Taylor & Francis
Taylor & Francis Group

Journal of Experimental & Theoretical Artificial Intelligence

ISSN: 0952-813X (Print) 1362-3079 (Online) Journal homepage: https://www.tandfonline.com/loi/teta20

# A Two-Level Function Evaluation Management Model for Multi-Population Methods in Dynamic Environments: Hierarchical Learning Automata Approach

Javidan Kazemi Kordestani, Mohammad Reza Meybodi & Amir Masoud Rahmani

**To cite this article:** Javidan Kazemi Kordestani, Mohammad Reza Meybodi & Amir Masoud Rahmani (2020): A Two-Level Function Evaluation Management Model for Multi-Population Methods in Dynamic Environments: Hierarchical Learning Automata Approach, Journal of Experimental & Theoretical Artificial Intelligence, DOI: 10.1080/0952813X.2020.1721568

**To link to this article:** https://doi.org/10.1080/0952813X.2020.1721568

Published online: 05 Feb 2020.

JOURNAL OF EXPERIMENTAL & THEORETICAL ARTIFICIAL INTELLIGENCE
https://doi.org/10.1080/0952813X.2020.1721568

Taylor & Francis
Taylor & Francis Group

ARTICLE



# A Two-Level Function Evaluation Management Model for Multi-Population Methods in Dynamic Environments: Hierarchical Learning Automata Approach

Javidan Kazemi Kordestani[a], Mohammad Reza Meybodi[b] and Amir Masoud Rahmani[a]

[a]Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran; [b]Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran

**ABSTRACT**

The fitness evaluation (FE) management has been successfully applied to improve the performance of multi-population methods for dynamic optimisation problems (DOPs). In this work, we extend one of its variants to address DOPs which was recently proposed by the authors. The aim of our proposal is to increase the efficiency of the FE management. To this end, we propose a technique based on hierarchical learning automata that manages FEs at two level: at first level the algorithm decides which population should be executed, and at the second level it specifies the operation that should be performed by the selected population. A detailed experimental analysis shows the effectiveness of our proposal.

**ARTICLE HISTORY**
Received 10 May 2019
Accepted 21 January 2020

**KEYWORDS**
Dynamic optimisation problems; differential evolution; moving peaks benchmark; evolutionary computation; hierarchical learning automata; function evaluation management

## Introduction

Several real-world optimisation problems are dynamic, meaning that the optimal solution(s) may change over time. A few real-world examples of such problems include dynamic resource allocation in shared hosting platforms (Shirali, Kazemi Kordestani, & Meybodi, 2018), dynamic railway junction rescheduling problem with multiple delays (Jayne, Yang, & Mavrovouniotis, 2016), vehicle routing problem with online arrival of customers (Pillac, Gendreau, Guéret, & Medaglia, 2013), dynamic multicast routing in MANETs (Haribaskar & Karnan, 2013), contaminant source identification problem in water distribution networks (Liu, Ranjithan, & Mahinthakumar, 2011) and shortest path routing in a changing network environment (Yang, Cheng, & Wang, 2010). In general, several processes of science and engineering involve the optimisation of a set of complex problems, in which the objectives of the optimisation, some restrictions or other elements of the problems may vary over time. In these cases, the optimum solution(s) to the problem are subject to change as well. As can be understood from above examples, unlike static optimisation problems, the goal of optimisation in dynamic problems is no longer locating the optimal solution(s), but tracking their trajectories over time with a high level of accuracy.

Each dynamic optimisation problem $\mathcal{P}$ can be defined by a quintuple $\{\Omega, \vec{x}, \phi, f, t\}$ where $\Omega$ denotes to the search space, $\vec{x}$ is a feasible solution in $\Omega$, $\phi$ represents the system control parameters which determine the distribution of the solutions in the fitness landscape, $f$ is the static objective function and $t$ is the time. $\mathcal{P}$ is then can be modelled as follows (Li, Yang, & Pelta, 2011):

**CONTACT** Mohammad Reza Meybodi mmeybodi@aut.ac.ir Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran

$$\mathcal{P} = \sum_{t=0}^{end} f_t(\vec{x}, \phi) \tag{1}$$

As can be seen in the above equation, the dynamic optimisation problem $\mathcal{P}$ is composed of a sequence of static instances. Hence, the goal of the optimisation in such problems is no longer just locating the optimal solution(s), but rather tracking the shifting optima over the time.

The dynamism of the problem can then be obtained by tuning the system control parameters as follows:

$$\phi_{t+1} = \phi_t \oplus \Delta\phi \tag{2}$$

where $\Delta\phi$ is the deviation of the control parameters from their current values and $\oplus$ represents the way the parameters are changed. The next state of the environment then can be defined using the current state of the environment as follows:

$$f_{t+1}(\vec{x}, \phi) = f_t(\vec{x}, \phi_t \oplus \Delta\phi) \tag{3}$$

Different change types can be defined, using $\Delta\phi$ and $\oplus$.

Due to their adaptive nature, evolutionary algorithms (EAs) and swarm intelligence methods (SIMs) have proven to be good optimisers for dynamic optimisation problems (DOPs) (Cruz, González, & Pelta, 2011; Nguyen, Yang, & Branke, 2012). Over the past years, various attempts have been done by researchers to improve the efficiency of traditional EAs and SIMs on DOPs. According to (Nguyen et al., 2012) the existing proposals can be grouped into the following approaches:

(i) increasing the diversity after detecting a change in the environment (Cobb, 1990; Hu & Eberhart, 2002; Vavak, Jukes, & Fogarty, 1997, 1998),
(ii) maintaining diversity during the optimisation process (Janson & Middendorf, 2006; Mori, Kita, & Nishikawa, 2001),
(iii) employing memory schemes to retrieve information about previously found solutions (Branke, 1999; Wang, Wang, & Yang, 2007),
(iv) predicting the location of the next optimal solution(s) after a change is detected (Hatzakis & Wallace, 2006; Rossi, Abderrahim, & Díaz, 2008),
(v) making use of the self-adaptive mechanisms of EAs, SIMs and other meta-heuristics (Grefenstette, 1999),
(vi) using multiple sub-populations to handle separate areas of the search space concurrently (Blackwell & Branke, 2006; Hashemi & Meybodi, 2009a, 2009b; Kamosi, Hashemi, & Meybodi, 2010a, 2010b; Kazemi Kordestani, Abedi Firouzjaee, & Meybodi, 2018; Li & Yang, 2008; Nabizadeh, Rezvanian, & Meybodi, 2012; Noroozi, Hashemi, & Meybodi, 2011, 2012; Sharifi, Noroozi, Bashiri, Hashemi, & Meybodi, 2012; Yang & Li, 2010).

Among the above-mentioned approaches, multi-population approach has shown to be very effective for handling DOPs, especially for multimodal fitness landscapes. The success of this approach can be contributed to three reasons (Li, Nguyen, Yang, Yang, & Zeng, 2015):

(i) As long as different populations search in different sub-areas in the fitness landscape, the overall population diversity can be maintained at the global level.
(ii) It is possible to locate and track multiple changing optima simultaneously. This feature can facilitate tracking of the global optimum, given that one of the being-tracked local optima may become the new global optimum when changes occur in the environment.
(iii) It is easy to extend any single-population approach, e.g. diversity increasing/maintain schemes, memory schemes, adaptive schemes, etc., to multi-population version.

Although being effective, the current multi-population approaches suffer from two shortcomings, from function evaluation (FE) management point of view, that limit their usefulness: (1) FEs are equally distributed among the populations, and (2) all populations go through the same process. To address the first issue, Kazemi Kordestani, Ranginkaman, Meybodi, and Novoa-Hernández (2019) proposed an FE management scheme using learning automaton (LA). In this work, an approach is introduced which addresses both the issues at the same time. Thus, the novelty of the present work is that it seeks to combine FE management with a mechanism to control the operation performed by each population.

The rest of the paper is structured as follows: First, a brief overview on multi-population methods for dynamic environments is given in Section 2. The FE management for DOPs is introduced in Section 3. In Section 4, learning automata and hierarchical learning automata are briefly reviewed. In Section 5, we explain our two-level FE management approach for multi-population methods. In Section 6, we describe how to apply our proposal in DynDE (a well-known multi-population algorithm for DOPs). An experimental study for analysing the impact of the proposed approach is presented in Section 7. Finally, Section 8 concludes the paper with some future works.

## Multi-population approach for DOPs

The following sections review some of the well-known multi-population algorithms for DOPs in terms of the way they determine the number of populations.

### *Multi-population methods with a fixed number of populations*

The main idea of these methods is to divide the task of optimisation among a number of fixed-size populations. One way to do this is by establishing a mutual repulsion among a predefined number of sub-populations to place them over different promising areas of the search space. The representative work in this category is that of Blackwell and Branke (2004). They proposed two multi-swarm algorithms based on the particle swarm optimisation (PSO), namely mCPSO and mQSO. In mCPSO, each swarm is composed of neutral and charged particles. Neutral particles update their velocity and position according to the principles of pure PSO. On the other hand, charged particles move in the same way as neutral particles, but they are also mutually repelled from other charged particles residing in their own swarm. Therefore, charged particles help to maintain the diversity inside the swarm. In mQSO, instead of having charged particles, each swarm contains quantum particles. Quantum particles change their positions around the centre of the best particle of the swarm according to a random uniform distribution with radius $r_{cloud}$. Consequently, they never converge and provide a suitable level of diversity to swarm in order to follow the shifting optimum. The authors also introduced the *exclusion* operator, which prevents populations from settling on the same peak. A complete survey on the exclusion operator can be found in (Kazemi Kordestani, Meybodi, & Rahmani, 2019).

After the introduction of mQSO, similar ideas were adopted in other algorithms such as DynDE (Mendes & Mohais, 2005).

Different from the above studies, some researchers combined desirable features of various optimisation algorithms into a single collaborative method for DOPs. In these methods, each population plays a different role and information can be shared between populations. For example, Lung and Dumitrescu (2007) introduced a hybrid collaborative method called collaborative evolutionary-swarm optimisation (CESO). CESO has two equal-size populations: a main population to maintain a set of local and global optimum during the search process using crowding-based DE, and a PSO population acting as a local search operator around solutions provided by the first population. During the search process, information is transmitted between both populations via collaboration mechanisms. In another work (Lung & Dumitrescu, 2010), a third population is incorporated to CESO which acts as a memory to recall some promising information from past generations of the algorithm. Inspired by CESO, a dual-population method with multiple strategies was also used in (Kazemi Kordestani, Rezvanian, & Meybodi, 2014).

### Methods with a variable number of populations

Another group of studies has investigated the multi-population schemes with a variable number of sub-populations. These methods can be further categorised into (1) methods with a parent population and variable number of child populations, (2) methods based on population clustering, and (3) methods based on space partitioning.

#### Methods with a parent population and variable number of child populations

The major strategy in this approach is to divide the search space into different sub-regions, using a parent population, and carefully exploit each sub-region with a distinct child population. The pioneer work in this category is self-organising scouts (SOS) algorithm proposed by Branke et al. (Branke, Kaussler, Smidt, & Schmeck, 2000). In SOS, a large parent population regularly explores the entire fitness landscape with the aim to find promising areas. When such areas are located by the parent, a child population is split off from the parent population and independently explores the respective sub-space. The search area of each child population is defined as a sphere with radius $r$ and centred at the best individual. The size of each population, including the parent, is calculated using a quality measure as defined in Branke et al., 2000.

Later, similar ideas have also been adopted by other authors in multi-swarm PSO algorithms. For instance, Li and Yang (2008) proposed FMSO that uses a large parent swarm to locate the promising areas of the search space. Whenever the quality of the best particle of the parent swarm improves, a child swarm is created with the best particle and particles within a specific radius from the best one. Another similar approach can be found in Kamosi et al., 2010a, 2010b.

#### Methods based on population clustering

Another way to create multiple populations is to divide the main population of the algorithm into several clusters, i.e. sub-populations, via different clustering methods. For instance, Parrott and Li (2006) proposed a speciation-based PSO for tracking multiple optima in dynamic environments, which dynamically distributes particles of the main swarm over a variable number of so-called *species*. In (Bird & Li, 2007) the performance of speciation-based PSO was improved by estimating the location of the peaks using a least squares regression method. The speciation scheme was also adopted by Nasiri and Meybodi (2012) in a speciation-based firefly algorithm to address DOPs.

