2.4410e+00**	6.8326e+00	1.6079e+01	1.8807e+01	1.9391e+01	2.7747e+01	1.7832e+01
	T6	**3.3815e+00**	8.3474e+00	1.9842e+01	1.4563e+01	1.9709e+01	3.7988e+01	2.0526e+01
F1-50	T1	**5.5437e+00**	1.4028e+01	2.1120e+01	1.8423e+01	2.5229e+01	2.6309e+01	2.6811e+01
	T2	**1.3433e+01**	2.2264e+01	3.5528e+01	3.9134e+01	4.0343e+01	4.5845e+01	4.1944e+01
	T3	1.1723e+01	1.0361e+01	3.5502e+01	1.8628e+01	3.5635e+01	2.7071e+01	2.9606e+01
	T4	**3.6569e+00**	5.7260e+00	2.3287e+01	2.1632e+01	2.6609e+01	4.6668e+01	2.4936e+01
	T5	**4.7226e+00**	6.0327e+00	1.4778e+01	1.3800e+01	1.0625e+01	2.9513e+01	1.8274e+01
	T6	**8.5314e+00**	1.0723e+01	2.1656e+01	1.8981e+01	1.5940e+01	2.4776e+01	1.8410e+01
F2	T1	**2.5003e+01**	4.5930e+01	7.5504e+01	5.1271e+01	6.8118e+01	4.8020e+01	7.1341e+01
	T2	2.1777e+01	**1.4901e+01**	1.2240e+02	1.6758e+02	9.9032e+01	9.6421e+01	9.0728e+01

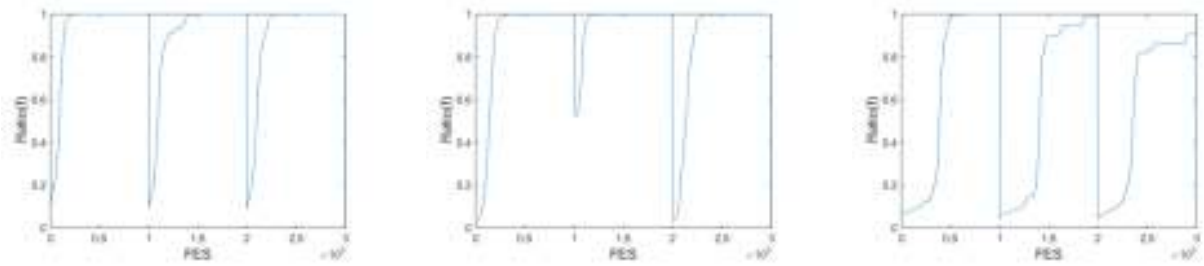
	T3	**2.0138e+01**	3.8882e+01	9.5797e+01	1.5994e+02	8.4678e+01	9.9275e+01	4.8944e+01
	T4	**2.4580e+01**	5.6044e+01	7.7503e+01	5.9131e+01	6.1141e+01	7.9904e+01	6.8494e+01
	T5	**1.4267e+01**	4.7152e+01	1.3406e+02	1.5861e+02	7.8448e+01	1.6815e+02	7.9605e+01
	T6	**1.2215e+01**	4.0506e+01	6.8778e+01	4.8635e+01	5.5767e+01	6.9765e+01	4.8776e+01
F3	T1	1.3580e+02	**9.4334e+01**	1.1117e+02	9.6233e+01	1.1634e+02	9.9131e+02	1.5054e+02
	T2	**1.9712e+02**	9.8854e+02	9.9450e+02	6.9403e+02	1.0221e+03	1.2311e+03	1.0476e+03
	T3	**1.8847e+02**	6.7059e+02	8.9888e+02	7.0136e+02	7.7089e+02	1.1059e+03	9.8518e+02
	T4	2.7062e+02	3.4425e+02	5.7039e+02	**2.0879e+02**	5.7214e+02	1.2522e+03	6.9225e+02
	T5	**1.3811e+02**	6.2300e+02	8.8159e+02	6.5050e+02	1.1348e+03	1.1257e+03	1.2516e+03
	T6	**9.5286e+01**	5.0056e+02	7.5254e+02	3.9880e+02	7.1317e+02	1.3911e+03	9.9674e+02

