An Adaptive Differential Evolution Algorithm for Global Optimization in Dynamic Environments

Swagatam Das, *Senior Member, IEEE,* Ankush Mandal, and Rohan Mukherjee

***Abstract*—This article proposes a multipopulation-based adaptive differential evolution (DE) algorithm to solve dynamic optimization problems (DOPs) in an efficient way. The algorithm uses Brownian and adaptive quantum individuals in conjunction with the DE individuals to maintain the diversity and exploration ability of the population. This algorithm, denoted as dynamic DE with Brownian and quantum individuals (DDEBQ), uses a neighborhood-driven double mutation strategy to control the perturbation and thereby prevents the algorithm from converging too quickly. In addition, an exclusion rule is used to spread the subpopulations over a larger portion of the search space as this enhances the optima tracking ability of the algorithm. Furthermore, an aging mechanism is incorporated to prevent the algorithm from stagnating at any local optimum. The performance of DDEBQ is compared with several state-of-the-art evolutionary algorithms using a suite of benchmarks from the generalized dynamic benchmark generator (GDBG) system used in the competition on evolutionary computation in dynamic and uncertain environments, held under the 2009 IEEE Congress on Evolutionary Computation (CEC). The simulation results indicate that DDEBQ outperforms other algorithms for most of the tested DOP instances in a statistically meaningful way.**

***Index Terms*—Differential evolution, diversity, double mutation strategy, dynamic optimization problems.**

I. Introduction

DIFFERENTIAL evolution (DE) [1], [2] has emerged as one of the most powerful real-parameter optimizers currently in use. DE implements similar computational steps to that of standard evolutionary algorithms (EAs). However, unlike traditional EAs, DE-variants perturb the current-generation population members with the scaled differences of randomly selected and distinct population members. Therefore, no separate probability distribution (like the Gaussian distributions used in evolutionary programming (EP) and evolution strategies (ES) or the Cauchy distributions used in case of the fast EPs) is used to generate offspring.

Several optimization problems in the real world are dynamic in nature. For these dynamic optimization problems (DOPs), the function landscape changes with time, i.e., optima of the problem to be solved change their locations over time and, thus, the optimizer should be able to track the optima continually by responding to the dynamic environment [3], [4]. Practical examples of such situations are price fluctuations, financial variations, stochastic arrival of new tasks in a scheduling problem, machine breakdown, or maintenance. Under dynamic environments, converging tendency of a conventional EA (implying the tendency of the population members of an EA to concentrate within a small basin of the search space as the iterations progress) imposes severe limitations on performance of the EA. If the population members of the EA converge rapidly, they will be unable to effectively respond to the environmental changes. Therefore, in case of DOPs the main challenge is to maintain a diverse population and at the same time to produce high quality solutions by tracking the moving optima. At this point, we would like to mention that there are also DOP instances where the optimal solution does not need to be tracked. For example, Allmendinger and Knowles [5] investigate DOPs where the constraints [ephemeral resource constraints (ERCs)] change over time but not the landscape and thus, also not the optimal solutions. In this paper, we focus on the real parameter-bound constrained DOPs where the objective function landscape explicitly changes with time and not on the problems with ERCs.

Classical DE faces difficulties when applied to DOPs due to two main factors. Firstly, DE individuals have a tendency to converge prematurely into small basins of attraction surrounding the local and global optima as the search progresses [6]. 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 individuals being similar and the consequently small perturbations. Secondly, DE may occasionally stop proceeding toward the global optimum even though the population has not converged to a local optimum or any other point [2], [6]. Researchers have made some attempts to introduce suitable algorithmic modifications in DE for enabling it to continually track changing optima under

Manuscript received July 30, 2012; revised July 11, 2013; accepted July 27, 2013. Date of publication August 28, 2013; date of current version May 13, 2014. This paper was recommended by Associate Editor Y. Tan.

This paper has supplementary downloadable multimedia material available at http://ieeexplore.ieee.org provided by the authors. The Supplementary document contains additional experimental and empirical evidences to validate the various algorithmic components of the proposed DDEBQ algorithm. It also provides empirical justification for the graded diversity preserving mechanisms induced by the quantum, Brownian, and Differential Evolution individuals. A complete pseudo-code of the algorithm has also been provided in this document. Moreover the choices of various control parameters of DDEBQ have been justified on the basis of extensive experimental results. This includes a PDF file, This material is 207 kB (0.2 MB) in size.

S. Das is with the Electronics and Communication Sciences Unit (ECSU), Indian Statistical Institute (ISI), Kolkata 700108, India (e-mail: swagatam.das@isical.ac.in).

A. Mandal and R. Mukherjee are with the Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata 700108, India (e-mail: ankushmandal19@gmail.com; rohan.mukherjii@gmail.com).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TCYB.2013.2278188

dynamic conditions. A brief account of such approaches is presented in Section II-B.

This article proposes a multipopulation-based adaptive DE with Brownian and Quantum individuals (DDEBQ) to address DOPs. In each subpopulation, one individual is an adaptive quantum individual, analogous to particles following the laws of quantum mechanics and one individual is a Brownian individual, whose trajectory is similar to the trajectory of any particle in Brownian motion. Other individuals in the subpopulation evolve in the DE framework with a neighborhood-based double mutation strategy. As quantum and Brownian individuals do not follow the same DE rules as others, they help in controlling the diversity of the population and thereby, in enhancing the search efficiency of the algorithm. A double mutation strategy is developed under the DE-framework to prevent the algorithm from converging quickly. In order to circumvent the problem of stagnation, an aging mechanism is integrated within DE. Also an exclusion scheme is used so that subpopulations may distribute themselves evenly over the entire search space. This increases the explorative power and the ability of the algorithm to track the global optimum. The performance of the proposed algorithm is primarily tested on a suite of DOPs generated by the generalized dynamic benchmark generator (GDBG) that was proposed for the special session and competition on "Evolutionary Computation in Dynamic and Uncertain Environments," held under the IEEE Congress on Evolutionary Computation (CEC) 2009 [7]. A comparison of DDEBQ with several state-of-the-art dynamic evolutionary optimizers reflects the statistical superiority of the algorithm over a wide variety of real-parameter DOPs. A list of terminologies used to describe DDEBQ can be found in Table IV of the appendix.

The rest of the paper is organized as follows. Section II provides a brief description of classical DE and also presents a compact survey of the different modified DE schemes previously used for solving DOPs. Section III describes the proposed DDEBQ algorithm with all its salient features in sufficient detail. Section IV describes the benchmarks used and explains the simulation strategies used for undertaking the experiments reported in the subsequent sections. Results of comparing DDEBQ with several state-of-the-art EAs are presented and discussed in Section V. Finally, conclusions are drawn in Section VI.

II. Background

A. Classical DE

A generation of the classical DE algorithm consists of four basic steps—initialization, mutation, crossover, and selection, of which, only last three steps are employed into DE generations. The generations continue till some termination criterion (such as exhaustion of maximum functional evaluations) is satisfied.

1) *Initialization of Population::* DE searches for a global optimum within a continuous search space of dimensionality D. It begins with an initial population of target vectors $\vec{X}_i = [x_i^1, x_i^2, ..., x_i^D]$, where $i = 1, 2, 3 \ldots .Np$ (Np is the population size). The individuals of the initial population are randomly generated from a uniform distribution within the search-space. The search-space has maximum and minimum bounds in each dimension and the bounds can be expressed as

$$\vec{X}_{\max} = \left[x_{\max}^1, x_{\max}^2, ..., x_{\max}^D\right] \text{ and } \vec{X}_{\min} = \left[x_{\min}^1, x_{\min}^2, ..., x_{\min}^D\right].$$

The jth component of the ith individual is initialized in the following way:

$$x_{i,0}^j = x_{\min}^j + rand_i^j(0,1) \cdot (x_{\max}^j - x_{\min}^j),\ j \in \{1, 2, \ldots, D\} \quad (1)$$

where $rand_i^j(0, 1)$ is a uniformly distributed random number in (0, 1) and it is instantiated independently for each jth component of the ith individual.

2) *Mutation:* 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 three most frequently referred mutation strategies for DE are listed below as

$$\text{DE/rand/1} : \vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}) \quad (2)$$

$$\text{DE/best/1} : \vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}) \quad (3)$$

$$\text{DE/current-to-best/1:}$$
$$\vec{V}_{i,G} = \vec{X}_{i,G} + F \cdot (\vec{X}_{best,G} - \vec{X}_{i,G}) + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}). \quad (4)$$

The indices r_1^i, r_2^i, and r_3^i are mutually exclusive integers randomly chosen from the range $\{1, 2, \ldots, Np\}$, 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 for scaling the difference vectors. $\vec{X}_{best,G}$ is the best individual vector with the best fitness (i.e., having the highest objective function value for a maximization problem) in the population at generation G. The general convention used for naming the various offspring generation strategies of DE is DE/x/y/z, where x represents a string denoting the vector to be perturbed and y is the number of difference vectors considered for perturbation of x. z stands for the type of crossover being used (*exp*: exponential; *bin*: binomial).

3) *Crossover:* The donor vector mixes its components with the target vector $\vec{X}_{i,G}$ under the crossover operation to form a trial vector of the same index denoted as $\vec{U}_{i,G} = [u_{i,G}^1, u_{i,G}^2, ..., u_{i,G}^D]$. The DE family of algorithms primarily uses two kinds of crossover schemes— exponential (or two-point modulo) and binomial (or uniform) [2]. The binomial crossover scheme is briefly explained below since it is used in the proposed algorithm. Under this scheme the trial vector is created as follows:

$$u_{i,G}^j = \begin{cases} v_{i,G}^j & \text{if } rand_i^j(0,1) \le CR \quad \text{or} \quad j = j_{rand} \\ x_{i,G}^j & \text{otherwise} \end{cases} \quad (5)$$

where Cr is a user-specified parameter (crossover rate) in the range [0, 1) and $j_{rand} \in \{1, 2, ..., D\}$ is a randomly chosen index, which ensures that the trial vector $\vec{U}_{i,G}$ differs from its corresponding target vector $\vec{X}_{i,G}$ by at least one component.

