Differential Evolution Using a Neighborhood-Based Mutation Operator

Swagatam Das, Ajith Abraham, *Senior Member, IEEE*, Uday K. Chakraborty, and Amit Konar, *Member, IEEE*

***Abstract*— Differential evolution (DE) is well known as a simple and efficient scheme for global optimization over continuous spaces. It has reportedly outperformed a few evolutionary algorithms (EAs) and other search heuristics like the particle swarm optimization (PSO) when tested over both benchmark and real-world problems. DE, however, is not completely free from the problems of slow and/or premature convergence. This paper describes a family of improved variants of the DE/target-to-best/1/bin scheme, which utilizes the concept of the neighborhood of each population member. The idea of small neighborhoods, defined over the index-graph of parameter vectors, draws inspiration from the community of the PSO algorithms. The proposed schemes balance the exploration and exploitation abilities of DE without imposing serious additional burdens in terms of function evaluations. They are shown to be statistically significantly better than or at least comparable to several existing DE variants as well as a few other significant evolutionary computing techniques over a test suite of 24 benchmark functions. The paper also investigates the applications of the new DE variants to two real-life problems concerning parameter estimation for frequency modulated sound waves and spread spectrum radar poly-phase code design.**

***Index Terms*— Differential evolution, evolutionary algorithms, meta-heuristics, numerical optimization, particle swarm optimization.**

I. Introduction

DIFFERENTIAL EVOLUTION (DE), proposed by Storn and Price [1]–[3], is a simple yet powerful algorithm for real parameter optimization. Recently, the DE algorithm has become quite popular in the machine intelligence and cybernetics communities. It has successfully been applied to diverse domains of science and engineering, such as mechanical engineering design [4], [5], signal processing [6], chemical engineering [7], [8], machine intelligence, and pattern recognition [9], [10]. It has been shown to perform better than the genetic algorithm (GA) [11] or the particle swarm optimization (PSO) [12] over several numerical benchmarks [13]. Many of the most recent developments in DE algorithm design and applications can be found in [14]. Like other evolutionary algorithms, two fundamental processes drive the evolution of a DE population: the *variation* process, which enables exploring different regions of the search space, and the *selection* process, which ensures the exploitation of previous knowledge about the fitness landscape.

Practical experience, however, shows 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 [15]. Occasionally, even new individuals may enter the population, but the algorithm does not progress by finding any better solutions. This situation is usually referred to as *stagnation*. DE also suffers from the problem of premature convergence, where the population converges to some local optima of a multimodal objective function, losing its diversity. The probability of stagnation depends on how many different potential trial solutions are available and also on their capability to enter into the population of the subsequent generations [15]. Like other evolutionary computing algorithms, the performance of DE deteriorates with the growth of the dimensionality of the search space as well. There exists a good volume of works (a review of which can be found in Section III), attempting to improve the convergence speed and robustness (ability to produce similar results over repeated runs) of DE by tuning the parameters like population size NP, the scale factor F, and the crossover rate Cr.

In the present work, we propose a family of variants of the DE/target-to-best/1 scheme [3, p.140], which was also referred to as "Scheme DE2" in the first technical paper on DE [1]. In some DE literature this algorithm is referred to as DE/current-to-best/1 [16], [17]. To combine the exploration and exploitation capabilities of DE, we propose a new hybrid mutation scheme that utilizes an explorative and an exploitive mutation operator, with an objective of balancing their effects. The explorative mutation operator (referred to as the *local* mutation model) has a greater possibility of locating the minima of the objective function, but generally needs more iterations (generations). On the other hand, the exploitative mutation operator (called by us the *global* mutation model) rapidly converges to a minimum of the objective function. In this case there exists the danger of premature convergence to a suboptimal solution. In the hybrid model we linearly combine the two mutation operators using a new parameter, called the weight factor. Four different schemes have been proposed and investigated for adjusting the weight factor, with a view to alleviating user intervention and hand tuning as much as possible.

Manuscript received November 6, 2007; revised March 23, 2008 and July 11, 2008; accepted September 2, 2008. Current version published June 10, 2009.

S. Das and A. Konar are with the Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata 700032, India (e-mail: swagatamdas19@yahoo.co.in; konaramit@yahhoo.co.in).