Table 8. Adaptability of each compared method on the problems f_4-f_6

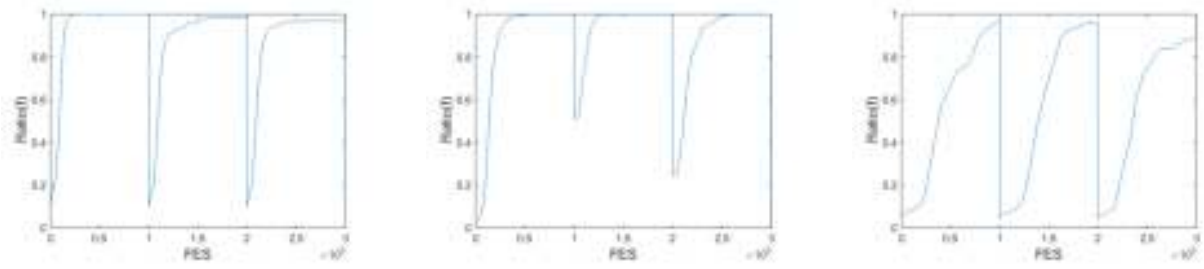
		CLA-MPD	MDELiGO	DASA	jDE	DynDE	doptaiNET	CPSO
F4	T1	2.9533e+01	**2.3051e+01**	6.0325e+01	3.6346e+01	3.9477e+01	3.2453e+01	4.0589e+01
	T2	3.0784e+01	**3.0312e+01**	9.8113e+01	8.8504e+01	8.7563e+01	2.1650e+02	6.7391e+01
	T3	**2.5978e+01**	2.8173e+01	1.0475e+02	9.6034e+01	8.7158e+01	1.5909e+02	9.7883e+01
	T4	**3.2844e+01**	3.7444e+01	5.0064e+01	4.8606e+01	4.3620e+01	5.9627e+01	4.8008e+01
	T5	**1.9948e+01**	3.2051e+01	1.6955e+02	9.9182e+01	9.2058e+01	4.1288e+02	9.9418e+01
	T6	**1.5341e+01**	4.4860e+01	5.8618e+01	5.7466e+01	5.5755e+01	7.9602e+01	6.1141e+01
F5	T1	6.2217e+01	6.0441e+01	6.9037e+01	**6.0136e+01**	6.8809e+01	2.0195e+02	7.9674e+01
	T2	7.4697e+01	**3.9736e+01**	4.6383e+01	4.5952e+01	7.3577e+01	1.5441e+02	7.7140e+01
	T3	7.0598e+01	**1.7304e+01**	4.9403e+01	4.7796e+01	5.4838e+01	6.2179e+01	8.8265e+01
	T4	8.6046e+01	**7.9683e+01**	8.8311e+01	8.3183e+01	8.9469e+01	2.1596e+02	9.4506e+01
	T5	4.9242e+01	**3.5159e+01**	4.0176e+01	4.6269e+01	9.8412e+01	1.2542e+03	9.2239e+01
	T6	**4.2546e+01**	4.4275e+01	4.7899e+01	4.7357e+01	7.5309e+01	5.4754e+02	8.8739e+01
F6	T1	**3.8867e+01**	5.6334e+01	8.5604e+01	5.8018e+01	1.0442e+02	1.9105e+02	6.3243e+01
	T2	5.4392e+01	**5.2352e+01**	1.4248e+02	6.2851e+01	1.3194e+02	4.8046e+02	1.3548e+02
	T3	5.1769e+01	5.2849e+01	8.8831e+01	**4.6495e+01**	1.2170e+02	5.5548e+02	1.3758e+02
	T4	**6.1676e+01**	9.2771e+01	9.4025e+01	8.3155e+01	1.0667e+02	2.5570e+02	9.9185e+01
	T5	**4.7950e+01**	7.0091e+01	1.5858e+02	7.6436e+01	1.5673e+02	1.2164e+03	1.4017e+02
	T6	**4.3789e+01**	5.1056e+01	1.2632e+02	6.1038e+01	1.1099e+02	9.4585e+02	1.2002e+02

