An Adaptive Bi-flight Cuckoo Search with Variable Nests for Continuous Dynamic Optimization Problems

Javidan Kazemi Kordestani [a,*], Hossein Abedi Firouzjaee [b], Mohammad Reza Meybodi [b]

[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), 424 Hafez Ave., Tehran, Iran*

Javidan.kazemi@gmail.com*, abedi.hossein@aut.ac.ir, mmeybodi@aut.ac.ir

Abstract

This paper presents an adaptive bi-flight cuckoo search algorithm for continuous dynamic optimization problems. Unlike the standard cuckoo search which relies on Levy flight, the proposed method uses two types of flight that are chosen adaptively by a learning automaton to control the global and local search ability of the method during the run. Furthermore, a variable nest scheme and a new cuckoo addition mechanism are introduced. A greedy local search method is also integrated to refine the best found solution. An extensive set of experiments is conducted on a variety of dynamic environments generated by the moving peaks benchmark, to evaluate the performance of the proposed approach. Results are also compared with those of other state-of-the-art algorithms from the literature. The experimental results indicate the effectiveness of the proposed approach.

Keywords: dynamic optimization problems, moving peaks benchmark, DOPs, MPB, cuckoo search, learning automata.

1. Introduction

Optimization in dynamic environments is of great importance because in many real-world optimization problems the objective function, the problem instance or other elements may change during the course of optimization. Unlike optimization in static environments, where the task of optimization is limited to finding the optimum solution(s) in the search space, the goal of optimization in dynamic environments is to track the trajectory of the moving optima. This characteristic poses additional challenges to optimization algorithms, creating a need for designing new strategies and techniques.

Due to their adaptive characteristics, nature-inspired algorithms have proven to be good optimizers for dynamic optimization problems (DOPs). Nguyen et al. [1] identified six major types of approach that have been proposed to meet the challenges of solving DOPs:

i. Increasing the diversity after detecting a change in the environment [2–5].
ii. Maintaining diversity during the optimization process [6, 7].
iii. Employing memory schemes to retrieve information about previously-found solutions [8, 9].
iv. Predicting the location of the next optimal solution(s) after a change is detected [10, 11].
v. Making use of the self-adaptive mechanisms of evolutionary algorithms (EAs), swarm intelligence methods and other metaheuristics [12].
vi. Using multiple sub-populations to handle separate areas of the search space concurrently [13–34].

One major challenge when dealing with DOPs is to determine the correct number of individuals to maintain the diversity. To address this issue, this paper proposes an adaptive bi-flight cuckoo search with variable nests (BfCS-wVN) where the population size is adjusted during the run. To further enhance the performance of BfCS-wVN, various other components are also integrated:

i. A variable nest scheme which helps the algorithm to spend more fitness evaluations (FEs) around the most promising areas of the search space by giving the opportunity to the most successful cuckoos to lay more eggs.
ii. A new cuckoo addition mechanism which generates cuckoos in the areas that have not been populated.
iii. A greedy Hooke-Jeeves (HJ) local search to allocate more FEs to the best found positions.

The rest of this paper is organized as follows: Section 2 gives an overview on some previous attempts for dealing with DOPs. Section 3 presents the basic principles of CS. In Section 4 a brief description about LA is provided. Section 5 describes the proposed BFMCS. Empirical study of the effects of the proposed method on the moving peaks benchmark (MPB) is given in Section 6. Finally, Section 7 concludes the paper with some suggestions for future work.

2. Related Work

The following sections discuss some of the well-known algorithms developed for DOPs in terms of their respective algorithmic types.

2.1. Swarm Intelligent Algorithms for DOPs

Swarm intelligent algorithms are a group of methods that have been applied extensively in DOPs. In this domain, a large number of studies have been carried out using particle swarm optimization (PSO). An example of these methods is the re-randomization PSO proposed by Hu and Eberhart [3]. In their approach, when a change in the environment is detected, the entire (or part of the) swarm is randomly relocated in the search space. Their results suggest that when the step size of the environmental change is small, the re-randomization of a small portion of the swarm can improve the tracking ability of the PSO.

Applying a mutual repulsion between particles, swarms or captured optima is another strategy to maintain a suitable level of diversity during the search process. The atomic model [35, 36] used in multi-swarm charged PSO is an example of utilizing repulsion for dynamic environments. In this approach, each swarm contains a number of *charged classical* particles. Charged classical particles in each swarm repel each other to provide a sustainable diversity among the individuals.

Another approach is to modify the topology of information-sharing among the individuals of the population in order to enhance the performance of the algorithms in dealing with dynamic situations. For example, Janson and Middendorf [6] studied a hierarchical PSO (H-PSO) for noisy and dynamic environments. H-PSO uses a tree-like structure to establish a hierarchy between particles. These authors also proposed a partitioned hierarchical PSO where the tree is temporarily partitioned into sub-swarms after an environmental change is detected. Both approaches reported a significant improvement over standard PSO on different dynamic and noisy functions.

Many researchers have attempted to establish a mutual repulsion between a number of populations with the aim of positioning each of those populations on different promising areas of the search

space. For example, Blackwell and Branke [13] proposed a multi-swarm algorithm based on PSO, namely mQSO. In mQSO, each swarm is composed of a number of *neutral* particles and a number of *quantum* particles. Neutral particles in any swarm update their velocity and position according to the principles of pure PSO. On the other hand, quantum particles do not move according to the PSO dynamics but change their positions around the center of the best particle of the swarm according to a random uniform distribution with radius r_{cloud}. Consequently, they never converge, but provide the swarm with the sustainable level of diversity needed for capturing the shifting optimum. These authors also introduced *exclusion*, which prevents populations from settling on the same peak by reinitializing the worst-performing swarm in the search space. In another work [14], they proposed a second operator referred to as *anti-convergence* which is triggered whenever all swarms converge, and removes the worst swarm from its peak and reinitializes it in the search space.

The critical drawback of the multi-swarm algorithm proposed in [14] is that the number of swarms must be defined prior to the optimization process, whereas in real-world problems the information about the environment may not be available. The very first attempt to adapt the number of populations in dynamic environments was made by Blackwell [37]. He developed an adaptive mQSO (AmQSO) which determines the number of swarms either by spawning new swarms into the search space, or by destroying redundant swarms. In this algorithm, swarms are categorized 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 a converged swarm. In AmQSO, when the number of free swarms (M_{free}) drops to zero, a free swarm is initialized in the search space for capturing undetected peaks. On the other hand, free swarms are removed from the search space if M_{free} is higher than a threshold n_{excess}.

Another group of multi-population methods tries to make use of a variable number of populations by splitting off sub-populations from a main population. Inspired by the self-organizing scouts (SOS) [15], Yang and Li [33] proposed a multi-swarm algorithm for DOPs named FMSO. FMSO starts with a large parent swarm exploring the search space with the aim of locating promising regions. If the quality of the best particle in the parent swarm improves, this implies that a promising search area may have been found. Therefore, a child swarm is split off from the parent swarm to exploit its own sub-space. The search territory of each child swarm is determined by a

radius *r* which is relative to the range of the landscape and width of the peaks. If the best particle of one child swarm ventures to the area captured by another child swarm, the worse child swarm will be removed to prevent child populations residing on the same peak. Furthermore, if a child swarm fails to improve its quality in a certain number of iterations, the best particle of the child swarm jumps to a new position according to a scaled Gaussian distribution. Another similar idea can be found in [23]. An interesting approach is the hibernating multi-swarm optimization algorithm proposed by Kamosi et al. [22], where unproductive searches in child swarms are stopped by "hibernating" converged child swarms to make more FEs available for productive child swarms.

Yazdani et al. [38] proposed a multi-swarm algorithm called finder-tracker multi-swarm PSO (FTMSPO). FTMPSO uses several mechanisms to tackle existing challenges in dynamic environments including: a peak finding scheme, a tracking scheme, a wakening and sleeping mechanism, a change detection method and a local search procedure.

Different from the above algorithms, another way to create multiple populations is to divide the main population of the algorithm into several clusters, i.e., sub-populations, via clustering methods. For example, Parrott and Li [39] proposed a speciation-based PSO (SPSO) for tracking multiple optima in dynamic environments, which dynamically distributes particles of the main swarm over a variable number of so-called *species*. Bird and Li [40] improved the performance of SPSO by estimating the location of the peaks using a least squares regression method.

Yang and Li [32] proposed a clustering PSO (CPSO) for locating and tracking multiple optima in dynamic environments. 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 then uses the PSO with a modified *gbest* model to exploit the respective region. In order to avoid different clusters crowding one another, they applied a redundancy control check. In this regard, when two sub-swarms are located on a single peak, they are first merged together to form a single sub-swarm and then the worst-performing individuals of the sub-swarm are removed until its size is equal to a predefined threshold. In another work [41], the authors extended the fundamental idea of CPSO and introduced a framework for multi-population methods in undetectable environments. Nickabadi et al. [42] proposed a competitive clustering PSO (CCPSO) which employs a multi-stage clustering procedure to split the particles of the main swarm over a varying number of sub-swarms based on the particles' positions as well

as their objective function values. They also used a group of free particles to locate new emerging optima or to track the unpopulated optima.

Another approach for creating multiple sub-populations is to partition the search space and keep the number of individuals in each partition below a predefined threshold. Hashemi and Meybodi [21] incorporated PSO and cellular automata into an algorithm referred to as cellular PSO. In cellular PSO, the search space is partitioned into equally sized cells using cellular automata. Then, particles of the swarm are allocated to different cells according to their positions in the search space. At any given time, particles residing in each cell use their personal best positions and the best solution found in their neighborhood cells for searching for an optimum. Moreover, whenever the number of particles within each cell exceeds a predefined threshold, randomly selected particles from the saturated cells are transferred to random cells within the search space. In addition, each cell has a memory that is used to keep track of the best position found within the boundary of the cell and its neighbors. In another work [20], they improved the performance of cellular PSO by temporarily converting the particles to quantum particles whenever a change was detected. Other examples of cellular methods are a multi-swarm cellular PSO based on clonal selection [26], and two phased cellular PSO [31].

Moving away from multi-population approaches, Liu et al. [43] introduced a new variant of PSO, called PSO with Composite Particles (PSO-CP) for DOPs. PSO-CP contains four major components: (1) composite particles which are formed by iteratively grouping the worst particles with two of their nearest particles, (2) a scattering operator which aims at enhancing the local diversity within composite particles, (3) a velocity-anisotropic reflection scheme that can improve the ability of particles to track the promising peaks in a new period and (4) an integral movement strategy which favors the detection of new peaks in the search space by promoting the swarm diversity. Inspired by microbial life, Karimi et al. [44] proposed the dynamic hybrid PSO (DHPSO) where a new birth and death mechanism was incorporated into the PSO to dynamically adjust the swarm size during the search process. In DHPSO, each three neighboring particles can produce a new infant, based on the quadratic interpolation distribution, with a certain probability. Moreover, an aging scheme was used to remove the old particles.

2.2. Evolutionary Algorithms for DOPs

Initially, most of the approaches proposed for dealing with DOPs were based on EAs. Early EAs for solving DOPs endeavored to increase population diversity after detecting a change in the

environment. For example, Cob [2] incorporated a hyper-mutation operator into a genetic algorithm (GA) which temporarily creates a dramatic increase in the mutation rate after a change occurs. Vavak et al. [4, 5] also suggested a variable local search for controlling the mutation size. Another approach for dealing with DOPs is to maintain a sustainable level of diversity during the entire optimization process. For example, Grefenstette [45] proposed the random immigrants GA, where at each generation a part of the population is replaced by randomly generated individuals. Other examples of mechanisms for maintaining diversity are fitness sharing [46] and sentinel placement [47].

Apart from the above approaches, many researchers have equipped EAs with a memory scheme that restores useful information about previously found positions in the search space. This technique is especially useful in periodical or recurrent environments, where the optima may repeatedly reappear in regions near to their previous positions [1].

A group of studies has distributed the individuals of EAs among multiple populations where each population is responsible for handling a separate area of the search space. Borrowing the concept of forking, SOS [15], shifting balance GA [48] and multinational GA [49] are three well-known examples of multi-population GAs for DOPs.