A. Abraham is with the Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology, Trondheim, NO-7491, Norway and Machine Intelligence Research Labs (MIR Labs), USA (e-mail: ajith.abraham@ieee.org).

U. K. Chakraborty is with the Department of Math and Computer Science, University of Missouri, St. Louis, MO 63121 USA (e-mail: chakrabortyu@umsl.edu).

Digital Object Identifier 10.1109/TEVC.2008.2009457

Here we would like to mention that although a preliminary version of this paper appeared as a conference paper in [18], the present version has been considerably enhanced and it differs in many aspects from [18]. It critically examines the effects of the global and local neighborhoods on the performance of DE and explores a few different ways of tuning of the weight factor (see Section IV) used for unification of the neighborhood models. In addition, it compares the performance of the proposed approaches with several state-of-the-art DE variants as well as other evolutionary algorithms over a testbed of 24 well-known numerical benchmarks and one real-world optimization problem in contrast to [18], which uses only six benchmarks and provides limited comparison results.

The remainder of this paper is organized as follows. In Section II, we provide a brief outline of the DE family of algorithms. Section III provides a short survey of previous research on improving the performance of DE. Section IV introduces the proposed family of variants of the DE/target-to-best/1 algorithm. Experimental settings for the benchmarks and simulation strategies are explained in Section V. Results are presented and discussed in Section VI. Finally, conclusions are drawn in Section VII.

II. DE Algorithm

Like any other evolutionary algorithm, DE starts with a population of NP D-dimensional parameter vectors representing the candidate solutions. 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 may adopt the following notation for representing the ith vector of the population at the current generation as

$$\vec{X}_{i,G} = \left[x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, \ldots, x_{D,i,G}\right]. \quad (1)$$

For each parameter of the problem, there may be a certain range within which the value of the parameter should lie for better search results. The initial population (at $G = 0$) should cover the entire search space as much as possible by uniformly randomizing individuals within the search space constrained by the prescribed minimum and maximum bounds: $\vec{X}_{\min} = \{x_{1,\min}, x_{2,\min}, \ldots, x_{D,\min}\}$ and $\vec{X}_{\max} = \{x_{1,\max}, x_{2,\max}, \ldots, x_{D,\max}\}$. Hence we may initialize the jth component of the ith vector as

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

where $rand_{i,j}(0,1)$ is a uniformly distributed random number lying between 0 and 1 and is instantiated independently for each component of the i-th vector. The following steps are taken next: mutation, crossover, and selection (in that order), which are explained in the following subsections.

A. 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 and sometimes using arithmetic recombination too. It is the method of creating this donor vector that differentiates one DE scheme from another. Five most frequently referred strategies implemented in the public-domain DE codes for producing the donor vectors (available online at http://www.icsi.berkeley.edu/storn/code.html) are listed below

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

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

$$\text{"DE/target-to-best/1"}: \vec{V}_{i,G} = \vec{X}_{i,G} + F \cdot (\vec{X}_{best,G} - \vec{X}_{i,G}) + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}) \quad (5)$$

$$\text{"DE/best/2"}: \vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}) + F \cdot (\vec{X}_{r_3^i,G} - \vec{X}_{r_4^i,G}) \quad (6)$$

$$\text{"DE/rand/2"}: \vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}) + F \cdot (\vec{X}_{r_4^i,G} - \vec{X}_{r_5^i,G}). \quad (7)$$

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 once 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., lowest objective function value for a minimization problem) in the population at generation G. Note that some of the strategies for creating the donor vector may be mutated recombinants, for example, (5) listed above, basically mutates a two-vector recombinant: $\vec{X}_{i,G} + F \cdot (\vec{X}_{best,G} - \vec{X}_{i,G})$. 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, y is the number of difference vectors considered for perturbation of x, and z stands for the type of crossover being used (exp: exponential; bin: binomial). The following section discusses the crossover step in DE.

