A Cluster-Based Differential Evolution Algorithm With External Archive for Optimization in Dynamic Environments

Udit Halder, Swagatam Das, *Member, IEEE*, and Dipankar Maity

***Abstract*—This paper presents a *C*luster-based *D*ynamic *D*ifferential *E*volution with external *Ar*chive (CDDE_Ar) for global optimization in dynamic fitness landscape. The algorithm uses a multipopulation method where the entire population is partitioned into several clusters according to the spatial locations of the trial solutions. The clusters are evolved separately using a standard differential evolution algorithm. The number of clusters is an adaptive parameter, and its value is updated after a certain number of iterations. Accordingly, the total population is redistributed into a new number of clusters. In this way, a certain sharing of information occurs periodically during the optimization process. The performance of CDDE_Ar is compared with six state-of-the-art dynamic optimizers over the moving peaks benchmark problems and dynamic optimization problem (DOP) benchmarks generated with the generalized-dynamic-benchmark-generator system for the competition and special session on dynamic optimization held under the 2009 IEEE Congress on Evolutionary Computation. Experimental results indicate that CDDE_Ar can enjoy a statistically superior performance on a wide range of DOPs in comparison to some of the best known dynamic evolutionary optimizers.**

***Index Terms*—Clustering, differential evolution (DE), dynamic optimization problems, evolutionary algorithms (EAs), self-adaptation.**

I. Introduction

RESEARCH WORKS on and with evolutionary algorithms (EAs) mostly deal with the static optimization problems where the functional landscape does not change over time. However, real-world problems can very often be dynamic in nature. For such problems, the function landscape changes with time, resulting into change in locations of the optima as well. For solving dynamic optimization problems (DOPs), the optimizer should be able to track the movement of the optima or detect emerging new optima in the search space [1]. Practical examples of such situations are price fluctuations, financial variations, the stochastic arrival of new tasks in a scheduling problem, machine breakdown or maintenance, etc. In dynamic environment, convergence tendency imposes severe limitation on a conventional EA. If the EA converges rapidly, it will be unable to respond effectively to the dynamically changing environment. Therefore, in case of DOPs, our main challenge is to maintain the diversity and, at the same time, to produce high-quality solutions by tracking the moving optima. Changing environments necessitate efficient algorithms, which should not only be able to find the global optimum but also detect whether an environmental change has occurred or not, and if yes, they should track the new optimum in the changed environment.

Differential evolution (DE) [2]–[5] has emerged as one of the most powerful real-parameter optimizers of current interest. DE operates through the similar computational steps as employed by a standard EA. However, unlike the traditional EAs, the DE variants perturb the current-generation population members with the scaled differences of randomly selected and distinct population members. Therefore, no separate probability distribution has to be used, which makes the scheme self-organizing in this respect. Classical DE faces difficulties when applied to DOPs, mainly due to two factors. First, DE individuals have a tendency to converge prematurely into small basins of attraction surrounding the local and global optima with progress of the search [6], [7]. Thereafter, if any change occurs in the position of the optima, DE starts lacking sufficient explorative power to track down the new optima due to the similar individuals and the consequent perturbations with small difference vectors. Second, that DE may occasionally stop proceeding toward the global optimum even though the population has not converged to a local optimum or any other point [7]. Occasionally, even new individuals may enter the population, as in DE, where an offspring, having the same fitness as that of the parent, replaces the parent in the next generation, and this can frequently happen in the flat portions of the functional landscape. In this case, however, the algorithm does not progress by finding any better solution than the current best. This situation is usually referred to as *stagnation*. Evidently, researchers have made attempts to introduce suitable algorithmic modifications in DE for enabling it to continually track the changing optima under dynamic conditions. A brief account of such approaches has been presented in Section II-B.

This paper proposes an adaptive DE variant called *C*luster-based *D*ynamic *D*ifferential *E*volution with external *Ar*chive (CDDE_Ar) to provide elegant solutions to real-parameter DOPs. In the proposed algorithm, the entire population is divided into a number of clusters according to the spatial

Manuscript received November 15, 2011; revised May 7, 2012; accepted August 28, 2012. Date of publication October 18, 2012; date of current version May 10, 2013. This paper was recommended by Associate Editor F. Hoffmann.

U. Halder and D. Maity are with the Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata 700 032, India (e-mail: udithalder99@gmail.com; dipankarmaity1991@gmail.com).

S. Das is with Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata 700 108, India (e-mail: swagatam.das@isical.ac.in).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2012.2217491

positions of the individuals, and the number of clusters can vary throughout the optimization process in an adaptive manner. Every member of a cluster evolves through DE-type mutation and recombination. Moreover, there is no information sharing among the clusters until the redistribution of the individuals occurs due to change in the number of clusters. To detect an environmental change, a memory is used. For adaptation to the new environment, an external archive (Ar) is employed. Members of the archive are the best individuals from the clusters that have already converged in the present environment.

The performance of the proposed algorithm is compared with those of six other state-of-the-art EAs that are specifically tailored to solve single-objective and real-parameter DOPs. For such comparative study, we use the moving peaks benchmark (MPB) problems [8] and a suite of DOPs generated from the generalized dynamic benchmark generator (GDBG) [9], [10], originally proposed for the Competition on Evolutionary Computation in Dynamic and Uncertain Environments held under the 2009 IEEE Congress on Evolutionary Computation (CEC 2009).

This paper is organized in the following way. Section II gives a brief description of classical DE and also briefly reviews the literature on the application of EAs to solve DOPs. The proposed algorithm is described in sufficient details under Section III. Section IV focuses on an empirical study of the search mechanism as well as on the parametric setup of the algorithm. Section V provides and discusses the results of comparing the proposed algorithm with six peer algorithms on both MPB and GDBG problems. Finally, Section VI concludes this paper and unfolds some important future research issues.

II. Scientific Background

A. DE Algorithm

DE is arguably one of the most powerful real-parameter optimization algorithms currently in use. It works through a simple cycle of stages, including initialization, mutation, crossover, and selection. Each stage is briefly described hereinafter.

1) Initialization of the Parameter Vectors: DE searches for a global optimum point in the D-dimensional real-parameter space $\Re^D$. It begins with a randomly initiated population of NP D-dimensional real-valued parameter vectors. Each vector forms a candidate solution of the multidimensional optimization problem. We shall denote subsequent generations in DE by $G = 0, 1, \ldots, G_{\max}$. Since the parameter vectors are likely to be changed over different generations, we adopt the following notation for representing the ith vector of the population at the current generation:

$$\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, \ldots\ldots, x_{D,i,G}]. \tag{1}$$

For each parameter of the problem, there may be a certain range within which the value of the parameter should be restricted, often because parameters are related to physical components or measures that have natural bounds (for example, if one parameter is a length or mass, it cannot be negative). The initial population (at $G = 0$) should cover this range as much as possible by uniformly randomizing individuals within the search space constrained by the prescribed minimum and maximum bounds

$$\vec{X}_{\min} = \{x_{1,\min}, x_{2,\min}, \ldots, x_{D,\min}\} \text{ and}$$
$$\vec{X}_{\max} = \{x_{1,\max}, x_{2,\max}, \ldots, x_{D,\max}\}.$$

Hence, the jth component of the ith vector is initialized as

$$x_{j,i,0} = x_{j,\min} + rand_{i,j}[0,1] \cdot (x_{j,\max} - x_{j,\min}) \tag{2}$$

where $rand_{i,j}[0,1]$ is a uniformly distributed random number in [0, 1] and is instantiated independently for each component of the ith vector.

2) Mutation With Difference Vectors: After initialization, DE creates a *donor* vector $\vec{V}_{i,G}$ corresponding to each population member or *target* vector $\vec{X}_{i,G}$ in the current generation through mutation. The five most popular mutation strategies of DE are listed as follows:

$$\text{“DE/rand/1"}: \vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot \left(\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}\right). \tag{3a}$$

$$\text{“DE/best/1"}: \vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot \left(\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}\right). \tag{3b}$$

$$\text{“DE/current-to-best/1"}: \vec{V}_{i,G} = \vec{X}_{i,G} + F \cdot (\vec{X}_{best,G} - \vec{X}_{i,G}) + F \cdot \left(\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}\right). \tag{3c}$$

$$\text{“DE/best/2"}: \vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot \left(\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}\right) + F \cdot \left(\vec{X}_{r_3^i,G} - \vec{X}_{r_4^i,G}\right). \tag{3d}$$

$$\text{“DE/rand/2"}: \vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot \left(\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}\right) + F \cdot \left(\vec{X}_{r_4^i,G} - \vec{X}_{r_5^i,G}\right). \tag{3e}$$

The indices r_1^i, r_2^i, r_3^i, r_4^i, and r_5^i are mutually exclusive integers randomly chosen from the range $[1, Np]$, and all are different from the base index i. These indices are randomly generated anew for each donor vector. The scaling factor F is a positive control parameter. $\vec{X}_{best,G}$ is the vector having the best fitness (i.e., the lowest objective function value for a minimization problem) in the population at generation G. The general convention used for naming the various mutation strategies is DE/x/y/z, where DE stands for differential evolution, x represents a string denoting the vector to be perturbed, and y is the number of difference vectors considered for the perturbation of x. z stands for the type of crossover being used (*exp*: exponential; *bin*: binomial).

3) Crossover: Through crossover, the donor vector mixes its components with the target vector $\vec{X}_{i,G}$ to form a trial vector $\vec{U}_{i,G} = [u_{1,i,G}, u_{2,i,G}, u_{3,i,G}, \ldots, u_{D,i,G}]$. The DE family of algorithms uses mainly two kinds of crossover methods—*exponential* (or two-point modulo) and *binomial* (or uniform) [2]. We, here, only outline the binomial crossover as the proposed DE variant uses this scheme. Binomial crossover is performed on each of the D variables whenever a randomly generated number between 0 and 1 is less than or equal to the Cr value. In this case, the number of parameters inherited from the donor has a (nearly) binomial distribution. The scheme may be outlined as

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{if } (rand_{i,j}[0,1] \le Cr \text{ or } j = j_{rand}) \\ x_{j,i,G}, & \text{otherwise} \end{cases} \tag{4}$$

where, as before, $rand_{i,j}[0,1]$ is a uniformly distributed random number, which is called anew for each jth component of the ith parameter vector. $j_{rand} \in [1, 2, \ldots, D]$ is a randomly chosen index, which ensures that $\vec{U}_{i,G}$ gets at least one component from $\vec{V}_{i,G}$. It is instantiated once for each vector per generation.

4) Selection: Selection determines which one between the target and the trial vector survives to the next generation i.e., at $G = G + 1$. The selection operation for minimization problems can be outlined as

$$\begin{aligned} \vec{X}_{i,G+1} &= \vec{U}_{i,G}, \qquad \text{if } f(\vec{U}_{i,G}) \le f(\vec{X}_{i,G}) \\ &= \vec{X}_{i,G}, \qquad \text{if } f(\vec{U}_{i,G}) > f(\vec{X}_{i,G}) \end{aligned} \tag{5}$$

where $f(\vec{X})$ is the objective function to be minimized. Therefore, if the new trial vector yields an equal or lower value of the objective function, it replaces the corresponding target vector in the next generation; otherwise, the target is retained in the population.

B. EAs for DOPs—An Overview

The first attempt to use EAs for solving DOPs dates back to 1966 and was due to Fogel and his colleagues [11]. Since late 1980s, the topic started to attract the attention of more and more researchers throughout the world, resulting into a steady increase in the publication in this area. Comprehensive surveys on the application of EAs for solving DOPs can be found in [1], [12], and [13].

A number of interesting approaches were taken by the researchers in order to make an EA adapt itself to a changing fitness landscape. For example, the hypermutation strategy [14] attempts to generate diversity after an environmental change has been detected by escalating the mutation rate drastically for some generations after the change has occurred. Morrison and De Jong [15] indicated that larger hypermutation bursts track the optimum better when the environmental changes are frequent while lower hypermutation levels perform better when the changes are less frequent. On the other hand, the mutation rate is gradually increased in variable local search [16] that basically replaces the information about the previously successful individuals with random information with an objective of increasing the diversity. Under the random immigrant scheme [17], randomly generated individuals are regularly inserted in the population in order to keep up the diversity throughout the runs. Fitness sharing and crowding [18] can also be employed for maintaining the diversity in dynamic optimization. For tracking multiple peaks in the functional landscape, the population may be divided into a number of subpopulations. These subpopulations can then store information about several promising regions of the search space and thus can serve as a form of diverse self-adaptive memory. Examples of this approach can be found in the multinational genetic algorithms (GA) [19], the shifting balance GA [20], and the self-organizing scouts [21], [22].