Apart from the above GA-based methods for DOPs, many researchers applied other EAs to DOPs. Inspired by the concept of exclusion, Mendes and Mohais [27] introduced a multi-population differential evolution (DE) algorithm for dynamic environments, called DynDE. In DynDE several populations of genomes are initialized into the search space and explore for multiple peaks in the environment incorporating *exclusion*, which prevents populations from converging to the same optima, and *increasing diversity*, which enables a partially-converged population to track the shifting optimum.

Du Plessis and Engelbrecht [50] improved the performance of DynDE by proposing a function evaluation management scheme. In their approach, FEs between two successive changes in the environment are divided into three phases: (1) all populations are evolved according to normal DynDE for ζ_1 generations in order to locate peaks, (2) the weaker populations are frozen and the stronger populations are executed for another ζ_2 generations and (3) frozen populations are put back into the search process and all populations are evolved for ζ_3 generations in a normal DynDE manner.

Inspired by the cellular PSO, Noroozi et al. [28] adopted the concept of cellular PSO and proposed an algorithm based on differential evolution, called CellularDE. CellularDE employs a DE/rand-to-best/1/bin scheme to provide local exploration capability for genomes residing in each cell. After a change is detected in the environment, the population performs a random local search for several subsequent iterations. In another work [51], the authors equipped CellularDE with a mechanism to detect the presence of a peak within each cell's boundary or in its neighbors.

Recently, Halder et al. [19] proposed a cluster-based differential evolution with external archive which uses the *k-means* clustering method to create a variable number of populations.

Du Plessis and Engelbrecht [18] proposed the dynamic population differential evolution (DynPopDE) algorithm, in which the populations are adaptively spawned and removed as required, based on their performance. In this approach, whenever all the current populations fail to improve their fitness after spending their last respective FEs, DynPopDE produces a new population of random individuals in the search space. On the other hand, when the number of populations exceeds the number of peaks in the landscape, redundant populations can be identified as those which are frequently reinitialized by the exclusion operator.

2.3. Other Metaheuristics for DOPs

Apart from the above major methods, some researchers applied other metaheuristics for dealing with DOPs. For example, Rezvanian and Meybodi [52] proposed an immune-based algorithm for solving DOPs. In their approach, the learning automata were used for tuning the mutation rate of antibodies in an artificial immune system to establish a balance between the global and local search ability of the algorithm. Other immune-based algorithms can be found in [53].

Nasiri and Meybodi [54] proposed a speciation-based firefly algorithm for DOPs. In their approach, an initial population of fireflies is randomly scattered in the search space. After a while, the fireflies are divided into different species each of which explores different regions of the search space using a modified firefly algorithm.

Inspired by AmQSO [37], Yazdani et al. [34] proposed a dynamic modified multi-population artificial fish swarm algorithm. In this approach, they modified the basic parameters, behaviors and general procedure of the standard artificial fish swarm algorithm to fulfill the requirements for optimization in dynamic environments.

Recently, Turkey and Abdullah [55] proposed a multi-population harmony search with an external archive for DOPs.

Fouladgar and Lotfi [56] adapted the ideas from [37] to a cuckoo search algorithm for DOPs. Their approach uses a modified cuckoo search algorithm to increase the convergence rate of populations for finding the peaks, and uses exclusion to prevent populations from converging to the same areas of the search space.

2.4. Hybrid Approaches

Some researchers have tried to combine desirable features of different optimization algorithms into a single method for DOPs. For example, Lung and Dumitrescu [57] introduced a hybrid collaborative approach for dynamic environments called collaborative evolutionary-swarm optimization (CESO). CESO has two equal-size populations: a main population to maintain a set of local and global optima during the search process using crowding-based differential evolution, and a PSO population acting as a local search operator around solutions provided by the first population. During the search process, information is transmitted between both populations via collaboration mechanisms. In another work [25], a third population is incorporated with CESO which acts as a memory to recall some promising information from past generations of the algorithm. Kordestani et al. [24] extended the general idea of CESO with different strategies and a hill-climbing local search to prevent wasting of FEs and to spend more FEs around the best found positions of the search space.

Moser and Hendtlass [58] proposed a hybrid algorithm of extremal optimization (EO) and local search methods. In their approach, EO is used to select initial solutions for the local search procedure. EO does not utilize a population of solutions, but it uses mutation to improve a single solution. This algorithm uses a "stepwise" sampling method to take samples at equal distances from every dimension of the search space. Then, a hill-climbing procedure is performed on the best sample. Finally, the solution is stored in memory, and the algorithm is applied again on another randomly generated solution. An enhanced and more general variant of this algorithm was proposed by Moser and Chiong [59].

Recently, Sharifi et al. [60] proposed a hybrid approach based on PSO and local search. In the proposed approach, a swarm of particles following the fuzzy social-only model is frequently applied to estimate the location of the peaks in the problem landscape. Upon convergence of the swarm to a previously-undetected position in the search space, a local search agent is created to exploit that region with naive direct search. Moreover, a density control mechanism and three

function evaluation management schemes were also introduced to ensure that FEs are spent in an efficient way.

3. Cuckoo Search

CS is a stochastic population-based algorithm which was recently proposed by Yang and Deb [61], motivated by the breeding strategy of some cuckoos. These birds show an extraordinary behavior when they want to lay their eggs. Instead of building their own nests, cuckoos lay their eggs in the nests of other species making those species responsible for caring for their chicks. This behavior is the main inspiration of the CS algorithm.

For the sake of simplicity of description and implementation, Yang and Deb [61] made three idealized assumptions as follows:

(1) Each cuckoo only lays one egg at a time and dumps it in a randomly-chosen nest.

(2) The best nests with high-quality eggs are transferred to the next generations.

(3) The number of nests is fixed and there is a probability that a host can recognize a cuckoo egg. If this happens, the host can either throw out the egg or simply leave the nest and construct a new one.

Based on the above assumption, the search process for standard CS can be summarized as shown in Algorithm 1.

Algorithm 1. CS via Levy flight

begin
  Objective function $f(x)$, $x = (x_1, x_2, \dots, x_n)^T$;
  Generate initial population of n host nests $x_i (i = 1,2, \dots, n)$;
  while ($e <$ *MaxEvals*) *or* (*Stopping criteria not met*) **do**
    Get a cuckoo randomly by Levy flight;
    Evaluate its quality/fitness F_i;
    Choose a nest among n (say, j) randomly;
    if $F_i > F_j$ **then**
      Replace j by the new solution;
    end-if
    A fraction (p_a) of worst nests are abandoned and new ones are built;
    Keep the best solutions (or nests with quality solutions);
    Rank the solutions and find the current best;
  end-while
End

CS has been successfully applied to a vast variety of science and engineering problems, including phase equilibrium and stability calculations [62], design of power systems for energy self-sufficient farms [63], selection of optimal machining parameters in milling operations [64], multi-objective optimization problems [65, 66], tunneling [67] and many others.

4. Learning Automata

LA are adaptive decision-making devices operating in an unknown random environment that learn to choose the optimal action from a set of available actions through iterative interaction with the environment [68, 69]. The learning process of the automaton can be divided into three steps:

(1) The automaton selects an action from a finite set of actions and applies it to the environment.

(2) The environment evaluates the effectiveness of the selected action and generates a response for the learning automaton.

(3) The automaton uses the response of the environment to update its internal action probabilities.

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

LA have a wide variety of applications in parameter control [52, 70–72], wireless sensor networks [73, 74], vertex coloring problems [75], traffic signal control [76], information retrieval [77] and hard rock tunnel-boring machine penetration rate-prediction problems [67], to name just a few.

4.1. Variable Structure Learning Automaton

A variable structure learning automaton (VSLA) is a type of LA which is defined with a quadruple $\langle \alpha, \phi, p, T \rangle$ where $\alpha = \{\alpha_1, \dots, \alpha_r\}$ is the set of actions, $\phi = \{\phi_1, \dots, \phi_r\}$ is the set of inputs and a set of outputs, $p = [p_1, \dots, p_r]$ denotes the vector of action probabilities which its elements should satisfy under the following conditions:

$$\sum_{i=1}^{r} p_i = 1 \tag{1}$$

$$\forall i \epsilon \{1, \dots, r\}: 0 \leq p_i \tag{2}$$

Finally, T is the learning algorithm that modifies the action probability vector according to the

received environmental feedback.

4.2. Learning Algorithm

There are many algorithms for updating the action probabilities of an automaton. In this paper, a learning algorithm called *linear reward-inaction* [78] was applied to update p as described below. Let $\alpha_i(k) \in \alpha_i$, and $p(k)$ denote the action selected by the learning automaton and the action probability vector at instant k. The action probability $p(k)$ is then updated according to the following linear learning scheme:

$$p_i(k+1) = p_i(k) + \eta\phi(k).(1 - p_i(k)) \tag{3}$$

$$p_j(k+1) = p_j(k) + \eta\phi(k)p_j(k), \quad \forall i \neq j \tag{4}$$

where the parameters k and $\phi(k)$ are the index representing the iteration number and the feedback received from the environment at iteration k respectively. Finally, η is a positive learning rate in (0,1).

4.3. Environment

Different environments can be modeled depending on the nature of the reinforcement signal. In the *P-model*, the reinforcement signal can only take two binary values 0 and 1 [78]. The framework for the learning process of a VSLA in a random unknown environment is shown in Algorithm 2.

Algorithm 2. The framework for the learning process of VSLA with linear reward-inaction learning scheme in a random unknown environment

begin
  let $\alpha = \{\alpha_1, \ldots, \alpha_n\}$ be the set of finite actions, where n is the number of actions
  let $p = [p_1, \ldots, p_n]$ be the action probability vector of the LA
  while (the automaton converges to one of its actions) **do**
    The learning automaton chooses one of its actions based on the probability distribution p
    The environment evaluates the action and calculates a reinforcement signal
    The environment sends a feedback to the learning automaton
    The automaton updates its probability vector using Eq. (3) and Eq. (4)
  end while
end

5. Proposed Algorithms

5.1. General Procedure

The structure of BfCS-wVN is given in Algorithm 3. In the rest of this sub-section, the details of the proposed algorithm, including *initialization*, *bi-flight cuckoo search*, *detection and response to environmental changes*, *stagnation and cuckoo addition*, *local search around the best found position*, *overlapping and nest removal* and *adjusting the number of nests* are described.

Algorithm 3. General Procedure of BfCS-wVN

Require: a matrix of randomly initialized N_p cuckoos with N_n eggs: $Cuckoo\left(Cuckoo_1, Cuckoo_2, \dots, Cuckoo_p\right)$, searching the space with length X and dimension D, initial values for probability vectors of learning automata, (β_l, β_s) and the parameters for HJ

1. $best \leftarrow$ nest with maximum fitness value
2. **while** a stopping criterion not met **do**
3. **if** a change is detected in the environment **then**
4. Re-evaluate all nests in $Cuckoo$
5. $count \leftarrow 0$
6. **end if**
7. **for** $k = 1, \dots, N_p$ **do**
8. *UpdateNests*$(Cuckoo_k)$
9. **end for**
10. **if** $(count > 2)$ and (*Stagnation*( ) = *TRUE*) **then**
11. Generate a new cuckoo with minimum Euclidean distance of r_p from all other existing cuckoos in the search space using Algorithm 8
12. **end if**
13. Evolve the $best$ using modified *HJ*( )
14. *PreventOverlapping* ( )
15. *AdjustCuckooNests* ( )
16. $count \leftarrow count + 1$
17. **end while**

5.1.1. Initialization

The initial population of nests is simply randomized within the boundary of the search space according to a uniform distribution as follows:

$$x_{ij} = lb_j + rand[0,1] \times (ub_j - lb_j) \quad (5)$$

where $i \in \{1,2,\dots,N_n\}$ is the index of i^{th} nest, $j \in \{1,2,\dots,D\}$ represents the j^{th} dimension of the search space, $rand[0,1]$ is a uniformly distributed random number in the range of [0,1]. Finally, lb_j and ub_j are the lower and upper bounds of the search space corresponding to the j^{th} dimension.