B. Crossover

To increase the potential diversity of the population, a crossover operation comes into play after generating the donor vector through mutation. The DE family of algorithms can use two kinds of crossover schemes—*exponential* and *binomial* [1]–[3]. The donor vector exchanges its components with the target vector $\vec{X}_{i,G}$ under this operation to form the *trial* vector $\vec{U}_{i,G} = \left[u_{1,i,G}, u_{2,i,G}, u_{3,i,G}, \ldots, u_{D,i,G}\right]$. In exponential crossover, we first choose an integer n randomly among the numbers $[1, D]$. This integer acts as a starting point in the target vector, from where the crossover or exchange of components with the donor vector starts. We also choose another integer L from the interval $[1, D]$. L denotes the number of components; the donor vector actually contributes

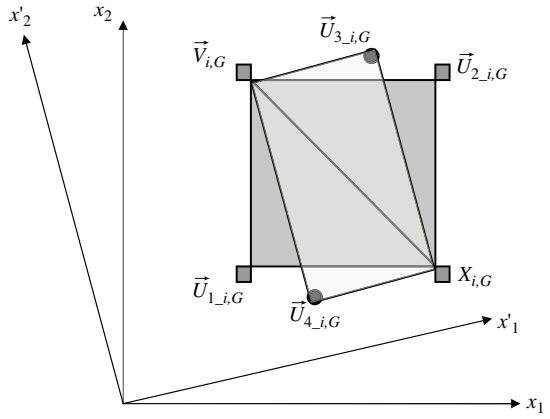


Fig. 1. Change of the trial vectors generated through the crossover operation described in (9) due to rotation of the coordinate system.

to the target. After a choice of n and L, the trial vector is obtained as

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{for } j = \langle n \rangle_D, \langle n+1 \rangle_D, \ldots, \langle n+L-1 \rangle_D \\ x_{j,i,G}, & \text{for all other } j \in [1, D] \end{cases} \tag{8}$$

where the angular brackets $\langle\ \rangle_D$ denote a modulo function with modulus D. The integer L is drawn from $[1, D]$ according to the following pseudo-code:

$L = 0$;
DO
{
$L = L + 1$;
} WHILE (((rand$(0, 1) < Cr$) AND $(L < D)$).

"Cr" is called the *crossover rate* and appears as a control parameter of DE just like F. Hence in effect, probability $(L \geq v) = (Cr)^{v-1}$ for any $v > 0$. For each donor vector, a new set of n and L must be chosen randomly as shown above.

On the other hand, binomial crossover is performed on each of the D variables whenever a randomly picked 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

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{if } (rand_{i,j}(0, 1) \leq Cr \text{ or } j = j_{rand}) \\ x_{j,i,G}, & \text{otherwise} \end{cases} \tag{9}$$

where $rand_{i,j}(0, 1) \in [0, 1]$ is a uniformly distributed random number, which is called a new 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 $\vec{V}_{i,G}$.

The crossover operation described in (9) is basically a discrete recombination [3]. Fig. 1 illustrates a two-dimensional example of recombining the parameters of two vectors $\vec{X}_{i,G}$ and $\vec{V}_{i,G}$, according to this crossover operator, where the potential trial vectors are generated at the corners of a rectangle. Note that $\vec{V}_{i,G}$ can itself be the trial vector (i.e., $\vec{U}_{i,G} = \vec{V}_{i,G}$) when $Cr = 1$. As can be seen from Fig. 1, discrete recombination is a rotationally variant operation. Rotation transforms the coordinates of both vectors and thus changes the shape of the rectangle as shown in Fig. 1. Consequently, the potential location of the trial vector moves from the possible set ($\vec{U}_{1\_i,G}$, $\vec{U}_{2\_i,G}$) to ($\vec{U}_{3\_i,G}$, $\vec{U}_{4\_i,G}$). To overcome this limitation, a new trial vector generation strategy "DE/current-to-rand/1" is proposed in [19], which replaces the crossover operator prescribed in (9) with the rotationally invariant arithmetic crossover operator to generate the trial vector $\vec{U}_{i,G}$ by linearly combining the target vector $\vec{X}_{i,G}$ and the corresponding donor vector $\vec{V}_{i,G}$ as follows:

$$\vec{U}_{i,G} = \vec{X}_{i,G} + K \cdot (\vec{V}_{i,G} - \vec{X}_{i,G}).$$

Now incorporating (3) in (10) we have

$$\vec{U}_{i,G} = \vec{X}_{i,G} + K \cdot (\vec{X}_{r_1,G} + F \cdot (\vec{X}_{r_2,G} - \vec{X}_{r_3,G}) - \vec{X}_{i,G})$$

which further simplifies to

$$\vec{U}_{i,G} = \vec{X}_{i,G} + K \cdot (\vec{X}_{r_1,G} - \vec{X}_{i,G}) + F^{/} \cdot (\vec{X}_{r_2,G} - \vec{X}_{r_3,G}) \tag{10}$$

where K is the combination coefficient, which has been shown [19] to be effective when it is chosen with a uniform random distribution from [0, 1] and $F^{/} = K \cdot F$ is a new constant here.