4) *Selection:* The next step of the algorithm calls for selection to determine which of the target or the trial vectors survives to the next generation, i.e., at $G = G + 1$. For a maximization problem, if the objective function value of the trial vector is not less than that of the corresponding target vector, then the trial vector is selected for the next generation; otherwise the target vector is selected for the next generation. Obviously, for a minimization problem the condition for selection is just the opposite.

B. Dynamic Optimization With DE—Brief Overview

Since the late 1990s, DE started to receive attention from DOP researchers. Mendes and Mohais presented DynDE [8], 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 secondly, by 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 Moving Peaks Benchmark (MPB) problems efficiently. Brest *et al*. [9] 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 [10]–[13]. Recently, Halder *et al*. [14] proposed a multipopulation DE for solving DOPs. In this proposal, 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 DE algorithm. The number of clusters is an adaptive parameter and its value is updated after a certain number of iterations.

Various niching strategies [15] have been proposed by the EA researchers to adapt an EA for detecting and maintaining multiple optima over a multimodal functional landscape. Niching also helps in preserving the population diversity in the course of an EA and track moving peaks in dynamic optimization. DE has been modified to induce efficient niching behavior on multimodal landscapes in some prominent works, such as bi-objective DE with mean distance-based selection [16], crowding-based DE [17], and DE-based multimodal optimization using the principle of locality [18]. Parrott and Li [19] used the speciation technique to track multiple peaks in a dynamic environment. Subsequently in 2006, Li *et al.* [20] used speciation-based particle swarm optimization (SPSO) to tackle DOPs by using detection and response. The method is designed for solving problems with primarily unknown numbers of peaks. Lung and Dumitrescu [21] used crowding DE to maintain diversity and combined it with PSO, called collaborative evolutionary-swarm optimization (CESO) to solve dynamic optimization problems. In 2009, Lung and Dumitrescu [22] further improved and extended their work by introducing one more crowing DE population that acted as a memory for the main population. However, most of the dynamic niching techniques necessitate the use of niching parameters, such as the niching radius or the crowding factor, which in turn require prior knowledge about the functional landscape for proper tuning [15]. This may lead to poor performance on complicated dynamic functions like those designed with the GDBG system.

III. DDEBQ Algorithm

A. Dynamic DE Scheme

In order to maintain diversity of the population to a larger extent, DDEBQ introduces adaptive quantum and Brownian individuals along with the DE individuals in the population. These quantum and Brownian individuals do not follow the same rule as the DE individuals. Actually, within a subpopulation, two individuals are randomly chosen at each generation. The quantum individual generation rules are applied to one of them and the Brownian individual generation rules to the other. If one of the chosen individuals happens to be the best individual of that subpopulation, then the choice is discarded and another individual is randomly picked for subjecting it to the Brownian or quantum individual generation processes.

1) *Quantum Individuals:* In quantum mechanics, due to the uncertainty in position measurement, the position of a particle is probabilistically defined. This idea is used here to generate individuals within a specific region around the local best position. The steps for stochastically generating an individual, whose position is inside a hyper-sphere of radius R and centered at the local best position $\vec{L}b$ can be outlined as follows.

1) Generate a radial distance randomly from a uniform distribution within the range $(0, R)$ as:$r \sim U(0, R)$. This implies $0 \leq r \leq R$.
2) Generate a vector with each component being sampled at random from a normal distribution having zero mean and unity variance: $\vec{X} = [x_1, x_2, ..., x_D]$; $x_d = N(0, 1)$, where $1 \leq d \leq D$ and $N(\mu, \sigma)$ denotes the normal distribution with mean μ and standard deviation σ.
3) Compute the distance of the vector from the origin $\|\vec{X}\| = \sqrt{\sum_{i=1}^{D} x_i^2}$.
4) The new quantum individual's position will be

$$\vec{X}_q = \vec{L}b + \left(\frac{r}{\|\vec{X}\|}\right)\vec{X}. \tag{6}$$

In DDEBQ, the radius R within which the quantum individuals are generated is adaptive in nature, i.e., the radius is automatically updated according to certain conditions and with the progress of the search. The adaptation of R is explained in Section III-E as it depends on the control parameter C.

2) *Brownian Individuals:* Brownian motion is used to describe the random movement of particles suspended in a fluid. In mathematics, Brownian motion is described by the Wiener process (a continuous-time stochastic process named

in honor of Norbert Wiener). The Wiener process $W_t$is characterized by the following three facts: 1)$W_0 = 0$; 2) is almost surely continuous; and 3) $W_t$has independent increments with distribution, i.e., $W_t - W_s \sim N(0, t - s)$. for $0 \leq s \leq t$. DDEBQ employs a very simple method to simulate the Brownian motion. New individuals are generated within a Gaussian hyper-ellipsoid centered at the local best position. If the local best position is $\vec{L}b$, then, the new Brownian individual's position will be

$$\vec{X}_B = \vec{L}b + \vec{\Delta} \tag{7}$$

where the Gaussian perturbation vector $\vec{\Delta} = [\Delta_1, \Delta_2, ..., \Delta_D]$; $\Delta_d = N(0, \sigma)$, with $1 \leq d \leq D$ and σ is the standard deviation of the multivariate normal distribution from which each component of the perturbation is randomly sampled. Here, the value $\sigma = 0.2$ is used following [8] and considering the fact that this value gave the best results for most of the tested benchmark instances.

We would like to mention here that Wong *et al.* [23] proposed a niching algorithm where, in the species-specific exploration stage, random individuals are generated to maintain the diversity of the population for detecting multiple peaks on a static landscape. However, the proposed Brownian and quantum individual generation schemes are considerably different from what was done in [23].

3) *DE Individuals:* These individuals evolve following the standard DE algorithm. The donor vectors are generated following a new mutation scheme that is detailed below. However, these individuals follow the same binomial crossover and selection process as that of the standard DE algorithm.

B. Double Mutation Strategy

In a dynamic environment, if the population is concentrated around the global optimum, then the individuals will lose their ability to detect the global optimum again when the position of the latter changes. Thus, here the idea is to control the perturbation to slow down the searching process and make the subpopulations evenly distributed over the entire search space. An exclusion rule is employed to meet the second objective and the rule is discussed in Subsection III-C. For the first objective, DDEBQ follows a double mutation scheme, which is conceptually motivated by the work of Das *et al.* [24] in a different context. Under this scheme, first a mutant vector is generated according to a neighborhood-based mutation scheme and then the final donor vector is produced as a linear combination of the mutant vector with the local best vector (of the corresponding subpopulation) formed through a constant weight factor.

1) *Neighborhood-Based Mutation Strategy:* In order to overcome the limitations of the fast but less reliable convergence characteristics of *DE/current-to-best/*1*/bin*, some changes are introduced in the process of generating the difference vectors. For the first difference vector, the original scheme uses the difference between the global best individual and the current individual; however, in the modified scheme, the difference between the nearest memory individual and the current individual is considered. The memory archive contains a collection of the best individuals from the previous subpopulations. This modification is done to control the convergence of the population toward global optima and to encourage the subpopulations to explore the vicinity of the corresponding local best positions. For the second differential vector, instead of taking the difference between two randomly chosen individuals, DDEBQ uses the difference between the best individual in the neighborhood and the worst individual in the neighborhood with respect to the current individual. This modification is likely to guide the mutant vector to explore the neighborhood of the current individual within the subpopulation.

Note that the concept of neighborhood in [24] is solely based on the index graph of the DE vectors and two given vectors are neighbors if they have adjacent indices, albeit they may not be adjacent geographically or according to fitness values. In this paper, the neighborhoods bear a completely different meaning as will be evident from the following discussion. The first mutation can be expressed as

$$v^j_{mut,G} = x^j_{i,G} + F^j_{mem} \cdot (x^j_{mem,G} - x^j_{i,G}) + F^j_{bw} \cdot (x^j_{n\_best,G} - x^j_{n\_worst,G}) \tag{8}$$

where $j \in \{1, 2, \ldots, D\}$ and $x^j_{i,G}$ is the jth component of $\vec{X}_{i,G}$ that is the current vector. Similarly $x^j_{n_best,G}$is the jth component of $\vec{X}_{n\_best,G}$ that is the best vector in the neighborhood with respect to the current vector. It is the vector within the corresponding subpopulation for which $\frac{1}{r_{ik}} \left(\frac{f(\vec{X}_{k,G})}{f(\vec{X}_{i,G})} - 1 \right)$ ($k = 1, 2, \ldots, m$, where m = number of individuals in the subpopulation and $k \neq i$) is maximum. Here, r_{ik} is the Euclidean distance between the vectors $\vec{X}_{i,G}$ and $\vec{X}_{k,G}$. $x^j_{n\_worst,G}$ denotes the jth component of $\vec{X}_{n\_worst,G}$, which is the worst vector in the neighborhood with respect to the current vector. For this vector, $\frac{1}{r_{ik}} \left(1 - \frac{f(\vec{X}_{k,G})}{f(\vec{X}_{i,G})} \right)$ ($k = 1, 2, \ldots, m$ and $k \neq i$) is maximum among all individuals within the subpopulation. $x^j_{mem,G}$ denotes jth component of the nearest memory individual (memory individuals are the best individuals from the previous subpopulations) to $\vec{X}_{i,G}$ in terms of Euclidean distance. During the process of generating the mutant vector, for each dimension of each difference vector, the respective scaling factors are randomly generated from a uniform distribution within a range and this range is varied inversely with the magnitude of the differential vector along the corresponding dimension. DDEBQ generates the scaling factors for each jth component in the following way:

$$F^j_{mem} = 0.3 + 0.7 \cdot rand^j_i[0, 1] \cdot \left(1 - \frac{|x^j_{mem,G} - x^j_{i,G}|}{|SR^j|} \right) \tag{9a}$$

$$F^j_{bw} = 0.3 + 0.7 \cdot rand^j_i[0, 1] \cdot \left(1 - \frac{|x^j_{best,G} - x^j_{worst,G}|}{|SR^j|} \right) \tag{9b}$$

where $|SR^j|$ is the search range corresponding to the jth dimension. Clearly, as the difference increases, i.e., approaches $|SR^j|$, the value of the scaling factor reduces to