5.1.2. Bi-flight Cuckoo Search

There is a class of probability distributions of an infinite second moment that yields a stable process. Such a probability distribution is called a Levy probability distribution and has the following form [79]:

$$f(x;\beta,\gamma) = \frac{1}{\pi}\int_0^{\infty} exp(-\gamma s^{\beta})\cos(sx)\,ds, \quad x \in (-\infty,+\infty) \quad (6)$$

where the parameter $\gamma > 0$ is the scale factor, $0 < \beta < 2$ is the shape parameter, and the distribution is symmetrical with respect to $x = 0$.

Without losing generality, one can set the parameter $\gamma = 1$. Consequently, a special case of the Levy distribution will be obtained with a single parameter β.

A Levy distribution at extreme values follows a heavy-tailed behavior where its probability density function (PDF) and its cumulative density function (CDF) could be approximated with the tail functions. The right tail PDF and CDF are shown below:

$$f(x;\beta,1) = x^{-(\beta+1)}, \qquad (pdf) \quad (7)$$

$$F(x;\beta,1) = x^{-\beta}, \qquad (cdf) \quad (8)$$

One of the main features of the samples generated by a Levy distribution is that the absolute values (magnitudes) are probabilistically governed by the value of β. As the value of β decreases and becomes closer to 0 the magnitudes of the samples increase. Conversely, as the value of β increases and becomes closer to 2 the absolute values of the samples get closer to zero. Figure 1 shows the importance of the β parameter in the above equation.

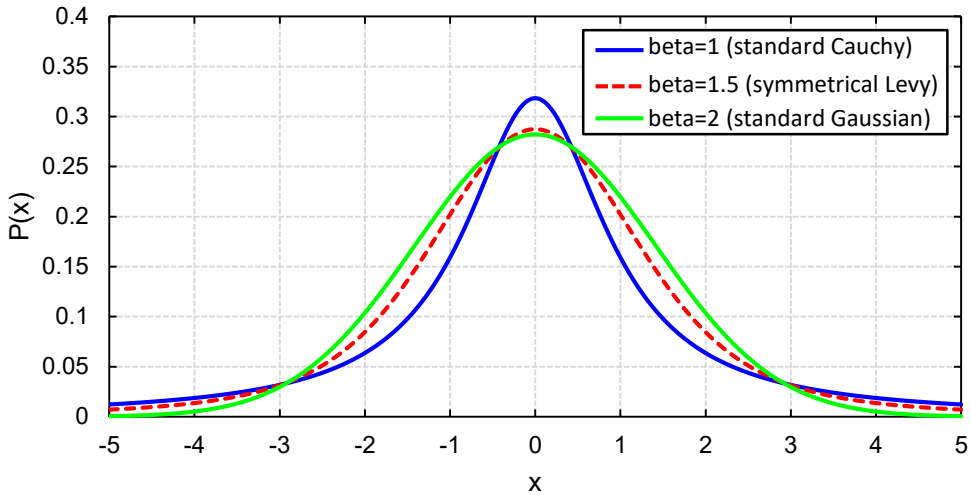

Figure 1. Levy probability distribution for different values of β [67].

With regard to Figure 1, decreasing the value of β results in generating more samples farther away from zero (distributions are symmetrical around zero). This phenomenon shows the critical importance of β in controlling the exploration and exploitation of the CS. High values of β result in the CS having much more exploitation; conversely, low values of β increase the exploration of the CS.

Unlike the standard cuckoo search which relies on the Levy flight, in the proposed method two types of Levy distributions were used with two different β values; one for long jumps and the other for short ones. These two types of Levy distributions represent two different search strategies: (a) an explorative search where the algorithm tends to search areas far from the current candidate solution and (b) an exploitative search where the algorithm is focused on searching local areas around the current candidate solution.

In the proposed approach, a search strategy is chosen for each egg with a certain probability vector $p = (p_l, p_s)$ using a learning automaton. In the initial stage of the execution of the algorithm, the probability of long jumps is expected to be higher than that of short ones. The root of this assumption is that an algorithm should explore wide areas of the search space during the early stages of the optimization to locate promising regions. At the final stages of the search process, however, the algorithm should exploit relatively small regions of the search space around the suspected optima to further refine the results.

The implementation of the updating nests procedure is outlined in Algorithm 4.

Algorithm 4. *UpdateNests* ($Cuckoo$)

Require: a $Cuckoo$ with N_n permissible eggs, $P = (p_l, p_s)$ representing the learning automaton probability vector for each egg in the nest and $\beta = (\beta_l, \beta_s)$
Ensure: updated $Cuckoo$
1. $best \leftarrow$ nest with maximum fitness value
2. **for** $k = 1, \ldots, N_n$ **do**
3. select a distribution β_k by LA with respect to P using Algorithm 5
4. $x_{candidate} \leftarrow best + uniform(0,1,D) \times Levy(\beta_k)$
5. $f_{candidate} \leftarrow f(x_{candidate})$
6. **if** $f_{candidate} > f_{best}$ **then**
7. $egg_k \leftarrow x_{candidate}$
8. $f_k \leftarrow f_{candidate}$
9. update the probability vector $p = (p_l, p_s)$ for the current egg using Algorithm 6
10. **end if**
11. **end for**

At each generation, the LA chooses a flight operator from the set of its actions according to Algorithm 5.

Algorithm 5. Pseudo-code for selecting the distribution by LA

Require: probability vector of a LA: $(p_l, p_s)^T$
Ensure: selected flight for generating a candidate solution (an egg)
1. $rnd \leftarrow uniform(0,1,1)$
2. $selectedDistribution \leftarrow Null$
3. **if** $(rnd < p_l)$ **then**
4. $selectedDistribution \leftarrow l$
5. **else**
6. $selectedDistribution \leftarrow s$
7. **end if**
8. **return** $selectedDistribution$

Afterwards, all existing eggs are evolved according to their selected flight and the best nest. Once each egg is evolved with the selected flight, the automata update their probability vector based on the reinforcement signal they receive from the environment. Based on the received reinforcement signals, the automata can determine whether their actions were right or wrong, and update their probability vectors accordingly. In this work, the reinforcement signal is generated as follows:

$$Reinforcement\ signal\ \phi = \begin{cases} 1 & if\ the\ candidate\ solution\ improved \\ 0 & otherwise \end{cases} \quad (9)$$

Then, the corresponding probability vector of the LA for each egg is modified based on the learning algorithm of the automaton using Algorithm 6.

Algorithm 6. Pseudo-code for updating the action probability vector of the LA

Require: reinforcement signal ϕ, probability vector $P = (p_l;\ p_s)^T$, the index of the chosen flight type $index$ and learning rate of the automaton η

1. **Ensure:** updated probability vector
2. **if** $(index = l)$ **then**
3. $\quad p_l \leftarrow p_l + \phi \times \eta \times (1 - p_l)$
4. $\quad p_s \leftarrow p_s - \phi \times \eta \times p_s$
5. **else if** $(index = s)$ **then**
6. $\quad p_s \leftarrow p_s + \phi \times \eta \times (1 - p_s)$
7. $\quad p_l \leftarrow p_l - \phi \times \eta \times p_l$
8. **end if**

5.1.3. Detection of and Response to Environmental Changes

Several methods have been suggested for change detection [1]. In this work, three eggs are used as "sensors" to detect changes in the environment. In fact, the fitness values of the selected eggs are re-evaluated in each iteration. If the fitness of one of the selected eggs differs from its previous value, we can conclude that a change has occurred in the environment.

Once an environmental change is detected, all eggs are re-evaluated.

5.1.4. Stagnation and Cuckoo Addition

Since the number of optima in real-world DOPs is not known during the optimization process, there is no way to define a preset and constant value for the number of cuckoos required. The ideal situation would be to have an equal number of cuckoos and optima, so each peak could be covered with a separate cuckoo. Otherwise, if the number of cuckoos is greater than the number of optima, some fraction of FEs is wasted on non-optimal areas of the search space and/or redundant searches. Conversely, if the number of cuckoos is smaller than the number of optima, the algorithm inevitably loses some promising areas of the search space. Therefore, different mechanisms for addition and removal of new cuckoos are essential.

For cuckoo addition, a concept called *stagnation* is applied which shows that the BfCS-wVN has reached a point of no further improvement in the fitness of the current cuckoos. The stagnation is defined with a function $\Phi(t)$, which indicates if all existing cuckoos in the search space have been stagnated, as follows [18]:

$$\Phi(t) = \begin{cases} true & if\ (\Delta f_n(t) = 0)\ \forall n \in N_p \\ false & otherwise \end{cases} \tag{10}$$

where N_p is the set of current cuckoos, and $\Delta f_n(t)$ is the amount of improvement in the quality of the n^{th} cuckoo at time t, calculated as follows:

$$\Delta f_n(t) = |f_n(t) - f_n(t-1)| \tag{11}$$

Algorithm 7 summarizes the procedure for detecting stagnation of the BfCS-wVN.

Algorithm 7. *Stagnation*( )

Require: a vector of values about improvement of the nests between two consecutive generations: $A = (\Delta f_1, \Delta f_2, \dots, \Delta f_n)$
Ensure: a Boolean value as a flag for occurrence of stagnation

```
n ← length of A
stagnated ← TRUE
for k = 1,...,n do
    if Δf_k > 0 then
        stagnated ← FALSE
        return stagnated
    end if
end for
return stagnated
```

Once stagnation occurs, a nest is added to the search space. The main motivation here is to actively scatter new eggs in the areas far from the existing nests, to further explore new regions of the search space. To achieve this goal, a prohibition distance r_p is defined as follows:

$$r_p \leftarrow \frac{X}{N_p^{\frac{1}{D}}} \tag{12}$$

where X is the range of the search space for each dimension, N_p is the number of existing cuckoos, and D is the dimensionality of the problem.

After detecting stagnation, a nest will be generated in a position which is at least a distance r_p away from the current nests. The pseudo-code for generating a new nest is shown in Algorithm 8.

Algorithm 8. Generating a new nest

Require: (lb, ub) as the lower and upper bound of search space in different dimensions, a matrix $bests = (best_1, best_2, \dots, best_n)$ representing best nest of the cuckoos, searching the space with length X and dimension D, (N_p, N_n) representing the number of cuckoos and maximum number of allowable eggs for a cuckoo.

Ensure: addition of a cuckoo

$r_p \leftarrow \frac{x}{N_p^{\frac{1}{D}}}$
$counter \leftarrow 0$
$success \leftarrow FALSE$
$distances \leftarrow [\,]$
while $success = FALSE$ and $counter < 100$ **do**
    $cuckoo_{new} \leftarrow lb + uniform(0,1, N_n, D) \times (ub - lb)$
    for each $best$ in $bests$ **do**
        $distances.append\big(EuclideanDistance(cuckoo_{new}, best)\big)$
    end for
    if $minimum(distances) > r_p$ **then**
        $success \leftarrow TRUE$
        $Cuckoo.append(cuckoo_{new})$
        $N_p \leftarrow N_p + 1$
    end if
$counter \leftarrow counter + 1$
end while

The proposed method ensures that the newly generated cuckoo is at least as far as r_p from all other existing cuckoos. This mechanism helps the BfCS-wVN to explore non-visited areas of the search space.

5.1.5. Local Search Around the Best Found Positions

In [60], the authors showed that spending more FEs around the best found positions of the search space can be favorable to the performance of the optimization. Therefore, inspired by [60], at the end of each generation a modified HJ procedure will be performed on the best found solution. The

advantage of this adaptation is that better-performing cuckoos will receive more FEs which otherwise would have been spent on exploiting the maxima of the sub-optimal peaks.

As described by Hooke and Jeeves [80], HJ pattern search is a direct search routine which uses two types of move:

i. An *exploratory move* in which all decision variables are changed in turn by a predefined step size s_0 to acquire the behavior of the function to be optimized.
ii. The *pattern move* which repeats all changes that are found to be successful in the exploratory move.