Table 9. Average Friedman ranks of the compared methods based on the adaptability measure

f	CLA-MPD	MDELiGO	DASA	jDE	DynDE	doptaiNET	CPSO
F1-10	1.33	1.83	5.50	3.17	4.67	6.84	4.67
F1-50	1.17	1.83	4.67	3.67	4.83	6.33	5.50
F2	1.17	1.83	5.83	5.00	4.33	5.67	4.17
F3	1.83	2.33	4.17	2.17	4.67	6.67	6.17
F4	1.33	1.67	6.00	4.33	3.50	6.33	4.83
F5	3.67	1.33	3.50	2.17	4.67	6.67	6.00
F6	1.33	2.17	5.17	2.50	5.00	7.00	4.83

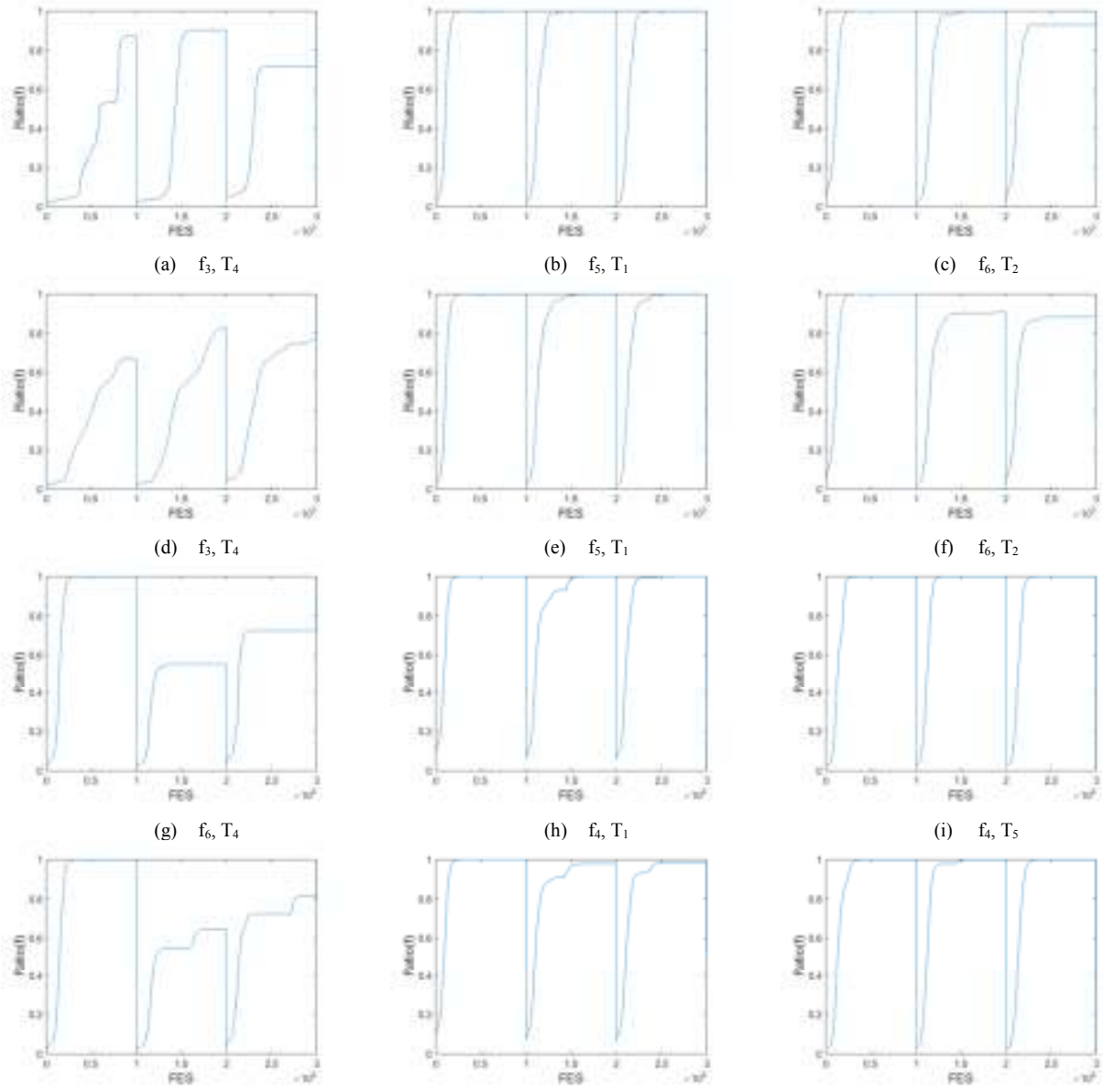


(a) f_2, T_3 (b) f_2, T_6 (c) f_3, T_3



(d) f_2, T_3 (e) f_2, T_6 (f) f_3, T_3

Figure 13. Convergence graphs for different GDBG problems: median convergence graphs on f_2 with T_3 (a), f_2 with T_6 (b), f_3 with T_3 (c); mean convergence graphs on f_2 with T_3 (d), f_2 with T_6 (e), f_3 with T_3 (f). FEs = The number of fitness evaluations.



(a) f_3, T_4 (b) f_5, T_1 (c) f_6, T_2

(d) f_3, T_4 (e) f_5, T_1 (f) f_6, T_2

(g) f_6, T_4 (h) f_4, T_1 (i) f_4, T_5

(j) f_6, T_4 (k) f_4, T_1 (r) f_4, T_5

Figure 14. Convergence graphs for different GDBG problems: median convergence graphs on f_3 with T_4 (a), f_5 with T_1 (b), f_6 with T_2 (c), f_6 with T_4 (g), f_4 with T_1 (h), and f_4 with T_5 (i); mean convergence graphs on f_3 with T_4 (d), f_5 with T_1 (e), f_6 with T_2 (f), f_6 with T_4 (j), f_4 with T_1 (k), and f_4 with T_5 (l). FEs = The number of fitness evaluations.

5.3. Complexity Issues

This section investigates the time complexity of the proposed method through asymptotic and experimental analysis.

5.3.1. Asymptotic analysis

Fitness evaluation is considered to be the most time-consuming operation; accordingly, the effectiveness of different nature-inspired optimization methods is evaluated based on an equal number of fitness evaluations. Here, we examine the time complexity of different building blocks of the proposed method without considering the cost