C. Selection

To keep the population size constant over subsequent generations, the next step of the algorithm calls for *selection* to determine whether the target or the trial vector survives to the next generation i.e., at $G = G + 1$. The selection operation is described as

$$\begin{aligned} \vec{X}_{i,G+1} &= \vec{U}_{i,G}, \quad \text{if } f(\vec{U}_{i,G}) \leq f(\vec{X}_{i,G}) \\ &= \vec{X}_{i,G}, \quad \text{if } f(\vec{U}_{i,G}) > f(\vec{X}_{i,G}) \end{aligned} \tag{11}$$

where $f(\vec{X})$ is the function to be minimized. So 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. Hence the population either gets better (with respect to the minimization of the objective function) or remains the same in fitness status, but never deteriorates. The complete pseudo-code of the DE is given below:

1) Pseudo-Code for the DE Algorithm Family:

Step 1. Set the generation number $G = 0$ and randomly initialize a population of NP individuals $P_G = \{\vec{X}_{1,G}, \ldots, \vec{X}_{NP,G}\}$ with $\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, \ldots, x_{D,i,G}]$ and each individual uniformly distributed in the range $[\vec{X}_{\min}, \vec{X}_{\max}]$, where $\vec{X}_{\min} = \{x_{1,\min}, x_{2,\min}, \ldots, x_{D,\min}\}$ and $\vec{X}_{\max} = \{x_{1,\max}, x_{2,\max}, \ldots, x_{D,\max}\}$ with $i = [1, 2, \ldots, NP]$.

Step 2. WHILE the stopping criterion is not satisfied
DO
FOR $i = 1$ to NP //do for each individual sequentially

Step 2.1 ***Mutation Step***

Generate a donor vector $\vec{V}_{i,G} = \{v_{1,i,G}, \ldots, v_{D,i,G}\}$ corresponding to the ith target vector $\vec{X}_{i,G}$ via one of the different mutation schemes of DE [(3) to (7)].

Step 2.2 ***Crossover Step***

Generate a trial vector $\vec{U}_{i,G} = \{u_{1,i,G}, \ldots, u_{D,i,G}\}$ for the ith target vector $\vec{X}_{i,G}$ through binomial

crossover (9) or exponential crossover (8) or through the arithmetic crossover (10).

Step 2.3 ***Selection Step***

Evaluate the trial vector $\vec{U}_{i,G}$.
IF $f(\vec{U}_{i,G}) \leq f(\vec{X}_{i,G})$
THEN $\vec{X}_{i,G+1} = \vec{U}_{i,G}$, $f(\vec{X}_{i,G+1}) = f(\vec{U}_{i,G})$
    IF $f(\vec{U}_{i,G}) < f(\vec{X}_{best,G})$
    THEN $\vec{X}_{best,G} = \vec{U}_{i,G}$, $f(\vec{X}_{best,G}) = f(\vec{U}_{i,G})$
    END IF
END IF
    ELSE $\vec{X}_{i,G+1} = \vec{X}_{i,G}$, $f(\vec{X}_{i,G+1}) = f(\vec{X}_{i,G})$.
END FOR

Step 2.4 Increase the Generation Count $G = G + 1$.
END WHILE

III. A Review of Previous Work on Improving the DE Algorithm

Over the past few years researchers have been investigating ways of improving the ultimate performance of the DE algorithm by tuning its control parameters. Storn and Price in [1] have indicated that a reasonable value for NP could be between $5D$ and $10D$ (D being the dimensionality of the problem), and a good initial choice of F could be 0.5. The effective value of F usually in the range [0.4, 1].

Gamperle *et al.* [20] evaluated different parameter settings for DE on the Sphere, Rosenbrock's, and Rastrigin's functions. Their experimental results revealed that the global optimum searching capability and the convergence speed are very sensitive to the choice of control parameters NP, F, and Cr. Furthermore, a plausible choice of the population size NP is between $3D$ and $8D$, with the scaling factor $F = 0.6$ and the crossover rate Cr in [0.3, 0.9]. Recently, the authors in [16] claim that typically $0.4 < F < 0.95$ with $F = 0.9$ is a good first choice. Cr typically lies in (0, 0.2) when the function is separable, while in (0.9, 1) when the function's parameters are dependent.