In the above procedure, there is a step size $s_0 > 0$ which is equal in all dimensions. If the algorithm fails to improve the candidate solution in all dimensions using the given step size, the value of the step size is reduced by multiplying it by a real number b where $0 < b < 1$. The process can be stopped if no further improvement is possible or if a certain number of FEs have been spent.

The aforementioned HJ algorithm has two implicit assumptions: (1) all dimensions of the problem contribute evenly to the improvement of the candidate solutions and (2) the step size could be reduced by the same amount for all dimensions. However, there are problems in which some specific dimensions are more prone to improve the candidate solutions, (e.g., the 2D ellipse function). Therefore, it is desirable to apply HJ in such a way that each dimension can be searched with a different step size.

Algorithm 9 shows the pseudo-code of the modified HJ algorithm.

Algorithm 9. Modified HJ Algorithm

Require: $x_0 = (x_1, x_2, \dots, x_n)^T$, objective function f, $s_0 > 0$, $b \in (0,1)$
Ensure: x_{best} (where $f(x_{best})$ is maximum)
1. $\alpha \leftarrow ones(1,n) \times s_0$
2. $\beta \leftarrow b$
3. $x_{best} \leftarrow x_0$
4. $f_{best} \leftarrow f(x_0)$
5. $exploit \leftarrow TRUE$
6. **while** $exploit = TRUE$ **do**
7. $exploit \leftarrow FALSE$
8. **for** j=1, …, n **do**
9. $x_{candidate} \leftarrow x_{best}$
10. $x^j_{candidate} \leftarrow x^j_{best} + \alpha_j$
11. $f_{candidate} \leftarrow f(x_{candidate})$
12. **if** $f_{candidate} > f_{best}$ **then**
13. $exploit \leftarrow TRUE$
14. $x_{best} \leftarrow x_{candidate}$
15. $f_{best} \leftarrow f_{candidate}$
16. **else**
17. $x_{candidate} \leftarrow x_{best}$
18. $x^j_{candidate} \leftarrow x^j_{best} - \alpha_j$
19. $f_{candidate} \leftarrow f(x_{candidate})$
20. **if** $f_{candidate} > f_{best}$ **then**
21. $exploit \leftarrow TRUE$
22. $x_{best} \leftarrow x_{candidate}$
23. $f_{best} \leftarrow f_{candidate}$
24. **else**
25. $\alpha_j \leftarrow \alpha_j \times \beta$
26. **end if**
27. **end if**
28. **end for**
29. **end while**

5.1.6. Overlapping and Nest Removal

To prevent populations from settling on a single peak, an exclusion operator with radius r_{exc} is applied between every pair of the best nests of all cuckoos. In this regard, when the Euclidean distance of the best nests of two cuckoos is smaller than r_{exc}, the worst performing nest will be deleted. The outline of the mechanism for preventing redundant searches is shown in Algorithm 10.

Algorithm 10. *PreventOverlapping* ( )

Require: a matrix $bests = (best_1, best_2, \dots, best_n)$ and a vector $f = (f_1, f_2, \dots, f_n)$ representing best nest of the cuckoos and their values respectively, n_{peaks}, D (dimension of search space), X (length of search space)
Ensure: a set of nests with minimum distance between their bests

$r_{exc} \leftarrow \frac{X}{2n_{peaks}^{\frac{1}{D}}}$
$n \leftarrow length\ of\ bests$
$deleted \leftarrow []$
for $k = 1, \dots, n$ **do**
  if $k\ is\ not\ in\ deleted$ **then**
    $p \leftarrow best_k$
    for $l = 2, \dots, n$ **do**
      if $EuclideanDistance(best_k, best_l) \leq r_{exc}$ **then**
        if $(f_k \geq f_l)$ **then**
          $remove(Cuckoo\ l)$
          $deleted.append(l)$
          $N_p \leftarrow N_p - 1$
        else
          $remove(Cuckoo\ k)$
          $deleted.append(k)$
          $N_p \leftarrow N_p - 1$
        end if
      end if
    end for
  end if
end for

5.1.7. Adjusting the Number of Nests

The most expensive part of the optimization process in many real-world problems is usually the calculation of FEs. So, it is reasonable to devise a method which allocates the FEs to the populations wisely, rather than uniformly. To achieve this, a procedure for adjusting the number of nests is performed as follows:

i. Calculate the pairwise distances between the best nest of each cuckoo and the global best solution.
ii. Find the maximum distance l_{max} (the greatest distance).
iii. If the best nest of each cuckoo is in the radius $l_{max} \times \sigma_{max}$ of the global best solution, and if the number of nests is less than N_n, add a nest to the cuckoo. Conversely, if the best nest of each cuckoo is outside the radius $l_{max} \times \sigma_{max}$ of the global best solution, and if the number of nests is greater than N_n, remove a nest from the cuckoo.

After each iteration, the cuckoos that are closer to the global best nest will have more fitness evaluations than the distant ones. Algorithm 11 shows different steps for adjusting the number of nests.

Algorithm 11. *AdjustCuckooNests* ( )

Require: a matrix $bests = (best_1, best_2, \dots, best_n)$ representing the best nests of the cuckoos, a vector $dists = (dist_{1,2}, dist_{1,3}, \dots, dist_{i,j,i\neq j}, \dots, dist_{n,n-1})$ of pairwise distances of best nests, a vector $N_{nests} = (N^1, N^2, \dots, N^n)^T$ representing the list of eggs in each cuckoo nest, N_n as the maximum number of allowable eggs that could be laid in a nest, global best $gBest$

Ensure: cuckoos with adjusted number of permissible eggs

1. $l_{max} \leftarrow maximum(dists)$
2. $l_p \leftarrow l_{max} \times \sigma_{max}$
3. $n \leftarrow length(bests)$
4. **for** $k = 1, \dots, n$ **do**
5. **if** $EuclideanDistance(best_k, gBest) \leq l_p$ **then**
6. **if** $(N^k < N_n)$ **then**
7. $N^K \leftarrow N^K + 1$
8. **end if**
9. **else**
10. **if** $(N^k > 1)$ **then**
11. $N^K \leftarrow N^K - 1$
12. **end if**
13. **end if**
14. **end for**

6. Experimental Study

6.1. Experimental Setup

6.1.1. Dynamic Test Function

One of the most widely used synthetic dynamic test functions in the literature is the moving peaks benchmark (MPB) proposed by Branke [8], which is highly regarded due to its configurability. MPB is a real-valued dynamic environment with a *D*-dimensional landscape consisting of *m* peaks, in which the height, the width and the position of each peak are changed slightly every time a change occurs in the environment. Different landscapes can be defined by specifying the shape of the peaks. A typical peak shape is conical and is defined as follows:

$$f(\vec{x},t) = \max_{i=1,\dots,m} H_t(i) - W_t(i)\sqrt{\sum_{j=1}^{D}(x_t(j) - X_t(i,j))^2} \quad (13)$$

where $H_t(i)$ and $W_t(i)$ are the height and the width of peak i at time t, respectively. The coordinates of each dimension $j \in \{1,2,\dots,D\}$ related to the location of peak i at time t, are expressed by $X_t(i,j)$, and D is the problem dimensionality. A typical change of a single peak can be modeled as follows:

$$H_{t+1}(i) = H_t(i) + height_{severity}.\sigma_h \quad (14)$$

$$W_{t+1}(i) = W_t(i) + width_{severity}.\sigma_w \quad (15)$$

$$\vec{X}_{t+1}(i) = \vec{X}_t(i) + \vec{v}_{t+1}(i) \quad (16)$$

$$\vec{v}_{t+1}(i) = \frac{s}{|\vec{r}+\vec{v}_t(i)|}\left((1-\lambda)\vec{r} + \lambda\vec{v}_t(i)\right) \quad (17)$$

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

Different instances of the MPB can be obtained by changing the environmental parameters. Three sets of configurations have been introduced to provide a unified test bed for the researchers to investigate the performance of their approaches under the same conditions. Among them, the second configuration (*Scenario 2*) is the most widely used configuration and was also the one used as the base configuration for the experiments conducted in this paper.

Table 1. Parameter settings for the moving peaks benchmark (Scenario 2).

Parameter	Default value	Other tested values

Number of peaks (m)	10	1, 5, 20, 30, 40, 50, 100, 200
Height severity	7.0	
Width severity	1.0	
Peak function	cone	
Number of dimensions (d)	5	
Height range (H)	[30, 70]	
Width range (W)	[1, 12]	
Standard height (I)	50.0	
Search space range (A)	$[0, 100]^d$	
Frequency of change (f)	5000	500, 1000, 2500, 10000
Shift severity (s)	1.0	0.0, 2.0, 3.0, 4.0, 5.0
Correlation coefficient (λ)	0.0	
Basic function	No	

6.1.2. Performance Measure

In order to evaluate the efficiency of different algorithms, the offline error suggested by Branke and Schmeck [81], being the most well-known metric for dynamic environments, is used as a performance measure. This measure is defined as the average of the smallest error found since the last change in the environment over the entire run:

$$E_O = \frac{1}{T}\sum_{t=1}^{T} e_t^* \tag{18}$$

where T is the maximum number of function evaluations so far and e_t^* is the minimum error gained by the optimization algorithm since the last change at the t^{th} fitness evaluation.

6.1.3. Experimental Settings

For each run of the algorithm, 500,000 FEs was considered to be the stopping criterion. All the experimental results reported in the paper are averaged over 50 independent runs. The parameter setting for the BfCS-wVN is as listed in Table 2.

Table 2. Default parameter settings for the proposed BfCS-wVN.

Parameter	Default value
η	0.01
P_l	0.85
P_s	0.15
β_l	1.90
β_s	0.75
N_p	20
N_n	2

σ_{max}	0.05
S_0	0.50
b	0.25

6.2. Experimental Results

6.2.1. Parameter Sensitivity Analysis for BfCS-wVN in Dynamic Environments

In this sub-section, the effects of key parameters and components on the performance of BfCS-wVN were analyzed on different DOPs. Moreover, the proposed BfCS-wVN was compared to the basic CS.

A. Effect of the learning rate η

In this set of experiments, we examine the effect of the learning rate η on the performance of BfCS-wVN in different dynamic environments. Figure 2 shows the average offline error of BfCS-wVN with varying learning rate parameters on DOPs, generated by MPB with $m = \{1,5,10,20,30,50\}$. It can be concluded from Figure 2 that a learning rate of 0.01 produces the best results. It can also be seen that the performance of BfCS-wVN is degraded as the learning rate increases. The reason lies mainly in the fact that the learning rate η is responsible for updating the action probability of the automaton LA_{flight}. The bigger the learning rate η, the faster the convergence of LA_{flight} to one of its actions, which may be a non-optimal action.

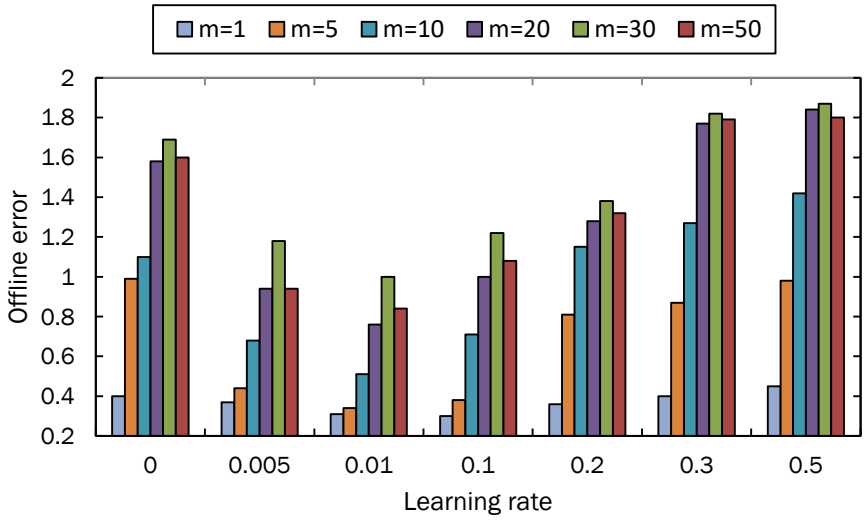

Figure 2. Average offline error of BfCS-wVN with different learning rates η on dynamic environments with various numbers of peaks.