Similarly, Yang and Li (2010) proposed a clustering PSO (CPSO) for locating and tracking multiple optimums. CPSO employed a single linkage hierarchical clustering method to create a variable number of sub-swarms, and assign them to different promising sub-regions of the search space. Each created sub-swarm uses the PSO with a modified *gbest* model to exploit the respective region. In order to avoid different clusters from crowding, they applied a redundancy control mechanism. In this regard, when two sub-swarms are located on a single peak they are merged together to form a single sub-swarm. So, the worst-performing individuals of the sub-swarm are removed until its size is equal to a predefined threshold. In another work (Li & Yang, 2012), the fundamental idea of CPSO is extended by introducing a novel framework for covering undetectable dynamic environments.

Recently, Halder, Das, and Maity, (2013) proposed a cluster-based DE with external archive which uses *k-means* clustering method to create a variable number of populations.

#### Methods based on space partitioning

Another approach for creating multiple sub-populations from the main population is to divide the search space into several partitions and keep the number of individuals in each partition less than a predefined threshold. In this group, we find the proposal of Hashemi and Meybodi (2009a). Here, cellular automata are incorporated into a PSO algorithm (cellular PSO). In cellular PSO, the search space is partitioned into some equally sized cells using cellular automata. Then, particles of the swarm are allocated to different cells according to their positions in the search space. Particles residing in each cell use their personal best positions and the best solution found in their neighbourhood cells for searching

an optimum. Moreover, whenever the number of particles within each cell exceeds a predefined threshold, randomly selected particles from the saturated cells are transferred to random cells within the search space. In addition, each cell has a memory that is used to keep track of the best position found within the boundary of the cell and its neighbours. In another work (Hashemi & Meybodi, 2009b), the same authors improved the performance of cellular PSO by changing the role of particles to quantum particles just at the moment when a change occurs.

Inspired by the cellular PSO, Noroozi et al. (2011) proposed an algorithm based on DE, called CellularDE. CellularDE employs the *DE/rand-to-best/1/bin* scheme to provide local exploration capability for genomes residing in each cell. After detecting a change in the environment, the population performs a random local search for several upcoming iterations. Later, the authors improved the performance of CellularDE by using a peak detection mechanism and hill climbing local search (Noroozi et al., 2012).

### *Methods with an adaptive number of populations*

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

The critical drawback of the multi-swarm algorithm proposed in (Blackwell & Branke, 2006) is that the number of swarms should be defined before the optimisation process. This is a clear limitation in real-world problems where information about the environment might be not available. The very first attempt to adapt the number of populations in dynamic environments was made by Blackwell (2007). He developed an adaptive mQSO (AmQSO) which adaptively determines the number of swarms either by spawning new swarms into the search space or by destroying redundant swarms. In this algorithm, swarms are categorised into two groups: (1) *free* swarms, whose expansion, i.e. the maximum distance between any two particles in the swarm in all dimensions, is larger than a predefined radius $r_{conv}$, and (2) *converged* swarms. Once the expansion of a free swarm becomes smaller than a radius $r_{conv}$, it is converted to the converged swarm. In AmQSO, when the number of free swarms ($M_{free}$) is dropped to zero, a free swarm is initialised in the search space for capturing undetected peaks. On the other hand, free swarms are removed from the search space if $M_{free}$ is higher than a threshold $n_{excess}$. A similar idea was adopted in artificial fish swarm algorithm (Yazdani, Akbarzadeh-Totonchi, Nasiri, & Meybodi, 2012) and cuckoo search (Fouladgar & Lotfi, 2016).

Different from the above algorithms which use the expansion of the populations as a feedback parameter to adapt the number of populations, du Plessis and Engelbrecht (2013) proposed the dynamic population differential evolution (DynPopDE), in which the populations are adaptively spawned and removed based on their performance. In this approach, when all of the current populations fail to improve their fitness, i.e. $\forall k \in \kappa \rightarrow \Delta f_k(t) = |f_k(t) - f_k(t-1)| = 0$ where $\kappa$ is the set of current populations, DynPopDE produces a new population of random individuals in the search space. On the other hand, a population $k$ will be removed from the search space when it is marked for restart due to exclusion, and it has not improved since its last FEs ($\Delta f_k(t) \neq 0$).

Recently, Li, Nguyen, Yang, Mavrovouniotis, and Yang (2016) proposed an adaptive multi-population framework to identify the correct number of populations. Their framework has three major procedures: clustering, tracking and adapting. Moreover, they have employed various components to further enhance the overall performance of the proposed approach, including a hibernation scheme, a peak hiding scheme, and two movement schemes for the best individuals.

Although majority of the reviewed multi-population methods in this section were based on PSO and DE, other multi-population metaheuristics were also proposed for DOPs such as artificial bee colony (Nseef, Abdullah, Turky, & Kendall, 2016), harmony search algorithm (Turky, Abdullah, & Dawod, 2018), bacterial foraging optimisation (Niu, Liu, & Wang, 2019), etc.

## Managing FEs in dynamic environments

Fitness evaluation is the most expensive component of the meta-heuristics for solving dynamic optimisation problems. From this point of view, dynamic optimisation can be considered as a computing resource management in which the objective is to allocate the major portion of FEs around the most promising areas of the search space.

FE management schemes for DOPs can be divided into two major categories: (1) approaches for reducing the wastage of computing resources (2) approaches for intelligent distribution of FEs among populations. The proposed FE management in this paper belongs to the second category.

### *Approaches for reducing the wastage of computing resources*

One way to manage the FEs is by reducing the wastage of computing resources, i.e. FEs, with the aim of saving FEs and make more FEs available to be spent around the most promising areas of the search space. Therefore, the first step is to identify the causes of FE dissipation and then find new ways to deal with them.

The very first straightforward source of FE wastage is allocating FEs to sub-populations which are not contributing to the search process. In this case, the unproductive populations are stopped by the algorithm until they become productive again (typically after a change in the environment). Various studies exist in the literature that used different techniques to detect and stop the unproductive populations. For example, du Plessis and Engelbrecht (2008) proposed an extension to the DynDE referred to as favoured populations DE. In this method, at each environment, weaker populations are frozen for the specific number of generations. Kamosi et al. (Kamosi et al., 2010b) proposed an interesting mechanism called *hibernation* for stopping the execution of unproductive populations in a multi-swarm algorithm. In their method, when the radius of a child swarm, say $c$, becomes less than a constant parameter $r_{conv}$, i.e. $(r_c < r_{conv})$, and the difference between fitness of the child swarm and the global best-found position is less than a threshold, i.e. $(f_c < f_{gbest} - \xi)$, the corresponding child swarm is deactivated. In turn, more FEs are available for productive populations.

In (Novoa-Hernández, Corona, & Pelta, 2011; Novoa-Hernández, Pelta, & Corona, 2010), the authors proposed a resource management mechanism, called *swarm control mechanism*, for DOPs. The swarm control mechanism is activated on the swarms with low diversity and bad fitness, and stops them from consuming FEs.

Recently, Sharifi et al. (Sharifi, Kazemi Kordestani, Mahdaviani, & Meybodi, 2015) studied different FE management schemes in a hybrid approach based on PSO and local search. Their first adapted algorithm uses the idea of hibernation to stop the local search agents that are not contributing to the search process.

### *Approaches for intelligent distribution of FEs among populations*

As the computing resources (i.e. fitness evaluations) are limited, it is wise to spend them around the most promising areas of the search space. In the current form of the multi-population methods, all populations are executed one by one based on their index. However, this strategy does not guarantee that the majority number of FEs are spent around the most promising areas of the search space. Kazemi Kordestani et al. (Kazemi Kordestani et al., 2019) pointed to three reasons why the equal distribution of FEs among populations is not a good choice:

(i) the main goal in dynamic optimisation is to locate the global optimum in a minimum amount of time and track its movements in the solution space. Therefore, strategies for quickly locating the global optimum are preferable.

(ii) a DOP may contain several peaks (local optimum). However, as the heights of the peaks are different, they do not have the same importance from the optimality point of view. Therefore, spending an equal number of FEs on all of them postpones the process of reaching a lower error, and
(iii) many real-world optimisation problems are large scale in nature. For these problems, the equal distribution of FEs among populations would be detrimental to the performance of the algorithms.

Therefore, the optimisation algorithms should be able to execute the populations in a manner that better-performing populations receive more FEs. To this end, du Plessis and Engelbrecht (2012) proposed a new mechanism, namely competitive population evaluation (CPE) to manage resources (i.e. FEs) in a way that enables the optimisation algorithm to reach the lowest error faster. In CPE, populations compete with each other to obtain FEs, which are allocated to populations according to their performance. The performance of each population is measured based on the current fitness of the best individual in the population and the improvement amount of the best individual during the previous evaluation of the population. Hence, the best-performing population will thus be the population with the highest fitness and improvement values. At each iteration, the best-performing population takes the FEs and evolves itself until its performance drops below that of another population, where the other population takes the resource, and this process continues during the run. Kazemi Kordestani et al. (2019) proposed two FE management methods to suitably exploit the FEs assigned to each sub-population. The first method combines the *success rate* and the quality of the best solution found by the sub-populations in a single criterion called *performance index*. The performance index is then used to distribute the FEs among populations. The second method uses variable-structure learning automata (VSLA) for choosing the sub-population that should be executed at each time step.

Off course, in all the above algorithms, the role of populations is fixed during the entire optimisation process. In contrast, in this work the operation performed by populations can vary over time.

## Theory of automata

In this section, learning automata are first introduced in brief. Then, hierarchical learning automata (HLA) is presented as the basis for our work.

### *Learning automata*

A learning automaton (Narendra & Thathachar, 1974, 2012) is an adaptive decision-making unit that improves its performance by learning the way to choose the optimal action from a finite set of allowable actions through iterative interactions with an unknown random environment. The action is chosen at random based on a probability distribution kept over the action-set and at each instant the given action is served as the input to the random environment. The environment responds the taken action in turn with a reinforcement signal. The action probability vector is updated based on the reinforcement feedback from the environment. The objective of a learning automaton is to find the optimal action from the action-set so that the average penalty received from the environment is minimised. Figure 1 shows the relationship between the learning automaton and random environment.

The environment can be described by a triple $E \equiv \{\alpha, \beta, c\}$ where $\alpha \equiv \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ represents the finite set of the inputs, $\beta \equiv \{\beta_1, \beta_2, \ldots, \beta_m\}$ denotes the set of the values that can be taken by the reinforcement signal, and $c \equiv \{c_1, c_2, \ldots, c_r\}$ denotes the set of the penalty probabilities, where the element $c_i$ is associated with the given action $\alpha_i$. If the penalty probabilities are constant, the random environment is said to be a stationary random environment, and if they vary with time, the environment is called a non-stationary environment. The environments depending on the nature of the