0.3. Zaharie [25] suggested that the values of F, which satisfy the equation, $2F^2 - {}^{2}/_{m} + {}^{Cr}/_{m} = 0$ can be considered to be critical. Here, m is the number of individuals in a subpopulation. In case of a single population algorithm, m should be replaced by Np. In DDEBQ, Cr is kept constant at 0.9 and m is six. Putting these values in the equation, the critical value for the scaling factor F becomes 0.285. Therefore, the lowest value of the scaling factor is set to 0.3 for convenience. However, the above equation can be used only for the DE/rand/1/bin scheme (2). For the DE-variants involving best individuals, the expression describing the influence of F and Cr becomes more complicated. The above equation is used here only to provide an indication of the actual critical value of F; it is not meant to give a precise estimate.

2) *Second Stage Mutation:* A linear combination of the mutant vector from the 1st stage of mutation to the local best vector is formed by using a weight factor. This way the local best vector is perturbed in a controlled manner. The second mutation can be expressed as

$$\vec{V}_{final,G} = (1 - \omega) \cdot \vec{L}_{b,G} + \omega \cdot \vec{V}_{mut,G} \tag{10}$$

where $\vec{L}_{b,G}$ is the local best vector, i.e., the best vector of the corresponding subpopulation, $\vec{V}_{mut,G}$ is the mutant vector generated from 1st stage mutation and ω is the weight factor.

C. Exclusion Rule

In DDEBQ, an exclusion rule is employed to ensure that different subpopulations are located around different basins of attraction. However, this rule is slightly different from the existing one [8] as it uses a new empirical formula to calculate the marginal distance between two subpopulations. Here, the strategy is to calculate the Euclidean distance between the best individuals from two different subpopulations at each generation. If the distance between the best individuals of any two subpopulations falls below a marginal value, then the subpopulation having the best individual of lower objective function value (i.e., worse fitness for a maximization problem) between the two is marked for reinitialization. The marginal value of the distance is calculated according to the following rule:

If there are D dimensions with search ranges SR and there are Nsub subpopulations, then the marginal value for the distance is

$$Dis\_marginal = SR/(Nsub \cdot D). \tag{11}$$

Here, the idea is to partition the search space almost equally among the Nsub subpopulations. Note that the DyneDE [8] algorithm uses the linear diameter of the basin of attraction as an indicator for this exclusion radius. Unlike DyneDE's exclusion scheme [(1) of [8]], the formula given in (12) does not make implicit assumption that the peaks are evenly distributed in the search space. It also eliminates the need for knowing the number of peaks of the objective function beforehand.

Algorithm 1 Algorithm for Aging Mechanism: (Considering *j*th individual of the *i*th subpopulation)

1. **if** the *i*th subpopulation contains the global best individual, **then** do not perform aging mechanism on the subpopulation.
2. **else if** the *j*-th individual is the best individual in the *i*-th subpopulation,
   then $Age\_best(i, j) = Age\_best(i, j) + 1$.
   if $Age\_best(i, j) \geq 30$, **then** reinitialize the *i*-th subpopulation and reset $Age\_best(i,:)$ and $Age\_worst(i,:)$ entries to 0.
3. **else if** *j-th* individual is the worst individual in the *i*-th subpopulation, **then** $Age\_worst(i, j) = Age\_worst(i, j) + 1$.
   if $Age\_worst(i, j) \geq 20$, **then** reinitialize the individual and reset $Age\_worst(i, j)$ entries to 0 leaving other members of the subpopulation intact.
4. **else** the $Age\_worst(i, j)$ and $Age\_best(i, j)$ of the *j-th* individual are reset to 0.

D. Aging Mechanism

DDEBQ employs a simple aging mechanism to get rid of the individuals stagnating at some local optimum. Algorithm 1 shows a schematic procedure to implement the aging mechanism. Age_best and Age_worst are two matrices with dimensions (*Nsub*, *m*), *m* being the number of individuals per subpopulation. The (*i*, *j*)th entry of Age_best matrix represents how many times consecutively the *j*th individual of *i*th subpopulation has been the best individual of the *i*th subpopulation. In the same way, the (*i*, *j*)th entry of Age_worst matrix represents how many times consecutively the *j*th individual of *i*th subpopulation has been the worst individual of the *i*th subpopulation. If an individual is reinitialized owing to its consistently bad performance then the corresponding entry of the Age_worst matrix is reset to 0 but the Age_best matrix remains unaltered. If a subpopulation is reinitialized due to stagnating at any local optimum, then the corresponding row entries of the Age_best and Age_worst matrices are all reset to 0. The reinitialization is done randomly covering the entire search space.

Aging is a heuristic method and its objective is to reinitialize the individuals that may be trapped at some local optimum. Except for the experimental results, it is difficult to justify the choice for the thresholds. They should be set in such a fashion that the reinitialization may occur only when stagnation is heuristically sensed. In DDEBQ, the aging thresholds are set to 30 and 20 for Age_best and Age_worst, respectively, through a series of experiments carried on the available benchmarks. A lower aging threshold will mean more reinitializations that might be unnecessary whereas a high aging threshold will mean more wastage of FEs to achieve the same level of accuracy.

E. Adaptation of Control Parameter and Radius of Generating Quantum Individuals

In order to actuate the diversity within the subpopulations, a control parameter is introduced in DDEBQ. Depending on the conditions, this parameter can take any value among 0, 1, and 2. This parameter, denoted by C, helps the search

process to achieve better convergence characteristics. If C becomes one, then, the quantum individuals are not generated; if C becomes two, then, the Brownian individuals are not generated, and if C becomes 0 then the search process progresses in normal way, i.e., with quantum, Brownian, and DE individuals.

As mentioned previously, when C is 0, the algorithm generates all the individuals (DE, Brownian, and quantum) to maintain the diversity at a higher level. When the population concentrates around a global best position before the occurrence of a dynamic change, as determined by the control parameter C, the diversity should be reduced separately within each of the subpopulations to ensure high precision in locating the global optima. If the diversity of the individuals in each of the subpopulations is reduced separately, then irrespective of the whole population's diversity, the subpopulation containing the global best individual converges to the global best position and the algorithm is likely to achieve a high degree of accuracy. In this way, while preserving the population diversity as a whole, DDEBQ can also obtain high quality solutions. This is possible because the subpopulations are located at different regions of the search space due to the exclusion rule, which is described earlier. In DDEBQ, the diversity is reduced in two steps, first by stopping the generation of quantum individuals and then by stopping the generation of Brownian individuals and starting the generation of quantum individuals. As quantum individuals are likely to possess less diversity than Brownian individuals [26], after the second step, the diversity is expected to decrease more.

The value of C is chosen in the following way. First, the difference of the global best objective function values before and after the first update interval (UI) generations is defined as PR. From this point onward, if the global best objective function values over UI generations have a difference greater than PR, then the current value of PR is replaced by this new value. If the difference becomes less than (PR /10) but greater than (PR /50), then C is set to one. If the difference is less than (PR /50), then C is set to two. A value too low as indicated by (PR /50), indicates that the algorithm has not experienced severe explorations in last UI generations and it can be concluded to be incisively searching around a possible optima. A higher value, even greater than (PR /10), can be referred to be in its explorative phase. A moderate value within these extremes can indicate an algorithm in its balanced explorative and exploitative phase. With respect to these values the control parameter C can be determined, which wheels the dynamics of the search process in DDEBQ by controlling the generation of the Brownian and quantum individuals. The strategy for adapting control parameter C is presented as Algorithm 2.

Note that the adaptation of C depends on monitoring of the progress of search (in terms of the frequency of variation of the globally best individual) at regular intervals and this bears some conceptual resemblance with the cooling schedules used in adaptive simulated annealing (SA) algorithms [27]. For example, the cooling schedule in hide-and-seek SA [28] depends on the best objective function values obtained up to a certain number of generations and an estimation of the unknown global optimum after the same number of generations. The performance is monitored after a specific UI (defined in the terminology list of the appendix). Based on the rate of change of the globally best solution during the intervals of the UI number of generations, the diversity is controlled by generating either Brownian or adaptive quantum or both kinds of individuals. Hence, selecting a proper value for frequency of update is decisive to performance of the algorithm. If the test is conducted too frequently, i.e., UI is very low, the search agents may not get enough scope to thoroughly explore the space. On the other hand, if UI is relatively high with respect to the frequency of occurrence of the dynamical changes, the detection of proper stages of optimization may be hampered. For GDBG problems, where E is 100000 FEs, $UI = 20$ gives optimal performance. In fact, the performance of the algorithm is not sensitive to UI values lying in the range of 15 to 35 and remains more or less consistent on different benchmarks from the GDBG suite. Our simulation experiments (not reported in the paper for space economy) indicate that a lower value of UI is suitable for lower change intervals while a higher value of UI is suited to higher change intervals. From our detailed empirical study, a value of $UI = 20$ can provide optimized performance over a wide range of functions. It can be noted that UI of 20 is in same range as aging thresholds age_best and age_worst. In the experimentation part, UI is fixed to 20 for all benchmark instances and no problem specific tunings were allowed.