Our experiments revealed that the BfCS-wVN without the learning algorithm ($\eta = 0$) fails to achieve a good performance due to the fixed amounts of exploration and exploitation governed by p_l and p_s. However, the $L_{R\text{-}I}$ learning algorithm used in this work can establish a balance between exploration and exploitation of the cuckoos by adaptively changing the probability vector of the automaton.

B. Effect of β_l and β_s

The parameters β_l and β_s are important in the proposed method as they can strongly affect the performance of BfCS-wVN. This experiment aims at examining the effect of β_l and β_s on the search behavior of BfCS-wVN. Figure 3 illustrates the average offline error of BfCS-wVN with varying values of β_l and β_s on a DOP generated by MPB in scenario 2.

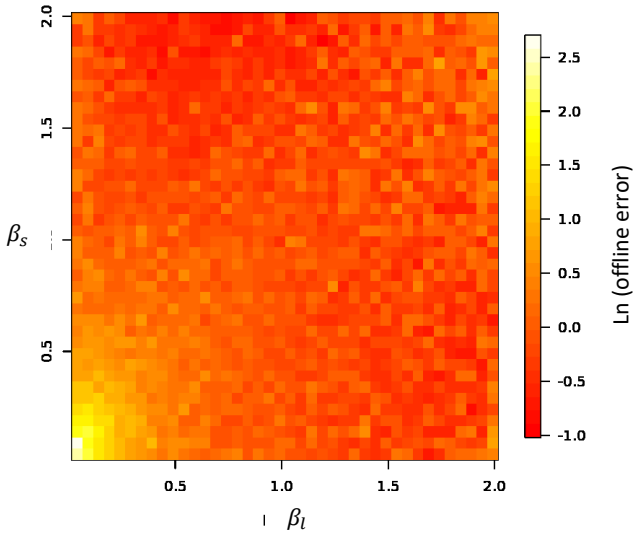

Figure 3. Natural logarithm (average offline error) of BfCS-wVN for varying β_l and β_s in Scenario 2 of MPB.

As can be seen in Figure 3, the combination (β_l, β_s) can clearly affect the behavior of BfCS-wVN. Several conclusions can be drawn from Figure 3:

i. The worst results are obtained when both β_l and β_s are small (< 0.5).

ii. If $\beta_l = 0$ or $\beta_s = 0$, the BfCS-wVN fails to produce acceptable results.

iii. The best results are obtained when $\beta_l = 1.9$ and $\beta_s = 0.75$.

C. Effect of the initial probabilities p_l and p_s

The main goal of this experiment is to investigate the effect of the initial probabilities of p_l and p_s on the performance of BfCS-wVN. As we mentioned in Section 4, p_l and p_s determine the initial probabilities of selecting long or short flights. Therefore, it is expected that these parameters will have a significant influence on the performance of BfCS-wVN. Figure 4 shows the performance of BfCS-wVN with various combinations of (p_l, p_s) on a DOP generated by MPB in scenario 2.

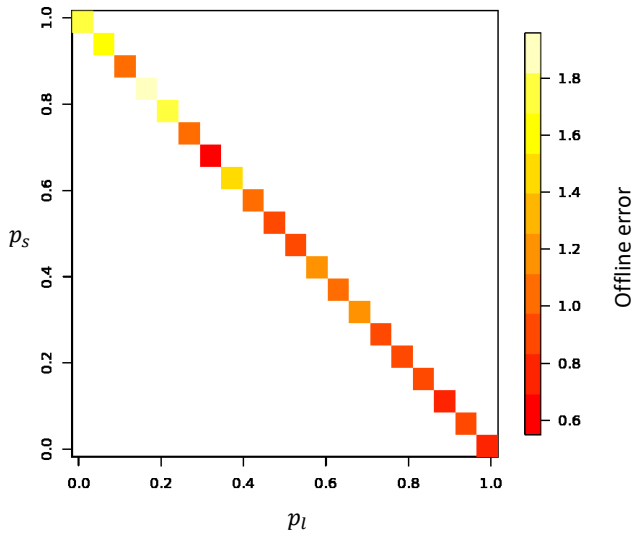

Figure 4. Average offline error of BfCS-wVN for varying p_l and p_s in Scenario 2 of MPB.

It can be observed that the best result is obtained when $p_l = 0.85$ and $p_s = 0.15$. The reason for this is that at the early stages of the optimization process, longer jumps increase the exploration ability of the algorithm which is needed for locating the optima. On the other hand, as the individuals come closer to the optima, smaller jumps help BfCS-wVN to refine the candidate solutions.

D. Effect of the threshold σ_{max}

The aim of this experiment is to investigate the effect of σ_{max} on the performance of BfCS-wVN. Figure 5 shows the average offline error of BfCS-wVN with different σ_{max} across different dynamic environments. With regard to Figure 5, the value of 0.05 is a suitable value for σ_{max}.

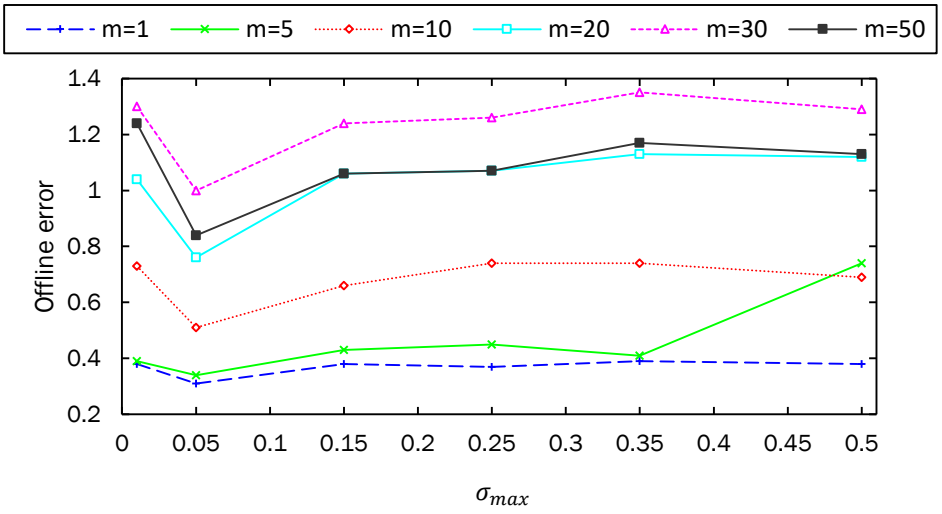

Figure 5. Average offline error of BfCS-wVN with different σ_{max} in dynamic environments with various numbers of peaks

E. Effect of Np and Nn

This experiment has been designed to examine the effect of Np and Nn on the performance of BfCS-wVN on a DOP generated by MPB using scenario 2. The values of the parameters are different combinations of $Np \in \{1, 5, 10, 20, 30, 50, 100\}$ and $Nn \in \{1, 3, 5, 10, 20\}$. The results are given in Table 3.

Table 3. Effect of parameters Np and Nn on the offline error of BfCS-wVN in Scenario 2 of MPB.

Parameters		Nn				
		1	2	3	5	10
Np	1	0.72 ± 0.15	0.63 ± 0.13	0.68 ± 0.21	0.66 ± 0.19	0.71 ± 0.19
	5	0.64 ± 0.18	0.56 ± 0.12	0.55 ± 0.16	0.57 ± 0.17	0.72 ± 0.18
	10	0.59 ± 0.09	0.55 ± 0.12	0.56 ± 0.18	0.55 ± 0.14	0.65 ± 0.12
	20	0.62 ± 0.14	0.51 ± 0.11	0.52 ± 0.11	0.58 ± 0.15	0.71 ± 0.22
	30	0.57 ± 0.13	0.60 ± 0.14	0.56 ± 0.16	0.56 ± 0.15	0.72 ± 0.24
	100	0.55 ± 0.10	0.54 ± 0.12	0.55 ± 0.14	0.64 ± 0.16	2.31 ± 1.12

From Table 3, it can be observed that the combination of $Np = 20$ and $Nn = 2$ produces the best result.

F. Effect of adaptive flight selection via learning automata

To show the effectiveness of the automata approach, an experiment was carried out in which the learning automata were replaced with pure-chance automata. Therefore, the pure chance approach differs from BfCS-wVN in that the flight for generating a new candidate solution is selected randomly at each stage. Table 4 presents the comparison of BfCS-wVN with the pure chance approach. From Table 4, it can be observed that the results of the BfCS-wVN are better than those of the pure chance approach in most of the tested DOPs.

Table 4. The effect of adaptive flight selection via learning automata on the performance of the proposed algorithm.

Method	*m*			*s*			*f*		
	1	10	100	0	3	5	500	2500	10000
Pure chance approach	**0.28 ± 0.09**	0.69 ± 0.15	1.18 ± 0.12	0.54 ± 0.13	1.47 ± 0.22	**2.30 ± 0.18**	6.43 ± 1.20	1.38 ± 0.28	0.40 ± 0.10
BfCS-wVN	0.30 ± 0.06	**0.51 ± 0.11**	**1.11 ± 0.06**	**0.40 ± 0.08**	**1.26 ± 0.13**	2.39 ± 0.20	**4.28 ± 0.42**	**0.99 ± 0.24**	**0.30 ± 0.06**

To verify whether significant differences exist between the two algorithms, we applied the Wilcoxon Signed-Rank Test. The obtained p-value ($0.02734 < 0.05$) is lower than the significance level, which confirms that the distributions of the corresponding results are not identical.

G. Comparison with basic cuckoo search

This experiment has been specifically designed to verify the effectiveness of hybridizing CS with the proposed strategies. Therefore, in this experiment, the performance of the proposed BfCS-wVN is compared with the basic CS.

For the basic CS, a population size of 20 cuckoos was used. Furthermore, the parameters p_a and β were set to 0.25 and 1.5 respectively. Moreover, when a change in the environment is detected, all cuckoos are re-evaluated.

Table 5 shows the results of CS and the proposed method on nine different DOPs generated by MPB. As can be observed, the results of our proposed algorithm are significantly better than those provided by pure CS.

Table 5. Numerical comparison between CS and the proposed algorithm on DOPs generated by MPB.

Method	*m*			*s*			*f*		
	1	10	100	0	3	5	500	2500	10000
CS	73.03±0.34	75.73±19.02	37.48±15.86	11.63±2.34	107.55±38.60	76.22±1.72	75.75±24.32	84.38±27.37	65.01±30.94
BfCS-wVN	**0.30±0.06**	**0.51±0.11**	**1.11±0.06**	**0.40±0.08**	**1.26±0.13**	**2.39±0.20**	**4.28±0.42**	**0.99±0.24**	**0.30±0.06**

6.2.2. Varying the Number of Peaks and Change Frequencies

The aim of this experiment is to investigate the performance comparison between BfCS-wVN and other algorithms on DOPs with different numbers of peaks $m \in \{1, 5, 10, 20, 30, 40, 50, 100, 200\}$ and change frequencies $f \in \{500, 2500, 10000\}$. The other environmental parameters were set according to the MPB settings shown in Table 1. The competing algorithms are mQSO [14], AmQSO [37], CellularPSO [21], HmSO [22], SFA [54], Adaptive-SFA [82], TP-CPSO [31], FTMPSO [38] and the proposed BfCS-wVN. The various combinations of (f, m) resulted in 27 different DOPs being modeled by the MPB. The numerical results of the different algorithms are reported in Table 6.