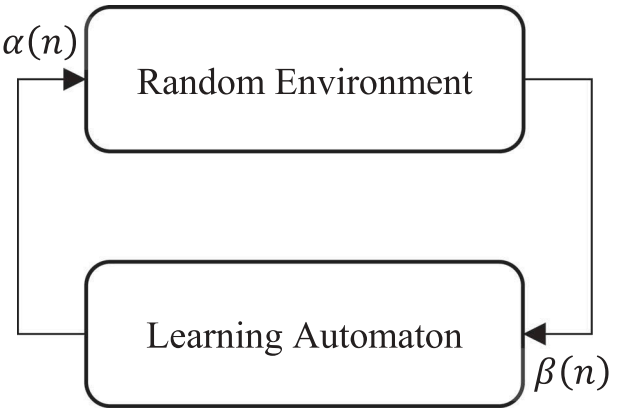


**Figure 1.** The schematic interaction of learning automata with an unknown random environment.

reinforcement signal $\beta$ can be classified into P-model, Q-model, and S-model. The environments in which the reinforcement signal can only take two binary values 0 and 1 are referred to as P-model environments. Another class of the environment allows a finite number of the values in the interval [0, 1] can be taken by the reinforcement signal. Such an environment is referred to as Q-model environment. In S-model environments, the reinforcement signal lies in the interval [a, b]. Learning automata can be classified into two main families (Narendra & Thathachar, 2012): fixed structure learning automata and variable-structure learning automata. In this work, we rely on a VSLA for choosing the sub-population that should be executed and the operation that should be performed at each time step. VSLA are represented by a quadruple $VSLA \equiv \{\beta, \alpha, p, T\}$ where $\beta \equiv \{\beta_1, \beta_2, \ldots, \beta_m\}$ is the set of inputs, $\alpha \equiv \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of actions, $p \equiv \{p_1, p_2, \ldots, p_r\}$ is the probability vector which determines the selection probability of each action and $T$ is learning algorithm which is used to modify the action probability vector, i.e. $p(n+1) = T[(n), (n), p(n)]$. Let $\alpha(n)$ and $p(n)$ denote the action chosen at instant $n$ and the action probability vector on which the chosen action is based, respectively. The recurrence equations shown by (4) and (5) are a linear learning algorithm by which the action probability vector $p$ is updated as follows:

$$p_j(n+1) = \begin{cases} p_j(n) + a.\big(1 - p_j(n)\big) & \text{if } i = j \\ p_j(n).(1-a) & \text{if } i \neq j \end{cases} \tag{4}$$

when the taken action is rewarded by the environment (i.e. $\beta(n) = 0$), and

$$p_j(n+1) = \begin{cases} p_j(n).(1-b) & \text{if } i = j \\ \frac{b}{r-1} + (1-b).p_j(n) & \text{if } i \neq j \end{cases} \tag{5}$$

when the taken action is penalised by the environment (i.e. $\beta(n) = 1$). In the above equations, $r$ is the number of actions. Finally, $a$ and $b$ denote the reward and penalty parameters and determine the amount of increases and decreases of the action probabilities, respectively. If $a = b$, the recurrence Equations (4) and (5) are called linear reward–penalty ($L_{R-P}$) algorithm, if $a \gg b$ the given equations are called linear reward–$\varepsilon$ penalty ($L_{R-\varepsilon P}$), and finally if $b = 0$ they are called linear reward–inaction ($L_{R-I}$). In the latter case, the action probability vectors remain unchanged when the taken action is penalised by the environment.

Learning automaton has been shown to perform well in parameter adjustment (Abedi Firouzjaee, Kazemi Kordestani, & Meybodi, 2017; Kazemi Kordestani, Ahmadi, & Meybodi, 2014; Mahdaviani, Kazemi Kordestani, Rezvanian, & Meybodi, 2015), networking (Rezvanian, Saghiri, Vahidipour, Esnaashari, & Meybodi, 2018), social networks (Rezvanian et al., 2018), etc.

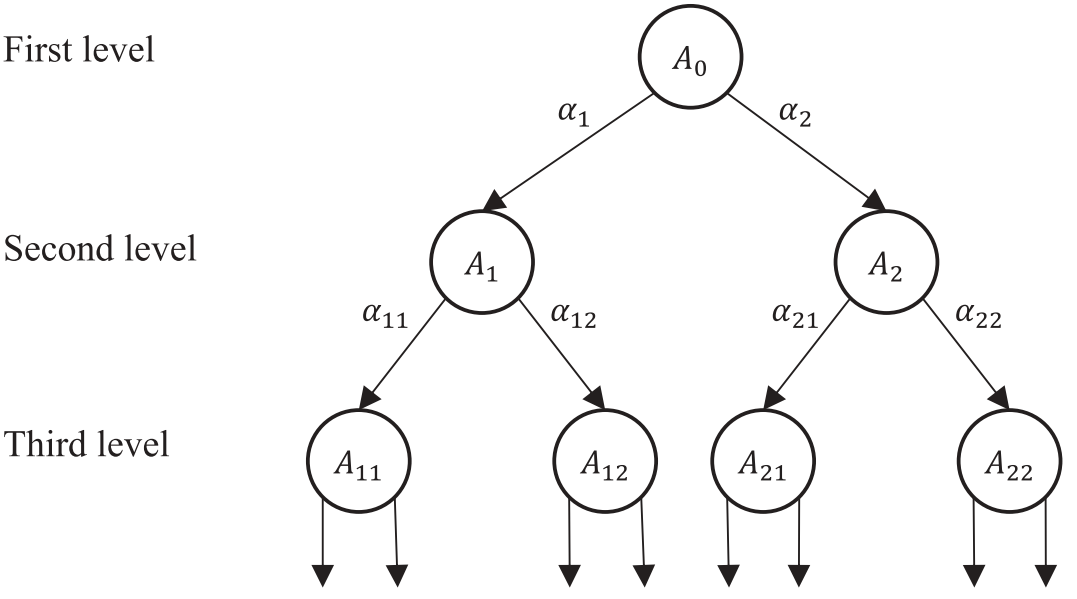


**Figure 2.** The schematic of a hierarchical system of learning automata.

### *Hierarchical learning automata*

A hierarchical learning automaton (Thathachar & Sastry, 1987) is a rooted tree structure where each vertex corresponds to an automaton and the edges emanating from that vertex correspond to actions of that automaton. The root vertex corresponds to an automaton called top-level automaton. Each of the actions of this automaton leads to a distinct automaton at the second level. In this way, the structure can be extended to an arbitrary number of levels. Figure 2 illustrates a schematic representation of a typical HLA with three levels.

The operation of this hierarchical learning system is as follows:

(1) At any instant $t$, the top-level automaton $A_0$ selects one of its actions, say $\alpha_1$ or $\alpha_2$, at random according to its action probability distribution.
(2) This activates the corresponding automaton $A_1$ or $A_2$, at the second level, which chooses an action. This, in turn, activates one of the automata in the third level, and so on.
(3) When the whole path is traversed, the environment responds with a set of $N$ feedbacks $\{\beta_1, \ldots, \beta_n\}$, one for each level.
(4) Using these feedbacks, the system updates the states of its automata through the learning algorithm.

By repeating the above procedure, the HLA will be gradually guided towards selecting the optimal path.

## Proposed method

The structure of the proposed scheduling method is given in Algorithm 1. In the rest of this subsection, the details of the proposed algorithm, including managing FEs at top level, managing FEs at second level, updating the probability vector of the automata and detection and response to environmental changes are described.

**Algorithm 1.** The proposed hierarchical learning automata based function management algorithm

```
Set the parameters N, pop_size;/*multi-population cardinality and population size*/
Randomly initialize N population of individuals in the search space;/*Initialize populations into the search space*/
Initialize a hierarchical learning automata with an N-action top-level learning automaton A_0 and N three-action second-level
    learning automata A_1 ... A_N;
Set action probability vector of A_0 to p_0 = {(1/N), (1/N), ..., (1/N)};
Set action probability vector of all A_i to p_i = {(1/3), (1/3), (1/3)};
while termination criterion not met do
  if not change in the environment is detected then
    Select a sub-population by A_0, say i, and activate the corresponding second-level A_i;
    Select an operation to be performed by sub-population i using A_i;
    Execute the selected operation by sub-population i;
    Compute the reinforcement signal β_1 and β_2 according to Equation (6);
    Update the probability vector A_i according to β_2;
    Update the probability vector A_0 according to β_1;
  else
    Re-evaluate all sub-populations;
    Set action probability of A_0 according to Equation (7);
    Reset the action probability vector of all second-level LAs;
  end
end
```

### Managing FEs at top level

The top-level LA $A_0$ is responsible for choosing the sub-population that should be executed. The $A_0$ is an $N$-action learning automaton, where its actions correspond to different sub-populations, i.e. $n_1, n_2, \ldots, n_N$. In order to select a new sub-population to be executed, the learning automaton $A_0$ selects one of its actions, e.g. $\alpha_{0i}$, according to its probability vector. Then, depending on the selected action, the corresponding $A_i$ is activated to select an operation among its set of allowable actions.

### Managing FEs at second level

The second-level automata are responsible for choosing the operation that should be performed by the selected sub-population. Different operations can be considered in this stage. In this paper, each second-level $A_i$ is a 3-action LA, which its actions correspond to different operations: *normal manner, local search around best solution found by its sub-population, local search around global best solution*. This way, a population either explores, exploits or donates its FEs to the best-performing individual of the algorithm. In order to choose an operation for the selected population $n_i$, corresponding learning automaton $A_i$ selects one of its actions, i.e. $\alpha_{i1}$, $\alpha_{i2}$ or $\alpha_{i3}$. Then, depending on the selected action, the operation for the selected population will be set accordingly.

Next, the LA $A_0$ and the activated $A_i$ update their probability vectors using the feedbacks received as the results of the execution of the evolved sub-population.

### Updating the probability vector of the automata

After the selected sub-population executed its operation, i.e. a path from top-level automaton to a leaf automaton is traversed, the environment responds with two reinforcement signals $\beta_1$ and $\beta_2$ one for each level. Based on the received reinforcement signals, the automata determine whether the taken actions were right or wrong, updating their probability vector accordingly. In this work, the reinforcement signals are generated as follows:

$$\textit{Reinforcement signal } \beta_1 = \beta_2 = \begin{cases} 0 & \textit{if } f\left(\vec{X}_{g,t}\right) < f\left(\vec{X}_{g,t+1}\right) \\ 1 & \textit{otherwise} \end{cases} \tag{6}$$

where $f$ is the fitness function, $\vec{X}_{g,t}$ is the global best individual of the algorithm at time step $t$. Equation 6 implies that if the quality of the best-found solution improves, then a favourable signal is received by the automata $A_0$, and $A_i$. Afterwards, the corresponding probability vectors of the system are modified using Equations (4) and (5).

### *Detection and response to the changes*

Several methods have been suggested for detecting changes in dynamic environments. See for example Nguyen et al., 2012 for a good survey on the topic. The mostly used method consists in re-evaluate memories of the algorithm for detecting inconsistencies in their corresponding fitness values. In this work, we considered the best solution for each sub-population. At the beginning of each iteration, the algorithm re-evaluates these solutions and compares their current fitness values against those from the previous iteration. If at least one inconsistency is found from these comparisons, then we will assume that the environment has changed.

Once a change is detected, all sub-populations are re-evaluated. Then, the action probability vector of the top-level automaton $A_0$ is modified as follows:

$$p_i = \frac{f_{g_i}}{\sum_{j=1}^{N} f_{g_j}} \tag{7}$$

where $p_i$ is the probability of selecting the ith sub-population, and $f_{g_i}$ is the fitness value of the best individual of the ith sub-population. Regarding Equation (7), the selection probability of each sub-population is modified according to its fitness value in the new environment. It means that the fittest sub-population in the new environment has a higher probability for being executed. Finally, the action probabilities of the second-level learning automata are set to their initial values.

## Application of the proposed FE management scheme

In what follows we describe how the proposed two-level FE management scheme can be incorporated into an existing multi-population method for DOPs. Although several options exist, we have considered DynDE, the algorithm proposed in (Mendes & Mohais, 2005). DynDE has been extensively studied in the past, showing very good results in challenging DOPs. However, we believe that it is necessary to describe the DE paradigm first, which is the underlying optimisation method in DynDE. Later, we give a brief overview of DynDE.

### *Differential evolution*

DE is one of the most powerful stochastic real-parameter optimisation algorithms, which was originally proposed by Storn and Price (1995, 1997). Over the past decade, DE becomes very popular because of its simplicity, effectiveness, and robustness.

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

(i) *Initialisation*: A number of $NP$ points are randomly sampled from the $D$-dimensional search space to form the initial population.
(ii) Repeat steps (iii), (iv) and (v) for each vector $\vec{x}_i$ ($i \epsilon \{1, 2, \ldots, NP\}$) of the current population.
(iii) *Mutation*: A mutant vector $\vec{v}_i$ is generated for $\vec{x}_i$ according to a specified mutation strategy.
(iv) *Repair*: if the mutant vector $\vec{v}_i$ is out of the feasible region, a repair operator is utilised to make it feasible.

(v) *Crossover*: A crossover operator is applied to combine the information from $\vec{x}_i$ and $\vec{v}_i$, and form a trial vector $\vec{u}_i$.
(vi) *Selection*: The vector with best fitness among $\vec{x}_i$ and $\vec{u}_i$ is transferred to the next generation.
(vii) If the termination condition is not met, go to step (ii).