Adaptation of the radius R for the generation of control parameters depend on C. If C is 0, then R is set to one. If C is two, then R changes according to the following rule depending upon the difference ($Diff$) of global best objective function values before and after UI generations

$$R = Diff \cdot \log_{10}\left(10 + \frac{PR}{50 \cdot Diff}\right). \tag{12}$$

Algorithm 2 Algorithm for Control Parameter (C) Adaptation

1. Initialize generation counter $G = 0$ and calculate initial $Gbest\_fit_0 = f(\vec{X}_{best,0})$, $C = 0$.
   //$\vec{X}_{best,G}$ is the globally best solution at generation G and f (.) is the function under test.
2. Initialize counter $k = 1$.
3. Start Loop
4. Carry out the optimization steps of DDEBQ
5. **if** $mod(Gen, UI) == 0$
6. Calculate new $Gbest\_fit_k = f(\vec{X}_{best,G})$.
7. Calculate PR_kas:$PR_k = |Gbest\_fit_k - Gbest\_fit_{k-1}|$
8. **if** $PR_k > PR_{k-1}$
9. Update PR.
10. **if** $(PR_k - PR_{k-1}) < PR_{k-1}/50$, $C = 2$
11. **else if** $PR_{k-1}/50 < (PR_k - PR_{k-1}) < PR_{k-1}/10$, $C = 1$
12. **else** $C = 0$.
13. $k = k + 1$.
14. G=G+1
15. **if** termination condition satisfies break Loop, **else** return to Step 4.

F. Dynamic Dimensional Change Addressing

In addition to changing the search landscape and thereby changing the functional values of the individuals, some challenging benchmark problems, such as GDBG with change type T7 [7], accompanies altering height, width, position of the optima, varying dynamics in orientation, scalability of the problem as well as a dimensionality contrast after a specific number of FEs. In that case, the algorithm needs to detect whether a dimensional change has occurred or not. The objective function of GDBG 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. Hence, the occurrence of a dimensional change can be detected by examining the test solution vector returned by the cost function in every generation. Whenever the dimension of the new test solution returned by the cost function does not match the dimension of the previous one, it can be inferred that a dimensional change has occurred in the environment. If the dimension is increased by one, then an extra dimension is added to the other individuals within the population. The values of extra dimensions of the individuals are randomly sampled from a uniform distribution within the corresponding bounds of the search space. If the dimension is decreased by one, then the additional dimension of other individuals within the population is eliminated.

G. Complexity Issues—Empirical Discussion

Apart from the computational burdens of evaluating the objective function (measured in terms of the number of FEs), another aspect of complexity of the algorithm can arise from the calculations of Euclidian distances between the current individual and the memory individuals during construction of the mutant vector for the current individual. This is because computing the Euclidean distances can demand a considerable amount of processor time. If each subpopulation contains m number of individuals and the total population size is denoted by Np then the number of subpopulations is $N_{sub} = (Np/m)$). As the memory archive contains the best individuals from each subpopulation, the number of memory individuals is also(Np/m). Hence, the total number of evaluations of Euclidian distances in one generation is total population size $\times$ number of evaluations of Euclidian distances for each individual. Therefore, the total number of evaluations of Euclidian distances in one generation is $\left(Np^2/m\right)$. As can be observed, if the number of individuals in each subpopulation is increased, i.e., as the multipopulation scheme approaches to a single-population scheme, part of the complexity of the algorithm decreases but it also loses the effectiveness of having multiple subpopulations. On the other hand, if the number of subpopulations is increased, the number of individuals in each subpopulation decreases and the complexity of the algorithm increases (the complexity gradually approaches $O(Np^2)$), but the effectiveness of the multipopulation scheme increases. Note that, according to Yang and Li [29], the timing complexity of the clustering operation in the clustering PSO algorithm that also uses Euclidean distance calculations heavily is $O(Np^2)$, Np being the initial population size.

H. Repairing Rule

For every newly generated individual (whether it is a DE, Brownian, or adaptive quantum individual), the algorithm checks whether any component of the new individual is outside the bounds. If any component is outside the bound, then it is randomly reinitialized by sampling from a uniform distribution within the bounds as per (1).

IV. Experimental Settings

A. Benchmark Problems

CEC 2009 benchmark problems for dynamic optimization were generated by using the GDBG system proposed in [7], which constructs dynamic environments for the location, height, and width of peaks. Li *et al.* [7] introduced a rotation method instead of shifting the positions of peaks as done in the MPB [30] 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 test functions in the GDBG system, which are small step change, large step change, random change, chaotic change, recurrent change, recurrent change with noise, and dimensional change.

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 functions. Only F1 is a maximization problem and others are minimization problems. In F1, there are two tests, one using 10 peaks and another using 50 peaks.

B. Simulation Strategies

Simulation environment (hardware and software) used for carrying out the experiments described in the subsequent sections can be summarized as CPU: 3.2 GHz Intel Core *i*5, RAM: 2 GB DDR3, and MATLAB 2009b edition. The performance of DDEBQ is measured in terms of the mean error [7] and the adaptability metric [31] obtained in 20 independent runs. The mean error is calculated according to the following expression [7]:

$$E_{mean} = \frac{1}{(runs * num\_change)} \sum_{i=1}^{runs} \sum_{j=1}^{num\_change} E_{i,j}^{last}. \tag{13}$$

Here, runs is the total number of runs, num_change is the number of dynamic changes that occur during each independent run, and $E_{i,j}^{last}$ is the error recorded before the jth dynamic change of ith independent run. Note that the error E^{last} corresponds to the absolute fitness difference between the best solution found by an EA (before a landscape change) and the known best solution (for that landscape), i.e., $E^{last}(t) =$

TABLE I
EXPERIMENTALLY DETERMINED BEST PARAMETRIC SETTINGS FOR DDEBQ

Parameters	Values
Population size, *Np*	60
Number of subpopulations, *Nsub*	10
Update Interval *UI*	20
Control parameter *C*	Depending on the condition it becomes 0, 1 or 2.
Radius within which quantum individuals are generated, *R*	If $C = 0$, then $R = 1$ If $C = 2$, then R varies according to (8)
Marginal value of the distance between the best individuals of the subpopulations *Dis_marginal*	If $C = 0$, then *Dis_marginal* = 0.08 If $C = 1$ or 2, then *Dis_marginal* = 0.03
Crossover Probability *Cr*	0.9
Weight factor for double mutation strategy	0.1

$\left|f(\vec{X}_{best}(t)) - f(\vec{X} * (t))\right|$. In all result tables, the best results are marked in boldface.

The adaptability metric measures a difference between the value of the current best individual of each generation and the optimum value averaged over the entire run and can be expressed as

$$Ada = \frac{1}{num\_change} \sum_{i=1}^{num\_change} \left[\frac{1}{\tau} \sum_{j=0}^{\tau-1} err_{i,j} \right] \tag{14}$$

where τ is the number of generations between changes when the environment remains static. $err_{i,\ j}$ denotes the absolute difference between the fitness values of the current best individual in the population of the jth generation and after the last change and the optimum value for the fitness landscape after the ith change. Evidently for both mean error and adaptability, the smaller the measured values are, the better the result is.

For results of the comparative studies, a nonparametric statistical test, called the Wilcoxon's rank sum test for independent samples [32], is conducted at the 5% significance level, in order to judge the statistical significance of the best results obtained in each experimental scenario. The statistical test results are indicated within parentheses throughout all the result tables as "+", " - ", or "≈", when the result of DDEBQ is statistically significantly better than, worse than, or statistically equivalent to the corresponding result, respectively. The rank sum test is conducted between the results of DDEBQ and the other dynamic EAs considered.

C. Parameter Settings

Table 1 lists the parametric values that keep the performance of DDEBQ considerably good over a wide range of benchmarks. Please refer to the supplementary document for a detailed account of the simulation experiments that empirically validate these values. Also once set, the same parameter values are used for DDEBQ on all the benchmark instances of GDBG in Section V, where performance of the algorithm is compared with some of the best-known evolutionary DOP solvers. No function-dependent tuning of the parameters is allowed anywhere for DDEBQ.

V. RESULTS AND DISCUSSIONS

This section presents a comparative study of the performance of DDEBQ with several other state-of-the-art evolutionary dynamic optimizers on the GDBG benchmarks. The performance of DDEBQ is compared with the following seven algorithms by using the benchmark suite of the GDBG system: Differential Ant-Stigmergy Algorithm (DASA) [33], jDE [9], [33], DynDE [9], dopt-aiNET [34], CPSO [28], CESO [21], and PSO with Composite particles (PSO-CP) [35]. DASA is based on the classical ant colony optimization methods. CPSO uses a hierarchical clustering method to locate and track multiple peaks. In addition, CPSO incorporates a fast local search method to search for optimal solutions in a promising subregion found by the clustering method. PSO-CP uses the idea of composite particles from physics to maintain the diversity of the population through a scattering operator. Dopt-aiNet introduces a set of complementary mutation operators and a better mechanism to maintain the diversity of solutions in the original opt-aiNet [36] algorithm, which was meant for solving static and multimodal function optimization problems

For the competitor algorithms, the best parametric setup is employed in accordance with their respective literatures. An identical experimental condition guided by the technical report of [7] is maintained for all the algorithms compared. Tables II and III provide the simulation results obtained over all the test cases mentioned in [7] by using DDEBQ and seven other algorithms in terms of the mean best-of the-run error values and the adaptability metric values achieved over 20 independent runs. The tables also show the average runtime (in seconds) consumed by all the algorithms compared. Sample convergence graphs are provided for functions F1 (number of peaks = 10), F2, F3, F4, F5, and F6 with change type T7 over 300,000 FEs in Fig. 1. In this case, dimension of the search space changes when the dynamic change occurs. This change type is similar to T3 (random change) except for the dimensional change. The y-axis of these plots contains the relative value $r(t)$ that is calculated as $f(\vec{X}_{best}(t))/f(\vec{X}^*(t))$ for function F1 (as it is a maximization problem) and for other functions as $f(\vec{X}^*(t))/f(\vec{X}_{best}(t))$. The highest possible value of $r(t)$ is one. As can be observed from the convergence graphs, the relative value is lowest in case of function F3 and it is highest in case of function F1. The convergence characteristics also indicate that as each dynamic change occurs, the relative value $r(t)$ attains a sharp downfall.