Table 6. Comparison of offline error of nine different algorithms on DOPs with *d*=5, *s*=1 and varying number of peaks and change intervals.

f	*m*	Method								
		mQSO	AmQSO	CellularPSO	HmSO	SFA	Adaptive-SFA	TP-CPSO	FTMPSO	BfCS-wVN
500	1	40.30 ± 1.30	10.41 ± 0.34	22.37 ± 3.87	9.40 ± 0.31	4.72 ± 0.12	4.80 ± 0.08	5.96 ± 0.22	**1.76 ± 0.09**	6.53 ± 1.45
	5	11.50 ± 0.21	6.95 ± 0.12	14.20 ± 0.65	7.83 ± 0.17	4.88 ± 0.12	4.98 ± 0.11	5.12 ± 0.11	**2.93 ± 0.18**	4.84 ± 1.72
	10	8.96 ± 0.19	6.97 ± 0.13	13.55 ± 0.52	8.04 ± 0.14	5.11 ± 0.14	5.09 ± 0.10	5.56 ± 0.11	**3.91 ± 0.19**	4.28 ± 0.42
	20	8.29 ± 0.11	7.61 ± 0.10	12.77 ± 0.38	8.25 ± 0.12	5.72 ± 0.13	5.67 ± 0.14	5.67 ± 0.10	**4.83 ± 0.19**	5.44 ± 0.74
	30	8.32 ± 0.09	8.99 ± 0.14	12.55 ± 0.42	8.62 ± 0.12	5.97 ± 0.12	5.84 ± 0.11	5.59 ± 0.08	5.05 ± 0.21	**4.97 ± 0.54**
	40	8.18 ± 0.08	10.26 ± 0.15	12.33 ± 0.38	8.67 ± 0.12	-	-	5.53 ± 0.07	-	**5.02 ± 0.38**
	50	8.12 ± 0.08	11.45 ± 0.17	12.19 ± 0.31	8.81 ± 0.12	5.94 ± 0.15	5.86 ± 0.10	5.57 ± 0.08	4.98 ± 0.15	**4.85 ± 0.34**
	100	7.81 ± 0.08	12.35 ± 0.15	11.38 ± 0.29	8.88 ± 0.11	6.15 ± 0.08	6.04 ± 0.11	5.36 ± 0.06	5.31 ± 0.11	**4.28 ± 0.42**
	200	7.59 ± 0.07	12.82 ± 0.13	11.34 ± 0.27	9.06 ± 0.11	6.18 ± 0.11	6.08 ± 0.17	5.33 ± 0.05	5.52 ± 0.21	**5.27 ± 0.28**
2500	1	9.48 ± 0.31	2.44 ± 0.07	4.57 ± 0.31	2.02 ± 0.07	0.85 ± 0.06	1.12 ± 0.08	0.92 ± 0.03	**0.39 ± 0.02**	0.50 ± 0.13
	5	3.33 ± 0.08	2.44 ± 0.08	3.15 ± 0.21	2.12 ± 0.09	1.49 ± 0.07	1.32 ± 0.08	1.17 ± 0.05	0.91 ± 0.08	**0.64 ± 040**
	10	2.85 ± 0.07	2.65 ± 0.05	3.09 ± 0.16	2.43 ± 0.07	1.95 ± 0.04	1.77 ± 0.07	1.59 ± 0.06	1.21 ± 0.06	**0.99 ± 0.24**
	20	3.41 ± 0.06	2.99 ± 0.05	3.60 ± 0.13	2.63 ± 0.05	2.28 ± 0.05	2.09 ± 0.06	1.82 ± 0.04	1.66 ± 0.05	**1.28 ± 0.18**
	30	3.45 ± 0.05	3.01 ± 0.03	3.88 ± 0.12	2.73 ± 0.04	2.46 ± 0.05	2.28 ± 0.07	1.99 ± 0.04	1.87 ± 0.05	**1.44 ± 0.17**
	40	3.45 ± 0.04	3.04 ± 0.03	4.17 ± 0.12	2.77 ± 0.04	-	-	1.96 ± 0.03	-	**1.56 ± 0.10**
	50	3.38 ± 0.04	3.03 ± 0.03	4.25 ± 0.12	2.76 ± 0.04	2.57 ± 0.05	2.42 ± 0.06	2.01 ± 0.03	2.09 ± 0.07	**1.40 ± 0.11**
	100	3.07 ± 0.03	2.96 ± 0.02	4.25 ± 0.13	2.83 ± 0.03	2.74 ± 0.04	2.64 ± 0.05	2.09 ± 0.03	2.22 ± 0.06	**1.59 ± 0.10**
	200	2.92 ± 0.02	2.93 ± 0.02	4.20 ± 0.09	2.94 ± 0.02	2.76 ± 0.04	2.65 ± 0.06	2.06 ± 0.02	2.22 ± 0.07	**1.81 ± 0.12**
10000	1	2.34 ± 0.07	0.60 ± 0.01	1.63 ± 0.12	0.52 ± 0.02	0.26 ± 0.03	0.35 ± 0.03	0.20 ± 0.01	**0.09 ± 0.00**	0.18 ± 0.04
	5	0.97 ± 0.04	0.75 ± 0.05	1.20 ± 0.15	0.76 ± 0.07	0.53 ± 0.04	0.49 ± 0.03	0.37 ± 0.03	0.31 ± 0.04	**0.20 ± 0.02**
	10	1.10 ± 0.03	1.01 ± 0.04	1.19 ± 0.09	0.90 ± 0.04	0.72 ± 0.02	0.63 ± 0.03	0.54 ± 0.03	0.43 ± 0.03	**0.30 ± 0.06**
	20	1.89 ± 0.05	1.35 ± 0.03	2.13 ± 0.10	1.09 ± 0.03	0.91 ± 0.03	0.77 ± 0.03	0.74 ± 0.02	**0.56 ± 0.01**	0.84 ± 0.05
	30	1.98 ± 0.03	1.44 ± 0.02	2.59 ± 0.12	1.14 ± 0.03	0.99 ± 0.04	0.84 ± 0.03	0.82 ± 0.01	0.69 ± 0.09	**0.66 ± 0.08**

40	2.01 ± 0.03	1.53 ± 0.02	2.72 ± 0.09	1.19 ± 0.03	-	-	0.93 ± 0.02	-	**0.80 ± 0.05**
50	1.95 ± 0.03	1.55 ± 0.02	2.91 ± 0.11	1.17 ± 0.01	1.19 ± 0.04	0.97 ± 0.03	0.99 ± 0.01	0.86 ± 0.02	**0.54 ± 0.70**
100	1.75 ± 0.02	1.52 ± 0.01	3.02 ± 0.11	1.15 ± 0.01	1.44 ± 0.04	1.27 ± 0.05	1.19 ± 0.01	1.08 ± 0.03	**0.85 ± 0.05**
200	1.60 ± 0.01	1.52 ± 0.01	2.94 ± 0.10	1.15 ± 0.01	1.52 ± 0.03	1.31 ± 0.03	1.29 ± 0.01	1.13 ± 0.04	**0.85 ± 0.05**

From Table 6 it can be observed that the proposed BfCS-wVN can outperform the other contestant algorithms on most of the tested DOPs (20 out of 27). The results confirmed that the proposed approach can successfully establish a good balance between the explorative and exploitative ability of the CS needed for locating and tracking the shifting optima.

6.2.3. Effect of Varying the Shift Length Severity

In this experiment, the effect of changing the shift severity on the performance of different algorithms is examined. The numerical results of mQSO [14], AmQSO [37], CellularPSO [21], SPSO [39], rSPSO [40], mPSO [23], HmSO [22], TP-CPSO [31], FTMPSO [38] and the proposed BfCS-wVN are reported in Table 7.

Table 7. Comparison of offline error of different algorithms on DOPs with *m*=10, *d*=5 and different shift lengths.

Method	Shift length (*s*)					
	0.0	1.0	2.0	3.0	4.0	5.0
mQSO	1.37 ± 0.06	1.77 ± 0.05	2.35 ± 0.04	2.90 ± 0.05	3.47 ± 0.06	3.95 ± 0.06
AmQSO	1.01 ± 0.05	1.64 ± 0.05	2.26 ± 0.05	2.86 ± 0.06	3.35 ± 0.06	3.80 ± 0.07
CellularPSO	0.93 ± 0.09	1.93 ± 0.08	2.72 ± 0.10	3.70 ± 0.11	4.71 ± 0.14	5.68 ± 0.18
SPSO	0.60 ± 0.04	3.02 ± 0.07	4.49 ± 0.09	5.65 ± 0.09	7.02 ± 8.32	8.32 ± 0.14
rSPSO	0.49 ± 0.06	1.41 ± 0.04	2.10 ± 0.06	2.79 ± 0.07	3.33 ± 0.07	3.85 ± 0.08
mPSO	1.05 ± 0.09	1.66 ± 0.08	2.54 ± 0.11	3.46 ± 0.11	4.51 ± 0.15	5.32 ± 0.09
HmSO	1.03 ± 0.11	1.47 ± 0.04	2.54 ± 0.13	3.79 ± 0.16	5.03 ± 0.19	6.04 ± 0.22
TP-CPSO	0.48 ± 0.03	0.96 ± 0.05	1.25 ± 0.04	1.69 ± 0.04	2.12 ± 0.05	2.46 ± 0.05
FTMPSO	-	0.67 ± 0.04	1.20 ± 0.06	1.40 ± 0.09	-	**1.69 ± 0.07**
BfCS-wVN	**0.40 ± 0.08**	**0.51 ± 0.11**	**0.89 ± 0.16**	**1.26 ± 0.13**	**1.89 ± 0.18**	2.39 ± 0.20

As shown in Table 7, BfCS-wVN outperforms the other methods in five out of six DOPs. In a DOP with *s*=5, the offline error achieved by FTMPSO is lower than that achieved by BfCS-wVN. The reason lies in the fact that FTMPSO utilizes the information about the shift severity, i.e., *s*, to respond to environmental changes, whereas this information may not be available in real-world situations.

6.3. Comparison with Other Dynamic Optimization Methods

In this sub-section, we study the performance of the proposed method in comparison with several well-known and recently-proposed algorithms taken from the literature. The algorithms considered

include: mQSO [14], AmQSO [37], CellularPSO [21], mPSO [23], HmSO [22], MDUMDA [83], APSO [84], CellularDE [28], SFA [54], CCPSO [42], DMAFSA [34], TP-CPSO [31], DynPopDE [18], CDEPSO [24], FTMPSO [38], mNAFSA [85], AMP/DE [86], AMP/PSO [86] and the proposed BfCS-wVN. Table 8 shows the numerical results of different algorithms on DOPs with different numbers of peaks $m \in \{1, 5, 10, 20, 30, 40, 50, 100, 200\}$. The other environmental parameters were set according to scenario 2 of MPB.

In order to compare the algorithms across all tested DOPs, a *normalized score* is calculated for each algorithm according to [1] as follows:

$$Score(i) = \frac{1}{m}\sum_{j=1}^{m}\frac{|e_{max}(j)-e(i,j)|}{|e_{max}(j)-e_{min}(j)|}, \quad \forall i = 1{:}n \tag{19}$$

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

Table 8. Comparison of offline error of the recent or well-known algorithms for dynamic environments modeled with MPB. The normalized score for each algorithm on a DOP is given in parentheses.