Different extensions of DE can be specified using the general convention *DE/x/y/z*, where DE stands for 'Differential Evolution', *x* represents a string denoting the base vector to be perturbed, *y* is the number of difference vectors considered for perturbation of *x*, and *z* stands for the type of crossover being used, i.e. *exponential* or *binomial* (Das & Suganthan, 2011).

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

## DynDE

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

$$\vec{v} = \vec{x}_{best} + F.(\vec{x}_1 + \vec{x}_2 - \vec{x}_3 - \vec{x}_4) \tag{8}$$

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

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

$$v_{ij} = \begin{cases} lb_j & if\ v_{ij} < lb_j \\ ub_j & if\ v_{ij} > ub_j \end{cases} \tag{9}$$

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

$$u_{ij} = \begin{cases} v_{ij} & if\ rand_{ij}[0,1] \leq CR\ or\ j = jrand \\ x_{ij} & otherwise \end{cases} \tag{10}$$

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

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

$$\vec{x}_{Brownian} = \vec{x}_{best} + \vec{\mathrm{N}}(0, \sigma) \tag{11}$$

where $\vec{\mathrm{N}}(0,\sigma)$ is a Gaussian distribution with zero mean and standard deviation $\sigma$. As shown in (Mendes & Mohais, 2005), $\sigma = 0.2$ is a suitable value for this parameter.

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

$$r_{excl} = \frac{X}{2p^{1/d}} \tag{12}$$

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

### *Local search method*

In order to enhance the exploitation ability of each population, we use the naive directed local search (NDS) (Sharifi et al., 2012). In this method, an individual is used as the base point to start the local search as follows: For individual $i$, where $i$ is the best individual of a population or the global best solution, a binary direction vector $dir^i = \left(dir^i_1, dir^i_2, \ldots, dir^i_D\right), dir^i_j \in \{-1, 1\}$ is defined which specifies the direction of the search in each dimension. This vector is initialised randomly at the beginning of the local search. In each iteration, a random point around the individual $i$ is considered such that it only differs from individual $i$ in the jth dimension. The location of the selected random point is calculated using Equation (13):

$$x'_{i,j} = x_{i,j} + dir^i_j \left(\delta_{init} \times d^{ndrs}\right) \tag{13}$$

where $\delta_{init}$ is the initial step size of random search. $d$ is a constant in (0,1) which controls the speed of random search convergence. $ndrs_{i,j}$ is the number of unsuccessful random search performed on dimension j of the individual $i$. If the fitness of the new position is better than the fitness of $i$, then $i$ is replaced by the new position. Otherwise, the search is performed on the opposite direction on dimension j. If the search in the new direction for dimension j is not successful too, the search process in the corresponding dimension is failed. If the algorithm fails to improve the quality of the solution $i$ in all dimensions, then the $ndrs$ is increased. The above procedure is performed until the step size of the local search reaches a predefined threshold. For the sake of completeness and because of the important role that NDS plays in the proposed approach, here we give the pseudo-code for NDS in Algorithm 2.

**Algorithm 2.** Naive directed search algorithm

```
Initialize the position of the base point x randomly;
stepCounter: = 0;
for each dimension j do
    select a value for dir[j] from {-1,1} randomly
end-for
failedDims: = NULL;
repeat
    stepSize: = initialStepSize . discountFactor^stepCounter;
    for each Dimension j that j ∉ failedDims do
        x_j' = x_j + dir[j] . stepSize
        if fitness of x' is greater than fitness of x then
            x: = x';
        else
            dir[j]: = −dir[j];
            x_j' = x_j + dir[j] .stepSize;
            if fitness of x' is greater than fitness of x then
                x: = x';
            else
                Add j to failedDims Set;
            end-if
        end-if
    end-for
    if |failedDims| is equal to the number of dimensions of the problem space then
        Increase stepCounter by One;
        failedDims: = NULL;
    end-if
until stepSize > desiredStepSize
```

### Proposed DynDE+HLA

Bearing in mind the inner working of DynDE, it will be easy to understand how our proposals can be included in it. In short, the DynDE+HLA is summarised as follows:

(i) At the beginning of the run, randomly initialise sub-populations into the search space.
(ii) If a change in the environment is detected, re-evaluate all sub-populations and (*a*) modify action probability vector of the top-level automaton $A_0$ according to Equation (7), then (*b*) set the action probabilities of the second-level learning automata to their initial values.
(iii) Select a sub-population using top-level automaton.
(iv) Select an operation using the corresponding second-level automaton.
(v) Perform the selected operation.
(vi) Calculate the reinforcement signal $\beta_1$, $\beta_2$ according to Equation (6).
(vii) Update the probability vector $A_0$ and $A_1$ according to $\beta_1$and $\beta_2$.
(viii) Return to step (ii).

## Experimental study

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

### Dynamic test function

One of the most widely used dynamic test-beds in the literature is the *Moving Peaks Benchmark* (MPB) (Branke, 1999). MPB is a real-valued dynamic environment with a *D*-dimensional landscape consisting of *m* peaks, where the height, width and position of each peak are changed slightly every time a change occurs in the environment (Branke, 2002). Different landscapes can be defined by specifying the shape of the peaks and some other parameters. A typical peak shape is conical which is defined as follows:

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

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

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

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

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

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

where $\sigma_h$ and $\sigma_w$ are two random Gaussian numbers with zero mean and standard deviation one. Moreover, the shift vector $\vec{v}_{t+1}(i)$ is a combination of a random vector $\vec{r}$, which is created by drawing random numbers in [-0.5, 0.5] for each dimension, and the current shift vector $\vec{v}_t(i)$, and normalised

**Table 1.** Parameter settings for the moving peaks benchmark.

| Parameter | Default values (Scenario 2) | Other tested values |
|---|---|---|
| Number of peaks ($m$) | 10 | 1,2, 5, 7, 20, 30, 40, 50, 100, 200 |
| Height severity | 7.0 | |
| Width severity | 1.0 | |
| Peak function | Cone | |
| Number of dimensions ($d$) | 5 | 10, 15, 20, 25 |
| Height range ($H$) | $\in$ [30, 70] | |
| Width range ($W$) | $\in$ [1, 12] | |
| Standard height ($I$) | 50.0 | |
| Search space range ($A$) | $[0, 100]^d$ | |
| Frequency of change ($f$) | 5000 | 1000, 2000, 3000, 4000 |
| Shift severity ($s$) | 1 | 3, 5 |
| Correlation coefficient ($\lambda$) | 0.0 | |
| Basic function | No | |

to the length $s$. Parameter $\lambda \in [0.0, 1.0]$ specifies the correlation of each peak's changes to the previous one. This parameter determines the trajectory of changes, where $\lambda = 0$ means that the peaks are shifted in completely random directions and $\lambda = 1$ means that the peaks always follow the same direction, until they hit the boundaries where they bounce off.

In this paper, we consider the so-called *Scenario 2* which is the most widely used configuration of MPB. Unless stated otherwise, the MPB's parameters are set according to the values listed in Table 1 (*default values*). In addition, to investigate the effect of the environmental parameters (i.e. the change severity, change period, number of peaks, number of dimensions, number of peaks, and peak shape) on the performance of the proposed approach, various experiments were carried out with different combinations of other tested values listed in Table 1.

## Performance metric

Several criteria have been already proposed in the literature in order to measure the optimality of the optimisation algorithms for DOPs (Kazemi Kordestani, Rezvanian, & Meybodi, 2019). For measuring the efficiency of the tested algorithms, we considered the *offline error*, which is the most well-known metric for dynamic environments (Cruz et al., 2011). This measure is defined as the average of the smallest error found by the algorithm in every time step (Branke & Schmeck, 2003):

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

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

## Experimental results

The computational experiments were organised in order to study:

(1) Different parameter settings for the DynDE+HLA;
(2) The effect of varying the change frequency and shift severity;
(3) The effect of varying the number of peaks;
(4) The effect of varying the search space dimension;
(5) The behaviour of DynDE+HLA against some other multi-population methods.

In all experiments we performed 50 independent runs with different random seeds. Specifically, each run finished when the algorithm reaches 500,000 FEs.

To determine the differences among tested algorithms, we applied nonparametric tests. First, we applied the Friedman's test ($P < 0.05$) to verify whether significant differences exist at group level, that is, among all algorithms. In case if such differences exist, then a Wilcoxon test ($\alpha = 0.05$) is applied to compare our DynDE+HLA with best-performing algorithm among DynDE (Mendes & Mohais, 2005), DynDE+CPE (du Plessis & Engelbrecht, 2012), DynDE+PI (Kazemi Kordestani et al., 2019) or DynDE+LA (Kazemi Kordestani et al., 2019). The outcomes of the Wilcoxon test revealing that no significant differences exist between the algorithms are highlighted with an asterisk symbol.

In addition, when reporting the results, the last column of each table includes the improvement rate (%Imp) of the DynDE+HLA vs. the best of either DynDE, DynDE+CPE, DynDE+PI or DynDE+LA. This measure is computed for every problem instance $i$ as follows:

$$\%Imp_i = 100 \times \left(1 - \frac{e_{i,\ DynDE\ +\ HLA}}{e_{i,\ Best}}\right) \tag{20}$$

where $e_{i,\ DynDE\ +\ HLA}$ is the best (minimum) offline error, which is obtained by our algorithm for the problem $i$. Similarly, $e_{i,\ Best} = min\left\{e_{i,\ DynDE}, e_{i,\ DynDE\ +\ CPE}, e_{i,\ DynDE\ +\ PI}, e_{i,\ DynDE\ +LA}\right\}$.

The parameter settings of the DynDE, the NDS and the LA parameters of the proposed approach are as shown in Table 2.

In order to draw valid conclusions regarding the effect of different FE management strategies in DynDE, we have employed the same parameter settings.

### *Sensitivity analysis for DynDE+HLA for dynamic environments*

In this sub-section, effects of key parameters (i.e. learning rate $a$ and $b$, and initial probabilities $p_e$,$p_{sl}$ *and* $p_{gl}$) and components (i.e. second-level resource management scheme) on the performance of DynDE+HLA were analysed on different DOPs.

***Effect of learning rate*** $a_i$ ***and*** $b_i$**.** As shown in (Kazemi Kordestani et al., 2019), $a = 0.15$ and $b = 0.05$ is the best configuration for top-level automaton. In this section, the performance of DynDE+HLA is analysed using different configurations for $a_i$ and $b_i$ of the second-level LA. Here, we aim at obtaining some insights for suitable values for those parameters. To this end, in this experiment, we consider to perform multifactorial experiments over different problem instances with $f \in \{1000, 3000, 5000\}$ and $s \in \{1, 3, 5\}$. The configuration analysed for parameter $a_i$ and $b_i$ of the second-level LA are as follows: $a_i = \{0.00, 0.05, 0.10, 0.15, 0.20, 0.25\}$ and $b_i = \{0.00, 0.05, 0.10, 0.15, 0.20, 0.25\}$.

In order to have a better picture of the performance of DynDE+HLA, we have employed colour maps to represent the results of the experiments (see Figure 3). Each graphic comprises 36 values of average offline error corresponding to the possible combinations of parameters $a_i$ and $b_i$ in a particular DOP.