A close scrutiny of Tables II and III reveals that DDEBQ outperforms all the seven evolutionary dynamic optimizers in a statistically significant fashion over 36 out of the 49 test instances. It yielded statistically inferior results compared to any one of the competitor algorithms in four test instances and statistically equivalent results with one or two competitors over the rest nine instances. For function F3, the jDE algorithm could attain lower best error values than DDEBQ over change types T2, T4, and T6. However, results of the Wilcoxon's rank sum test reveals that for change types T1, T3, T5, and T7, the differences between the results of jDE and DDEBQ are not statistically significant. Also DDEBQ exhibited a statistically better performance than the other six EAs for all the change

TABLE II

MEAN ERROR VALUES, ADA METRIC VALUES, AND AVERAGE RUNTIME (IN SECONDS) ACHIEVED BY THE ALGORITHMS COMPARED FOR TEST FUNCTIONS F1-F3 OF THE GDBG SYSTEM. WILCOXON'S RANK SUM TEST RESULTS OF COMPARING DDEBQ WITH THE CONTENDER ALGORITHMS INDICATED IN PARENTHESES.

Func.	Algorithm	Performance Measures	T1	T2	T3	T4	T5	T6	T7
F1 (No. of peaks= 10)	DDEBQ	Mean Error	**2.03E-09**	**1.84E-07**	**6.52E-09**	**4.42E-09**	**2.98E-06**	**2.80E-06**	**0.2934**
		Ada	**13.1401**	**22.3389**	**3.2469**	**12.1296**	**6.5727**	**7.5671**	**3.6621**
		Runtime	1037.3542	1210.4513	1174.2241	1319.3932	1417.7332	1214.2819	1102.3617
	DASA	Average	0.1864(+)	4.2017(+)	6.4126(+)	0.5003(+)	2.2173(+)	2.8562(+)	4.9502(+)
		Ada	19.3184	30.7419	23.1854	21.5219	15.2789	19.4518	21.4971
		Runtime	1012.8291	1209.3728	1004.2781	1382.8264	1302.8281	1028.3512	991.2617
	jDE	Average	0.0342(+)	3.6019(+)	3.1058(+)	0.0214(+)	2.3256(+)	1.2411(+)	3.4162(+)
		Ada	15.2875	26.3854	8.3048	14.3189	18.3751	14.3215	15.3654
		Runtime	972.3728	1202.3512	1096.3676	1292.6271	1301.2716	992.1822	962.7366
	DynDE	Average	0.0792(+)	2.5813(+)	5.5624(+)	0.1186(+)	1.4674(+)	1.6761(+)	4.3809(+)
		Ada	18.9748	26.3517	15.4168	18.2148	18.4804	19.3047	25.3405
		Runtime	986.3627	1008.6351	1020.6353	1437.2873	1302.9787	1128.6737	909.3916
	dopt-aiNET	Average	0.1721(+)	4.5714(+)	5.0291(+)	6.3438(+)	5.8372(+)	10.2713(+)	3.9298 (+)
		Ada	19.3487	30.3150	31.9102	30.9294	27.5648	30.8912	23.5948
		Runtime	1129.3671	1198.3627	1203.5362	1300.6718	1387.9092	1189.3625	1072.9837
	CPSO	Average	0.0478(+)	2.4785(+)	4.6915(+)	0.0648(+)	1.3892(+)	1.3362(+)	3.7823(+)
		Ada	17.3654	29.3487	19.3578	17.3258	17.5684	20.3945	27.3589
		Runtime	1318.2713	1376.3619	1298.2617	1450.3526	1615.7632	1432.6378	1201.5362
	CESO	Average	0.0768 (+)	2.4536(+)	5.6436(+)	0.0975 (+)	1.6124 (+)	1.0342(+)	3.7903(+)
		Ada	15.3947	31.3680	25.0248	18.3697	19.3891	15.3594	28.0590
		Runtime	1152.6371	1162.5363	1024.6374	1043.5243	997.5361	1143.5209	993.0925
	PSO-CP	Average	0.0419(+)	2.7014(+)	4.6873(+)	0.0521(+)	1.5861(+)	1.5281(+)	3.9369(+)
		Ada	15.3018	28.2874	8.3599	15.9877	18.3515	21.0121	26.3897
		Runtime	1242.6358	1297.5358	1182.4426	1393.6259	1497.5361	1143.5209	1193.0925
F1 (No. of peaks= 50)	DDEBQ	Mean Error	**6.44E-09**	**5.53E-08**	**0.8546**	**7.53E-09**	**2.64E-06**	**2.94E-06**	**0.5083**
		Ada	**13.0701**	**21.8257**	**9.5465**	**5.5604**	**5.1634**	**10.148**	**14.536**
		Runtime	1564.2252	1632.2544	1506.3034	1501.2597	1702.1570	1584.1892	1496.1785
	DASA	Mean Error	0.3862(+)	3.9034(+)	7.3182(+)	0.3165(+)	1.1026(+)	3.0105(+)	6.3349(+)
		Ada	20.1547	25.1466	25.2587	22.6846	14.1986	21.5958	23.1587
		Runtime	1382.6473	1592.3672	1516.7469	1414.4637	1763.7482	1452.7361	1413.5463
	jDE	Mean Error	0.1958(+)	4.1662(+)	4.4473(+)	0.0957(+)	1.0022(+)	1.8836(+)	4.3892(+)
		Ada	18.2654	28.3689	17.6984	21.3691	13.2501	18.7458	25.6841
		Runtime	1423.6354	1588.5609	1502.5434	1478.5408	1683.0963	1503.9805	1532.8625
	DynDE	Mean Error	0.3352(+)	4.7542(+)	6.1538(+)	0.2108(+)	1.1036(+)	0.9372(+)	6.8359(+)
		Ada	24.2587	29.5478	25.2845	25.9547	10.4801	15.5864	28.6541
		Runtime	1402.5421	1574.6523	1507.4324	1490.5879	1628.7013	1597.1109	1489.4428
	dopt-aiNET	Mean Error	0.4535(+)	4.1429(+)	4.9847(+)	3.0872(+)	3.2754(+)	5.9397(+)	4.1208(+)
		Ada	25.3521	25.6584	26.8745	30.9784	28.6597	23.9548	30.2897
		Runtime	1582.6539	1662.5476	1597.7879	1513.4352	1709.4352	1529.5409	1485.7839
	CPSO	Mean Error	0.3738(+)	4.1642(+)	4.8370(+)	0.3129(+)	2.0927(+)	1.6323(+)	3.7722(+)
		Ada	26.3257	31.4547	29.3547	24.1875	17.6520	18.3951	23.5487
		Runtime	1692.4767	1773.9283	1798.5608	1643.3241	1812.4352	1779.6376	1798.2314
	CESO	Mean Error	0.4536 (+)	4.7367 (+)	6.8375 (+)	0.1763 (+)	1.0372 (+)	0.9257 (+)	5.8693 (+)
		Ada	26.8107	28.6742	29.2145	22.3648	13.6987	18.6008	26.9016
		Runtime	1482.3524	1598.8843	1562.452	1530.2921	1622.4627	1601.7381	1417.9086
	PSO-CP	Mean Error	0.9456(+)	3.4686(+)	3.9523(+)	0.1077(+)	1.1282(+)	0.9699(+)	5.4468(+)
		Ada	19.1267	29.6512	17.9856	25.6847	21.6845	22.3245	35.8945
		Runtime	1584.7526	1634.7782	1729.3781	1672.3679	1642.9681	1487.6479	1476.2898
F2	DDEBQ	Mean Error	**0.6679**	**4.4666**	**3.7863**	**0.3042**	**11.33**	**0.7194**	**0.8355**
		Ada	**45.5085**	**44.1915**	**38.5298**	**55.8153**	**46.7502**	**39.8567**	**25.701**
		Runtime	2156.8751	2155.8693	2079.3692	2154.2303	1888.0309	1400.4820	1859.1188
	DASA	Mean Error	2.6794(+)	23.8215(+)	16.8836(+)	1.7829(+)	51.7035(+)	2.2710(+)	3.1327(+)
		Ada	74.5879	121.6487	94.9658	76.5897	133.9871	68.0459	45.3871
		Runtime	2083.6721	2018.7731	1966.0392	2081.2209	1786.9807	1346.3425	1812.7462
	jDE	Mean Error	0.9873(+)	42.1125(+)	52.2487(+)	0.7659(+)	65.5274(+)	3.4558(+)	12.8369(+)
		Ada	50.4792	167.3025	129.3547	58.9784	128.3674	47.9871	68.9987
		Runtime	1802.5362	2108.5463	1983.6558	1949.4342	1793.2271	1318.4315	1793.6574
	DynDE	Mean Error	1.3627 (+)	14.6783 (+)	12.7537 (+)	0.7928 (+)	21.5784 (+)	2.6940 (+)	2.9836 (+)
		Ada	67.1587	78.3525	84.1279	60.3148	78.3247	55.3158	59.3974
		Runtime	1883.6525	1942.6573	1993.5463	1934.6578	1719.4355	1216.7684	1707.6572
	dopt-aiNET	Mean Error	0.837(+)	10.7653(+)	20.1653(+)	1.8582(+)	98.6526(+)	5.9871(+)	4.1873(+)
		Ada	47.3645	95.7654	98.3574	79.3654	137.9614	69.2179	63.3489
		Runtime	1567.7683	1902.3629	1846.9705	1673.5908	1675.0692	1458.5462	1892.5462
	CPSO	Mean Error	1.1728(+)	12.4367(+)	8.5362(+)	0.6046(+)	23.6722(+)	1.6903(+)	4.1226(+)
		Ada	71.3057	90.5657	48.6587	67.4978	79.3654	48.4792	50.7456
		Runtime	2295.6471	2285.6832	2179.5431	2298.9428	1903.0461	1679.5162	2037.4432
	CESO	Mean Error	1.4672 (+)	13.4575 (+)	12.6472(+)	0.7637(+)	23.6502(+)	2.1636(+)	2.4782(+)
		Ada	78.6512	101.2544	88.2401	70.9845	78.8742	58.6848	67.6984
		Runtime	2198.3728	2102.6453	2118.5678	2092.1928	1963.8796	1216.2948	1608.3871
	PSO-CP	Mean Error	0.9563(+)	9.2044(+)	10.2548(+)	4.4855(+)	19.4729(+)	1.8689(+)	3.9687(+)
		Ada	56.7456	62.1456	56.6412	74.1258	78.5718	50.3248	69.4578
		Runtime	2196.4893	2263.0951	2219.9172	2275.1817	1881.7683	1517.3827	1732.2472
F3	DDEBQ	Mean Error	**12.7332**	750.1274	**568.0162**	96.1268	**487.2735**	339.1475	**151.9471**
		Ada	**93.655**	987.5773	**670.2121**	344.1393	**622.0945**	499.876	**258.3248**
		Runtime	2061.5036	2068.3955	2068.0575	1930.9352	2381.8595	1510.0609	2674.7382
	DASA	Mean Error	16.3416(+)	809.4177(+)	676.1128(+)	443.3980(+)	701.1427(+)	618.4158(+)	426.0946(+)
		Ada	110.4169	994.1596	898.3154	569.4279	880.6501	752.3594	674.3548
		Runtime	1983.6571	2094.8724	1928.8942	1921.7683	2285.7382	1562.7684	2669.4971
	jDE	Mean Error	13.6735(≈)	**563.7835(–)**	582.6456(≈)	**68.2357 (–)**	491.2671(≈)	**240.4814(–)**	162.2456(≈)
		Ada	95.4894	**693.4479**	701.2874	**208.7854**	650.0107	**398.4297**	264.3028
		Runtime	1952.6472	1934.4574	1995.1473	1925.4631	2377.0946	1608.9784	2693.1837
	DynDE	Mean Error	21.8460(+)	794.4378(+)	643.4592(+)	336.5654(+)	763.8548(+)	512.4632(+)	424.9346(+)
		Ada	115.9474	1021.8971	770.8324	571.3698	1134.3571	712.5489	689.3801
		Runtime	2018.5342	2108.6452	2002.6657	1984.0892	2399.7613	1528.5461	2613.6524
	dopt-aiNET	Mean Error	764.897(+)	1011.762(+)	1002.653(+)	1103.675(+)	1035.798(+)	1291.357(+)	1108.432(+)
		Ada	990.6510	1230.3158	1105.3597	1251.3694	1125.3694	1390.3548	1298.3954
		Runtime	2139.7683	2105.7980	2098.4814	1981.6074	2405.7684	1533.2108	2635.0794
	CPSO	Mean Error	124.575(+)	803.6472(+)	784.5447(+)	455.7382(+)	836.4890(+)	769.7464(+)	596.7684(+)
		Ada	190.3695	1047.3598	984.3981	691.3847	1250.6980	996.6584	975.6848
		Runtime	2203.7583	2296.9382	2275.9705	2103.1752	2506.4631	1802.7584	2846.9956
	CESO	Mean Error	20.546 (+)	795.7266(+)	646.0372(+)	350.6273(+)	752.7006(+)	505.872(+)	420.657(+)
		Ada	130.3258	1160.3486	805.6981	584.3641	892.1764	743.0591	678.0486
		Runtime	1894.7583	2104.7682	2092.9887	1975.2281	2319.5462	1603.7584	2609.8322
	PSO-CP	Mean Error	15.7312(+)	774.1627(+)	625.3457(+)	319.5279(+)	736.5267(+)	501.1201(+)	401.9820(+)
		Ada	115.3058	1067.2587	894.1459	597.3249	914.2574	786.6581	654.2589
		Runtime	2164.8824	2198.5362	2137.4623	2004.6302	2435.7582	1726.8320	2764.8391