of fitness evaluations. The most time-consuming operation of the proposed method (aside from fitness evaluation) is the distance calculation between different individuals. CLA-MPD requires the following distances to be calculated: the distances between the best individuals of different subpopulations, the distances between each pair of individuals within a subpopulation, and the distances between every individual and its nearby best individuals. To reduce the runtime of the algorithm, we maintain the former two types of distances in two different matrices. Each entry of these matrices is updated whenever one of the corresponding individuals is changed.

Using the maintained distances, the context of each subpopulation can be obtained in $O(N^2)$ time. The time complexity of action selection step of each subpopulation is O(1). The time complexities of mutation schemes 1, 2, and 3 for each subpopulation are $O(N\log(N)+ND)$, $O(N^2+ND)$, and $O(ND)$ respectively. Change detection step requires $O(M)$ time, and once a change is detected the re-initialization of all subpopulations can be performed in $O(MND)$ time (this initialization happens only once per each environmental change). The time complexity of maintaining the previously mentioned distance matrices is $O(MN^2D)$. CLA adaptation phase has a time complexity of $O(NM)$. The time complexity of examining convergence or overlapping in a subpopulation is $O(N^2 + M)$. Once a subpopulation is marked as overlapping or converged, it is reinitialized, which requires $O(ND)$ time. Also, the best individual of a fit

converged subpopulation is added to the external memory. Adding an individual to the memory requires at most O(MND) time (whenever the external memory is full). It should be noted that adding an individual to the external memory is an infrequent operation.