**Table 2.** Default parameter settings for DynDE, NDS and LA.

| Parameter | Default value |
|---|---|
| Number of populations | 10 |
| Number of Brownian individuals | 2 |
| Number DE individuals | 4 |
| $\sigma$ | 0.2 |
| Exclusion radius | 31.5 |
| $F$ | 0.5 |
| $CR$ | 0.5 |
| Initial step size ($\delta_{init}$) | 2.00 |
| Discount factor ($d$) | 0.10 |
| Desired step size ($S_{min}$) | 0.05 |
| Reward parameter for $LA_{top}$ ($a$) | 0.15 |
| Penalty parameter $LA_{top}$ ($b$) | 0.05 |
| Reward parameter for $LA_i$ ($a_i$) | 0.00 |
| Penalty parameter for $LA_i$ ($b_i$) | 0.15 |

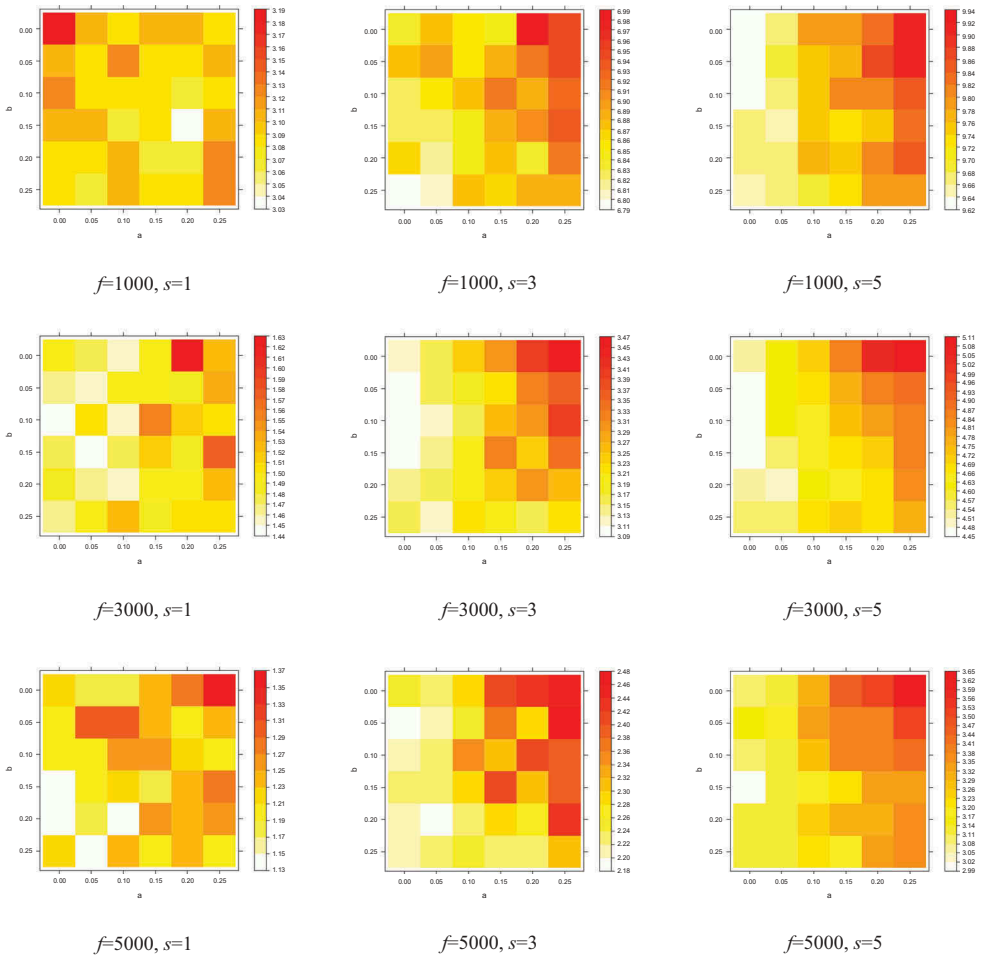


**Figure 3.** Colour maps for average offline error from DynDE+HLA with different combinations of parameters $a_i$ and $b_i$ in various instances of MPB.

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

***Effect of initial probabilities of second-level automata.*** As we mentioned before in Section 5, each second-level automaton has three actions: *normal manner, local search around best solution found by its sub-population, local search around global best solution*. Here, we show the corresponding initial probability of each action as $p_{nm}$, $p_{lb}$ and $p_{gb}$, respectively. The main goal of this experiment is to investigate the effect of the initial probabilities of $p_{nm}$, $p_{lb}$ and $p_{gb}$ on the performance of DynDE +HLA. It is expected that these parameters will have a significant influence on the performance of DynDE+HLA. Since the feasible region for $p_{nm} + p_{lb} + p_{gb}$ is large (See Figure 4), we only considered the values reported in Table 3.

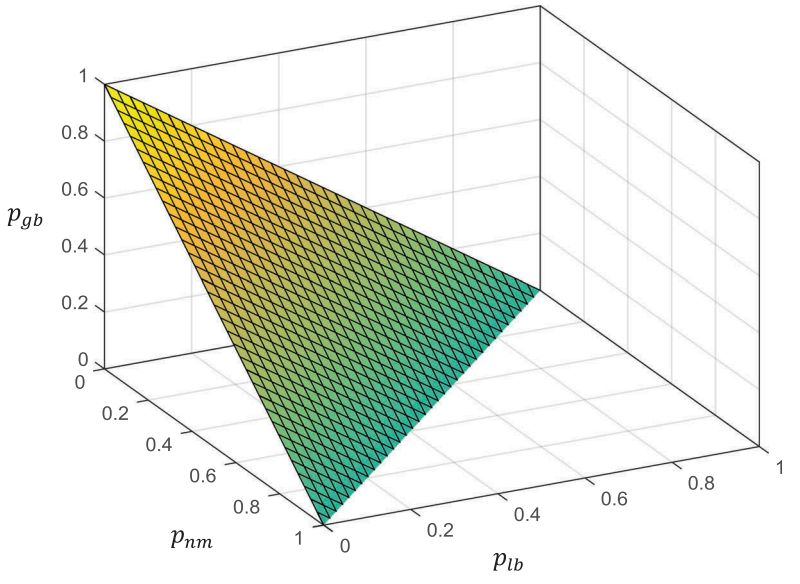


**Figure 4.** Surface plot for possible combination of parameters $p_{nm}$, $p_{lb}$ and $p_{gb}$.

As expected, $p_{nm}$, $p_{lb}$ and $p_{gb}$ affect the performance of DynDE+HLA. From Table 3, it can be observed that $p_{nm} = 0.0$, $p_{lb} = 0.8$ and $p_{gb} = 0.2$ is the worst choice for these parameters. The reason for this is that at the early stages of the optimisation process, the exploration ability of the algorithm is needed for locating the optima. On the other hand, as the individuals come closer to the optima, exploitation helps the algorithm to refine the candidate solutions. It is also observed in Table 3, that the combination of $p_{nm} = 0.6$, $p_{lb} = 0.2$ and $p_{gb} = 0.2$ produces the best results. However, as we consider that the optimisation process is unknown to the algorithm we consider the equal initial probabilities for all the parameters.

***Effect of local search parameters.*** The aim of this experiment is to investigate the effect of initial step size and discount factor on the performance of the proposed method. Similar to the first set of experiments, in this experiment we perform multifactorial experiments over different problem instances with $f \in \{1000, 3000, 5000\}$ and $s \in \{1, 3, 5\}$. The configurations analysed for parameter initial step size ($\delta_{init}$) and discount factor ($\delta_{init}$) of NDS is as follows: $\delta_{init} = \{2, 5, 10\}$ and $(d) = \{0.1, 0.2, 0.5\}$. The results are visualized in Figure 5 using colormaps.

As can be observed in Figure 5, the best performance corresponds to configuration $\delta_{init} = 2$ and $d = 0.1$.

### *Effect of varying the change frequency and shift length*

In this experiment, the effect of the change severity and change frequency on the performance of different DynDE variants is examined. The various combinations of ($f$, $s$) resulted in 15 different problem instances. The numerical results of different DynDE variants are reported in Table 4.

From Table 4 it is clearly observed that the proposed HLA strategy is the best FE management scheme among five and can clearly improve the performance of DynDE. The obtained results confirmed that controlling the operation performed by each sub-population gives significant benefits to DynDE. Note that for the problem instance with $s$= 1 and $f$= 1000 the result of DynDE +LA is better than the result of DynDE+HLA. However, this superiority is not statistically significant.

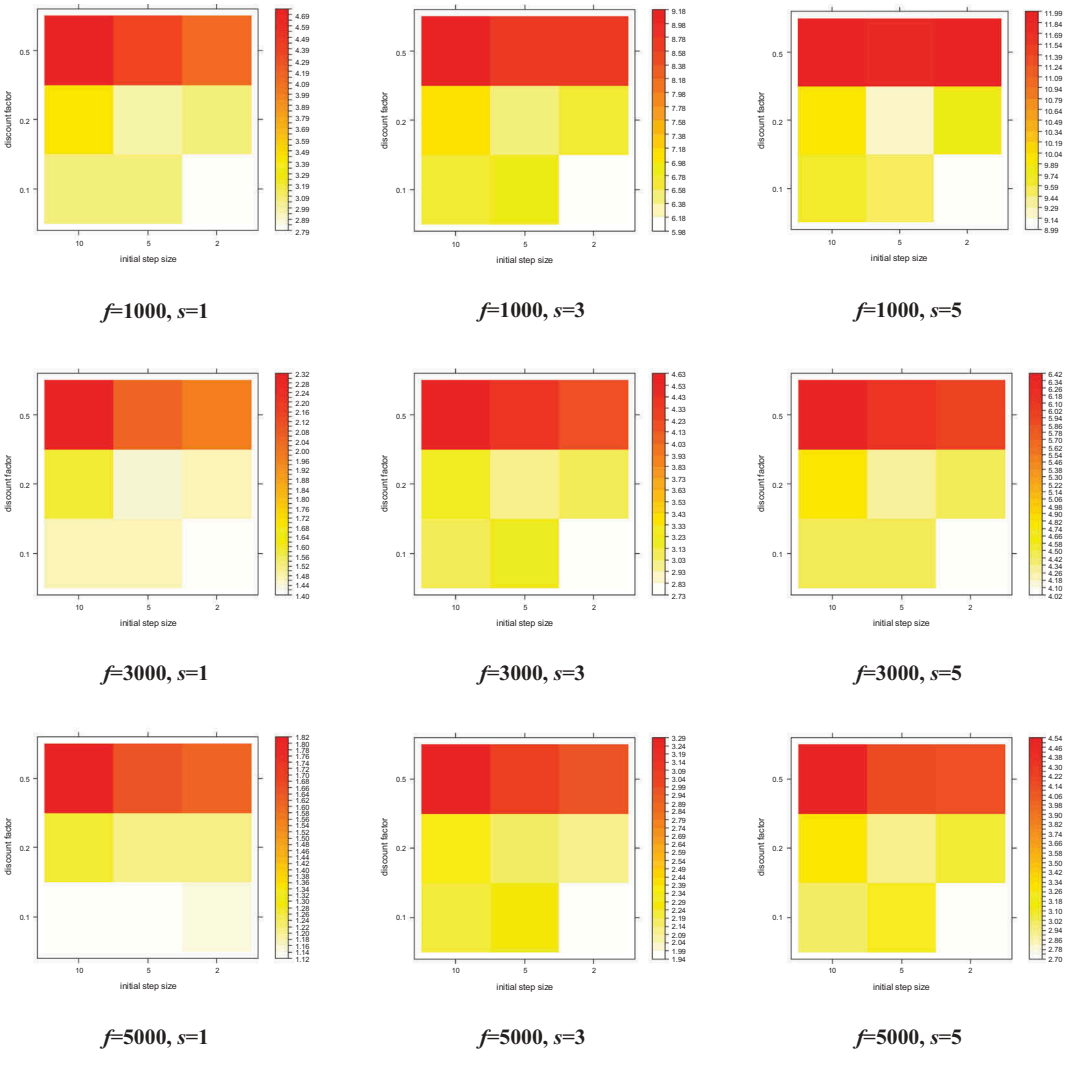


**Figure 5.** Colour maps for average offline error from DynDE+HLA with different combinations of parameters *a* and *b* in various instances of MPB.

### *Effect of varying number of peaks*

In this experiment, we studied how the proposed framework is affected by DOPs with different number of peaks. In this case, we considered the following setting $m \in \{1, 2, 5, 7, 10, 20, 30, 40, 50, 100, 200\}$. The other environmental parameters are set according to the default MPB settings shown in Table 1. The numerical results of DynDE with different FE management schemes are reported in Table 5.

Regarding Table 5, when the number of peaks is small, i.e. 1, 2 and 5, DynDE+HLA is considerably better than the other peer algorithms. For the problem with $m = 7$, although it is not significant, the result of DynDE+HLA is slightly better than that of DynDE+LA. As the number of peaks increases from 10 to 200, DynDE+HLA performs better than the best-performing counterpart by 7.50% on average. As the final remark, it is easy to observe that the proposed DynDE+HLA shows an overall superiority over the other methods.