TABLE III

MEAN ERROR VALUES AND STANDARD DEVIATIONS ACHIEVED BY DDEBQ AND OTHER ALGORITHMS FOR TEST FUNCTIONS F4-F6 OF GDBG SYSTEM. WILCOXON'S RANK SUM TEST RESULTS OF COMPARING DDEBQ WITH CONTENDER ALGORITHMS INDICATED IN PARENTHESES.

Func.	Algorithm	Performance Measures	T1	T2	T3	T4	T5	T6	T7
F4	DDEBQ	Mean Error	**0.8611**	**18.7268**	**13.2794**	**0.7061**	**31.1387**	**1.3693**	6.6937
		Ada	**22.1463**	**30.0743**	**27.6785**	**36.4655**	**31.878**	**44.5907**	64.8446
		Runtime	2151.2107	2279.6941	2372.4617	2054.7873	1882.2359	1544.6890	1881.1184
	DASA	Mean Error	4.5287(+)	67.2895(+)	49.3902(+)	2.9127(+)	115.4713(+)	3.1026(+)	28.2715(+)
		Ada	59.3458	97.6540	73.9748	49.3517	169.1578	58.1950	87.1258
		Runtime	2089.6473	2281.8321	2365.7462	2001.4233	1902.0573	1527.7482	1798.2834
	jDE	Mean Error	1.6398(+)	54.8754(+)	46.9674(+)	1.7583(+)	67.7002(+)	3.0376(+)	14.3352(+)
		Ada	27.9072	67.5412	45.3186	38.1057	88.3501	56.9184	79.3254
		Runtime	2188.9322	2241.8742	2493.1728	2192.7421	1784.6623	1619.9783	1784.5532
	DynDE	Mean Error	1.8846(+)	41.6474(+)	24.8453(+)	0.8836(+)	46.4667(+)	1.7637(+)	6.5458(≈)
		Ada	34.3657	67.0158	36.2541	43.1489	91.2548	54.8127	70.3584
		Runtime	2098.6752	2253.8103	2463.2516	2098.4627	1973.6271	1523.7381	1798.5361
	dopt-aiNET	Mean Error	1.0928(+)	134.8593(+)	86.8337(+)	5.7732(+)	300.7371(+)	15.8747(+)	55.9398(+)
		Ada	32.1954	215.9817	158.1974	59.5674	412.8219	79.1843	98.0176
		Runtime	2131.9885	2261.1332	2317.9283	2096.6321	1914.9702	1519.7263	1982.8273
	CPSO	Mean Error	2.8718(+)	34.7409(+)	32.6271(+)	0.7846(+)	61.4873(+)	5.2781(+)	10.7583(+)
		Ada	40.1807	67.1598	77.5489	47.3259	99.0187	60.1578	88.2549
		Runtime	2539.6174	2563.7721	2817.4627	2209.7583	2210.7583	1835.0694	2218.9972
	CESO	Mean Error	1.9124(+)	30.0192(+)	24.8673(+)	0.8639 (+)	48.6454 (+)	1.5624(≈)	6.9270 (+)
		Ada	40.6508	65.1987	90.4588	50.6478	78.6879	59.3514	78.3597
		Runtime	2019.7647	2281.6529	2209.1728	2100.6512	1902.6473	1811.0293	1982.9301
	PSO-CP	Mean Error	1.9743(+)	30.5689(+)	21.6393(+)	0.9958(+)	41.3289(+)	1.7541(+)	**5.4312(–)**
		Ada	43.3548	60.6897	87.4598	55.3154	78.3471	65.7418	**56.9748**
		Runtime	2451.8291	2372.6374	2618.8787	2198.4352	2159.5362	1794.7382	2093.827
F5	DDEBQ	Mean Error	**0.0611**	**0.1627**	**0.2982**	**0.0319**	**0.0927**	**0.0398**	**0.1132**
		Ada	**59.8381**	**39.0564**	**17.2733**	**79.1614**	**34.5024**	**43.6089**	**36.2352**
		Runtime	1911.6563	2136.6694	1975.5724	2514.2102	1893.2672	1888.1726	1856.3941
	DASA	Mean Error	1.1026(+)	1.0049(+)	1.1358(+)	0.3722(+)	2.4679(+)	0.4129(+)	1.1714(+)
		Ada	68.3254	45.9874	48.6589	88.2145	39.5478	47.3598	40.2874
		Runtime	1862.6342	2089.8273	1882.7921	2327.4891	1873.8472	1772.9029	1828.2371
	jDE	Mean Error	0.1602(+)	0.3251(+)	0.3628(+)	0.1057(+)	0.4637(+)	0.3260(+)	0.4481(+)
		Ada	59.9148	45.5841	47.2956	82.3647	45.9774	46.6587	38.2547
		Runtime	1782.7483	2097.5463	1927.8329	2496.8071	1673.8837	1792.5614	1802.0932
	DynDE	Mean Error	3.1726(+)	3.2748(+)	2.8562(+)	1.7611(+)	9.3453(+)	2.5837(+)	3.6537(+)
		Ada	68.6912	72.5894	54.3579	88.6512	67.9801	74.6421	487.5482
		Runtime	1873.1172	2109.8678	1982.6635	2448.4231	1857.0291	1903.8672	1811.7348
	dopt-aiNET	Mean Error	35.8392(+)	36.7382(+)	30.8875(+)	116.5646(+)	1008.647(+)	456.8390(+)	234.546(+)
		Ada	101.6579	104.3698	61.2743	215.2358	1254.2064	547.3594	335.8756
		Runtime	1925.8795	2172.6657	1962.8364	2674.9847	1794.9801	1974.8372	1904.5412
	CPSO	Mean Error	2.1647(+)	2.6460(+)	4.0372(+)	1.1218(+)	8.1649(+)	5.7387(+)	5.8913(+)
		Ada	79.3547	76.2549	87.6548	94.3560	91.2548	88.6107	61.2401
		Runtime	2385.7002	2451.6273	2097.5361	2766.4109	2279.1982	2107.4351	2173.0928
	CESO	Mean Error	2.9397(+)	2.5724(+)	3.2617(+)	1.6243 (+)	8.5453 (+)	2.1690(+)	3.7732 (+)
		Ada	88.6510	79.6597	89.5647	97.5427	99.6512	85.1954	94.5419
		Runtime	2093.6192	2198.1253	1997.4253	2609.5342	1992.5042	1922.9381	1923.5193
	PSO-CP	Mean Error	1.9730(+)	2.8051(+)	2.8879(+)	0.9068(+)	7.3517(+)	2.2074(+)	3.4678(+)
		Ada	79.6514	81.6548	59.3549	89.6149	94.2195	87.6497	98.2569
		Runtime	2308.4312	2271.4122	1991.4152	2533.9348	2160.4192	1853.4421	2082.6370
F6	DDEBQ	Mean Error	**5.3914**	**9.5653**	**10.2539**	**5.7937**	**13.9586**	**7.1033**	**8.0366**
		Ada	**56.2489**	**52.2535**	49.0431	91.7976	**69.7187**	**51.0232**	**41.9599**
		Runtime	1971.8619	1765.8937	2051.5664	1991.7595	1413.4759	1632.9108	1584.4324
	DASA	Mean Error	9.1137(+)	35.5782(+)	28.3789(+)	8.9961(+)	39.4678(+)	11.6279(+)	13.0281(+)
		Ada	85.3412	142.2154	88.2541	93.3755	128.3845	75.7619	71.2149
		Runtime	1874.8237	1726.5362	1896.1341	1874.7265	1386.5413	1611.0291	1539.4352
	jDE	Mean Error	7.3728(+)	13.7403 (≈)	12.5831 (≈)	8.8376 (+)	14.2024 (+)	8.1184 (≈)	9.8471(+)
		Ada	57.2168	62.5154	**47.3121**	**90.3548**	75.9467	60.1560	45.2579
		Runtime	1912.4350	1673.8928	2016.4351	1983.0921	1372.0937	1592.5091	1471.3094
	DynDE	Mean Error	7.2464(+)	22.5635(+)	20.8464(+)	9.2635(+)	41.6367(+)	13.6379(+)	17.2167(+)
		Ada	74.3878	71.2648	81.4589	106.2196	156.3894	60.3248	51.2649
		Runtime	1987.5463	1721.1362	2087.5110	2008.7192	1389.0212	1618.4251	1452.8827
	dopt-aiNET	Mean Error	23.6572(+)	347.6759(+)	428.6572(+)	86.7548(+)	873.6868(+)	526.7384(+)	361.6976(+)
		Ada	100.1252	480.3266	554.5946	255.2648	1215.4631	945.6548	454.5954
		Runtime	2019.1112	1982.5163	2047.4352	1876.1523	1519.4352	1656.9281	1499.4312
	CPSO	Mean Error	7.9309(+)	24.7235(+)	28.6496(+)	7.5365(+)	67.8734(+)	26.8274(+)	34.7615(+)
		Ada	62.5124	114.7594	73.5481	98.3599	139.2453	89.6521	82.3193
		Runtime	2277.5361	2081.8291	2309.5361	2471.5261	1692.4132	1922.4215	1724.7612
	CESO	Mean Error	5.9357(≈)	24.3468(+)	21.7559(+)	8.2136(+)	42.8745(+)	12.8678(+)	14.5743(+)
		Ada	62.5293	116.2519	131.0301	103.6249	120.2597	96.7821	80.3259
		Runtime	1971.4090	1774.3002	2109.1162	1982.4413	1438.5463	1720.1361	1613.0951
	PSO-CP	Mean Error	5.9931(≈)	15.1217(+)	16.6627(+)	7.5235(+)	40.2989(+)	8.1346(+)	8.7361(+)
		Ada	66.9878	79.6412	67.8945	95.3549	115.6982	64.2519	52.2147
		Runtime	2210.7821	1913.4531	2189.1351	1999.4412	1510.5461	1879.5091	1773.5131