Considering the time complexities of building blocks of CLA-MPD, it can be inferred that the time complexity of each iteration of the proposed method is O(MN^2D). It should be noted that distance calculation is a very common operation in dynamic optimizers. For instance MDELiGO calculates the distance of every pair of individuals in each iteration, which induces an O(N_p^2D) cost on MDELiGO. Here, N_p is the size of the population (N_p=MN in CLA-MPD).

5.3.2. Experimental analysis

The runtime of the proposed method on different problems is experimentally analyzed in this section. We obtain the average runtime of CLA-MPD between two consecutive environmental changes for different test instances. The obtained results are compared with those of jDE, mSQDE-i, CPSO, MDELiGO, and NoOp. For a fair comparison, all algorithms are implemented in C++ and are tested on an Intel Core i7-2670QM CPU @2.2GHz system. jDE and mSQDE-i do not use any time-consuming operations in comparison to the canonical DE algorithm. However, CPSO and MDELiGO are relatively complex dynamic optimizers, which employ some expensive operations such as clustering and Eigen decomposition. The runtime of each algorithm *algo* on each test instance f can be represented by T_f+$T_{algo}(f)$. T_f represents the time which is spent on the fitness evaluations and the operations related to environmental changes. $T_{algo}(f)$ represents the time spent on the exclusive operations of the algorithm *algo*. T_f can be estimated using a simple procedure, which we call NoOp. Each iteration of NoOp consists of two operations: generation of a random individual within the search space and its evaluation using the GDBG system.

The comparison results are provided in Table 10. It is clear that most of the runtime of each algorithm is consumed on its fitness evaluations. Additionally, all compared methods have close execution times (mainly due to the large T_f values of different test scenarios). In general, jDE and mSQDE-i require less execution time than CLA-MPD as each generation of the proposed method contains relatively more

operations. As stated previously, the most time-consuming operation of the proposed method is the distance calculation between different individuals. MDELiGO requires more distance calculation operations than CLA-MPD. Additionally, it computes an $N_P \times N_P$ Delta matrix and its Eigen decomposition in each generation. These operations significantly increase the execution times of MDELiGO in comparison to the other peer methods. The proposed algorithm is also faster than CPSO, which utilizes some expensive operations such as clustering.

Table 10. Execution time different peer methods in seconds.

		NoOp	mSQDE-i	jDE	CPSO	MDELiGO	CLA-MPD
F2	T1	4.98	5.37	5.16	5.90	6.45	5.49
	T2	5.03	5.46	5.07	6.03	6.07	5.62
	T3	4.96	5.35	5.12	5.98	6.63	5.57
	T4	4.97	5.48	5.11	5.85	6.28	5.53
	T5	4.91	5.31	5.08	5.88	6.51	5.47
	T6	4.90	5.27	5.09	5.97	6.22	5.53
F3	T1	5.38	5.61	5.55	6.72	6.97	5.80
	T2	5.35	5.61	5.52	6.52	7.21	5.79
	T3	5.39	5.61	5.54	6.23	7.16	5.81
	T4	5.36	5.60	5.55	6.12	7.01	5.79
	T5	5.37	5.61	5.53	6.13	7.08	5.79
	T6	5.36	5.61	5.58	6.17	7.06	5.79
F4	T1	5.67	5.90	5.83	6.45	8.08	6.13
	T2	5.66	5.90	5.81	6.49	8.04	6.12
	T3	5.65	5.89	5.84	6.41	7.99	6.11
	T4	5.67	5.90	5.81	6.48	7.90	6.11
	T5	5.68	5.90	5.82	6.42	7.99	6.07
	T6	5.65	5.93	5.85	6.49	7.98	6.10
F5	T1	5.36	5.59	5.54	6.21	6.93	5.79
	T2	5.35	5.61	5.57	6.26	7.16	5.79
	T3	5.35	5.62	5.54	6.27	6.89	5.80
	T4	5.38	5.62	5.61	6.21	7.02	5.78
	T5	5.37	5.60	5.52	6.20	6.93	5.77
	T6	5.35	5.58	5.60	6.37	7.20	5.80
F6	T1	21.47	21.93	21.58	26.06	26.24	22.04
	T2	21.52	21.74	21.68	23.93	25.85	22.13
	T3	21.46	21.96	21.65	23.64	25.79	22.07
	T4	21.50	22.00	21.67	23.91	25.93	22.14
	T5	21.33	21.81	21.50	23.34	26.09	21.97
	T6	21.52	21.96	21.62	23.67	25.65	22.08