**Table 3.** Effect of different initial probabilities of $p_{nm}$, $p_{lb}$ and $p_{gb}$ on the performance of the proposed method on MPB with Scenario 2.

| $p_{nm}$ | $p_{lb}$ | $p_{gb}$ | Results | $p_{nm}$ | $p_{lb}$ | $p_{gb}$ | Results | $p_{nm}$ | $p_{lb}$ | $p_{gb}$ | Results |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.0 | 1.0 | 1.18 ± 0.06 | 0.2 | 0.1 | 0.7 | 1.16 ±0.06 | 0.5 | 0.0 | 0.5 | 1.23 ± 0.07 |
| 0.0 | 0.1 | 0.9 | 1.17 ± 0.06 | 0.2 | 0.2 | 0.6 | 1.17 ± 0.06 | 0.5 | 0.1 | 0.4 | 1.16 ± 0.06 |
| 0.0 | 0.2 | 0.8 | 1.13 ± 0.06 | 0.2 | 0.3 | 0.5 | 1.20 ± 0.07 | 0.5 | 0.2 | 0.3 | 1.19 ± 0.06 |
| 0.0 | 0.3 | 0.7 | 1.20 ± 0.06 | 0.2 | 0.4 | 0.4 | 1.17 ± 0.06 | 0.5 | 0.3 | 0.2 | 1.11 ± 0.06 |
| 0.0 | 0.4 | 0.6 | 1.12 ± 0.05 | 0.2 | 0.5 | 0.3 | 1.14 ± 0.05 | 0.5 | 0.4 | 0.1 | 1.18 ± 0.06 |
| 0.0 | 0.5 | 0.5 | 1.18 ± 0.06 | 0.2 | 0.6 | 0.2 | 1.22 ± 0.06 | 0.5 | 0.5 | 0.0 | 1.20 ± 0.07 |
| 0.0 | 0.6 | 0.4 | 1.26 ± 0.07 | 0.2 | 0.7 | 0.1 | 1.18 ± 0.06 | 0.6 | 0.0 | 0.4 | 1.19 ± 0.06 |
| 0.0 | 0.7 | 0.3 | 1.13 ± 0.05 | 0.2 | 0.8 | 0.0 | 1.18 ± 0.06 | 0.6 | 0.1 | 0.3 | 1.12 ± 0.05 |
| 0.0 | 0.8 | 0.2 | 1.27 ± 0.07 | 0.3 | 0.0 | 0.7 | 1.20 ± 0.07 | 0.6 | 0.2 | 0.2 | **1.01 ± 0.04** |
| 0.0 | 0.9 | 0.1 | 1.20 ± 0.06 | 0.3 | 0.1 | 0.6 | 1.24 ± 0.09 | 0.6 | 0.3 | 0.1 | 1.16 ± 0.06 |
| 0.0 | 1.0 | 0.0 | 1.17 ± 0.06 | 0.3 | 0.2 | 0.5 | 1.17 ± 0.07 | 0.6 | 0.4 | 0.0 | 1.22 ± 0.07 |
| 0.1 | 0.0 | 0.9 | 1.24 ± 0.06 | 0.3 | 0.3 | 0.4 | 1.21 ± 0.06 | 0.7 | 0.0 | 0.3 | 1.22 ± 0.06 |
| 0.1 | 0.1 | 0.8 | 1.23 ± 0.07 | 0.3 | 0.4 | 0.3 | 1.18 ± 0.06 | 0.7 | 0.1 | 0.2 | 1.12 ± 0.04 |
| 0.1 | 0.2 | 0.7 | 1.25 ± 0.07 | 0.3 | 0.5 | 0.2 | 1.17 ± 0.07 | 0.7 | 0.2 | 0.1 | 1.20 ± 0.05 |
| 0.1 | 0.3 | 0.6 | 1.21 ± 0.07 | 0.3 | 0.6 | 0.1 | 1.23 ± 0.06 | 0.7 | 0.3 | 0.0 | 1.16 ± 0.07 |
| 0.1 | 0.4 | 0.5 | 1.18 ± 0.07 | 0.3 | 0.7 | 0.0 | 1.20 ± 0.07 | 0.8 | 0.0 | 0.2 | 1.17 ± 0.06 |
| 0.1 | 0.5 | 0.4 | 1.18 ± 0.06 | 0.4 | 0.0 | 0.6 | 1.24 ± 0.06 | 0.8 | 0.1 | 0.1 | 1.16 ± 0.06 |
| 0.1 | 0.6 | 0.3 | 1.17 ± 0.07 | 0.4 | 0.1 | 0.5 | 1.20 ± 0.06 | 0.8 | 0.2 | 0.0 | 1.14 ± 0.06 |
| 0.1 | 0.7 | 0.2 | 1.18 ± 0.05 | 0.4 | 0.2 | 0.4 | 1.22 ± 0.07 | 0.9 | 0.0 | 0.1 | 1.18 ± 0.06 |
| 0.1 | 0.8 | 0.1 | 1.19 ± 0.06 | 0.4 | 0.3 | 0.3 | 1.13 ± 0.05 | 0.9 | 0.1 | 0.0 | 1.19 ± 0.09 |
| 0.1 | 0.9 | 0.0 | 1.13 ± 0.07 | 0.4 | 0.5 | 0.1 | 1.19 ± 0.06 | 1.0 | 0.0 | 0.0 | 1.15 ± 0.06 |
| 0.2 | 0.0 | 0.8 | 1.23 ± 0.07 | 0.4 | 0.6 | 0.0 | 1.14 ± 0.06 | (1/3) | (1/3) | (1/3) | 1.16 ± 0.08 |

**Table 4.** Average offline error ± standard error of algorithms on DOPs with $m$= 10, $D$= 5 and varying number of shift lengths and change intervals.

| MPB Setting | | Method | | | | | |
|---|---|---|---|---|---|---|---|
| $s$ | $f$ | DynDE | DynDE+CPE | DynDE+PI | DynDE+LA | DynDE+HLA | %$imp$ |
| 1 | 1000 | 3.35 ± 0.04 | 3.15 ± 0.03 | 2.96 ± 0.05 | **2.83 ± 0.04** | **2.85 ± 0.05** | 0.00* |
| | 2000 | 2.31 ± 0.05 | 2.23 ± 0.05 | 2.05 ± 0.05 | 1.96 ± 0.04 | **1.71 ± 0.04** | 12.75 |
| | 3000 | 1.89 ± 0.05 | 1.78 ± 0.04 | 1.69 ± 0.06 | 1.65 ± 0.05 | **1.40 ± 0.06** | 15.15 |
| | 4000 | 1.67 ± 0.05 | 1.59 ± 0.04 | 1.56 ± 0.08 | 1.48 ± 0.06 | **1.26 ± 0.06** | 14.86 |
| | 5000 | 1.50 ± 0.05 | 1.49 ± 0.05 | 1.47 ± 0.08 | 1.32 ± 0.06 | **1.16 ± 0.08** | 12.12 |
| 3 | 1000 | 8.19 ± 0.06 | 7.96 ± 0.07 | 6.44 ± 0.07 | 6.48 ± 0.05 | **6.01 ± 0.05** | 6.67 |
| | 2000 | 5.13 ± 0.06 | 5.02 ± 0.05 | 4.01 ± 0.07 | 4.13 ± 0.06 | **3.66 ± 0.06** | 8.72 |
| | 3000 | 3.85 ± 0.07 | 3.71 ± 0.06 | 3.05 ± 0.08 | 3.18 ± 0.08 | **2.73 ± 0.07** | 10.49 |
| | 4000 | 3.14 ± 0.07 | 3.01 ± 0.06 | 2.52 ± 0.08 | 2.63 ± 0.07 | **2.33 ± 0.07** | 7.53 |
| | 5000 | 2.69 ± 0.07 | 2.54 ± 0.07 | 2.24 ± 0.08 | 2.23 ± 0.06 | **1.94 ± 0.08** | 13.00 |
| 5 | 1000 | 13.67 ± 0.10 | 12.37 ± 0.09 | 10.13 ± 0.09 | 10.44 ± 0.08 | **8.90 ± 0.07** | 12.44 |
| | 2000 | 8.66 ± 0.10 | 8.26 ± 0.09 | 6.13 ± 0.10 | 6.57 ± 0.09 | **5.38 ± 0.07** | 12.23 |
| | 3000 | 6.29 ± 0.09 | 6.09 ± 0.09 | 4.45 ± 0.10 | 4.88 ± 0.09 | **4.02 ± 0.08** | 9.66 |
| | 4000 | 4.98 ± 0.08 | 4.69 ± 0.08 | 3.59 ± 0.09 | 3.87 ± 0.08 | **3.13 ± 0.07** | 12.81 |
| | 5000 | 4.26 ± 0.10 | 4.11 ± 0.09 | 3.17 ± 0.10 | 3.33 ± 0.09 | **2.70 ± 0.09** | 14.82 |

### *Effect of increasing the search space dimension*

In this experiment, the performance of the five FE management schemes is investigated on the MPB with different dimensionalities $d \in \{5, 10, 15, 20, 25\}$. The other dynamic and complexity parameters are set according to Table 1. Table 6 shows the numerical results of different DynDE variants on DOPs with varying number of dimensions.

As can be seen in Table 6, an increase in the number of dimensions increases the complexity of the problem, which degrades the performance of all tested algorithms. Regarding the results in Table 6, DynDE+HLA can significantly outperform the other DynDE variants. Results also indicate that DynDE +HLA is less affected by the number of dimensions.

In order to get a better understanding of the performance of different FE management strategies, we performed a one-way ANOVA test with a 0.05 level of significance for all 50 instances of all algorithms for the MPB scenario 2. Table 7 presents whether pairwise performance comparisons

**Table 5.** Average offline error ± standard error of algorithms on DOPs with $D$=5, $s$= 1 and different number of peaks.

| $m$ | DynDE | DynDE+CPE | DynDE+PI | DynDE+LA | DynDE+HLA | %$imp$ |
|---|---|---|---|---|---|---|
| 1 | 3.80 ± 0.17 | 2.79 ± 0.16 | 2.70 ± 0.11 | 3.07 ± 0.12 | **0.92 ± 0.04** | 65.92 |
| 2 | 2.41 ± 0.17 | 1.93 ± 0.15 | 1.63 ± 0.13 | 1.88 ± 0.14 | **0.89 ± 0.03** | 45.39 |
| 5 | 1.64 ± 0.09 | 1.55 ± 0.08 | 1.32 ± 0.09 | 1.41 ± 0.08 | **0.98 ± 0.05** | 25.75 |
| 7 | 1.63 ± 0.08 | 1.59 ± 0.06 | 1.50 ± 0.09 | **1.32 ± 0.05** | **1.29 ± 0.09** | 2.27* |
| 10 | 1.50 ± 0.05 | 1.49 ± 0.05 | 1.47 ± 0.08 | 1.32 ± 0.06 | **1.16 ± 0.08** | 12.12 |
| 20 | 2.74 ± 0.07 | 2.66 ± 0.07 | 2.46 ± 0.08 | 2.60 ± 0.07 | **2.27 ± 0.07** | 7.72 |
| 30 | 3.22 ± 0.10 | 3.25 ± 0.11 | 2.91 ± 0.11 | 3.05 ± 0.10 | **2.79 ± 0.09** | 4.12 |
| 40 | 3.46 ± 0.08 | 3.53 ± 0.09 | 3.28 ± 0.09 | 3.34 ± 0.07 | **3.03 ± 0.08** | 7.62 |
| 50 | 3.81 ± 0.10 | 3.77 ± 0.09 | 3.38 ± 0.10 | 3.56 ± 0.09 | **3.22 ± 0.09** | 7.73 |
| 100 | 4.21 ± 0.12 | 4.15 ± 0.12 | 3.78 ± 0.11 | 3.88 ± 0.11 | **3.50 ± 0.08** | 7.40 |
| 200 | 3.99 ± 0.12 | 3.98 ± 0.11 | 3.62 ± 0.09 | 3.71 ± 0.09 | **3.41 ± 0.09** | 5.80 |

**Table 6.** Average offline error ± standard error of algorithms on DOPs with $m$= 10, $s$= 1 and different dimensionalities.

| $D$ | DynDE | DynDE+CPE | DynDE+PI | DynDE+LA | DynDE+HLA | %$imp$ |
|---|---|---|---|---|---|---|
| 5 | 1.50 ± 0.05 | 1.49 ± 0.05 | 1.47 ± 0.08 | 1.32 ± 0.09 | **1.16 ± 0.08** | 12.12 |
| 10 | 4.57 ± 0.17 | 4.33 ± 0.21 | 4.04 ± 0.24 | 3.81 ± 0.22 | **3.38 ± 0.23** | 11.28 |
| 15 | 6.83 ± 0.20 | 6.28 ± 0.21 | 5.88 ± 0.26 | 5.47 ± 0.23 | **4.61 ± 0.23** | 15.72 |
| 20 | 8.63 ± 0.25 | 7.96 ± 0.24 | 7.33 ± 0.26 | 7.00 ± 0.24 | **6.17 ± 0.26** | 11.86 |
| 25 | 11.37 ± 0.29 | 10.12 ± 0.30 | 8.90 ± 0.34 | 8.10 ± 0.27 | **7.25 ± 0.28** | 10.49 |

**Table 7.** One-way ANOVA test results based on offline error values on Scenario 2.

| | DynDE | DynDE+CPE | DynDE+PI | DynDE+LA | DynDE+HLA |
|---|---|---|---|---|---|
| DynDE | . | - | - | *S* - | *S* - |
| DynDE+CPE | + | . | - | *S* - | *S* - |
| DynDE+PI | + | + | . | *S* - | *S* - |
| DynDE+LA | *S* + | *S* + | *S* + | . | *S* - |
| DynDE+HLA | *S* + | *S* + | *S* + | *S* + | . |

between the algorithms are statistically significant or not. Each result given at the Xth row and the Yth column is marked with symbols 's+', 's-', '+' or '-' to indicate that the algorithm on the Xth row is significantly better than, significantly worse than, insignificantly better and insignificantly worse than the algorithm on the Yth column, respectively.