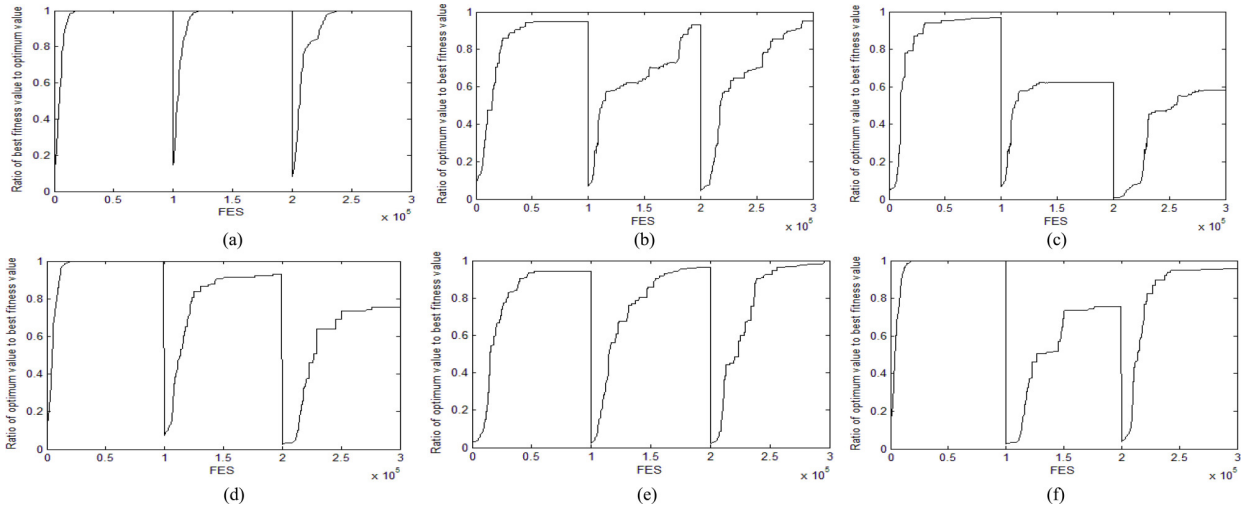


Fig. 1. Sample convergence graphs for DDEBQ algorithm. (a) For F1 with T7. (b) For F2 with T7. (c) For F3 with T7. (d) For F4 with T7. (e) For F5 with T7. (f) For F6 with T7.

types of F3. For all the change types of the functions F1, F2, F4, and F5, and for change types T1, T4, T5, T7 of F6, DDEBQ yielded statistically superior performance to jDE, which was the winner of 2009 IEEE CEC Competition on Evolutionary Dynamic Optimizers.

DDEBQ performed statistically better than CPSO in all test cases. There are two test instances where DDEBQ performed statistically similar to CESO (F4 with T6, F6 with T1). DDEBQ performed worse than PSO-CP and comparable to DynDE in only one instance: F4 with change type T7. In 43 out of the 49 tested instances, DDEBQ achieved the lowest values of the adaptability metric. This indicates that for majority of the tested instances the best individual in the DDEBQ population remained closer the optimum for all generations, i.e., the optimum was better tracked by the proposed algorithm. For function F3 with change types T2, T4, and T6, jDE yielded the best adaptability metric values while DDEBQ attained the second best values. For function F6 with change types T3 and T4, despite yielding the lowest mean errors, DDEBQ was marginally surpassed by jDE in terms of the adaptability values. Note that for the instances where DDEBQ is statistically outperformed by any one of the seven contender algorithms, it ranked second best outperforming the other six algorithms. No other evolutionary DOP solver considered in this article could keep such a consistent performance on the wide variety of the tested DOP instances. As the double mutation strategy prevents the population from converging too quickly and the aging mechanism helps the population to get rid of local optima, DDEBQ is able to perform very well over such highly complex and multimodal functions. Extremely good performances over the sphere function (F2), the Ackley's function (F5), and the composition function (F6) have resulted from the incorporation of the dynamic DE scheme and exclusion principle. As the dynamic DE scheme maintains a good diversity level of the population, DDEBQ is able to locate the global optimum after any dynamic change more efficiently than other algorithms. The exclusion rule also helps the algorithm to explore much greater portion of the search space—a feature that leads to high success rate in locating the global optimum. Also, a high degree of precision in locating the global optimum observed in rotation peak function (F1 with number of peaks = 10, 50) is a consequence of introducing the control parameter C that has an important role in controlling the diversity of the population and adaptively changing the radius within which the quantum individuals are to be generated. From the average runtimes listed in Tables II and III, it is evident that the runtime of DDEBQ is in several cases comparable to DyneDE, dopt-aiNet, and CESO. However, CPSO takes higher average runtime on most of the functions due to the incurrence of several Euclidean distance calculations. PSO-CP also involves various computational overheads and in general is slower or comparable to DDEBQ in majority of the cases. DASA and jDE appear to be marginally faster than DDEBQ. However, when the accuracy appears to be the major bottleneck, DDEBQ has several advantages to offer.