Method	*m*									*overall score*
	1	5	10	20	30	40	50	100	200	
mQSO	4.49 ± 0.14 (0)	1.72 ± 0.05 (0.1302)	1.77 ± 0.05 (0.1127)	2.49 ± 0.05 (0.2222)	2.58 ± 0.04 (0.4698)	2.58 ± 0.04 (0.4773)	2.50 ± 0.03 (0.5654)	2.27 ± 0.02 (0.5105)	2.12 ± 0.02 (0.5708)	0.3399
AmQSO	1.27 ± 0.04 (0.7318)	1.30 ± 0.06 (0.3787)	1.64 ± 0.05 (0.2042)	2.03 ± 0.04 (0.4267)	2.11 ± 0.03 (0.6275)	2.18 ± 0.02 (0.6071)	2.18 ± 0.02 (0.6492)	2.14 ± 0.01 (0.5654)	2.12 ± 0.01 (0.5708)	0.5290
CellularPSO	2.79 ± 0.19 (0.3864)	1.94 ± 0.18 (0)	1.93 ± 0.08 (0)	2.73 ± 0.12 (0.1156)	3.08 ± 0.11 (0.3020)	3.28 ± 0.11 (0.2500)	3.34 ± 0.07 (0.3455)	3.48 ± 0.11 (0)	3.37 ± 0.08 (0)	0.1555
mPSO	0.90 ± 0.05 (0.8159)	1.21 ± 0.12 (0.4320)	1.66 ± 0.08 (0.1901)	2.05 ± 0.08 (0.4178)	2.18 ± 0.06 (0.6040)	2.24 ± 0.06 (0.5877)	2.30 ± 0.04 (0.6178)	2.32 ± 0.04 (0.4895)	2.34 ± 0.03 (0.4703)	0.5139
HmSO	1.01 ± 0.03 (0.7909)	1.23 ± 0.05 (0.4201)	1.47 ± 0.04 (0.3239)	1.67 ± 0.04 (0.5867)	1.72 ± 0.03 (0.7584)	1.78 ± 0.03 (0.7370)	1.75 ± 0.02 (0.7618)	1.78 ± 0.02 (0.7173)	1.82 ± 0.01 (0.7078)	0.6449
MDUMDA	4.45 ± 0.69 (0.0091)	1.53 ± 0.47 (0.2426)	1.14 ± 0.39 (0.5563)	2.99 ± 0.50 (0)	3.98 ± 0.74 (0)	4.05 ± 0.46 (0)	4.66 ± 0.54 (0)	N/A	N/A	0.1154
APSO	0.53 ± 0.01 (0.9000)	1.05 ± 0.06 (0.5266)	1.31 ± 0.03 (0.4366)	1.69 ± 0.05 (0.5778)	1.78 ± 0.02 (0.7383)	1.86 ± 0.02 (0.7110)	1.95 ± 0.02 (0.7094)	1.95 ± 0.01 (0.6456)	1.90 ± 0.01 (0.6712)	0.6574
CellularDE	1.53 ± 0.07 (0.6727)	1.50 ± 0.04 (0.2604)	1.64 ± 0.03 (0.2042)	2.46 ± 0.05 (0.2356)	2.62 ± 0.05 (0.4564)	2.76 ± 0.05 (0.4188)	2.75 ± 0.05 (0.5000)	2.73 ± 0.03 (0.3165)	2.61 ± 0.02 (0.3470)	0.3791
SFA	0.42 ± 0.07	0.89 ± 0.09	1.05 ± 0.04	1.48 ± 0.05	1.56 ± 0.06	N/A	1.87 ± 0.05	2.01 ± 0.04	1.99 ± 0.06	0.7037

	(0.9250)	(0.6213)	(0.6197)	(0.6711)	(0.8121)		(0.7304)	(0.6203)	(0.6301)	
CCPSO	0.09 ± 0.00 (1)	0.25 ± 0.01 (1)	0.75 ± 0.06 (0.8310)	1.21 ± 0.08 (0.7911)	1.40 ± 0.07 (0.8658)	1.47 ± 0.08 (0.8377)	1.50 ± 0.09 (0.8272)	1.76 ± 0.09 (0.7257)	N/A	0.8598
DMAFSA	0.55 ± 0.06 (0.8955)	0.78±0.06 (0.6864)	1.01 ± 0.05 (0.6479)	1.42 ± 0.06 (0.6978)	1.63 ± 0.06 (0.7886)	N/A	1.84 ± 0.07 (0.7382)	1.95 ± 0.05 (0.6456)	1.99 ± 0.04 (0.6301)	0.7163
TP-CPSO	0.40 ± 0.01 (0.9295)	0.75 ± 0.07 (0.7041)	0.96 ± 0.05 (0.6831)	1.18 ± 0.03 (0.8044)	1.32 ± 0.03 (0.8926)	1.36 ± 0.03 (0.8734)	1.40 ± 0.03 (0.8534)	1.50 ± 0.02 (0.8354)	1.56 ± 0.02 (0.8265)	0.8225
DynPopDE	N/A	1.03 ± 0.13 (0.5385)	1.39 ± 0.07 (0.3803)	N/A	N/A	N/A	2.10 ± 0.06 (0.6702)	2.34 ± 0.05 (0.4810)	2.44 ± 0.05 (0.4247)	0.4989
FTMPSO	0.18 ± 0.01 (0.9795)	0.47 ± 0.05 (0.8698)	0.67 ± 0.04 (0.8873)	0.93 ± 0.04 (0.9156)	1.14 ± 0.04 (0.9530)	N/A	1.32 ± 0.04 (0.8743)	1.61 ± 0.03 (0.7890)	1.67 ± 0.03 (0.7763)	0.8806
CDEPSO	0.41 ± 0.00 (0.9273)	0.97 ± 0.01 (0.5740)	1.22 ± 0.01 (0.5000)	1.54 ± 0.01 (0.6444)	2.62 ± 0.01 (0.4564)	N/A	2.20 ± 0.01 (0.6440)	1.54 ± 0.01 (0.8186)	2.11 ± 0.01 (0.5753)	0.6425
mNAFSA	0.38 ± 0.06 (0.9341)	0.55 ± 0.04 (0.8225)	0.90 ± 0.03 (0.7254)	1.25 ± 0.06 (0.7733)	1.47 ± 0.05 (0.8423)	N/A	1.65 ± 0.05 (0.7880)	1.83 ± 0.05 (0.6962)	1.84 ± 0.05 (0.6986)	0.7851
AMP/DE	N/A	N/A	0.90 ± 0.10 (0.7254)	1.40 ± 0.10 (0.7067)	N/A	N/A	N/A	1.70 ± 0.08 (0.7511)	2.00 ± 0.04 (0.6256)	0.7022
AMP/PSO	N/A	N/A	0.69 ± 0.03 (0.8732)	1.10 ± 0.09 (0.8400)	N/A	N/A	N/A	1.40 ± 0.00 (0.8776)	1.80 ± 0.00 (0.7169)	0.8269
BfCS-wVN	0.30 ± 0.06 (0.9523)	0.38 ± 0.21 (0.9231)	0.51 ± 0.11 (1)	0.74 ± 0.12 (1)	1.00 ± 0.11 (1)	0.97 ± 0.07 (1)	0.84 ± 0.06 (1)	1.11 ± 0.06 (1)	1.18 ± 0.08 (1)	0.9862

As can be observed in Table 8, the results of the BfCS-wVN are considerably better than those of other peer algorithms in most of the tested DOPs. Considering Table 8, when the number of peaks *(m*) is 1 or 5, the reported results by CCPSO are slightly better than those of BfCS-wVN. However, as the number of the peaks in the landscape is increased, BfCS-wVN becomes the best performing algorithm. From the results of Table 8, we can conclude that the BfCS-wVN algorithm is an effective method for handling DOPs.

7. Conclusion

A new adaptive method based on the cuckoo search has been proposed for dealing with continuous dynamic optimization problems. The key novelty of the proposed method is that instead of using the Levy flight with a single β value, the proposed method utilizes two values (one for short mutations and the other for long ones) for β. The proposed mechanism helps the cuckoo search to

have more control over the exploration and exploitation of the algorithm during the search process. Some other novelties presented in this paper can be summarized as follows:

(1) Introducing an adaptive mechanism for controlling the number of allowable eggs that can be laid by a cuckoo, in such a way that more fitness evaluations are allocated to fitter cuckoos and more promising nests.
(2) Using a reinforcement learning method called learning automaton to select the right flight type for each egg laid by a cuckoo. This mechanism makes the choice of β used in generating the flights more flexible. Furthermore, the mechanism uses the prior knowledge that a higher exploration in the early stages of the optimization and a higher exploitation in the final stages of the optimization is required.
(3) Eliminating the need for hand-tuning the exact number of cuckoos by devising a method for adapting the appropriate number of cuckoos during the run.
(4) Applying a greedy Hooke-Jeeves method as a local search mechanism for refining the global best egg.

In order to validate the effectiveness of the proposed methods, various experiments were carried out to compare the proposed approach with various state-of-the-art algorithms on different configurations of the moving peaks benchmark. The experimental results confirmed the effectiveness of the proposed approach.

Several future research directions could be pursued based on this paper. First, the values of the parameters of BfCS-wVN were adjusted based on some preliminary experiments. Therefore, a comprehensive sensitivity analysis on the effect of the parameters could be a direction for future research. Second, instead of restricting the values of β to just two values, a continuous action-set learning automata can be used to regulate the β parameter. Third, it might be beneficial to consider other learning algorithms, e.g., the pursuit algorithm, for the learning automata and fourth, hybridizing other heuristics into the proposed method may also prove an interesting approach.

Acknowledgment The authors would like to thank S. Moreitz for reviewing the paper.

References

1. Nguyen TT, Yang S, Branke J (2012) Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evolutionary Computation 6:1–24. doi: 10.1016/j.swevo.2012.05.001

2. Cobb HG (1990) An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. DTIC Document, NAVAL RESEARCH LAB WASHINGTON USA

3. Hu X, Eberhart RC (2002) Adaptive particle swarm optimization: detection and response to dynamic systems. In: Proceedings of the IEEE Congress on Evolutionary Computation. pp 1666–1670

4. Vavak F, Jukes KA, Fogarty TC (1998) Performance of a genetic algorithm with variable local search range relative to frequency of the environmental changes. Genetic Programming 22–25.

5. Vavak F, Jukes K, Fogarty TC (1997) Learning the local search range for genetic optimisation in nonstationary environments. In: Evolutionary Computation, 1997., IEEE International Conference on. IEEE, pp 355–360

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

7. 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 14:33–41.

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

9. Wang H, Wang D, Yang S (2007) Triggered memory-based swarm optimization in dynamic environments. In: Applications of Evolutionary Computing. Springer, Berlin, Heidelberg, pp 637–646

10. Hatzakis I, Wallace D (2006) Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, Seattle, Washington, USA, pp 1201–1208

11. Rossi C, Abderrahim M, Díaz JC (2008) Tracking moving optima using kalman-based predictions. Evolutionary Computation 16:1–30. doi: 10.1162/evco.2008.16.1.1

12. Grefenstette JJ (1999) Evolvability in dynamic fitness landscapes: a genetic algorithm approach. In: Proceedings of the IEEE Congress on Evolutionary Computation. IEEE, p 1–2038 Vol. 3

13. Blackwell T, Branke J (2004) Multi-swarm optimization in dynamic environments. In: Workshops on Applications of Evolutionary Computation. Springer, pp 489–500

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

15. Branke J, Kaussler T, Smidt C, Schmeck H (2000) A Multi-population Approach to Dynamic Optimization Problems. In: Parmee IC (ed) Evolutionary Design and Manufacture. Springer London, pp 299–307

16. C. Li, T. T. Nguyen, M. Yang, et al (2016) An Adaptive Multipopulation Framework for Locating and Tracking Multiple Optima. IEEE Transactions on Evolutionary Computation 20:590–605. doi: 10.1109/TEVC.2015.2504383

17. du Plessis MC, Engelbrecht AP (2012) Using Competitive Population Evaluation in a differential evolution algorithm for dynamic environments. European Journal of Operational Research 218:7–20. doi: 10.1016/j.ejor.2011.08.031

18. du Plessis MC, Engelbrecht AP (2013) Differential evolution for dynamic environments with unknown numbers of optima. Journal of Global Optimization 55:73–99. doi: 10.1007/s10898-012-9864-9

19. 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 43:881–897. doi: 10.1109/TSMCB.2012.2217491

20. Hashemi AB, Meybodi MR (2009) A multi-role cellular PSO for dynamic environments. In: Proceedings of the 14th International CSI Computer Conference. IEEE, pp 412–417

21. Hashemi AB, Meybodi MR (2009) Cellular PSO: A PSO for dynamic environments. In: The 4th International Symposium on Intelligence Computation and Applications. Springer, Berlin, Heidelberg, pp 422–433

22. Kamosi M, Hashemi AB, Meybodi MR (2010) A hibernating multi-swarm optimization algorithm for dynamic environments. In: Proceedings of the Second World Congress on Nature and Biologically Inspired Computing. IEEE, pp 363–369

23. Kamosi M, Hashemi AB, Meybodi MR (2010) A new particle swarm optimization algorithm for dynamic environments. In: Proceedings of the First International Conference on Swarm, Evolutionary, and Memetic Computing. Springer, Berlin, Heidelberg, pp 129–138

24. Kordestani JK, Rezvanian A, Meybodi MR (2014) CDEPSO: A bi-population hybrid approach for dynamic optimization problems. Applied Intelligence 40:682–694.