6. Conclusion

This paper presents a novel multi-population approach for optimization in dynamic environments. The proposed method incorporates a set of updating strategies which are specifically designed for dynamic environments. A context-aware learning mechanism controls the utilization of each strategy. This mechanism enables each subpopulation to choose a proper strategy according to its context state. Accordingly, it can appropriately react to drastic changes such as re-initialization and environmental

changes. The proposed learning approach is based on cellular learning automata. The algorithm maintains a CLA, and each cell of the CLA resides a subpopulation. The LAs of each cell control the evolutionary procedure of their associated subpopulation. Beside the context-aware learning mechanism, CLA-MPD utilizes some exclusion and memory schemes to enhance its search process. The exclusion scheme helps the proposed algorithm in diversifying its population. The utilized memory scheme helps the algorithm in tracking previously detected optima. Experimental results on GDBG dynamic optimization benchmark set demonstrated the effectiveness of the proposed dynamic optimization approach in comparison to some other significant dynamic optimization methods.

References

[1] A.E. Eiben, J.E. Smith, Introduction to evolutionary computing, Springer, 2003.
[2] X. Yu, M. Gen, Introduction to evolutionary algorithms, Springer Science & Business Media, 2010.
[3] T.T. Nguyen, S. Yang, J. Branke, Evolutionary dynamic optimization: A survey of the state of the art, Swarm and Evolutionary Computation, 6 (2012) 1-24.
[4] G. Wu, R. Mallipeddi, P.N. Suganthan, Ensemble strategies for population-based optimization algorithms–A survey, Swarm and evolutionary computation, 44 (2019) 695-711.
[5] H. Samma, C.P. Lim, J.M. Saleh, A new reinforcement learning-based memetic particle swarm optimizer, Applied Soft Computing, 43 (2016) 276-297.
[6] I. Fister, P.N. Suganthan, D. Strnad, J. Brest, I. Fister, Artificial neural network regression on ensemble strategies in differential evolution, in: 20th International Conference on Soft Computing, Mendel, 2014, pp. 65-70.
[7] X. Lu, K. Tang, B. Sendhoff, X. Yao, A new self-adaptation scheme for differential evolution, Neurocomputing, 146 (2014) 2-16.
[8] G. Koulinas, K. Anagnostopoulos, A new tabu search-based hyper-heuristic algorithm for solving construction leveling problems with limited resource availabilities, Automation in Construction, 31 (2013) 169-175.
[9] H.R. Topcuoglu, A. Ucar, L. Altin, A hyper-heuristic based framework for dynamic optimization problems, Applied Soft Computing, 19 (2014) 236-251.
[10] G. Wu, R. Mallipeddi, P.N. Suganthan, R. Wang, H. Chen, Differential evolution with multi-population based ensemble of mutation strategies, Information Sciences, 329 (2016) 329-345.
[11] H. Beigy, M.R. Meybodi, A mathematical framework for cellular learning automata, Advances in complex systems, 7 (2004) 295-319.
[12] M.R. Mirsaleh, M.R. Meybodi, A new memetic algorithm based on cellular learning automata for solving the vertex coloring problem, Memetic Computing, 8 (2016) 211-222.
[13] R. Rastegar, M. Meybodi, A new evolutionary computing model based on cellular learning automata, in: IEEE Conference on Cybernetics and Intelligent Systems, 2004., IEEE, 2004, pp. 433-438.
[14] R. Vafashoar, M. Meybodi, A.M. Azandaryani, CLA-DE: a hybrid model based on cellular learning automata for numerical optimization, Applied Intelligence, 36 (2012) 735-748.
[15] R. Vafashoar, M.R. Meybodi, Multi swarm bare bones particle swarm optimization with distribution adaption, Applied Soft Computing, 47 (2016) 534-552.
[16] R. Vafashoar, M.R. Meybodi, Multi swarm optimization algorithm with adaptive connectivity degree, Applied Intelligence, 48 (2018) 909-941.
[17] R. Vafashoar, M.R. Meybodi, Cellular learning automata based bare bones PSO with maximum likelihood rotated mutations, Swarm and evolutionary computation, 44 (2019) 680-694.
[18] W. Luo, J. Sun, C. Bu, H. Liang, Species-based particle swarm optimizer enhanced by memory for dynamic optimization, Applied Soft Computing, 47 (2016) 130-140.
[19] R. Mukherjee, S. Debchoudhury, S. Das, Modified differential evolution with locality induced genetic operators for dynamic optimization, European Journal of Operational Research, 253 (2016) 337-355.
[20] W. Zhang, W. Zhang, G.G. Yen, H. Jing, A cluster-based clonal selection algorithm for optimization in dynamic environment, Swarm and Evolutionary Computation, (2018).
[21] S.K. Nseef, S. Abdullah, A. Turky, G. Kendall, An adaptive multi-population artificial bee colony algorithm for dynamic optimisation problems, Knowledge-based systems, 104 (2016) 14-23.
[22] M. Mavrovouniotis, F.M. Müller, S. Yang, Ant colony optimization with local search for dynamic traveling salesman problems, IEEE transactions on cybernetics, 47 (2017) 1743-1756.
[23] F.B. Ozsoydan, A. Baykasoglu, A multi-population firefly algorithm for dynamic optimization problems, in: 2015 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS), IEEE, 2015, pp. 1-7.
[24] H. Wang, D. Wang, S. Yang, A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems, Soft Computing, 13 (2009) 763-780.
[25] F.B. Ozsoydan, A. Baykasoğlu, Quantum firefly swarms for multimodal dynamic optimization problems, Expert Systems with Applications,