An external memory integrated with an EA may help the latter to recall useful information from the past generations, and this seems very helpful when the changing optimum repeatedly returns to the previous positions. In addition, the memory provides diversity by retaining former good solutions, which otherwise would have been lost in the selection process, and reintroducing (parts of) these solutions on a later occasion. Two types of memories can be distinguished for dynamic EAs—*explicit memory* and *implicit memory*. Explicit memory-oriented GAs incorporate strategies for storing solutions and reintroducing them on later occasions during the run [23]–[25]. On the other hand, GAs with implicit memory generally use redundant genetic representations. The most common example is employing a diploid genetic structure [12], [26]–[28]. A diploid GA possesses two sets of chromosomes instead of a common single set (haploid) possessed by the regular GAs. Consequently, in this type of GAs, two genes can involve into a competition for the same phenotypic feature in the same individual. Lewis *et al.* [28] indicated that a diploid structure alone is not enough for a diploid GA to adapt to changing environments. Frequently switching the values from dominant to recessive and vice versa can be useful for obtaining acceptable results. Some other useful approaches of using GAs to solve DOPs can be found in [29]–[33].

Aside from GAs, the particle swarm optimization (PSO) methods also received a great deal of attention from the researchers working on DOPs. An efficient PSO is expected to continuously track the moving optima in a dynamic environment [34]. Several modified PSO algorithms [35]–[40] were proposed for addressing DOPs. Liu *et al.* [41] presented a PSO algorithm with composite particles (PSO-CP) that groups the swarm into a set of composite particles based on their similarity and by using a "worst first" principle. Inspired by the composite particle phenomenon in physics, elementary members in each composite particle interact via a velocity-anisotropic reflection scheme, and a scattering operator is used for regaining local diversity within the composite particles. Moreover, an integral movement strategy is introduced to promote the swarm diversity. Parrott and Li proposed a PSO algorithm that uses the idea of species to detect multiple peaks over a dynamic fitness landscape [42].

Since late 1990s, DE started to receive attention from DOP researchers. Mendes and Mohais presented dynamic differential evolution (DynDE) [43], a multipopulation DE algorithm, developed specifically to optimize slowly time-varying objective functions. In DynDE, the diversity of the population is maintained in two ways: first, by reinitializing a population if the best individual of the population moves too close to the best individual of another population and, second, by the randomization of one or more population vectors by adding a random deviation to the components. The authors showed that DynDE is capable of solving the MPB problems efficiently. Brest *et al.* [44] investigated a self-adaptive DE algorithm (jDE), where the control parameters F and CR are self-adapted, and a multipopulation method with an aging mechanism is used to improve performance on DOPs. This algorithm ranked first in the Competition on "Evolutionary Computation in Dynamic and Uncertain Environments" under IEEE CEC 2009. Some other interesting research efforts on modifying DE for optimizing in dynamic environments can be found in [45]–[49].

Recently, a variant of the ant colony algorithm, called differential ant-stigmergy algorithm (DASA) [50], has been applied to solve DOPs. A key feature of DASA is that it remembers the move in the search space that improves the current best solution and can direct further search based on this move. In [51], the authors undertook a detailed comparative study of jDE and DASA on the GDBG benchmarks by using nonparametric statistical test procedures. The authors de Franca and Von Zuben proposed an artificial immune network for artificial immune network for dynamic optimization (dopt-aiNet) [52] by introducing a set of complementary mutation operators and a better mechanism to maintain the diversity of solutions in the original dopt-aiNet [53] algorithm meant for static optimization problems.

III. Proposed Algorithm: CDDE_Ar

This section provides a detailed account of the proposed CDDE_Ar algorithm. The algorithm partitions the whole population into several clusters based on spatial positions of the individuals by using the k-means algorithm [54]. The k-means clustering technique has been previously used for generating multiple populations in EAs running on static optimization problems. For example, Kennedy [55] proposed a PSO algorithm, where k-means clustering was used to partition the swarm into stereotypical groups of particles that are working toward particular optima in the multimodal problem space. The cluster centers were substituted for the individuals' and neighbors' best previous positions in the corresponding PSO algorithm. However, CDDE_Ar does not deal with the cluster centers, and also, it includes additional mechanisms for solving DOPs, while the algorithm in [55] is not tailored to optimize in dynamic environments. In CDDE_Ar, the number of clusters is dynamically varied throughout the optimization in an adaptive fashion. An external memory archive is also used to enhance the performance of the algorithm in the dynamic environment. Before describing the salient features of the algorithm, we first define some terms regarding the clusters.

1) If the ith cluster contains n_i members and the members are denoted as $\vec{X}_1, \vec{X}_2, \ldots, \vec{X}_{n_i}$, then the center $(\vec{C}_i)$ is determined as

$$\vec{C}_i = \frac{\sum_{i=1}^{n_i} \vec{X}_i}{n_i}. \tag{6}$$

2) The radius (R) of a cluster is defined as the mean distance (Euclidean) of the members from the center of the cluster. Thus, R can be written as

$$R_i = \frac{\sum_{p=1}^{n_i} \|\vec{X}_p - C_i\|}{n_i} \tag{7}$$

where $\|.\|$ denotes the standard Euclidean norm.

3) Any cluster cannot contain an arbitrarily high number of members. Hence, there is an upper limit of members called pop_max in each cluster.

4) For each cluster, we define a fixed value, called *convergence radius* (R_{conv}), which can be calculated as

$$R_{conv} = \|\vec{X}_{upper} - \vec{X}_{lower}\| \cdot 10^{-3} \tag{8}$$

where $\vec{X}_{upper}$ and $\vec{X}_{lower}$ are the upper bound and lower bound vectors in the search region. Scaling by 10^{-3} provides reasonably good results over a wide variety of benchmark functions as we have found through empirical experiments. Next, we elaborate on different components of the proposed DE scheme.

A. Initialization and Clustering

A certain number of individuals (NP) are initialized randomly in the feasible search space to cover the search volume as uniformly as possible. Then, the k-means clustering algorithm [54], [56] is invoked to partition the population into a number of clusters based on their geographical locations. Since k-means is a very well-known algorithm, we do not reiterate its steps here to save space. Initially, NP members are divided into k clusters. Clustering is done to discourage premature convergence, and the solutions within each cluster are improved separately by using the evolutionary operators of DE. There is no information exchange among the clusters in a specific iteration. However, we make room for sharing information about the search landscape among the clusters after a certain number of iterations called *time span (TS)*, which is discussed later in sufficient details.

B. Cluster Improvement

In every cluster, the members are updated according to the exploitative "DE/best/1/bin" scheme [see (3b)], where $\vec{X}_{best,G}$ is replaced by $\vec{X}_{best\_C_i,G}$, which is the best solution of the ith cluster in the current generation. Therefore, the optimization process runs separately in each cluster, and thus, solutions belonging to different clusters are attracted to different promising points of the fitness landscape. The parallel evolution of multiple clusters with an exploitative DE scheme permits detailed search in each cluster, at the same time discouraging the premature convergence of all individuals. It is expected that every cluster will discover a distinct optimum. The radius of a cluster [see (7)] is calculated in every iteration and for each cluster. If it is found that the radius of a cluster is sufficiently small, then we can infer that individuals of the cluster have converged well to a small Euclidean neighborhood surrounding an optimum. In this case, the solutions are pretty close to each other, and hence, the cluster will not improve further. Hence, we have to detect the convergence of a cluster in terms of convergence radius R_{conv} [see (8)]. If the radius of a cluster becomes smaller than R_{conv}, we delete the cluster storing its best individual in the external archive Ar. If the cluster contains the globally best individual of the current iteration and even if $R < R_{conv}$, the cluster is not deleted. It will be discussed later that the number of members of a specific cluster is not fixed throughout the optimization process; rather, it varies with the dynamic and self-adaptive behavior of the algorithm. Therefore, it may happen that the size of a cluster becomes very high at some point of time, but we restrict the algorithm from doing so. During optimization, if the number of individuals of a cluster exceeds pop_max, then the best pop_max solutions are selected and others are discarded.

C. Performance Evaluation and Redistribution

During optimization, it may so happen that two or more clusters come close to each other or get overlapped to a high degree. Then, they will practically search the same domain of the functional landscape, and the very purpose of clustering will be lost. To prevent this, we must have some redistribution strategy. For this purpose, we used a feedback technique through performance evaluation. After a specific time span *TS*, the performance of the algorithm is evaluated, and accordingly, the cluster numbers are updated. The performance of the algorithm is evaluated in terms of the number of changes in the global best value over TS. The number of improvements in the global best value over TS is counted, and this number of improvements is denoted as the change number $(changeno)$ over a TS. If the change is satisfactory, then we can say that the algorithm is performing well, and to reduce the function evaluation (FE) cost, the total cluster number is decreased by one. The total population is divided into a decreased number of clusters by another call to k-means. Thus, spatially nearer solutions are merged into one cluster. A change of more than 30% of TS (i.e., $changeno \geq 0.3 \cdot TS$) can be taken as satisfactory. If the change number is less than that, then it can be said that the algorithm is not performing well at all. There is a high chance that the global optimum lies in some other region which is not covered by the present clusters. Thus, the cluster number is increased by one, as if the current situation is demanding it. A new cluster is introduced in the search space whose individuals are generated randomly. Then, the total population is clustered into increased cluster numbers by k-means. Hence, after every TS, the feedback test arranges the population in a proper order according to the current situation. If the situation is acceptably good, the cluster number is decreased; otherwise, it is increased.

Overlapping handling is a serious issue when dealing with multiple populations. When two subpopulations come too close, it means that they are basically searching in the same area instead of searching in different regions, which they were intended for. In the proposed algorithm, we do not have to explicitly care about overlapping handling. This is because redistribution or reclustering is taking place after every TS. Therefore, even if two clusters come very close to each other, they get merged into a single cluster after the reclustering due to the property of the k-means algorithm. Thus, overlapping handling is automatically taken care of by the redistribution process.

It is to be noted here that the performance evaluation test is based on the number of changes in the best value. This implies that we exert stress on the consistency of change and not on the quality of the change. However, quality is affected when the change is very low, yet we are getting change in almost every time. This can happen when a cluster gets very near to an optimum and is just tuning itself to get further good result, and in that case, we get change in almost every iteration, although the quantity of change is very low. Under such a situation, we cannot say that the algorithm is doing very well because it is actually stagnant in the basin of attraction of an optimum, which may not be the global one. Therefore, if we apply the same strategy here also, our purpose of redistribution may not be fulfilled. Thus, in addition to calculating the total number of changes over TS, we also calculate the average percentage of change in the global best value over that period. The average percentage change (avg_change) is obtained by dividing the total amount of percentage change in the global best value $(totchange)$ over a TS by the total change number $(changeno)$ in that TS. If it is found that the average change is very poor (usually on the order of 10^{-3}), then we increase the cluster number treating the situation as a bad one, even if the number of changes is high.

The number of clusters is bounded from above and below. The bounds are designated as $Clust_{max}$ and $Clust_{min}$, respectively. The maximum number of clusters is taken as twice the initial cluster number (k), and minimum cluster numbers can assume a low value (for example, 1 or 2). Generally, the updated cluster number after a TS can be written as

$$\begin{aligned} clusterno &= \min\{Clust_{max}, prev\_clusterno + 1\}, \\ &\quad \text{for increment,} \\ clusterno &= \max\{Clust_{min}, prev\_clusterno - 1\}, \\ &\quad \text{for decrement.} \end{aligned} \tag{9}$$

The pseudocode of the performance evaluation technique is given in Algorithm 2.

Algorithm 2: Determine_increase$(changeno, totchange)$

$avg\_change \leftarrow totchange/changeno$
if $changeno \geq 0.3.TS$ & $avg\_change > 1e-03$**do**
  $increase \leftarrow 1$
else
  $increase \leftarrow -1$
end if
$temp \leftarrow clusterno - increase$
if $temp > Clust_{max}$ **or** $temp < Clust_{min}$ **do**
  $increase \leftarrow 0$
end if
Return *increase*

D. Detection of Environmental Change and Adaptation

An efficient dynamic optimizer should detect whether an environmental change has occurred or not and take actions accordingly. For the detection of change in the environment, we keep a memory. The functional values of the best individuals of each cluster are stored in that memory at the end of each iteration. At the start of the next iteration, we re-evaluate the best individuals of each cluster before evolving them and tally the new values with the values kept in the memory. This means that the new and the old values in memory represent the functional values of the best individuals of the clusters in the new generation and in the previous generation, respectively. Clearly, if there is a mismatch in those two values, it can be inferred that an environmental change must have taken place. Whenever a change in the environment is detected, we stop continuing the optimization process, and it is restarted afresh.