From Table 7, DynDE+HLA is the best-performing algorithm and DynDE+LA statistically outperforms all the other algorithms, except DynDE+HLA. Moreover, it is realised that all algorithms with FE management scheme are better than DynDE.

### *Comparison with other methods*

In this sub-section, we study the performance of the proposed method in comparison with several algorithms taken from the literature. The algorithms considered include RPSO (Hu & Eberhart, 2002), mQSO (T. Blackwell & Branke, 2006), mCPSO (T. Blackwell & Branke, 2006), SPSO (Parrott & Li, 2006), MEPSO (Du & Li, 2008), RVDEA/Mem (Woldesenbet & Yen, 2009), MDUMDA (Wu, Wang, Liu, & Ye, 2010), DHPSO (Karimi, Nobahari, & Pourtakdoust, 2012), and the proposed DynDE+HLA. Table 8 shows the numerical results of different algorithms on DOPs with different numbers of peaks $m \in \{1, 5, 10, 20, 30, 40, 50, 100, 200\}$. Notice that since the results of literature algorithms come from the references given above, certain algorithms like MEPSO, RVDEA/Mem, MDUMDA and DHPSO, have missing values. It means that these algorithms were not tested on the corresponding DOPs. In addition, see that we have highlighted the best results in boldface.

In order to compare the algorithms across all tested DOPs, a normalised score is calculated for each algorithm according to (Nguyen et al., 2012) as follows:

**Table 8.** Comparison of offline error of different multi-population algorithms on MPB's instances with different number of peaks. The normalised score for each algorithm on a DOP is printed in parentheses.

| | *m* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Method | 1 | 5 | 10 | 20 | 30 | 40 | 50 | 100 | 200 | *Overall score* |
| RPSO | 0.56 ± 0.04 (0.9933) | 12.22 ± 0.76 (0) | 12.98 ± 0.48 (0) | 12.79 ± 0.54 (0) | 12.35 ± 0.62 (0) | 11.37 ± 0.41 (0) | 11.34 ± 0.29 (0) | 9.73 ± 0.28 (0) | 8.90 ± 0.19 (0) | 0.1104 |
| mCPSO* | 4.93 ± 0.17 (0.0308) | 2.07 ± 0.08 (0.9030) | 2.05 ± 0.07 (0.9231) | 2.95 ± 0.08 (0.9353) | 3.38 ± 0.11 (0.9382) | 3.69 ± 0.11 (0.9208) | 3.68 ± 0.11 (0.9433) | 4.07 ± 0.09 (0.9085) | 3.97 ± 0.08 (0.8979) | 0.8231 |
| mQSO* | 5.07 ± 0.17 (0) | 1.81 ± 0.07 (0.9261) | 1.75 ± 0.07 (0.9484) | 2.74 ± 0.07 (0.9553) | 3.27 ± 0.11 (0.9497) | 3.60 ± 0.08 (0.9316) | 3.65 ± 0.11 (0.9470) | 3.93 ± 0.08 (0.9309) | 3.86 ± 0.07 (0.9180) | 0.8342 |
| SPSO | 2.64 ± 0.10 (0.5352) | 2.15 ± 0.07 (0.8959) | 2.51 ± 0.09 (0.8842) | 3.21 ± 0.07 (0.9106) | 3.64 ± 0.07 (0.9110) | 3.85 ± 0.08 (0.9016) | 3.86 ± 0.08 (0.9211) | 4.01 ± 0.07 (0.9181) | 3.82 ± 0.05 (0.9253) | 0.8671 |
| MEPSO | **0.53 ± 0.15 (1)** | 4.68 ± 1.01 (0.6708) | 4.02 ± 0.56 (0.7567) | 4.19 ± 0.57 (0.8174) | 4.27 ± 0.83 (0.8451) | N/A | 4.20 ± 0.47 (0.8793) | N/A | N/A | 0.8283 |
| RVDEA/Mem | 1.23 (0.8458) | N/A | 4.88 (0.6841) | 5.68 (0.6758) | 5.86 (0.6788) | 5.65 (0.6858) | 5.21 (0.7549) | 4.98 (0.7624) | 4.92 (0.7249) | 0.7266 |
| MDUMDA | 4.45 ± 0.69 (0.1365) | 1.53 ± 0.47 (0.9510) | **1.14 ± 0.39 (1)** | 2.99 ± 0.50 (0.9315) | 3.98 ± 0.74 (0.8755) | 4.05 ± 0.46 (0.8776) | 4.66 ± 0.54 (0.8226) | N/A | N/A | 0.7993 |
| DHPSO | 1.64 ± 0.01 (0.7555) | 1.62 ± 0.01 (0.9430) | 2.73 ± 0.03 (0.8657) | 2.82 ± 0.02 (0.9477) | 3.97 ± 0.03 (0.8765) | 4.67 ± 0.03 (0.8033) | 5.13 ± 0.03 (0.7647) | N/A | N/A | 0.8510 |
| DynDE+HLA | 0.92 ± 0.04 (0.9140) | **0.98 ± 0.05 (1)** | 1.16 ± 0.08 (0.9983) | **2.27 ± 0.07 (1)** | **2.79 ± 0.09 (1)** | **3.03 ± 0.08 (1)** | **3.22 ± 0.09 (1)** | **3.50 ± 0.08 (1)** | **3.41 ± 0.09 (1)** | **0.9903** |

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

$$Score(i) = \frac{1}{m}\sum_{j=1}^{m} \frac{|e_{max}(j) - e(i,j)|}{|e_{max}(j) - e_{min}(j)|}, \forall i = 1{:}n \tag{21}$$

where $e(i,j)$ is the offline error of $i$th algorithm on $j$th tested DOP; and $e_{max}(j)$ and $e_{min}(j)$ are the largest and smallest offline errors obtained by the algorithms on $j$th tested DOP. The normalised score is in the range of [0, 1] so that the best-performing algorithm on all $m$ tested DOPs gets an overall score 1. Similarly, the worst-performing algorithm on all $m$ tested DOPs will get an overall score 0.

As can be observed in Table 8, the proposed DynDE+HLA shows an overall superiority over the other tested methods. Only in two cases literature algorithms are slightly better than our method: (*a*) when the environment is unimodal (1 peak), MEPSO is the best-performing algorithm and (*b*) for DOP with $m = 10$, MDUMDA shows the best performance. For other tested DOPs, DynDE+HLA is the best-performing algorithm.

## Conclusion and future works

A new model based on hierarchical learning automaton has been presented in this paper to enhance the management of function evaluations in multi-population methods for dealing with dynamic optimisation problems (DOPs). The main motivation of the present study is the following questions about the multi-population approach in DOPs: (*a*) which sub-population should be selected for execution at each time, and (*b*) which operation should be performed by the selected sub-population. To this end, a model based on hierarchical learning automaton was introduced for addressing the above questions. The proposed method was then integrated into DynDE, a well-known multi-population algorithm for dynamic environments.

In order to validate the effectiveness of the proposed methods, a wide range of experiments were carried out on the moving peaks benchmark. The experimental results confirmed that the proposed approach is an effective method for handling DOPs.

Several research directions can be pursued as future works. First, the values of the parameters of the proposed method were adjusted based on some preliminary experiments. Therefore, a comprehensive sensitivity analysis on the effect of parameters could be a direction for future research. Second, the proposed approach can be used to improve the performance of other multi-population methods. Third, since change detection will not work in many real-world dynamic optimisation problems, e.g. noisy dynamic environments, it is interesting to propose a new method without change detection. Forth, it is also very valuable to apply the proposed approach to real-world dynamic problems.

## Disclosure statement

No potential conflict of interest was reported by the authors.

## References

Abedi Firouzjaee, H., Kazemi Kordestani, J., & Meybodi, M. R. (2017). Cuckoo search with composite flight operator for numerical optimization problems and its application in tunnelling. *Engineering Optimization*, *49*(4), 597–616.

Bird, S., & Li, X. (2007). Using regression to improve local convergence. *2007 IEEE Congress on Evolutionary Computation* pp. 592–599. doi:10.1109/CEC.2007.4424524

Blackwell, T., & Branke, J. (2006). Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, *10*(4), 459–472.

Blackwell, T. (2007). Particle swarm optimization in dynamic environments. In S. Yang, Y.-S. Ong, & Y. Jin (Eds.), *Evolutionary computation in dynamic and uncertain environments* (Vol. 51, pp. 29–49). Berlin Heidelberg: Springer.

Blackwell, T., & Branke, J. (2004). Multi-swarm optimization in dynamic environments. *Applications of Evolutionary Computing*, *3005*, 489–500.

Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, *3*, 1875–1882.

Branke, J. (2002). *Evolutionary optimization in dynamic environments*. Springer, Boston, MA: Kluwer Academic Publishers.

Branke, J., Kaussler, T., Smidt, C., & Schmeck, H. (2000). A multi-population approach to dynamic optimization problems. In I. C. Parmee(Ed., *Evolutionary design and manufacture: Selected papers from ACDM* (Vol. 00, pp. 299–307). London: Springer

Branke, J., & Schmeck, H. (2003). Designing evolutionary algorithms for dynamic optimization problems. In A. Ghosh & S. Tsutsui (Eds.), *Advances in evolutionary computing: Theory and applications* (pp. 239–262). Berlin Heidelberg: Springer. doi:10.1007/978-3-642-18965-4_9

Cobb, H. G. (1990). *An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments* (NRL-MR-6760, pp. 10–20). Washington DC: Naval Research lab

Cruz, C., González, J. R., & Pelta, D. A. (2011). Optimization in dynamic environments: A survey on problems, methods and measures. *Soft Computing*, *15*(7), 1427–1448.

Das, S., & Suganthan, P. N. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, *15*(1), 4–31.

du Plessis, M. C., & Engelbrecht, A. P. (2008). Improved differential evolution for dynamic optimization problems. *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)* pp. 229–234. doi:10.1109/CEC.2008.4630804

du Plessis, M. C., & Engelbrecht, A. P. (2012). Using competitive population evaluation in a differential evolution algorithm for dynamic environments. *European Journal of Operational Research*, *218*(1), 7–20.

du Plessis, M. C., & Engelbrecht, A. P. (2013). Differential evolution for dynamic environments with unknown numbers of optima. *Journal of Global Optimization*, *55*(1), 73–99.

Du, W., & Li, B. (2008). Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences*, *178*(15), 3096–3109.

Fouladgar, N., & Lotfi, S. (2016). A novel approach for optimization in dynamic environments based on modified cuckoo search algorithm. *Soft Computing*, *20*(7), 2889–2903.