115 (2019) 189-199.
[26] H. Ma, S. Shen, M. Yu, Z. Yang, M. Fei, H. Zhou, Multi-population techniques in nature inspired optimization algorithms: A comprehensive survey, Swarm and evolutionary computation, 44 (2019) 365-387.
[27] C. Li, T.T. Nguyen, M. Yang, S. Yang, S. Zeng, Multi-population methods in unconstrained continuous dynamic environments: The challenges, Information Sciences, 296 (2015) 95-118.
[28] J. Brest, A. Zamuda, B. Boskovic, M.S. Maucec, V. Zumer, Dynamic optimization using self-adaptive differential evolution, in: 2009 IEEE Congress on Evolutionary Computation, IEEE, 2009, pp. 415-422.
[29] M.C. Du Plessis, A.P. Engelbrecht, Differential evolution for dynamic environments with unknown numbers of optima, Journal of Global Optimization, 55 (2013) 73-99.
[30] L. Cao, L. Xu, E.D. Goodman, A neighbor-based learning particle swarm optimizer with short-term and long-term memory for dynamic optimization problems, Information Sciences, 453 (2018) 463-485.
[31] S. Das, A. Mandal, R. Mukherjee, An adaptive differential evolution algorithm for global optimization in dynamic environments, IEEE transactions on cybernetics, 44 (2014) 966-978.
[32] M. Kamosi, A.B. Hashemi, M.R. Meybodi, A hibernating multi-swarm optimization algorithm for dynamic environments, in: 2010 second world congress on nature and biologically inspired computing (NaBIC), IEEE, 2010, pp. 363-369.
[33] R.I. Lung, D. Dumitrescu, Evolutionary swarm cooperative optimization in dynamic environments, Natural Computing, 9 (2010) 83-94.
[34] X. Zheng, H. Liu, A cooperative dual-swarm pso for dynamic optimization problems, in: 2011 Seventh International Conference on Natural Computation, IEEE, 2011, pp. 1131-1135.
[35] U. Halder, S. Das, D. Maity, A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments, IEEE transactions on cybernetics, 43 (2013) 881-897.
[36] L. Cao, L. Xu, E.D. Goodman, A collaboration-based particle swarm optimizer with history-guided estimation for optimization in dynamic environments, Expert Systems with Applications, 120 (2019) 1-13.
[37] A.M. Turky, S. Abdullah, A multi-population electromagnetic algorithm for dynamic optimisation problems, Applied Soft Computing, 22 (2014) 474-482.
[38] A.M. Turky, S. Abdullah, A multi-population harmony search algorithm with external archive for dynamic optimization problems, Information Sciences, 272 (2014) 84-95.
[39] 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, Applied Soft Computing, 13 (2013) 2144-2158.