In addition to changing the search space and thereby changing the functional values of the individuals, some challenging benchmark problems, e.g., GDBG (in T7) [10], also change the dimensionality of the problem after a specific number of FEs. The objective function of the GDBG system changes the dimension by some rules within a limit. It modifies the current solution vector by adding or deleting dimensions of the current solution and returns the changed dimension of the problem along with the modified solution vector. Therefore, examining the objective value of the best individual vector in every iteration gives us an opportunity to detect whether a dimensional change has occurred or not. Whenever the dimension of the best individual vector returned by the cost function does not match with the dimension of the previous one, a dimensional change is detected in the environment. We have treated the dimensional change as a kind of environmental change, and the steps that we have taken in this case are similar to the functional landscape change case as described earlier. The only difference is that all members of the population are formed with the newly updated number of dimensions.

An algorithm can be stated efficient in dynamic environments if it can utilize the knowledge gained in the previous environment for responding to the changed one. For this purpose, we use an external archive (Ar). It is stated earlier that, whenever a cluster is about to converge, it is deleted and its best solution is preserved in the archive. Therefore, the archive is likely to contain the positions of the optimum of the previous environment. These solutions are added to the new fresh population when an environmental change is detected. The injection of the members of the archive in the population can be done only when there is no dimensional change. Once a dimensional change occurs in the environment, the idea of using the archive will not work because the new environment has more or less dimensions than that of the previous environment. Therefore, we restrict ourselves from using the archive if this type of change is detected. In case of other change types, we use the archive as stated earlier. The resulting population follows the same steps starting from clustering as described earlier. For most of the DOPs, the new environment is somewhat related to the old environment unless a dimensional change occurs. If the change is not too severe, then this method should help us in tracking the movement of the optima and hence improve the search process in the new environment. In summary, the pseudocode of the proposed algorithm is given in Algorithm 3.

Algorithm 3: CDDE_Ar

//Step 1: Initialization
Initialize a population of NP individuals in the search region randomly
Divide them into k clusters using k-means
Set appropriate values of $k, Clust_{max}, Clust_{min}$, and TS
Set R_{conv} using (8)
$clusterno \leftarrow k$, $t \leftarrow 1$, $Archive \leftarrow [\,]$, $Mem \leftarrow [\,]$
while stopping criteria are not satisfied **do**
//Step 4: Detection of Environment Change
for $i = 1$ to $clusterno$ **do**
if $Mem = [\,]$
Break
end if
Evaluate *the best individual of ith cluster*
if $f(best_i)_t \neq Mem(i)_{t-1}$ **or** $\dim(best)_t \neq \dim(best)_{t-1}$ **do**
//$\dim(\vec{A})$ denotes the dimension of vector $\vec{A}$
Reinitialize the total population
if *dimensional change has not occurred* **do**
Inject the members of $Archive$ in the new population
end if
$Archive \leftarrow [\,]$
break
end if
end for
//Step 2: Cluster Improvement
for $i = 1$ to $clusterno$ **do**
Calculate radius R_i by (7)
if $R_i > R_{conv}$ **do**
Improve cluster i using "DE/best/1/bin" scheme.
else
Add the best member of cluster i to $Archive$
Delete cluster i
end if
end for
//Step 3: Performance Evaluation and Redistribution
if $modulo(t, TS) = 0$ **do**
$increase =$ Determine_increase$(changeno, totchange)$
$changeno \leftarrow 0, totchange \leftarrow 0$
else
if $Globalbest_t < Globalbest_{t-1}$ **do**
Calculate *percentage change* in $Globalbest_{t-1}$
$changeno \leftarrow changeno + 1$
$totchange \leftarrow totchange + percentagechange$
end if
end if
if $modulo(t, TS) = 0$ **do**
Create $(clusterno - increase)$ clusters using k-means algorithm.
$clusterno \leftarrow clusterno - increase$
end if
for $i = 1$ to $clusterno$ **do**
$Mem(i) \leftarrow f(best_i)$
end for
$t \leftarrow t + 1$
end while

IV. Experimental Study for Parameters of CDDE_Ar

A. MPB Problem

The MPB problem set [8] has been widely used for benchmarking dynamic optimizers. Within the MPB framework, the optima can change with respect to three features: location,

TABLE I
DEFAULT SETTINGS FOR THE MPB PROBLEM

Parameter	Value
Number of peaks, p	10
Search range	$[0,100]^D$
Shift length, S	1.0
Dimension, D	5
Correlation coefficient, λ	0
H	[30, 70]
W	[1, 12]
Height Severity	7.0
Width Severity	1.0
Initial Height, I	50
Change Frequency, E	5000

height, and width of the peaks. Three standardized sets of parameter settings or scenarios have been defined for better comparisons of different algorithms. Details of the parametric setups used in these three scenarios can be found at http://people.aifb.kit.edu/jbr/MovPeaks/. The environment remains static for a certain number of FEs. After that, it changes following the rules described earlier. The change frequency is denoted by E, i.e., the environment changes after every E number of FEs. The default settings for the MPB problem are given in Table I. It is seen from the table that the height and width of the peaks are shifted randomly in the ranges [30, 70] and [1, 12], respectively. The performance measure used is the *offline error*, which can be defined as

$$\mu = \frac{1}{Q}\sum_{q=1}^{Q}(h_q - f_q) \tag{10}$$

where f_q is the best solution found by the algorithm just before the qth environmental change, h_q is the optimum value in the qth environment, and Q is the total number of environmental changes. For our experiments, we have used 20 environmental changes in a single run, and 25 such runs are conducted. The results provided here are the means and standard deviation values of 25 independent runs.

B. Working Mechanism of CDDE_Ar—An Empirical Study

In this section, the working mechanism of CDDE_Ar is investigated on the MPB problem. The variations of offline error and number of clusters are shown in Fig. 1(a) and (b) respectively. For convenience of simulation, only five environmental changes are allowed in the MPB frame with the other parameters kept in their default values. For our algorithm, the parameter values used are the following: the number of initial clusters $k = 10$, number of initial individuals $NP = 80$, time span $TS = 10$, $F = 0.5$, and $Cr = 0.9$. These values are used only to demonstrate how our algorithm works. The issue of detailed tuning of the parameters is discussed in the following section. The offline error is decreasing in nature in a specific environment. However, whenever an environmental change occurs, the error jumps to a higher value as can be seen observed from Fig. 1(a). This happens because, when the algorithm detects a change, it reinitializes the population in the search region and the optimization in the changed environment starts afresh. However, the number of clusters and the total population size are not monotonic in nature, but they vary self-adaptively as it was described earlier. This can be seen from Fig. 1(b).

The number of clusters decreases when the algorithm performs well, and the change in the number of clusters occurs after a predefined number of iterations. In this period, the cluster number remains constant. Thus, we get a stairlike structure in the variation of the cluster number. The jump in the value of the cluster number is also seen with the occurrence of an environmental change. Another interesting point is to measure the diversity level of the total population during optimization process because maintaining diversity is an important aspect regarding the DOPs. The "distance-to-average point" measurement for the diversity of the population $P(t)$ at generation t, as defined in [37], is introduced as follows:

$$diversity\,(P(t)) = \frac{1}{NP \times L}\sum_{i=1}^{NP}\sqrt{\sum_{j=1}^{D}(x_{ij} - \bar{x}_j)^2} \tag{11}$$

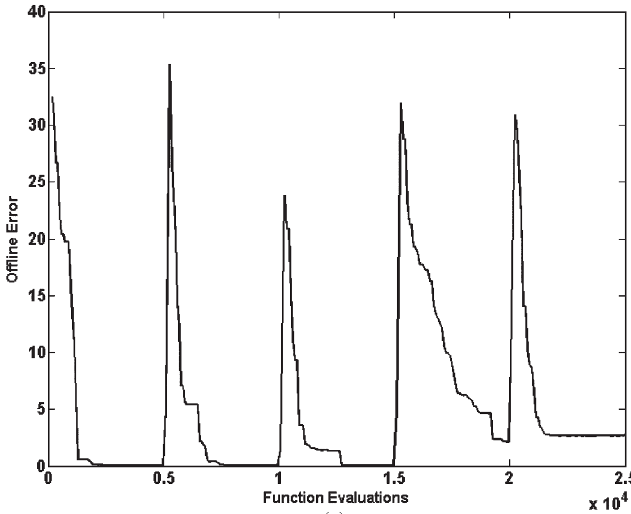
where NP is the population size, L is the length of the longest diagonal in the search space of dimension D, and $\overline{x}_j$ is the value of the jth dimension of the average vector. The variation of the diversity with FEs as defined in (11) for the population is plotted in Fig. 2. It can be seen from the figure that the diversity never becomes too small as multiple clustered population is used instead of a single population.

Now, to visualize the movement of individual members or the clusters, the positions of the population members at certain FEs are plotted in Fig. 3. CDDE_Ar is run on a 2-D MPB, and the behaviors of the members are shown in only a single environment, i.e., a static environment. For convenience of simulation, the initial cluster number is taken as 5 ($k = 5$), and $TS = 5$. The other parameters are kept the same as that of the previous simulation. The peaks in the space are shown by the filled black squares. The locations of members belonging to different clusters are shown by the plus sign of different color. The locations of the members are monitored at six different instances of the optimization process. The FEs and the corresponding cluster numbers are also indicated. The convergence of the clusters and the adaptive behavior of our algorithm in changing the cluster number and redistributing the total population can be perceived from Fig. 3.

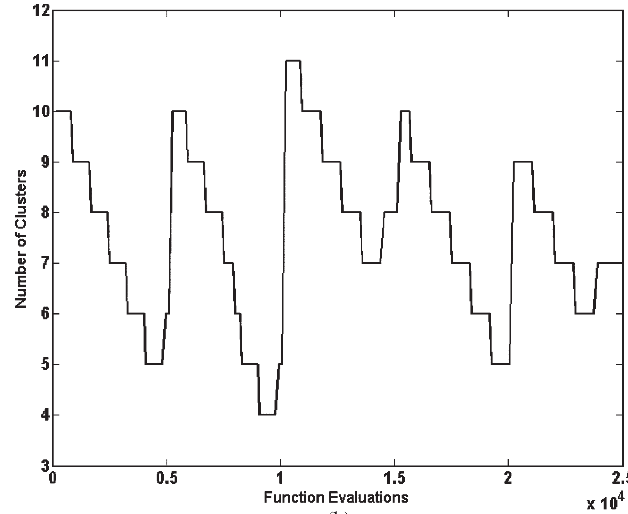
C. Parameter Settings

In this section, the parametric settings for CDDE_Ar are discussed. The best values of the parameters used in our algorithm are chosen through a series of tuning experiments. While studying the effects of the variation of one parameter, the others are fixed at their default values as described in the first paragraph of Section IV-B.

The performance of the algorithm is also given using the parameter settings other than the best one. However, most of the parameters of our algorithm are adaptive, i.e., they keep changing their values during the optimization process until an environmental change occurs. When a change is detected, they are reset to their initial values, and the process continues.



(a)



(b)

Fig. 1. Dynamic behavior of CDDE_Ar with variation of (a) offline error and (b) number of clusters with number of FEs.

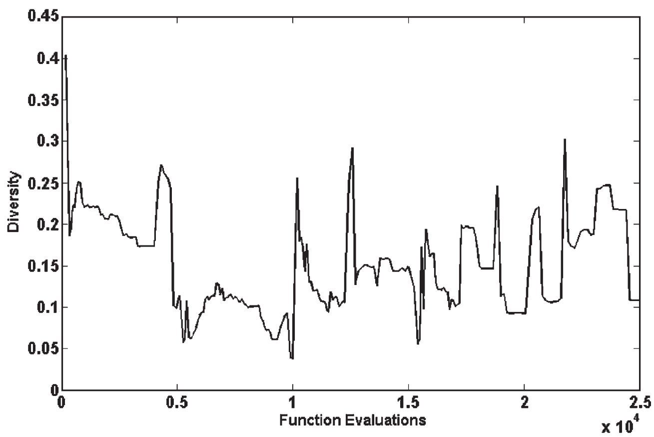


Fig. 2. Variation of the population diversity with the number of FEs.

We have undertaken extensive experimentation on MPB and GDBG benchmarks with various configurations and values of the parameters varying over wide ranges to find an optimal set of parameters that may cause the algorithm to perform considerably well on a broad class of DOPs. In this section, we provide only limited results from such experiments conducted on the MPB problems in order to save space. In all result tables, the best entries are marked in bold.

1) Initial Population Size (NP)*:* The population size affects the performance of the algorithm as too low a population size increases the chances of getting trapped in a local optimum, and on the other hand, although too high a population size can enhance the performance of the algorithm, it is also likely to increase the number of FEs consumed per generation and thereby reduce the efficiency of the algorithm. In our algorithm, the population size does not remain constant, but it is dynamic in nature. therefore, it is likely to have less impact on the performance. However, we have undertaken simulation experiments to obtain an acceptable value of the initial population size NP. For that purpose, CDDE_Ar is run on MPB, using a number of peaks $p \in \{1, 10, 100\}$, and the other parameters of MPB are kept at their default values. NP is varied from 50 to 100 in steps of 10. The mean offline error and standard deviation values of 25 independent runs are listed in Table II.