Grefenstette, J. J. (1999). Evolvability in dynamic fitness landscapes: A genetic algorithm approach. *Proceedings of the 1999 Congress on Evolutionary Computation* pp. 2031–2038. doi:10.1109/CEC.1999.785524

Halder, U., Das, S., & Maity, D. (2013). A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments. *IEEE Transactions on Cybernetics*, *43*(3), 881–897.

Haribaskar, K., & Karnan, M. (2013). Artificial bee colony: For detecting dynamic shortest path routing problems in mobile ad hoc networks. *European Journal of Scientific Research*, *98*, 7–15.

Hashemi, A. B., & Meybodi, M. R. (2009a). Cellular PSO: A PSO for dynamic environments. In Z. Cai, Z. Li, Z. Kang, & Y. Liu (Eds., *Advances in computation and intelligence. ISICA 2009* (Vol. 5821, pp. 422–433). Berlin Heidelberg: Springer. .

Hashemi, A. B., & Meybodi, M. R. (2009b). A multi-role cellular PSO for dynamic environments. *Computer Conference, 2009. CSICC 2009. 14th International CSI* pp. 412–417. doi:10.1109/CSICC.2009.5349615

Hatzakis, I., & Wallace, D. (2006). Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach. *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* pp. 1201–1208. doi:10.1145/1143997.1144187

Hu, X., & Eberhart, R. C. (2002). Adaptive particle swarm optimization: Detection and response to dynamic systems. *Proceedings of the 2002 Congress on Evolutionary Computation*, *2*, 1666–1670.

Janson, S., & Middendorf, M. (2006). A hierarchical particle swarm optimizer for noisy and dynamic environments. *Genetic Programming and Evolvable Machines*, *7*(4), 329–354.

Jayne, E., Yang, S., & Mavrovouniotis, M. (2016). Ant colony optimization with immigrants schemes for the dynamic railway junction rescheduling problem with multiple delays. *Soft Computing*, *20*(8), 2951–2966.

Kamosi, M., Hashemi, A. B., & Meybodi, M. R. (2010a). A new particle swarm optimization algorithm for dynamic environments. In B. K. Panigrahi, S. Das, P. N. Suganthan, & S. S. Dash(Eds., *Swarm, evolutionary, and memetic computing, SEMCCO 2010* (Vol. 6466, pp. 129–138). Berlin Heidelberg: Springer.

Kamosi, M., Hashemi, A. B., & Meybodi, M. R. (2010b). A hibernating multi-swarm optimization algorithm for dynamic environments. *2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)* pp. 363–369. doi:10.1109/NABIC.2010.5716372

Karimi, J., Nobahari, H., & Pourtakdoust, S. H. (2012). A new hybrid approach for dynamic continuous optimization problems. *Applied Soft Computing*, *12*(3), 1158–1167.

Kazemi Kordestani, J., Abedi Firouzjaee, H., & Meybodi, M. R. (2018). An adaptive bi-flight cuckoo search with variable nests for continuous dynamic optimization problems. *Applied Intelligence*, *48*(1), 97–117.

Kazemi Kordestani, J., Ahmadi, A., & Meybodi, M. R. (2014). An improved differential evolution algorithm using learning automata and population topologies. *Applied Intelligence*, *41*(4), 1150–1169.

Kazemi Kordestani, J., Meybodi, M. R., & Rahmani, A. M. (2019). A note on the exclusion operator in multi-swarm PSO algorithms for dynamic environments. *Connection Science, in Press*. doi:10.1080/09540091.2019.1700912

Kazemi Kordestani, J., Ranginkaman, A. E., Meybodi, M. R., & Novoa-Hernández, P. (2019). A novel framework for improving multi-population algorithms for dynamic optimization problems: A scheduling approach. *Swarm and Evolutionary Computation*, *44*, 788–805.

Kazemi Kordestani, J., Rezvanian, A., & Meybodi, M. R. (2014). CDEPSO: A bi-population hybrid approach for dynamic optimization problems. *Applied Intelligence*, *40*(4), 682–694.

Kazemi Kordestani, J., Rezvanian, A., & Meybodi, M. R. (2019). New measures for comparing optimization algorithms on dynamic optimization problems. *Natural Computing*, *18*(4), 705–720.

Li, C., Nguyen, T. T., Yang, M., Mavrovouniotis, M., & Yang, S. (2016). An adaptive multipopulation framework for locating and tracking multiple optima. *IEEE Transactions on Evolutionary Computation*, *20*(4), 590–605.

Li, C., Nguyen, T. T., Yang, M., Yang, S., & Zeng, S. (2015). Multi-population methods in unconstrained continuous dynamic environments: The challenges. *Information Sciences*, *296*, 95–118.

Li, C., & Yang, S. (2008). Fast multi-swarm optimization for dynamic optimization problems. *2008 Fourth International Conference on Natural Computation*, *7*, 624–628.

Li, C., & Yang, S. (2012). A general framework of multipopulation methods with clustering in undetectable dynamic environments. *IEEE Transactions on Evolutionary Computation*, *16*(4), 556–577.

Li, C., Yang, S., & Pelta, D. A. (2011). *Benchmark generator for the IEEE WCCI-2012 competition on evolutionary computation for dynamic optimization problems* (Technical Report, pp. 1–13). UK: Department of Information Systems and Computing, Brunel University.

Liu, L., Ranjithan, S. R., & Mahinthakumar, G. (2011). Contamination source identification in water distribution systems using an adaptive dynamic optimization procedure. *Journal of Water Resources Planning and Management*, *137*(2), 183–192.

Lung, R. I., & Dumitrescu, D. (2007). A new collaborative evolutionary-swarm optimization technique. *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation* pp. 2817–2820. doi:10.1145/1274000.1274043

Lung, R. I., & Dumitrescu, D. (2010). Evolutionary swarm cooperative optimization in dynamic environments. *Natural Computing*, *9*(1), 83–94.

Mahdaviani, M., Kazemi Kordestani, J., Rezvanian, A., & Meybodi, M. R. (2015). LADE: Learning automata based differential evolution. *International Journal on Artificial Intelligence Tools*, *24*(6), 1550023.

Mendes, R., & Mohais, A. S. (2005). DynDE: A differential evolution for dynamic optimization problems. *2005 IEEE Congress on Evolutionary Computation*, *3*, 2808–2815.

Mori, N., Kita, H., & Nishikawa, Y. (2001). Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. *Transactions of the Institute of Systems, Control and Information Engineers*, *14*(1), 33–41.

Nabizadeh, S., Rezvanian, A., & Meybodi, M. R. (2012). A multi-swarm cellular PSO based on clonal selection algorithm in dynamic environments. *2012 International Conference on Informatics, Electronics & Vision (ICIEV)* pp. 482–486. doi:10.1109/ICIEV.2012.6317524

Narendra, K. S., & Thathachar, M. A. L. (1974). Learning automata-a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, *4*, 323–334.

Narendra, K. S., & Thathachar, M. A. L. (2012). *Learning automata: An introduction*. Mineola, New York:Dover Publications, inc.

Nasiri, B., & Meybodi, M. R. (2012). Speciation based firefly algorithm for optimization in dynamic environments. *International Journal of Artificial Intelligence*, *8*(S12), 118–132.

Nguyen, T. T., Yang, S., & Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, *6*, 1–24.

Niu, B., Liu, Q., & Wang, J. (2019). Bacterial foraging optimization with memory and clone schemes for dynamic environments. In Y. Tan, Y. Shi, & B. Niu (Eds.), *Advances in swarm intelligence* (Vol. 11655, pp. 352–360). Cham: Springer.

Noroozi, V., Hashemi, A. B., & Meybodi, M. R. (2011). CellularDE: A cellular based differential evolution for dynamic optimization problems. In A. Dobnikar, U. Lotrič, & B. Šter(Eds., *Adaptive and natural computing algorithms. ICANNGA 2011* (Vol. 6593, pp. 340–349). Berlin Heidelberg: Springer.

Noroozi, V., Hashemi, A. B., & Meybodi, M. R. (2012). Alpinist cellularde: A cellular based optimization algorithm for dynamic environments. *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation* pp. 1519–1520. doi:10.1145/2330784.2331024

Novoa-Hernández, P., Corona, C. C., & Pelta, D. A. (2011). Efficient multi-swarm PSO algorithms for dynamic environments. *Memetic Computing*, *3*, 163–174.

Novoa-Hernández, P., Pelta, D. A., & Corona, C. C. (2010). Improvement strategies for multi-swarm PSO in dynamic environments. In J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, & N. Krasnogor (Eds.), *Nature inspired cooperative strategies for optimization (NICSO 2010)* (Vol. 284, pp. 371–383). Berlin Heidelberg: Springer.

Nseef, S. K., Abdullah, S., Turky, A., & Kendall, G. (2016). An adaptive multi-population artificial bee colony algorithm for dynamic optimisation problems. *Knowledge-Based Systems*, *104*, 14–23.

Parrott, D., & Li, X. (2006). Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation*, *10*(4), 440–458.

Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, *225*(1), 1–11.

Rezvanian, A., Saghiri, A. M., Vahidipour, S. M., Esnaashari, M., & Meybodi, M. R. (2018). *Recent advances in learning automata* (Vol. 754). Springer International Publishing.

Rossi, C., Abderrahim, M., & Díaz, J. C. (2008). Tracking moving optima using kalman-based predictions. *Evolutionary Computation*, *16*(1), 1–30.

Sharifi, A., Kazemi Kordestani, J., Mahdaviani, M., & Meybodi, M. R. (2015). A novel hybrid adaptive collaborative approach based on particle swarm optimization and local search for dynamic optimization problems. *Applied Soft Computing*, *32*, 432–448.

Sharifi, A., Noroozi, V., Bashiri, M., Hashemi, A. B., & Meybodi, M. R. (2012). Two phased cellular PSO: A new collaborative cellular algorithm for optimization in dynamic environments. *2012 IEEE Congress on Evolutionary Computation* pp. 1–8. doi:10.1109/CEC.2012.6256517

Shirali, A., Kazemi Kordestani, J., & Meybodi, M. R. (2018). Self-adaptive multi-population genetic algorithms for dynamic resource allocation in shared hosting platforms. *Genetic Programming and Evolvable Machines*, *19*(4), 505–534.

Storn, R., & Price, K. (1995). *Differential evolution–a simple and efficient adaptive scheme for global optimization over continuous spaces* (Technical Report TR-95-012). International Computer Science Institute.

Storn, R., & Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341–359.

Thathachar, M. A. L., & Sastry, P. S. (1987). A hierarchical system of learning automata that can learn die globally optimal path. *Information Sciences*, *42*(2), 143–166.

Turky, A., Abdullah, S., & Dawod, A. (2018). A dual-population multi operators harmony search algorithm for dynamic optimization problems. *Computers & Industrial Engineering*, *117*, 19–28.

Vavak, F., Jukes, K., & Fogarty, T. C. (1997). Learning the local search range for genetic optimisation in nonstationary environments. *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)* pp. 355–360. doi:10.1109/ICEC.1997.592335

Vavak, F., Jukes, K., & Fogarty, T. C. (1998). Performance of a genetic algorithm with variable local search range relative to frequency of the environmental changes.

Wang, H., Wang, D., & Yang, S. (2007). Triggered memory-based swarm optimization in dynamic environments. In M. Giacobini (Ed.), *Applications of evolutionary computing* (Vol. 4448, pp. 637–646). Berlin Heidelberg: Springer.

Woldesenbet, Y. G., & Yen, G. G. (2009). Dynamic evolutionary algorithm with variable relocation. *IEEE Transactions on Evolutionary Computation*, *13*(3), 500–513.

Wu, V., Wang, Y., Liu, X., & Ye, J. (2010). Multi-population and diffusion UMDA for dynamic multimodal problems. *Journal of Systems Engineering and Electronics*, *21*(5), 777–783.

Yang, S., Cheng, H., & Wang, F. (2010). Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *40*(1), 52–63.

Yang, S., & Li, C. (2010). A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *IEEE Transactions on Evolutionary Computation*, *14*(6), 959–974.

Yazdani, D., Akbarzadeh-Totonchi, M. R., Nasiri, B., & Meybodi, M. R. (2012). A new artificial fish swarm algorithm for dynamic optimization problems. *2012 IEEE Congress on Evolutionary Computation* pp. 1–8. doi:10.1109/CEC.2012.6256169