25. Lung RI, Dumitrescu D (2009) Evolutionary swarm cooperative optimization in dynamic environments. Natural Computing 9:83–94.

26. Nabizadeh S, Rezvanian A, Meybodi MR (2012) A multi-swarm cellular PSO based on clonal selection algorithm in dynamic environments. In: Proceedings of the International Conference on Informatics, Electronics & Vision. IEEE, pp 482–486

27. Mendes R, Mohais AS (2005) DynDE: A differential evolution for dynamic optimization problems. In: Proceedings of the IEEE Congress on Evolutionary Computation. IEEE, pp 2808–2815

28. Noroozi V, Hashemi AB, Meybodi MR (2011) CellularDE: A cellular based differential evolution for dynamic optimization problems. In: Proceedings of the 10th International Conference on Adaptive and Natural Computing Algorithms. Springer, Berlin, Heidelberg, pp 340–349

29. Novoa-Hernández P, Corona CC, Pelta DA (2011) Efficient multi-swarm PSO algorithms for dynamic environments. Memetic Computing 3:163–174.

30. Novoa-Hernández P, Corona CC, Pelta DA (2013) Self-adaptive, multipopulation differential evolution in dynamic environments. Soft Computing 17:1861–1881.

31. Sharifi A, Noroozi V, Bashiri M, et al (2012) Two phased cellular PSO: A new collaborative cellular algorithm for optimization in dynamic environments. In: Proceedings of the IEEE Congress on Evolutionary Computation. pp 1–8

32. Yang S, Li C (2010) A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Transactions on Evolutionary Computation 14:959–974. doi: 10.1109/TEVC.2010.2046667

33. Yang S, Li C (2008) Fast multi-swarm optimization for dynamic optimization problems. In: Proceeding of the Fourth International Conference on Natural Computation. IEEE, pp 624–628

34. Yazdani D, Akbarzadeh-Totonchi MR, Nasiri B, Meybodi MR (2012) A new artificial fish swarm algorithm for dynamic optimization problems. In: IEEE Congress on Evolutionary Computation. pp 1–8

35. Blackwell TM, Bentley P (2002) Don't push me! Collision-avoiding swarms. In: Proceedings of the 2002 Congress on Evolutionary Computation. pp 1691–1696

36. Blackwell TM (2002) Dynamic search with charged swarms. In: Genetic and Evolutionary Computation Conference. pp 9–13

37. Blackwell T (2007) Particle swarm optimization in dynamic environments. In: Evolutionary Computation in Dynamic and Uncertain Environments. Springer, Berlin, Heidelberg, pp 29–49

38. Yazdani D, Nasiri B, Sepas-Moghaddam A, Meybodi MR (2013) A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization. Applied Soft Computing 13:2144–2158. doi: 10.1016/j.asoc.2012.12.020

39. Parrott D, Li X (2006) Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Transactions on Evolutionary Computation 10:440–458. doi: 10.1109/TEVC.2005.859468

40. Bird S, Xiaodong Li (2007) Using regression to improve local convergence. pp 592–599

41. Li C, Yang S (2012) A general framework of multipopulation methods with clustering in undetectable dynamic environments. IEEE Transactions on Evolutionary Computation 16:556–577. doi: 10.1109/TEVC.2011.2169966

42. Nickabadi A, Ebadzadeh MM, Safabakhsh R (2012) A competitive clustering particle swarm optimizer for dynamic optimization problems. Swarm Intelligence 6:177–206.

43. Lili Liu, Shengxiang Yang, Qing Wang (2010) Particle Swarm Optimization With Composite Particles in Dynamic Environments. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 40:1634–1648. doi: 10.1109/TSMCB.2010.2043527

44. Karimi J, Nobahari H, Pourtakdoust SH (2012) A new hybrid approach for dynamic continuous optimization problems. Applied Soft Computing 12:1158–1167. doi: 10.1016/j.asoc.2011.11.005

45. Grefenstette JJ (1992) Genetic algorithms for changing environments. In: Parallel Problem Solving from Nature. North Holland, pp 137–144

46. Andersen HC (1991) An investigation into genetic algorithms, and the relationship between speciation and the tracking of optima in dynamic functions. Brisbane, Australia: Honors, Queensland Univ

47. Morrison RW (2004) Designing Evolutionary Algorithms for Dynamic Environments. Springer-Verlag, Berlin

48. Oppacher F, Wineberg M (1999) The shifting balance genetic algorithm: Improving the GA in a dynamic environment. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1. Morgan Kaufmann Publishers Inc., pp 504–510

49. Ursem RK Multinational GA Optimization Techniques in Dynamics Environments.

50. M. C. du Plessis, A. P. Engelbrecht (2008) Improved differential evolution for dynamic optimization problems. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence). pp 229–234

51. Noroozi V, Hashemi AB, Meybodi MR (2012) Alpinist CellularDE: A cellular based optimization algorithm for dynamic environments. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, Philadelphia, Pennsylvania, USA, pp 1519–1520

52. Rezvanian A, Meybodi MR (2010) LACAIS: Learning automata based cooperative artificial immune system for function optimization. In: Proceedings of the Third International Conference on Contemporary Computing. Springer, Berlin, Heidelberg, pp 64–75

53. Trojanowski K, Wierzchoń ST (2009) Immune-based algorithms for dynamic optimization. Information Sciences 179:1495–1515.

54. Nasiri B, Meybodi MR (2012) Speciation based firefly algorithm for optimization in dynamic environments. International Journal of Artificial Intelligence 8:118–132.

55. Turky AM, Abdullah S (2014) A multi-population harmony search algorithm with external archive for dynamic optimization problems. Information Sciences 272:84–95.

56. Fouladgar N, Lotfi S (2015) A novel approach for optimization in dynamic environments based on modified cuckoo search algorithm. Soft Computing 1–15.

57. Lung RI, Dumitrescu D (2007) A collaborative model for tracking optima in dynamic environments. In: Proceedings of the IEEE Congress on Evolutionary Computation. IEEE, pp 564–567

58. Moser I, Hendtlass T (2007) A simple and efficient multi-component algorithm for solving dynamic function optimisation problems. In: 2007 IEEE Congress on Evolutionary Computation. IEEE, pp 252–259

59. Moser I, Chiong R (2010) Dynamic function optimisation with hybridised extremal dynamics. Memetic Computing 2:137–148.

60. Sharifi A, Kordestani JK, Mahdaviani M, Meybodi MR (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.

61. Xin-She Yang, Deb S (2009) Cuckoo Search via Lévy flights. In: World Congress on Nature & Biologically Inspired Computing. pp 210–214

62. Bhargava V, Fateen S-EK, Bonilla-Petriciolet A (2013) Cuckoo search: a new nature-inspired optimization method for phase equilibrium calculations. Fluid Phase Equilibria 337:191–200.

63. Piechocki J, Ambroziak D, Palkowski A, Redlarski G (2014) Use of Modified Cuckoo Search algorithm in the design process of integrated power systems for modern and energy self-sufficient farms. Applied Energy 114:901–908. doi: 10.1016/j.apenergy.2013.07.057

64. Yildiz A (2013) Cuckoo search algorithm for the selection of optimal machining parameters in milling operations. Int J Adv Manuf Technol 64:55–61. doi: 10.1007/s00170-012-4013-7

65. Chandrasekaran K, Simon SP (2012) Multi-objective scheduling problem: Hybrid approach using fuzzy assisted cuckoo search algorithm. Swarm and Evolutionary Computation 5:1–16. doi: 10.1016/j.swevo.2012.01.001

66. Yang XS, Deb S (2013) Multiobjective cuckoo search for design optimization. Computers & Operations Research 40:1616 – 1624.

67. Abedi Firouzjaee H, Kordestani JK, Meybodi MR (2016) Cuckoo search with composite flight operator for numerical optimization problems and its application in tunnelling. Engineering Optimization 1–20.

68. Thathachar MAL, Sastry PS (2002) Varieties of learning automata: an overview. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 32:711–722. doi: 10.1109/TSMCB.2002.1049606

69. Narendra KS, Thathachar MAL (1974) Learning automata - A survey. IEEE Transactions on Systems, Man, and Cybernetics 4:323–334. doi: 10.1109/TSMC.1974.5408453

70. Hashemi AB, Meybodi MR (2011) A note on the learning automata based algorithms for adaptive parameter selection in PSO. Applied Soft Computing 11:689–705. doi: 10.1016/j.asoc.2009.12.030

71. Kordestani JK, Ahmadi A, Meybodi MR (2014) An improved differential evolution algorithm using learning automata and population topologies. Applied Intelligence 41:1150–1169.

72. Mahdaviani M, Kordestani JK, Rezvanian A, Meybodi MR (2015) LADE: Learning Automata Based Differential Evolution. International Journal on Artificial Intelligence Tools 24:1550023.

73. Moghiss V, Meybodi MR, Esnaashari M (2010) An intelligent protocol to channel assignment in wireless sensor networks: Learning automata approach. In: Proceedings of the International Conference on Information, Networking and Automation. IEEE, pp 338–343

74. Esnaashari M, Meybodi MR (2011) A cellular learning automata-based deployment strategy for mobile wireless sensor networks. Journal of Parallel and Distributed Computing 71:988–1001.

75. Akbari Torkestani J, Meybodi MR (2011) A cellular learning automata-based algorithm for solving the vertex coloring problem. Expert Systems with Applications 38:9237–9247. doi: 10.1016/j.eswa.2011.01.098

76. Barzegar S, Davoudpour M, Meybodi MR, et al (2011) Formalized learning automata with adaptive fuzzy coloured Petri net; an application specific to managing traffic signals. Scientia Iranica 18:554–565. doi: 10.1016/j.scient.2011.04.007

77. Akbari Torkestani J (2012) An adaptive learning automata-based ranking function discovery algorithm. Journal of Intelligent Information Systems 39:441–459.

78. Thathachar MA, Sastry PS (2003) Networks of learning automata: Techniques for online stochastic optimization. Springer Science & Business Media

79. Wang F, He XS, Wang Y, Yang SM (2012) Markov model and convergence analysis based on cuckoo search algorithm. Computer Engineering 11:55.

80. Hooke R, Jeeves TA (1961) Direct Search Solution of Numerical and Statistical Problems. Journal of the ACM (JACM) 8:212–229.

81. Branke J, Schmeck H (2003) Designing evolutionary algorithms for dynamic optimization problems. In: Advances in Evolutionary Computing: Theory and Applications. Springer, Berlin, Heidelberg, pp 239–262

82. Nasiri B, Meybodi MR (2016) Improved Speciation-Based Firefly Algorithm in Dynamic and Uncertain Environments. Journal of Information Science and Engineering 32:661–676.

83. Wu V, Wang Y, Liu X, Ye J (2010) Multi-population and diffusion UMDA for dynamic multimodal problems. Journal of Systems Engineering and Electronics 21:777–783. doi: 10.3969/j.issn.1004-4132.2010.05.010

84. Rezazadeh I, Meybodi M, Naebi A (2011) Adaptive Particle Swarm Optimization Algorithm for Dynamic Environments. In: Tan Y, Shi Y, Chai Y, Wang G (eds) Advances in Swarm Intelligence. Springer Berlin Heidelberg, pp 120–129

85. Yazdani D, Nasiri B, Sepas-Moghaddam A, et al (2014) mNAFSA: a novel approach for optimization in dynamic environments with global changes. Swarm and Evolutionary Computation 18:38–53.

86. C. Li, T. T. Nguyen, M. Yang, et al (2016) An Adaptive Multipopulation Framework for Locating and Tracking Multiple Optima. IEEE Transactions on Evolutionary Computation 20:590–605. doi: 10.1109/TEVC.2015.2504383