Table II indicates that the performance of the algorithm is very poor when NP is very small (∼50). The performance gets better with the increment of NP and remains almost the same after $NP = 80$. Therefore, we can take $NP = 80$ as a considerably good choice. We neglected the higher values of NP as they increase the required number of FEs per generation without giving a satisfactory improvement of the performance. The number of maximum individuals in one cluster (pop_max) can be set from here. It is only the maximum capacity of a cluster. When a cluster exceeds this limit, the best pop_max members are selected, and others are discarded as described earlier. If we take $NP = 80$, then pop_max can be reasonably set to 50. Our experiments indicate that this value provides reasonably good results over both the MPB and GDBG benchmark problems.

2) Initial Cluster Numbers (k)*:* Initially, the total population is partitioned into k clusters. Afterward, the cluster number is changed by the algorithm until an environmental change is detected. Therefore, as the cluster number is a self-adaptive parameter, its tuning may not have much effect on the performance. However, if k is taken too low, there are only a few overlapping clusters in the search space. This may significantly reduce the diversity of the evolving population. On the other hand, if k is taken too high, the number of individual in each cluster becomes very small, which may affect the convergence characteristics of the individual clusters. Hence, we must have a balance between these two tendencies. The effect of varying the initial number of clusters is empirically investigated, and some results are shown in Table III for choices of k as 5, 10, and 15. The MPB configurations selected for these experiments are $D \in \{5, 10\}$, $p \in \{1, 10, 100\}$, and the others are kept at their default values.

From Table III, we see that the algorithm performs the best when k is taken as 10. It is also observed that taking the cluster number too small (i.e., 5) makes the algorithm inefficient when the number of peaks rises. Similarly, a high initial cluster number (i.e., 15) affects the performance for problems with a

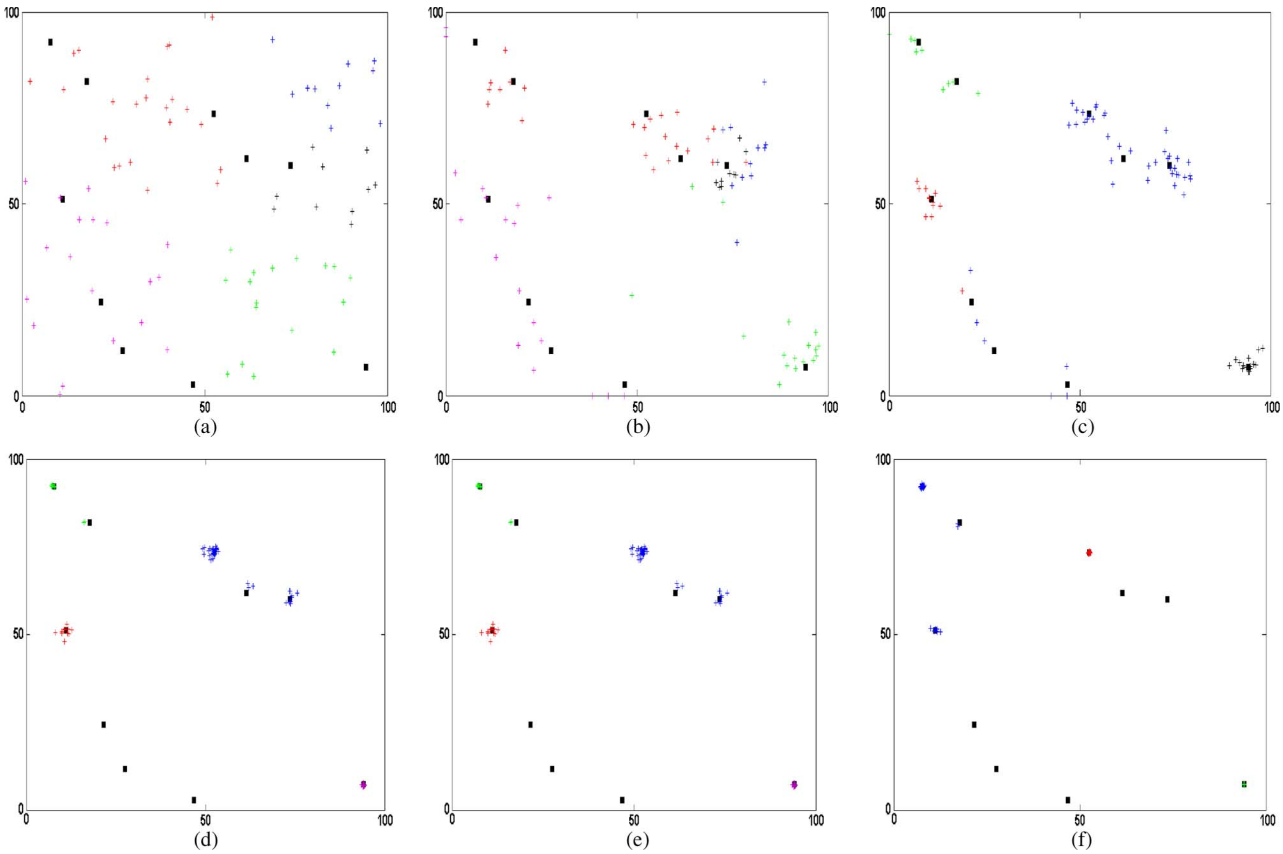


Fig. 3. (a)–(f) Positions of population members in a single environment on a 2-D landscape. (a) Clusters: 5; FEs: 0. (b) Clusters: 5; FEs: 324. (c) Clusters: 4; FEs: 567. (d) Clusters: 4; FEs: 810. (e) Clusters: 4; FEs: 1053. (f) Clusters: 3; FEs: 1296.

TABLE II
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF CDDE_Ar FOR DIFFERENT POPULATION SIZES

Population Size, *NP*	$p = 1$	$p = 10$	$p = 100$
50	2.41±0.13	2.92±0.09	3.30±0.18
60	1.71±0.06	2.32±0.03	2.63±0.12
70	1.58±0.17	1.52±0.21	2.45±0.19
80	1.23±0.04	1.51±0.12	2.20±0.05
90	1.25±0.12	1.51±0.07	2.08±0.15
100	**1.19±0.09**	**1.50±0.10**	**1.99±0.03**

TABLE III
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF CDDE_Ar FOR DIFFERENT INITIAL CLUSTER NUMBERS

k	5	10	15
$D,\ p$			
5, 1	**1.03±0.02**	1.25±0.15	1.31±0.05
5, 10	1.52±0.12	**1.31±0.06**	1.36±0.10
5, 100	2.99±0.09	2.08±0.10	**1.79±0.19**
10, 1	2.79±0.06	**2.05±0.09**	3.10±0.03
10, 10	2.86±0.12	**2.08±0.02**	2.19±0.07
10, 100	3.77±0.09	**2.95±0.13**	2.79±0.13

lower number of peaks. $k = 10$ maintains a balance between these two. The diversity plot of CDDE_Ar is shown in Fig. 4 for different k values. Here, for convenience, only one test instance of $(D, p) = (5, 10)$ is selected, and only five environmental changes are shown.

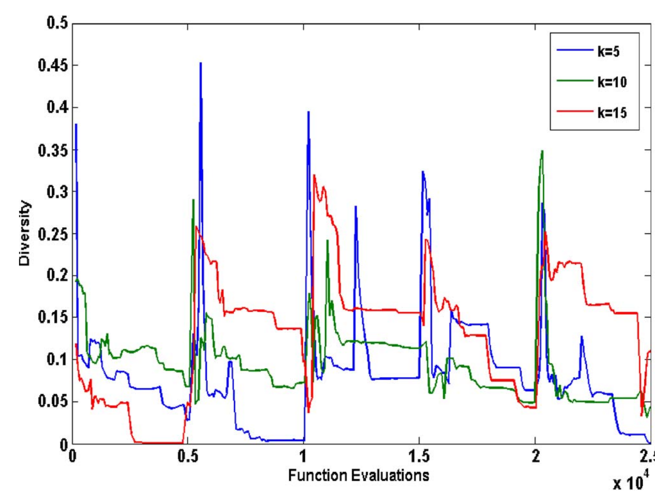


Fig. 4. Diversity plot for different values of k.

From Fig. 4, it is observed that, when $k = 5$, the diversity often becomes very small, and it also exhibits rapid variation that is not intended. When $k = 15$, although the population possesses a good diversity, the performance of CDDE_Ar in terms of offline error is worse than the case with $k = 10$. For $k = 10$, the diversity is good, and it also does not exhibit an excessively rapid variation. In addition to that, it performs the best as can be seen from Table III. Therefore, the optimum value of the initial cluster number k is fixed to 10. The parameters

TABLE IV
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF CDDE_Ar FOR DIFFERENT TIME SPANS

TS	5	10	15	20
D, p				
Change Frequency $E = 2000$				
5, 1	**3.01±0.10**	3.21±0.16	3.25±0.08	3.41±0.32
5, 10	**3.20±0.14**	3.32±0.06	3.43±0.27	3.82±0.08
5, 100	**3.34±0.05**	3.79±0.15	4.02±0.13	4.32±0.16
Change Frequency $E = 5000$				
5, 1	1.21±0.05	**1.15±0.12**	1.12±0.09	1.19±0.09
5, 10	1.32±0.12	**1.27±0.05**	1.38±0.12	1.32±0.12
5, 100	2.43±0.16	**1.71±0.02**	1.99±0.06	2.04±0.06
Change Frequency $E = 10000$				
5, 1	1.13±0.06	0.78±0.03	0.87±0.12	**0.73±0.03**
5, 10	1.27±0.12	**0.85±0.07**	0.91±0.05	0.88±0.02
5, 100	1.77±0.09	1.23±0.10	1.10±0.07	**0.95±0.07**

$Clust_{max}$ and $Clust_{min}$ bound the value of the number of clusters that change in an adaptive fashion as the algorithm progresses. $Clust_{max}$ should be quite high for the exploration of different regions but not too high such that the population size of a cluster becomes very low. $Clust_{max}$ can be intuitively related to k as follows: $Clust_{max} = 2k$. Taking $k = 10$ makes $Clust_{max} = 20$. $Clust_{min}$ should be low to allow the algorithm to reduce the number of clusters significantly. Thus, it is set to 1.

3) Time Span (TS)*:* The performance evaluation test is conducted after every TS iterations. Based on the result of the test, the adaptive strategy of the algorithm for the next TS is determined. Thus, it is very crucial to set the frequency of the performance evaluation test. If the test is conducted too frequently, i.e., TS is very low, it may happen that the members are redistributed too rapidly and the chance of getting trapped in a local optimum rises. On the other hand, if TS is very high, the clusters may not get enough scope of sharing the information among them. Therefore, proper mixing of the clusters should be done at an optimum rate, which we have empirically determined by applying the algorithm to MPB problems of different configurations. Results are shown here by varying TS from 5 to 20 in steps of 5, and the configuration of MPB used is given by $p \in \{1, 10, 100\}$, $E \in \{2000, 5000, 10000\}$; the others are set at their default values. The corresponding results are provided in Table IV. From Table IV, it can be seen that, when the change frequency (E) is low (i.e., 2000), the algorithm performs the best with $TS = 5$. However, as E rises, the performance with this configuration also degrades in comparison with the other configurations. When E is 10 000, $TS = 20$ gives the best result, but it does not provide good results in lower change frequencies. Hence, it is clear that TS should be low when E is low and high when E is high. However, we have to choose an acceptable value of TS that will provide satisfactory results over a wide range of functions. Referring to Table IV and also from our detailed empirical study, we find that $TS = 10$ can qualify as such a value.

4) Effect of Using the External Archive (Ar)*:* The external archive is used in CDDE_Ar to utilize the knowledge of the suspected optimum positions of one environment in the changed environment. The peaks are basically shifted randomly in the new environment from their previous positions, and the amount of movement is controlled by the parameter *shift severity (S).* The height and width of the peaks are also changed, making the job of tracking the global optimum more challenging. Now, if we have some knowledge about the peak positions of the earlier search space, searching in the new environment becomes somewhat easier when the shift of the optima is not too severe. However, if S is too high, the use of Ar is not expected to be of great help. We investigated the performance of CDDE_Ar on MPB problem using Ar and without using Ar. The configuration of MPB is set as $S \in \{1.0, 2.0, 5.0\}$, and the other parameters are set at their default values. The result of the experiment is given in Table V.