VI. Conclusion

In this paper, a variant of the DE algorithm referred to as DDEBQ is proposed to solve DOPs in a statistically efficient manner. The proposed algorithm uses a dynamic DE scheme that obviously shares the traditional DE framework. In addition to DE individuals, it uses adaptive quantum and Brownian individuals to increase the diversity and exploration ability of the search process. A control parameter is introduced to control the diversity as necessary. The algorithm also employs an aging mechanism to get rid of stagnation. The DE individuals produce the donor vectors according to a neighborhood-based double mutation strategy to control the perturbation. An exclusion scheme is used so that the subpopulations become evenly distributed over the entire search space.

The statistical summary of the simulation results indicates that DDEBQ can provide consistently superior performance as compared to the other state-of-the-art evolutionary dynamic

optimizers in terms of average level of accuracy. Future work may focus on introducing more co-operation and information exchange among the subpopulations in DDEBQ. It can also be fruitful to make the crossover probability adaptive to the condition of the fitness landscape. Algorithmic components of DDEBQ can be integrated with some of the adaptive DE variants ([37], [38]) to improve their performance on dynamic landscapes as well.

APPENDIX

TABLE IV
LIST OF TERMINOLOGY USED IN DDEBQ

Age_ best	A matrix that keeps track of the number of times a particular individual within a subpopulation is consecutively being the best member. This can subsequently be a check for stagnation and helps aging process.
Age_worst	A matrix that keeps track of the number of times a particular individual within a subpopulation is consecutively being the worst member. This can effectively verify whether a subpopulation is stagnant and guides aging mechanism.
Aging Mechanism	Mechanism that prevents stagnation of individuals at local optima.
Brownian individuals	Individuals that follow Brownian motion and act like local explorers within subpopulations.
Control Parameter	The control parameter C identifies the different stages of optimization that helps the search process by maintaining diversity within the subpopulation. Depending on the conditions, this parameter can take any value among 0, 1, and 2.
Double Mutation	A modified mutation scheme that involves a neighborhood scheme followed by a linear combination of mutant vector and local best solutions.
Dis Marginal	The marginal distance between two subpopulations less than which the subpopulations are considered to be searching the same locality.
Exclusion Rule	Rule to prevent subpopulations concentrate their search to same locations in the search space and diversify their search to different optima.
Locally best individuals	The best solutions of each subpopulation are termed locally best individuals.
Quantum Individuals	Individuals with probabilistically determined positions around local best explore the neighborhood of local optima while maintaining diversity.
Repairing Rule	A rule that confines the population members within bounds, as specified by the problem.
Update Interval (*UI*)	The fixed number of generations after which the change in the global fitness is monitored for adapting the control parameter C
Weight factor	Used in the second stage of double mutation strategy to form a linear combination of the mutant vector from the 1st stage of mutation to the local best vector.
Search Range (*SR*)	The bounds as defined by the algorithm within which the algorithmic search process operates.
Subpopulations	The whole population is partitioned into smaller groups called subpopulations.

REFERENCES

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

[2] 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.

[3] 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.

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

[5] R. Allmendinger and J. Knowles "On-line purchasing strategies for an evolutionary algorithm performing resource-constrained optimization," in *Proc. PPSN XI*, vol. II, LNCS 6239. 2010, pp. 161–170.

[6] J. Lampinen and I. Zelinka, "On stagnation of the differential evolution algorithm," in *Proc. 6th Int. Mendel Conf. Soft Comput.*, Jun. 2000, pp. 76–83.

[7] 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., Tech. Rep., Sep. 2008.

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

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

[10] 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.

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

[12] M. C. du Plessis and A. P. Engelbrecht, "Using competitive population evaluation in a differential evolution algorithm for dynamic environments," *Eur. J. Oper. Res.*, vol. 218, no. 1, pp. 7–20, Apr. 2012.

[13] V. Noroozi, A. B. Hashemi, and M. R. Meybodi, "CellularDE: A cellular based differential evolution for dynamic optimization problems," in *Proc. ICANNGA*, part I, LNCS 6593. 2011, pp. 340–349.

[14] U. Halder, S. Das, and D. Maity, "A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments," *IEEE Trans. Cybern.*, vol. 43, no. 3, pp. 881–897, Jun. 2013.

[15] S. Das, S. Maity, B-Y Qu, and P. N. Suganthan, "Real-parameter evolutionary multimodal optimization: A survey of the state-of-the-art," *Swarm Evol. Comput.*, vol. 1, no. 2, pp. 71–88, Jun. 2011.

[16] A. Basak, S. Das, and K. C. Tan, "Multimodal optimization using a bi-objective differential evolution algorithm enhanced with mean distance based selection," *IEEE Trans. Evol. Comput.,* vol. PP, no. 99, p. 1, Dec. 2012.

[17] R. Thomsen, "Multimodal optimization using crowding-based differential evolution," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2004, pp. 1382–1389.

[18] K.-C. Wong, C.-H. Wu, R. K. P. Mok, C. Peng, and Z. Zhang, "Evolutionary multimodal optimization using the principle of locality," *Inf. Sci*., vol. 194, pp. 138–170, Jul. 2012.

[19] 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.

[20] X. Li, J. Branke, and T. Blackwell, "Particle swarm with speciation and adaptation in a dynamic environment," in *Proc. GECCO*, 2006, pp. 51–58.

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

[22] R. Lung and D. Dumitrescu, "Evolutionary swarm cooperative optimization in dynamic environments," *Natural Comput*., vol. 9, no. 1, pp. 83–94, Mar. 2010.

[23] K.-C. Wong, K.-S. Leung, and M.-H. Wong, "An evolutionary algorithm with species-specific explosion for multimodal optimization," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2009, pp. 923–930.

[24] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, "Differential evolution using a neighborhood based mutation operator," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 526–553, Jun. 2009.

[25] D. Zaharie, "Critical values for the control parameters of differential evolution algorithms," in *Proc. 8th Int. Mendel Conf. Soft Comput.*, 2002, pp. 62–67.

[26] T. M. Blackwell, "Particle swarm optimization in dynamic environments," in *Evolutionary Computation in Dynamic and Uncertain Environments*, S. Yang, Y. S. Ong, and Y. Jin Eds. Berlin, Germany: Springer-Verlag, 2007, ch. 1, pp. 29–49.

[27] L. Ingber, "Adaptive simulated annealing (ASA): Lessons learned," *Control Cybern.*, vol. 25, no. 1, pp. 33–54, 1996.

[28] H. E. Romeijn and R. L. Smith, "Simulated annealing for constrained global optimization," *J. Global Optimization*, vol. 5, no. 2, pp. 101–126, Sep. 1994.

[29] 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.

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

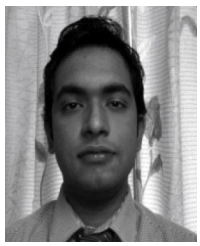
[31] K. Trojanowski and Z. Michalewicz, "Searching for optima in nonstationary environments," in *Proc. IEEE Congr. Evol. Comput.*, 1999, pp. 1843–1850.
[32] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, Mar. 2011.
[33] J. Brest, P. Korošec, J. Šilc, A. Zamuda, B. Bošković, and M. Sepesy Maučec, "Differential evolution and differential antstigmergy on dynamic optimisation problems," *Int. J. Syst. Sci.*, vol. 44, no. 4, pp. 663–679, 2013.
[34] 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.
[35] 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.
[36] L. N. de Castro and J. Timmis, "An artificial immune network for multimodal optimization," in *Proc. Congr. Evol. Comput. Part IEEE World Congr. Comput. Intell.*, May 2002, pp. 699–704.
[37] S. M. 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*, vol. 42, no. 2, pp. 482–500, Apr. 2012.
[38] W. Gong, Z. Cai, C. X. Ling, and Hui Li, "Enhanced differential evolution with adaptive strategies for numerical optimization," *IEEE Trans. Syst., Man, Cybern. B*, vol. 41, no. 2, pp. 397–413, Apr. 2011.

Swagatam Das (M'10–SM'12) is currently an Assistant Professor with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, India. He has published one research monograph, one edited volume, and over 150 research articles in peer-reviewed journals and international conferences. His current research interests include evolutionary computing and pattern recognition.

Mr. Das is the Founding Co-Editor-in-Chief of *Swarm and Evolutionary Computation*, an international journal from Elsevier. He also serves as an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, *Neurocomputing*, *Information Sciences*, and *Engineering Applications of Artificial Intelligence*. He is an Editorial Board member of *Progress in Artificial Intelligence* (Springer), *Mathematical Problems in Engineering*, *International Journal of Artificial Intelligence and Soft Computing*, and *International Journal of Adaptive and Autonomous Communication Systems*. He has been associated with international program committees and organizing committees of several regular international conferences, including IEEE WCCI, IEEE SSCI, SEAL, GECCO, and SEMCCO. He has acted as a Guest Editor for special issues in journals, such as the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the *ACM Transactions on Adaptive and Autonomous Systems*, and the IEEE TRANSACTIONS ON SYSTEM, MAN, AND CYBERNETICS, PART C. He was a recipient of the 2012 Young Engineer Award from the Indian National Academy of Engineering.

Ankush Mandal received the B.E. degree in electronics and telecommunication engineering from Jadavpur University, Kolkata, India, in 2012.

He is currently working as a Control and Instrumentation Engineer at the Engineering and Planning Department of the Damodar Valley Corporation, West Bengal, India. His current research interests include evolutionary optimization in nonstationary environments and evolutionary design of antennas.



Rohan Mukherjee was born in West Bengal, India, in 1992. He is currently pursuing the B.E. degree in electronics and telecommunication engineering at Jadavpur University, Kolkata, India.

He has published research articles in peer-reviewed journals and international conference proceedings under the guidance of his teacher Dr. Swagatam Das. He has acted as a reviewer for international conferences. His current research interests include smart grids, game theoretic applications, power systems, wireless communications, and evolutionary computing.