[40] K.S. Narendra, M.A. Thathachar, Learning automata: an introduction, Courier Corporation, 2012.
[41] M.A. Thathachar, P.S. Sastry, Networks of learning automata: Techniques for online stochastic optimization, Springer Science & Business Media, 2011.
[42] S. Wolfram, A new kind of science, Wolfram media Champaign, IL, 2002.
[43] H. Beigy, M.R. Meybodi, Cellular learning automata with multiple learning automata in each cell and its applications, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 40 (2010) 54-65.
[44] M. Esnaashari, M.R. Meybodi, Irregular cellular learning automata, IEEE transactions on cybernetics, 45 (2015) 1622-1632.
[45] B. Sareni, L. Krahenbuhl, Fitness sharing and niching methods revisited, IEEE transactions on Evolutionary Computation, 2 (1998) 97-106.
[46] C. Li, S. Yang, T. Nguyen, E.L. Yu, X. Yao, Y. Jin, H. Beyer, P. Suganthan, Benchmark generator for CEC 2009 competition on dynamic optimization, in, 2008.
[47] W.A. Hussein, S.N.H.S. Abdullah, S. Sahran, The Patch-Levy-Based Bees Algorithm Applied to Dynamic Optimization Problems, Discrete Dynamics in Nature and Society, 2017 (2017).
[48] C.-K. Au, H.-F. Leung, Cooperative coevolutionary algorithms for dynamic optimization: an experimental study, Evolutionary Intelligence, 7 (2014) 201-218.
[49] S. Hui, P.N. Suganthan, Ensemble differential evolution with dynamic subpopulations and adaptive clearing for solving dynamic optimization problems, in: 2012 IEEE Congress on Evolutionary Computation, IEEE, 2012, pp. 1-8.
[50] P. Novoa-Hernández, C.C. Corona, D.A. Pelta, Self-adaptive, multipopulation differential evolution in dynamic environments, Soft Computing, 17 (2013) 1861-1881.
[51] J. Brest, P. Korošec, J. Šilc, A. Zamuda, B. Bošković, M.S. Maučec, Differential evolution and differential ant-stigmergy on dynamic optimisation problems, International Journal of Systems Science, 44 (2013) 663-679.
[52] R. Mendes, A.S. Mohais, DynDE: a differential evolution for dynamic optimization problems, in: 2005 IEEE Congress on Evolutionary Computation, IEEE, 2005, pp. 2808-2815.
[53] F.O. De França, F.J. Von Zuben, A dynamic artificial immune algorithm applied to challenging benchmarking problems, in: 2009 IEEE Congress on Evolutionary Computation, IEEE, 2009, pp. 423-430.
[54] S. Yang, C. Li, A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments, IEEE Transactions on Evolutionary Computation, 14 (2010) 959-974.
[55] Competitions-benchmarks, benchmarks for evaluation of evolutionary algorithms. http://www.ntu.edu.sg/home/EPNSugan/, 2019 (accessed 10 March 2019).