TABLE V
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF CDDE_AR USING Ar AND WITHOUT USING Ar

	$S = 1.0$	$S = 2.0$	$S = 5.0$
Using Ar	**1.27±0.05**	**1.56±0.08**	**2.63±0.12**
Without using Ar	1.45±0.11	1.89±0.07	2.75±0.06

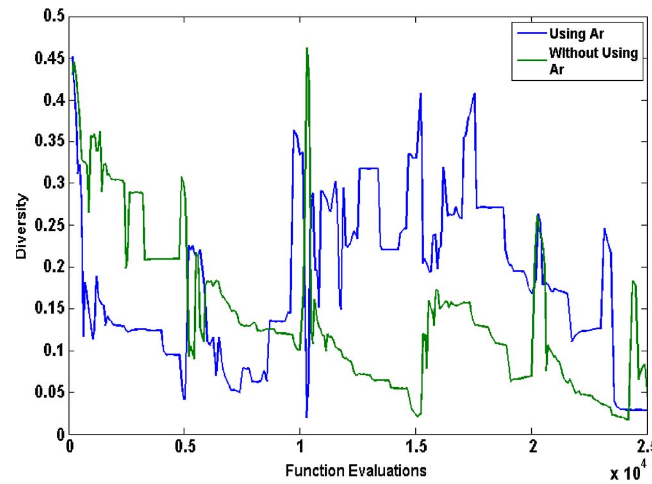


Fig. 5. Diversity plot using and without using Ar.

The experimental results show (see Table V) that the usage of Ar improves the performance of the algorithm when the shift severity is small ($S = 1.0$ or 2.0) and it cannot improve the performance of the algorithm to a great extent when the shift severity is high ($S = 5.0$). However, the usage of Ar maintains a high diversity in the population compared to that without using Ar. This is seen experimentally, and the corresponding diversity plot is provided in Fig. 5.

5) Effect of Varying the Cluster Number: CDDE_Ar varies the number of clusters throughout the optimization process based on the results of the performance evaluation test performed periodically. In this section, we investigate the effect of this scheme, i.e., how the performance of the algorithm is affected if the cluster number is not varied. The performance of CDDE_Ar on the MPB problem is noted with varying as well as fixed number of clusters. The parameters used for CDDE_Ar are the optimal values that we have set through extensive experiments. The configurations of MPB are given

TABLE VI
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF CDDE_Ar USING FIXED CLUSTER NUMBER AND USING VARIABLE CLUSTER NUMBER

	(*D*, *p*) = (5,1)	(*D*, *p*) = (5,10)	(*D*, *p*) = (5,100)
Using Fixed cluster number	1.15±0.08	1.40±0.10	2.51±0.12
Using variable cluster number	**1.15±0.12**	**1.27±0.05**	**1.71±0.02**

TABLE VII
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF CDDE_Ar USING DIFFERENT SCHEMES OF DE

Scheme	DE/best/1	DE/rand/1	DE/current-to-best/1	DE/best/2	DE/rand/2
D, *p*					
5, 1	**1.15±0.12**	1.22±0.08	1.25±0.07	1.16±0.07	1.21±0.07
5, 10	**1.27±0.05**	1.39±0.08	1.40±0.08	1.35±0.06	1.42±0.08
5, 100	**1.71±0.02**	2.05±0.08	1.98±0.09	1.91±0.08	2.07±0.09

as $p \in \{1, 10, 100\}$, and the other parameters are kept at their default values. The results of this experiment are given in Table VI.

6) Effect of Using DE/Best/1/Bin Scheme: In CDDE_Ar, the individuals of a cluster are evolved according to the DE/best/1/bin scheme [see (3b)]. The effects of choosing other DE schemes are investigated in this section. For this purpose, we applied the algorithm with different schemes of DE on MPB problems. The MPB configuration is taken as $p \in \{1, 10, 100\}$, and the other parameters are fixed at their default values. Table VII contains the results of this experiment.

Table VII shows that DE/best/1/bin is undoubtedly the best scheme for solving this problem.

V. EXPERIMENTAL STUDY ON COMPARING CDDE_Ar WITH OTHER ALGORITHMS

In this section, we give the comparative experimental results of our proposed algorithm with six other state-of-the-art dynamic evolutionary optimizers. The results on the MPB problem set [8] and the GDBG system [9], [10] are provided and discussed, respectively. A nonparametric statistical test called Wilcoxon's rank sum test for independent samples [57], [58] is conducted at the 5% significance level in order to judge whether the results obtained with the proposed algorithm differ from the final results of the rest of the algorithms in a statistically significant way. The statistical test results of comparing CDDE_Ar with the contender algorithms in terms of the offline error are indicated within parentheses throughout all the result tables, where the rank sum test results are marked as "+," "−," or "$\sim$" when CDDE_Ar is statistically significantly better than, worse than, or statistically equivalent to the corresponding algorithm, respectively.

A. Experimental Study on the MPB Problems

The performance of CDDE_Ar is compared with six other algorithms including DASA [50], jDE [44], DynDE [43], dopt-aiNET [51], clustering particle swarm optimizer (CPSO) [39], and PSO-CP [41]. For each algorithm, the best parametric setup is employed as reported in the corresponding papers. The parameters of CDDE_Ar are fixed as discussed in Section IV (i.e., $NP = 80$, $pop\_max = 50$, $k = 10$, $Clust_{max} = 2k = 20$, $Clust_{min} = 1$, and $TS = 10$). For the DE/best/1/bin scheme, the standard values of the parameter scaling factor (F) and crossover rate (Cr) are $F = 0.5$ and $Cr = 0.9$. The comparison is undertaken on various configurations of the MPB. First, the dimensionality (D) and number of peaks (p) are varied. The values used are given by $D \in \{5, 10\}$, $p \in \{1, 10, 100\}$, and the other parameters are set to their default values. The comparison results are shown in Table VIII. Table VIII indicates that CDDE_Ar outperforms six other dynamic optimizers over all cases except for the case with $D = 5$ and $p = 100$, where it managed to remain the third best algorithm, beaten by DASA and dopt-aiNET. Next, the shift severity (S) is varied, keeping other configurations of MPB at default values. Table IX shows the comparative results for the shift severity S varying from 0 to 6 in steps of 1. From this table, we observe that CDDE_Ar remains the best for all cases except for $S = 3$, where it managed to rank second while PSO-CP ranks first. The effect of varying the change frequency (E) on the comparative performances is indicated in Table X, where E is taken from {3000, 5000, 10 000}. CDDE_Ar performs the best as compared to all the six dynamic evolutionary optimizers for all the three values of E. The correlation coefficient (λ) is generally set to 0 in MPB. However, experiments have also been conducted to study the effect of varying λ in {0, 0.1, 0.3, 0.5, 0.7, 1} on the comparative performances. Corresponding results are shown in Table XI, where we find CDDE_Ar to outperform all the contestants for all the listed values of λ.

B. Experimental Study on the GDBG System

1) The GDBG System: CEC 2009 benchmark problems for dynamic optimization were generated by using the GDBG system proposed in [9] and [10], which constructs dynamic environments for the location, height, and width of peaks. The authors of [10] introduced a rotation method instead of shifting the positions of peaks as done in the MPB problems. The GDBG system poses greater challenges for optimization than the MPB problems due to the rotation method, larger number of local optima, and higher dimensionalities. There are seven change types for each of the test functions in the GDBG benchmark suite, and these are small step change, large step change, random change, chaotic change, recurrent change, recurrent change with noise, and dimensional change. Details of the change types can be found in [10]. The test functions in real space instance are as follows: F1—rotation peak function, F2—composition of sphere functions, F3—composition of Rastrigin's functions, F4—composition of Griewank's functions, F5—composition of Ackley's functions, and F6—hybrid composition function. Only F1 is a maximization problem, and the others are minimization problems. In F1, there are two tests, one using 10 peaks and another using 50 peaks.

2) Comparisons of Results: There are a total of 49 benchmark instances in the GDBG system created from six test functions and the different change types T1–T7. The measurements done for comparing the performance are offline error and the

TABLE VIII
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF ALGORITHMS ON MPB WITH $E = 5000$, $S = 1.0$, DIFFERENT DIMENSIONALITY, AND DIFFERENT NUMBER OF PEAKS. THE WILCOXON'S RANK SUM TEST RESULTS OF COMPARING CDDE_Ar WITH CONTENDER ALGORITHMS ARE SHOWN IN BRACKETS

D, p	CDDE_Ar	DASA	jDE	DynDE	Dopt-aiNET	CPSO	PSO-CP
5, 1	**1.15±0.12**	4.89±0.07(+)	2.02±0.07(+)	5.05±0.08(+)	4.88±0.07(+)	1.54±0.07(+)	3.45±0.06(+)
5, 10	**1.27±0.05**	1.70±0.06(+)	1.68±0.05(+)	1.76±0.06(+)	1.75±0.05(+)	1.80±0.06(+)	1.40±0.06(+)
5, 100	1.71±0.02	**1.69±0.05(~)**	2.65±0.08(+)	2.55±0.08(+)	1.76±0.06(~)	1.92±0.06(+)	2.07±0.07(+)
10, 1	**1.98±0.07**	5.12±0.07(+)	3.23±0.07(+)	5.32±0.08(+)	5.02±0.07(+)	2.07±0.06(+)	4.00±0.07(+)
10, 10	**2.14±0.05**	2.31±0.07(+)	2.50±0.08(+)	3.19±0.09(+)	2.35±0.06(+)	2.39±0.06(+)	2.17±0.08(+)
10, 100	**2.31±0.08**	2.47±0.08(+)	3.29±0.09(+)	3.38±0.09(+)	2.59±0.07(+)	2.51±0.07(+)	2.38±0.08(+)

TABLE IX
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF ALGORITHMS ON MPB WITH $E = 5000$, $D = 5$, $P = 10$, AND DIFFERENT SHIFT SEVERITIES (S). THE WILCOXON'S RANK SUM TEST RESULTS OF COMPARING CDDE_Ar WITH CONTENDER ALGORITHMS ARE SHOWN IN BRACKETS

S	CDDE_Ar	DASA	jDE	DynDE	Dopt-aiNET	CPSO	PSO-CP
0	**0.81±0.07**	0.98±0.07(+)	0.96±0.07(+)	1.15±0.06(+)	1.00±0.06(+)	0.97±0.06(+)	0.91±0.07(+)
1	**1.27±0.05**	1.70±0.06(+)	1.68±0.05(+)	1.76±0.06(+)	1.75±0.05(+)	1.80±0.06(+)	1.40±0.06(+)
2	**1.56±0.08**	2.17±0.07(+)	2.12±0.06(+)	2.27±0.06(+)	2.15±0.06(+)	2.23±0.06(+)	2.01±0.06(+)
3	**2.12±0.07**	2.75±0.07(+)	2.55±0.06(+)	2.70±0.06(+)	2.76±0.08(+)	2.83±0.08(+)	2.25±0.06(+)
4	2.67±0.12	2.96±0.07(+)	3.10±0.07(+)	3.29±0.07(+)	2.85±0.08(+)	3.09±0.09(+)	**2.66±0.11(~)**
5	**2.63±0.12**	3.00±0.08(+)	3.19±0.08(+)	4.32±0.08(+)	2.90±0.08(+)	3.17±0.09(+)	3.23±0.13(+)
6	**3.21±0.09**	3.37±0.08(+)	3.78±0.09(+)	4.55±0.09(+)	3.37±0.08(+)	3.45±0.09(+)	3.97±0.14(+)

TABLE X
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF ALGORITHMS ON MPB WITH $D = 5$, $P = 10$, $S = 1.0$, AND DIFFERENT CHANGE FREQUENCIES (E). THE WILCOXON'S RANK SUM TEST RESULTS OF COMPARING CDDE_Ar WITH CONTENDER ALGORITHMS ARE SHOWN IN BRACKETS

E	CDDE_Ar	DASA	jDE	DynDE	Dopt-aiNET	CPSO	PSO-CP
3000	**2.13±0.06**	2.42±0.13(+)	2.35±0.06(+)	2.39±0.17(+)	2.40±0.13(+)	2.37±0.13(+)	2.23±0.08(+)
5000	**1.27±0.05**	1.70±0.06(+)	1.68±0.05(+)	1.76±0.06(+)	1.75±0.05(+)	1.80±0.06(+)	1.40±0.06(+)
10000	**0.85±0.07**	0.97±0.06(+)	0.96±0.04(+)	1.33±0.04(+)	0.98±0.05(+)	0.99±0.05(+)	0.98±0.03(+)

TABLE XI
OFFLINE ERROR AND STANDARD DEVIATION VALUES OF ALGORITHMS ON MPB WITH $D = 5$, $P = 10$, $S = 1.0$, $E = 5000$, AND DIFFERENT CORRELATION COEFFICIENTS (λ). THE WILCOXON'S RANK SUM TEST RESULTS OF COMPARING CDDE_Ar WITH CONTENDER ALGORITHMS ARE SHOWN IN BRACKETS

λ	CDDE_Ar	DASA	jDE	DynDE	Dopt-aiNET	CPSO	PSO-CP
0.0	**1.27±0.05**	1.70±0.06(+)	1.68±0.05(+)	1.76±0.06(+)	1.75±0.05(+)	1.80±0.06(+)	1.40±0.06(+)
0.1	**1.31±0.04**	1.72±0.07(+)	1.68±0.05(+)	1.76±0.07(+)	1.74±0.06(+)	1.75±0.07(+)	1.42±0.05(+)
0.3	**1.33±0.08**	1.74±0.07(+)	1.75±0.05(+)	1.89±0.07(+)	1.76±0.06(+)	1.78±0.07(+)	1.45±0.05(+)
0.5	**1.36±0.07**	1.78±0.08(+)	1.78±0.05(+)	1.93±0.08(+)	1.77±0.06(+)	1.80±0.07(+)	1.48±0.06(+)
0.7	**1.38±0.06**	1.83±0.08(+)	1.85±0.06(+)	2.09±0.08(+)	1.84±0.07(+)	1.82±0.07(+)	1.50±0.04(+)
1.0	**1.39±0.07**	1.86±0.09(+)	1.87±0.06(+)	2.17±0.08(+)	1.86±0.07(+)	1.83±0.07(+)	1.51±0.06(+)

standard deviation values for each test instance, as described in [9] and [10]

$$e_{off} = \frac{1}{R \times K} \sum_{r=1}^{R} \sum_{k=1}^{K} e_{r,k}^{last}, \quad (12)$$

$$STD = \sqrt{\frac{\sum_{r=1}^{R} \sum_{k=1}^{K} \left(e_{r,k}^{last} - e_{off} \right)}{R \times K - 1}} \quad (13)$$

where R and K denote the total number of runs and total number of environmental changes for each run, respectively, and $e_{r,k}^{last} = |f(\vec{X}_{best}(r,k)) - f(\vec{X}^*(r,k))|$, where $\vec{X}^*(r,k)$ and $\vec{X}_{best}(r,k)$ are the global optimum position and the position of the best individual of the last generation of the kth environment during the rth run, respectively.

The overall performance of an algorithm on all 49 test cases is calculated using the method proposed in [10]. Each test case i is assigned a weight w_i, and the sum of the weights of all the test cases is 1.0; see [10] for details. The mark obtained by an algorithm on test case $i \in \{1, \ldots, 49\}$ is calculated by

$$mark_i = \frac{w_i}{R \times K} \sum_{r=1}^{R} \sum_{k=1}^{K} \left(\frac{r_{rk}^{last}}{1 + \frac{1}{S} \cdot \sum_{s=1}^{S} (1 - r_{rk}^{s})} \right)$$

where r_{rk}^{last} is the relative ratio of the best individual's fitness of the last generation to the global optimum of the kth environment. r_{rk}^{s} is the relative ratio of the best individual's fitness to the global optimum at the sth sampling during the kth environment, and S is the total number of samples for each environment. The relative ratio r_{rk}^{s} is defined by

$$r_{rk}^{s} = \begin{cases} \frac{f\left(\vec{X}_{best}(r,k,s)\right)}{f\left(\vec{X}^*(r,k)\right)}, & \text{for } f = F1, \\ \frac{f\left(\vec{X}^*(r,k)\right)}{f\left(\vec{X}_{best}(r,k,s)\right)}, & \text{for } f \in F2, F3, F4, F5, F6 \end{cases}$$

TABLE XII
OFFLINE ERROR AND STANDARD DEVIATION VALUES ACHIEVED BY CDDE_Ar AND OTHER ALGORITHMS ON THE GDBG SYSTEM AND DIFFERENT CHANGE FREQUENCIES (E). THE WILCOXON'S RANK SUM TEST RESULTS OF COMPARING CDDE_Ar WITH CONTENDER ALGORITHMS ARE SHOWN IN BRACKETS

Test Functions	Algorithm	Error	T1	T2	T3	T4	T5	T6	T7
F1 (Number of peaks = 10)	CDDE_Ar	Average	**0.00001**	**1.2623**	12.5468	**0.0007**	**0.7982**	**0.9729**	4.3525
		STD	**0.00003**	**5.8901**	29.7423	**0.0032**	**2.0001**	**3.2895**	7.0213
	DASA	Average	0.1864(+)	4.2017(+)	6.4126(+)	0.5013(+)	1.2173(+)	2.8562(+)	4.9502(+)
		STD	1.3253	7.2814	9.3362	1.7432	3.0941	8.2064	8.3348
	jDE	Average	0.0342(+)	3.6019(+)	3.1058(+)	0.0214(+)	2.3256(+)	1.2411(+)	3.4162(–)
		STD	0.4253	7.9241	8.2617	0.6282	5.3791	5.5375	7.7258
	DynDE	Average	0.0732(+)	2.5567(+)	5.4245(–)	0.1263(+)	1.5651(+)	1.3115(+)	4.1137(~)
		STD	2.9567	8.4313	9.2485	0.9426	4.6461	6.2511	8.5249
	dopt-aiNET	Average	0.1521(+)	4.5714(+)	5.0291(–)	6.3438(+)	5.8372(+)	10.2713(+)	4.1298 (~)
		STD	2.1274	8.4783	7.9298	5.2098	3.9483	13.6539	6.1117
	CPSO	Average	0.0351(+)	2.7185(+)	**4.1315(–)**	0.0944(+)	1.8698(+)	1.1569(+)	4.5401(+)
		STD	0.4262	6.5230	**8.9947**	0.7855	4.4910	4.8054	9.1194
	PSO-CP	Average	0.0419(+)	2.7014(+)	4.6873(+)	0.0521(+)	1.5861(+)	1.5281(+)	**3.9369(–)**
		STD	0.5214	7.1248	8.9752	0.8150	4.4532	5.9279	**8.4244**
F1 (Number of peaks = 50)	CDDE_Ar	Average	**0.0011**	**1.5414**	4.3130	**0.0003**	**0.0062**	**0.0372**	4.7207
		STD	**0.0293**	**6.0512**	7.1271	**0.0092**	**0.2701**	**2.3135**	5.8231
	DASA	Average	0.4367(+)	5.0284(+)	8.7684(+)	0.5392(+)	0.9816(+)	2.1646(+)	7.4493(+)
		STD	1.8573	8.5603	10.6753	1.6849	3.6872	4.8590	8.9750
	jDE	Average	0.1802(+)	4.0861(+)	**4.2909(~)**	0.0877(+)	0.9483(+)	1.7652(+)	**4.3691(~)**
		STD	0.4564	6.4546	**6.4538**	0.2461	1.7655	5.8252	**6.9321**
	DynDE	Average	0.3286(+)	4.6547(+)	6.7834(+)	0.1412(+)	1.0162(+)	0.9859(+)	6.2513(+)
		STD	1.5224	6.3453	8.9274	0.5914	2.6489	4.8631	9.06517
	dopt-aiNET	Average	0.4092(+)	4.8047(+)	5.7352(+)	3.0291(+)	2.6458(+)	5.9580(+)	4.9826(+)
		STD	0.7827	7.4663	8.4793	6.8274	5.1867	10.7879	8.5647
	CPSO	Average	0.2624(+)	3.2792(+)	6.3198(+)	0.1255(+)	0.8481(+)	1.4821(+)	6.6467(+)
		STD	0.9362	5.3034	7.4420	0.3859	1.779	4.3932	7.9411
	PSO-CP	Average	0.9456(+)	3.4686(+)	3.9523(+)	0.1077(+)	1.1282(+)	0.9699(+)	5.4468(+)
		STD	0.4251	5.2124	6.1689	0.4279	2.5913	3.7845	8.8542
F2	CDDE_Ar	Average	**0.0292**	**6.5825**	**4.5629**	**0.1042**	**16.3825**	**0.8341**	**2.1231**
		STD	**0.2590**	**3.2133**	**3.4932**	**0.2009**	**43.6944**	**2.9223**	**6.0309**
	DASA	Average	2.1464(+)	24.3479(+)	18.8654(+)	1.3645(+)	46.8748(+)	2.7567(+)	3.4738(+)
		STD	7.6456	88.4748	67.0934	4.7286	110.6753	4.9543	7.2178
	jDE	Average	0.9283(+)	44.0935(+)	52.5645(+)	0.7362(+)	62.8587(+)	3.9684(+)	13.6353(+)
		STD	2.7536	112.6937	120.6863	3.0653	135.7582	14.6543	44.2348
	DynDE	Average	2.4627(+)	26.0179(+)	19.9214(+)	1.7842(+)	20.7842(+)	2.1845(+)	2.4235(+)
		STD	5.0315	48.2532	45.7054	2.2248	64.5341	3.9643	7.1031
	dopt-aiNET	Average	0.0947(+)	8.1209(+)	18.7875(+)	1.3416(+)	102.0374(+)	5.9476(+)	4.1164(+)
		STD	0.0342	14.3832	68.3599	3.5653	131.7843	14.5672	8.6944
	CPSO	Average	1.2475(+)	10.1055(+)	10.2725(+)	0.5664(+)	25.1424(+)	1.9871(+)	3.6510(+)
		STD	4.1780	35.0601	33.4527	2.1371	64.2500	5.2175	6.9274
	PSO-CP	Average	0.9451(+)	9.1431(+)	11.5219(+)	4.4855(+)	19.5651(+)	1.8689(+)	3.9687(+)
		STD	2.6621	10.2519	43.6323	2.1625	62.7372	2.8846	7.0023
F3	CDDE_Ar	Average	**10.6014**	**497.6317**	**572.9469**	**60.6250**	594.3210	**233.0049**	**123.8847**
		STD	**32.3001**	**204.2876**	**152.0073**	**179.0932**	295.0094	**407.5190**	**256.6001**
	DASA	Average	15.3453(+)	821.8589(+)	691.3659(+)	437.4538(+)	699.3421(+)	632.8273(+)	432.7582(+)
		STD	65.9758	197.3436	312.7537	469.4938	326.0472	453.8756	393.1894
	jDE	Average	11.3673(+)	562.6455(+)	586.8382(+)	66.1216(+)	**476.3928 (–)**	262.6451(+)	154.7509(+)
		STD	61.8438	326.8581	375.0942	210.7584	**326.6572**	392.6487	316.6472
	DynDE	Average	21.2512(+)	792.4575(+)	735.6144(+)	541.7015(+)	749.2651(+)	679.5504(+)	465.3243(+)
		STD	73.6549	255.6163	342.7753	419.8116	280.9181	438.2467	390.3450
	dopt-aiNET	Average	813.7382(+)	1056.4367(+)	1101.7578(+)	1037.2378(+)	1020.4892(+)	1188.3415(+)	1068.4257(+)
		STD	77.6462	73.7673	62.8573	280.3526	61.5342	284.2867	121.4512
	CPSO	Average	137.5274(+)	855.1386(+)	765.9663(+)	430.6204(+)	859.7036(+)	753.0394(+)	653.7032(+)
		STD	221.6201	161.0024	235.8834	432.2391	121.5581	361.7855	334.4892
	PSO-CP	Average	15.7312(+)	774.1627(+)	634.5219(+)	319.5279(+)	736.5267(+)	501.1201(+)	401.9820(+)
		STD	73.6549	254.1579	341.2314	416.6123	281.3412	439.6415	388.3141
F4	CDDE_Ar	Average	**1.0091**	**2.1902**	**1.9041**	0.8792	56.0598	**0.6557**	9.7802
		STD	**6.6079**	**5.0007**	**5.8999**	2.0333	12.6544	**10.4409**	13.6432
	DASA	Average	5.4756(+)	66.0356(+)	52.9564(+)	3.6357(+)	120.9565(+)	2.3467(+)	27.3728(+)
		STD	28.7438	146.4367	147.9543	6.6765	189.4463	8.5356	78.7685
	jDE	Average	1.5653(+)	49.7642(+)	52.1324(+)	1.6735(+)	66.8794(+)	2.5637(+)	12.0423(+)
		STD	4.8765	133.2347	156.8447	5.0263	148.8274	5.8505	44.3746
	DynDE	Average	1.8616(+)	39.5923(+)	23.4921(+)	3.9691(+)	144.6713(+)	1.5624(+)	6.5213(–)
		STD	5.7531	98.6312	94.5314	3.1723	121.7162	6.2149	26.5951
	dopt-aiNET	Average	1.7536(+)	125.6746(+)	95.2373(+)	4.9786(+)	306.3421(+)	12.3423(+)	51.5463(+)
		STD	4.3785	221.5737	201.4632	10.3245	215.3754	59.4725	142.7464

TABLE XII

(Continued.) OFFLINE ERROR AND STANDARD DEVIATION VALUES ACHIEVED BY CDDE_Ar AND OTHER ALGORITHMS ON THE GDBG SYSTEM AND DIFFERENT CHANGE FREQUENCIES (E). THE WILCOXON'S RANK SUM TEST RESULTS OF COMPARING CDDE_Ar WITH CONTENDER ALGORITHMS ARE SHOWN IN BRACKETS

	CPSO	Average	2.6771(+)	37.1512(+)	36.6711(+)	**0.7926(–)**	67.1702(+)	4.8814(+)	12.7924(+)
		STD	7.0552	99.4352	97.1805	**2.775**	130.3059	15.3965	19.2105
	PSO-CP	Average	1.9743(+)	28.5925(+)	21.4561(+)	0.9958(~)	**44.4145(–)**	1.7541(+)	**5.4312(–)**
		STD	5.6079	98.6229	92.5379	2.9516	**119.9201**	6.1736	**7.6572**
F5	CDDE_Ar	Average	**0.0317**	0.9907	**0.2467**	**0.0297**	**0.1822**	**0.1791**	**0.2237**
		STD	**3.0409**	28.6789	**1.9972**	**0.4501**	**2.7691**	**0.3679**	**0.7691**
	DASA	Average	0.9745(+)	2.1145(+)	0.9378(+)	0.4097(+)	2.4324(+)	0.4609(+)	1.1217(+)
		STD	3.4245	4.7462	3.2694	1.7124	5.6535	1.6983	3.8326
	jDE	Average	0.1604(+)	**0.3462(–)**	0.3612(+)	0.1076(+)	0.4336(+)	0.3754(+)	0.4521(+)
		STD	1.2415	**1.3874**	2.0756	0.9183	1.5847	0.9867	2.6935
	DynDE	Average	2.9929(+)	2.9481(+)	2.9125(+)	1.3796(+)	8.4378(+)	2.3049(+)	1.5214(+)
		STD	6.8831	4.7179	5.3886	2.4199	12.1132	3.6182	0.7135
	dopt-aiNET	Average	42.7536(+)	35.9463(+)	33.6468(+)	123.4268(+)	945.6357(+)	487.3526(+)	224.7258(+)
		STD	215.8452	121.5343	131.7683	308.4256	580.3732	574.7462	434.6267
	CPSO	Average	1.8559(+)	2.8791(+)	3.403(+)	1.0954(+)	7.9869(+)	4.0535(+)	6.5278(+)
		STD	5.1812	6.7875	6.448	4.8651	13.8170	8.3719	22.8129
	PSO-CP	Average	1.9730(+)	2.8261(+)	2.8879(+)	0.9068(+)	7.3517(+)	2.2074(+)	3.4578(+)
		STD	6.2714	4.4479	5.2534	2.3017	11.7631	3.5149	11.7356
F6	CDDE_Ar	Average	**4.8190**	27.5017	**6.9145**	**3.4570**	**0.9897**	**3.0399**	14.9371
		STD	**11.4371**	19.2033	**17.5421**	**17.9109**	**23.9801**	**1.3305**	50.2756
	DASA	Average	6.3265(+)	26.2314(+)	18.6378(+)	7.4624(+)	36.7544(+)	13.6542(+)	15.8346(+)
		STD	16.3863	120.6402	69.3436	23.7335	121.5635	54.6783	37.3436
	jDE	Average	6.3425(+)	**10.5363(–)**	10.4415(+)	6.2435(+)	15.0372(+)	7.6548(+)	17.3439(+)
		STD	10.7454	**13.8342**	25.7565	13.3425	44.3274	10.7564	14.4772
	DynDE	Average	8.1471(+)	30.2205(+)	21.3782(+)	8.8731(+)	43.3514(+)	15.1784(+)	18.9644(+)
		STD	11.0458	62.2093	67.3585	26.6683	136.9062	25.2617	21.0744
	dopt-aiNET	Average	21.5754(+)	334.7268(+)	462.0574(+)	85.6756(+)	847.3436(+)	485.2867(+)	368.5643(+)
		STD	83.5927	401.3364	413.3435	223.7615	259.7643	456.2726	403.2342
	CPSO	Average	6.7254(+)	31.5738(+)	27.1358(+)	9.2742(+)	71.5704(+)	23.6757(+)	32.5842(+)
		STD	9.9747	63.5115	83.9873	24.2344	160.3211	51.5521	76.9105
	PSO-CP	Average	5.9931(+)	15.0895(–)	16.5418(+)	7.9072(+)	40.1644(+)	8.1575(+)	**8.2205(–)**
		STD	10.3175	60.9701	65.6901	24.394	99.0895	15.3906	**15.3943**

where $\vec{X}_{best}(r,k,s)$ is the best solution up to the sth sample in the kth environment during the ith run. The overall performance of the algorithm on all the test cases is then calculated as

$$performance = 100 \times \sum_{i=1}^{49} mark_i. \tag{14}$$

The comparative results in terms of mean offline errors and standard deviations calculated over 25 independent runs of each algorithm on each test case are provided in Table XII. In general, the GDBG system offers greater challenge to the optimization algorithms due to higher number of local optima and higher dimensionality. Comparing to the MPB case, we see that the performances of all the algorithms degrade in GDBG system. A close scrutiny of Table XII reveals that, out of the 49 test instances, CDDE_Ar outperforms all the six contestant algorithms in a statistically significant fashion over 39 instances. We also observe that no other algorithm could maintain such a consistent performance over a broad range of benchmark instances. The algorithm performs poorly for F1 with 10 peaks and change type T3. However, for the same change type over function F1 with 50 peaks, CDDE_Ar remains the second best, being outperformed only by jDE. For function F1 with 50 peaks and change type T7, although jDE beats CDDE_Ar by a small margin, the difference of their average offline errors is not statistically significant, as indicated by the rank sum test. CDDE_Ar managed to remain the second best performing algorithm for functions F3 with change type T5, F5 with change type T2, F4 with change type T4, and F6 with change type T7. For functions F5 with change type T2 and F3 with change type T5, jDE alone could provide statistically better results than CDDE_Ar, which was able to beat the other five algorithms in a statistically meaningful way. For function F4-T4, CDDE_Ar is outperformed by CPSO but remains statistically better than DASA, jDE, and dopt-aiNET and statistically comparable to PSO-CP. For function F6-T7, the results obtained with CDDE_Ar are statistically worse than PSO-CP but better than jDE, DynDE, DASA, CPSO, and dopt-aiNET. The consistently good performance over the majority of the benchmark instances indicate that the modifications introduced in CDDE_Ar for tackling dynamically changing fitness landscapes do have an edge over a set of well-known state-of-the-art DOP solvers.

The overall performances and average ranks of the algorithms on the 49 test cases are listed in Table XIII. In order to undertake multiple pairwise comparisons, we report the adjusted p-values obtained by using the following methods [59]: Nemeny's test, Holm's procedure, Shaffer's static procedure, and Bergmann–Hommel's dynamic procedure. Table XIV provides information about the state of retention or rejection of any hypothesis, comparing its associated adjusted p-value with the chosen $\alpha = 0.01$. If a p-value is less than α, then the corresponding hypothesis is considered as rejected. We note that hypotheses 1–6 are rejected by all the four procedures,

TABLE XIII
OVERALL PERFORMANCE AND AVERAGE RANKS OF CDDE_Ar AND PEER ALGORITHMS ON GDBG BENCHMARKS

Algorithm	CDDE_Ar	jDE	PSO-CP	DASA	CPSO	DynDE	Dopt-aiNet
Performance	69.47	64.51	58.37	57.46	51.63	47.47	38.29
Avg. Ranking	1.490	2.941	3.306	3.682	4.284	4.610	6.245

TABLE XIV
ADJUSTED p-VALUES FOR PAIRWISE STATISTICAL COMPARISONS

No.	Hypothesis	Holm	Schaffer	Nemeny	Bergman-Hommel
1.	CDDE_Ar vs. jDE	0.0062	0.0062	0.0094	0.0083
2.	CDDE_Ar vs. PSO-CP	3.29×10^{-5}	4.83×10^{-5}	2.61×10^{-4}	6.20×10^{-5}
3.	CDDE_Ar vs. Dopt-aiNet	5.26×10^{-24}	7.14×10^{-24}	5.47×10^{-23}	3.77×10^{-24}
4.	CDDE_Ar vs. DASA	4.17×10^{-8}	3.92×10^{-8}	4.85×10^{-7}	3.92×10^{-8}
5.	CDDE_Ar vs. CPSO	3.76×10^{-12}	7.52×10^{-12}	8.34×10^{-12}	7.58×10^{-12}
6.	CDDE_Ar vs. DynDE	3.45×10^{-14}	5.04×10^{-14}	5.36×10^{-13}	1.84×10^{-14}
7.	jDE vs. PSO-CP	0.0052	0.0073	0.0342	0.0129
8.	jDE vs. Dopt-aiNet	6.37×10^{-22}	2.84×10^{-22}	4.07×10^{-20}	4.62×10^{-22}
9.	jDE vs. DASA	0.0292	0.0292	0.0876	0.184
10.	jDE vs. CPSO	1.79×10^{-6}	1.79×10^{-6}	3.48×10^{-6}	1.36×10^{-6}
11.	jDE vs. DynDE	3.64×10^{-8}	3.75×10^{-8}	8.03×10^{-8}	2.46×10^{-8}
12.	PSO-CP vs. CPSO	2.79×10^{-3}	4.15×10^{-3}	6.72×10^{-3}	4.60×10^{-3}
13.	PSO-CP vs. DASA	0.0214	0.0173	0.0278	0.0469
14.	PSO-CP vs. DynDE	4.38×10^{-4}	4.38×10^{-4}	8.51×10^{-3}	4.55×10^{-4}
15.	PSO-CP vs. Dopt-aiNet	2.57×10^{-7}	3.63×10^{-7}	7.41×10^{-6}	9.42×10^{-7}
16.	DASA vs. CPSO	0.0283	0.0362	0.1135	0.0169
17.	DASA vs. DynDE	0.0018	0.0018	0.0076	0.0029
18.	DASA vs. Dopt-aiNet	5.18×10^{-4}	7.24×10^{-4}	4.02×10^{-4}	1.37×10^{-4}
19.	CPSO vs. DynDE	0.0038	0.0094	0.0107	0.0051
20.	CPSO vs. Dopt-aiNet	2.81×10^{-4}	6.02×10^{-4}	7.35×10^{-3}	1.36×10^{-4}
21.	DynDE vs. Dopt-aiNet	0.0031	0.0034	0.0062	0.0046

indicating the gross performance of CDDE_Ar to be significantly different from the six contender algorithms. We also observe that, for $\alpha = 0.01$, hypotheses 9, 13, and 16 are retained by all the four procedures and hypotheses 19 is retained by Nemeny's test only.

VI. CONCLUSION AND FUTURE WORK

A cluster-based dynamic DE algorithm has been proposed to solve DOPs efficiently. The key features of the algorithm can be summarized as follows.

1) Multipopulation strategy is used. The entire population is clustered according to spatial distribution, and every cluster evolves separately using an exploitative DE scheme.
2) The number of clusters is kept dynamic throughout the optimization process. A feedback system is implemented through the performance evaluation test, which is conducted after every certain number of iterations (TS), and it helps the algorithm to determine the change in cluster numbers. Accordingly, the algorithm redistributes the total population. Hence, a certain information sharing occurs among the separate clusters in certain points of time during optimization.
3) An external archive is used for better performance in dynamic environment. It is also seen experimentally that using Ar enhances the diversity level of the population.

The experimental results on both MPB and GDBG benchmark suites indicate that CDDE_Ar can achieve statistically better or competitive results as compared to several state-of-the-art algorithms over a wide spectrum of DOPs.

Our future work will focus on distributing the total population in the search region as uniformly as possible such that no part of the functional landscape may remain unexplored. When the increment in cluster number occurs, a fresh cluster is injected in the search region to explore the search region more. However, the individuals of the new cluster are generated randomly. This does not ensure that the new cluster will search in a less explored area. There is only a chance of that, and this is one possible downside of the proposed algorithm. The members of the new cluster should be generated in such a way that there is a good probability of exploring new areas. This may boost the performance of the algorithm in dynamic environments. For the basic DE scheme, we have resorted to standard values of F and Cr in order to impose no extra computational burden on the algorithm. However, exploring the use of randomized F and Cr values or making these parameters self-adaptive may improve the performance of the algorithm further. Future works may also investigate the effects of other DE schemes that indulge in balanced exploration and exploitation like DE/target-to-pbest/1 [60] and DE/current-to-gr_best/1 [61]. The effect of integrating memetic DEs (e.g., see [62]) in the clustering framework for solving DOPs can be investigated. The use of other sophisticated clustering schemes like fuzzy c-means and hierarchical clustering within the dynamic optimizer framework may also be studied.

REFERENCES

[1] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments—A survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, Jun. 2005.

[2] R. Storn and K. Price, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, Dec. 1997.

[3] R. Storn, K. Price, and J. Lampinen, *Differential Evolution—A Practical Approach to Global Optimization*. Berlin, Germany: Springer-Verlag, 2005.

[4] R. Storn and K. Price, "Differential evolution: A simple evolution strategy for fast optimization," *Dr. Dobb's J. Softw. Tools*, vol. 22, no. 4, pp. 18–24, Apr. 1997.

[5] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, Feb. 2011.

[6] K. Trojanowski and Z. Michalewicz, "Evolutionary optimization in non-stationary environments," *J. Comput. Sci. Technol.*, vol. 1, no. 2, pp. 93–124, 2000.

[7] J. Lampinen and I. Zelinka, "On stagnation of the differential evolution algorithm," in *Proc. MENDEL—6th Int. Mendel Conf. Soft Comput.*, P. Ošmera, Ed., Brno, Czech Republic, Jun. 7–9, 2000, pp. 76–83.

[8] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proc. Congr. Evol. Comput.*, 1999, vol. 3, pp. 1875–1882.

[9] C. Li and S. Yang, "A generalized approach to construct benchmark problems for dynamic optimization," in *Proc. 7th Int. Conf. Simul. Evol. Learn.*, 2008, pp. 391–400.

[10] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H. G. Beyer, and P. N. Suganthan, "Benchmark Generator for CEC 2009 Competition on Dynamic Optimization," Univ. Leicester, Univ. Birmingham, Nanyang Technol. Univ., 2008, Tech. Rep.

[11] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

[12] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Norwell, MA: Kluwer, 2001.

[13] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 6, pp. 1–24, Oct. 2012.

[14] H. G. Cobb, "An Investigation Into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments," Naval Res. Lab., Washington, DC, Tech. Rep. AIC-90-001, 1990.

[15] R. W. Morrison and K. A. De Jong, "Triggered hypermutation revisited," in *Proc. IEEE CEC*, 2000, pp. 1025–1032.

[16] F. Vavak, K. Jukes, and T. C. Fogarty, "Adaptive combustion balancing in multiple burner boiler using a genetic algorithm with variable range of local search," in *Proc. Int. Conf. Genetic Algorithms*, T. Back, Ed., 1997, pp. 719–726.

[17] J. J. Grefenstette, "Genetic algorithms for changing environments," in *Parallel Problem Solving From Nature*, R. Maenner and B. Manderick, Eds. Amsterdam, The Netherlands: North-Holland, 1992, pp. 137–144.

[18] W. Cedeno and V. R. Vemuri, "On the use of niching for dynamic landscapes," in *Proc. Int. Conf. Evol. Comput.*, 1997, pp. 361–366.

[19] R. K. Ursem, "Mutinational GA optimization techniques in dynamic environments," in *Proc. Genetic Evol. Comput. Conf.*, D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, and H. G. Beyer, Eds., 1998, pp. 19–26.

[20] M. Wineberg and F. Oppacher, "Enhancing the GA's ability to cope with dynamic environments," in *Proc. Genetic Evol. Comput. Conf.*, D. Whitley, L. Darrell, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. C. Parmee, and H.-G. Beyer, Eds., 2000, pp. 3–10.

[21] J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck, "A multipopulation approach to dynamic optimization problems," in *Adaptive Computing in Design and Manufacturing 2000*. Berlin, Germany: Springer-Verlag, 2000, ser. LNCS.

[22] J. Branke and H. Schmeck, "Designing evolutionary algorithms for dynamic optimization problems," in *Theory and Application of Evolutionary Computation: Recent Trends*, S. Tsutsui and A. Ghosh, Eds. Berlin, Germany: Springer-Verlag, 2002, pp. 239–262.

[23] S. J. Louis and Z. Xu, "Genetic algorithms for open shop scheduling and re-scheduling," in *Proc. 11th ISCA Int. Conf. Comput. Appl.*, 1996, pp. 99–102.

[24] C. L. Ramsey and J. J. Greffenstette, "Case-based initialization of genetic algorithms," in *Proc. 5th Int. Conf. Genetic Algorithms*, 1993, pp. 84–91.

[25] J. Eggermont and T. Lenaerts, "Dynamic optimization using evolutionary algorithms with a case-based memory," in *Proc. Belgium-Netherlands Conf. Artif. Intell.*, 2002, pp. 107–114.

[26] K. P. Ng and K. C. Wong, "A new diploid scheme and dominance change mechanism for non-stationary function optimization," in *Proc. 6th Int. Conf. Genetic Algorithms*, 1995, pp. 159–166.

[27] R. Calabretta, R. Galbiati, S. Nolfi, and D. Parisi, "Two is better than one: A diploid genotype for neural networks," *Neural Process. Lett.*, vol. 4, no. 3, pp. 149–155, Dec. 1996.

[28] J. Lewis, E. Hart, and G. Ritchie, "A comparison of dominance mechanisms and simple mutation on stationary problems," in *Parallel Problem Solving From Nature*, vol. 1498, *LNCS*, A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds. London, U.K.: Springer-Verlag, 1998, pp. 139–148.

[29] J. J. Grefenstette, "Genetic algorithms for changing environments," in *Proc. 2nd Int. Conf. Parallel Problem Solving From Nature*, 1992, pp. 137–144.

[30] S. Yang, "Genetic algorithms with memory- and elitism-based immigrants in dynamic environments," *Evol. Comput.*, vol. 16, no. 3, pp. 385–416, 2008.

[31] S. Yang, "Associative memory scheme for genetic algorithms in dynamic environments," in *Proc. Workshops Appl. Evol. Comput.*, vol. 3907, *Lecture Notes in Computer Science*, 2006, pp. 788–799.

[32] S. Yang, "A comparative study of immune system based genetic algorithms in dynamic environments," in *Proc. Genetic Evol. Comput. Conf.*, 2006, pp. 1377–1384.

[33] A. Simoes and E. Costa, "An immune system-based genetic algorithm to deal with dynamic environments: Diversity and memory," in *Proc. 6th Int. Conf. Neural Netw. Genetic Algorithms*, 2003, pp. 168–174.

[34] R. C. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *Proc. Congr. Evol. Comput.*, 2001, pp. 94–100.

[35] T. M. Blackwell and P. J. Bentley, "Dynamic search with charged swarms," in *Proc. Genetic Evol. Comput. Conf.*, 2002, pp. 19–26.

[36] T. S. Janson and M. Middendorf, "A hierarchical particle swarm optimizer for noisy and dynamic environments," *Genet. Program. Evolvable Mach.*, vol. 7, no. 4, pp. 329–354, Dec. 2006.

[37] J. Hu, J. Zeng, and Y. Tan, "A diversity-guided particle swarm optimizer for dynamic environments," in *Proc. Bio-Inspired Comput. Intell. Appl.*, vol. 9, *Lecture Notes in Computer Science*, 2007, no. 3, pp. 239–247.

[38] A. Carlisle and G. Dozier, "Adapting particle swarm optimization to dynamic environments," in *Proc. Int. Conf. Artif. Intell.*, 2000, pp. 429–434.

[39] S. Yang and C. Li, "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 959–974, Dec. 2010.

[40] X. Li and K. H. Dam, "Comparing particle swarms for tracking extrema in dynamic environments," in *Proc. Congr. Evol. Comput.*, 2003, pp. 1772–1779.

[41] L. Liu, D. Wang, and S. Yang, "Particle swarm optimization with composite particles in dynamic environments," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 6, pp. 1634–1648, Dec. 2010.

[42] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Trans. Evol. Comput.*, vol. 10, no. 4, pp. 440–458, Aug. 2006.

[43] R. Mendes and A. S. Mohais, "DynDE: A differential evolution for dynamic optimization problems," in *Proc. IEEE Congr. Evol. Comput.*, 2005, vol. 2, pp. 2808–2815.

[44] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, and V. Zumer, "Dynamic optimization using self-adaptive differential evolution," in *Proc. IEEE Cong. Evol. Comput.*, 2009, pp. 415–422.

[45] R. Angira and A. Santosh, "Optimization of dynamic systems: A trigonometric differential evolution approach," *Comput. Chem. Eng.*, vol. 31, no. 9, pp. 1055–1063, Sep. 2007.

[46] H.-Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," *J. Global Optim.*, vol. 27, no. 1, pp. 105–129, Sep. 2003.

[47] M. C. du Plessis and A. P. Engelbrecht, "Improved differential evolution for dynamic optimization problems," in *Proc. Congr. Evol. Comput.*, 2008, pp. 229–234.

[48] V. Noroozi, A. B. Hashemi, and M. R. Meybodi, "CellularDE: A cellular based differential evolution for dynamic optimization problems," in *Proc. ICANNGA—Part I*, vol. 6593, *LNCS*, A. Dobnikar, U. Lotrič, and B. Šter, Eds., 2011, pp. 340–349.

[49] R. I. Lung and D. Dumitrescu, "A collaborative model for tracking optima in dynamic environments," in *Proc. Congr. Evol. Comput.*, 2007, pp. 564–567.

[50] P. Korosec and J. Silc, “The differential ant-stigmergy algorithm applied to dynamic optimization problems,” in *Proc. Congr. Evol. Comput.*, 2009, pp. 407–414.
[51] J. Brest, P. Korošec, J. Šilc, A. Zamuda, B. Bošković, and M. Sepesy Maučec, “Differential evolution and differential ant-stigmergy on dynamic optimisation problems,” *Int. J. Syst. Sci.*, 2011. DOI:10.1080/00207721.2011.617899.
[52] F. O. de Franca and F. J. Von Zuben, “A dynamic artificial immune algorithm applied to challenging benchmarking problems,” in *Proc. Congr. Evol. Comput.*, 2009, pp. 423–430.
[53] L. N. de Castro and J. Timmis, “An artificial immune network for multimodal optimisation,” in *Proc. Congr. Evol. Comput. Part IEEE World Congr. Comput. Intell.*, Honolulu, HI, May 2002, pp. 699–704.
[54] J. B. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proc. 5th Berkeley Symp. Math. Stat. Probab.*, 1967, pp. 281–297.
[55] J. Kennedy, “Stereotyping: Improving particle swarm performance with cluster analysis,” in *Proc. IEEE Congr. Evol. Comput.*, 2000, pp. 1507–1512.
[56] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: A review,” *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999.
[57] F. Wilcoxon, “Individual comparisons by ranking methods,” *Biometrics*, vol. 1, no. 6, pp. 80–83, Dec. 1945.
[58] J. Derrac, S. García, D. Molina, and F. Herrera, “A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms,” *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, Mar. 2011.
[59] S. García, A. Fernández, J. Luengo, and F. Herrera, “Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power,” *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, May 2010.
[60] J. Zhang and A. C. Sanderson, “JADE: Adaptive differential evolution with optional external archive,” *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, Oct. 2009.
[61] S. Minhazul Islam, S. Das, S. Ghosh, S. Roy, and P. N. Suganthan, “An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 2, pp. 482–500, Apr. 2012.
[62] V. Tirronen, F. Neri, T. Kärkkäinen, K. Majava, and T. Rossi, “An enhanced memetic differential evolution in filter design for defect detection in paper production,” *Evol. Comput. J.*, vol. 16, no. 4, pp. 529–555, 2008.

Udit Halder is currently in the final year of the Bachelor of Engineering course in the Department of Electronics and Telecommunication Engineering, Jadavpur University, Calcutta, India.

He has published papers in peer-reviewed international journals and conferences. He has also acted as a reviewer for the journal “Swarm and Evolutionary Computation,” Elsevier. His current research interest includes evolutionary computation, mathematical analysis of various kinds of swarms, and machine intelligence.

Swagatam Das (M’10) received the B. E. Tel. E., M. E. Tel. E. (Control Engineering specialization), and Ph.D. degrees from Jadavpur University, Calcutta, India, in 2003, 2005, and 2009, respectively.

He served as an Assistant Professor with the Department of Electronics and Telecommunication Engineering, Jadavpur University, from 2006 to 2011. He is currently an Assistant Professor with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata. His current research interests include evolutionary computing, pattern recognition, multiagent systems, and wireless communication. He has published more than 140 research articles in peer-reviewed journals and international conferences. He is the Founding Coeditor-in-Chief of “Swarm and Evolutionary Computation,” an international journal from Elsevier. He serves as an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS (PART-A) and Information Sciences (Elsevier). He is an editorial board member of Progress in Artificial Intelligence (Springer), the International Journal of Artificial Intelligence and Soft Computing, and the International Journal of Adaptive and Autonomous Communication Systems. He has been acting as a regular reviewer for journals like Pattern Recognition, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE/ASSOCIATION FOR COMPUTING MACHINERY TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, and IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS PART A, PART B, AND PART C. He has acted as a Guest Editor for special issues in journals like IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and IEEE TRANSACTIONS ON SMC, PART C. He coauthored a research monograph on metaheuristic clustering techniques from Springer in 2009.

Dipankar Maity is currently in the final year of the Bachelor of Engineering course in the Department of Electronics and Telecommunication Engineering, Jadavpur University, Calcutta, India.

He has published several papers in peer-reviewed journals and in the proceedings of several conferences. He has acted as a reviewer for the journal *Swarm and Evolutionary Computation, Elsevier* and for the conference *Swarm Evolutionary and Memetic Computing*. His current research is in the fields of evolutionary computing, dynamical systems and chaos, neural networks, and machine learning.