As can be seen from the literature, several claims and counterclaims were reported concerning the rules for choosing the control parameters, confusing engineers who try to solve real-world optimization problems with DE. Further, many of these claims lack sufficient experimental justification. Therefore researchers consider techniques such as self-adaptation to avoid manual tuning of the parameters of DE. Usually self-adaptation is applied to tune the control parameters F and Cr. Liu and Lampinen introduced fuzzy adaptive differential evolution (FADE) [21] using fuzzy logic controllers, whose inputs incorporate the relative function values and individuals of successive generations to adapt the parameters for the mutation and crossover operation. Based on the experimental results over a set of benchmark functions, the FADE algorithm outperformed the conventional DE algorithm. In this context, Qin *et al*. proposed a self-adaptive DE (SaDE) [22] algorithm, in which both the trial vector generation strategies and their associated parameters are gradually self-adapted by learning from their previous experiences of generating promising solutions.

Zaharie proposed a parameter adaptation strategy for DE (ADE) based on the idea of controlling the population diversity, and implemented a multipopulation approach [23]. Following the same line of thinking, Zaharie and Petcu designed an adaptive Pareto DE algorithm for multiobjective optimization and also analyzed its parallel implementation [24]. [25] self-adapted the crossover rate Cr for multiobjective optimization problems, by encoding the value of Cr into each individual and simultaneously evolving it with other search variables. The scaling factor F was generated for each variable from a Gaussian distribution N (0, 1).

[26] introduced a self-adaptive scaling factor parameter F. They generated the value of Cr for each individual from a normal distribution N (0.5, 0.15). This approach (called SDE) was tested on four benchmark functions and performed better than other versions of DE. Besides adapting the control parameters F or Cr, some researchers also adapted the population size. Teo proposed DE with self-adapting populations (DESAP) [27], based on Abbass's self-adaptive Pareto DE [25]. Recently, [28] encoded control parameters F and Cr into the individual and evolved their values by using two new probabilities τ_1 and τ_2. In their algorithm (called SADE), a set of F values was assigned to each individual in the population. With probability τ_1, F is reinitialized to a new random value in the range [0.1, 1.0], otherwise it is kept unchanged. The control parameter Cr, assigned to each individual, is adapted in an identical fashion, but with a different re-initialization range [0, 1] and with the probability τ_2. With probability τ_2, Cr takes a random value in [0, 1], otherwise it retains its earlier value in the next generation.

[29] introduced two schemes for adapting the scale factor F in DE. In the first scheme (called DERSF: DE with random scale factor) they varied F randomly between 0.5 and 1.0 in successive iterations. They suggested decreasing F linearly from 1.0 to 0.5 in their second scheme (called DETVSF: DE with time varying scale factor). This encourages the individuals to sample diverse zones of the search space during the early stages of the search. During the later stages, a decaying scale factor helps to adjust the movements of trial solutions finely so that they can explore the interior of a relatively small space in which the suspected global optimum lies.

DE/rand/1/either-or is a state-of-the-art DE variant described by Price *et al.* [3, p.118]. In this algorithm, the trial vectors that are pure mutants occur with a probability p_F and those that are pure recombinants occur with a probability $1 - p_F$. The scheme for trial vector generation may be outlined as

$$\begin{aligned}\vec{U}_{i,G} &= \vec{X}_{r_1^i,G} + F \\ &\quad \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}), \qquad \text{if } rand_i(0,1) < p_F \\ &= \vec{X}_{r_1^i,G} + K \\ &\quad \cdot (\vec{X}_{r_2^i,G} + \vec{X}_{r_3^i,G} - 2.\vec{X}_{r_1^i,G}), \quad \text{otherwise} \end{aligned} \tag{12}$$

where, according to Price *et al.*, $K = 0.5 \cdot (F + 1)$ serves as a good choice of the parameter K for a given F.

Rahnamayan *et al.* have proposed an opposition-based DE (ODE) [30] that is specially suited for noisy optimization problems. The conventional DE algorithm was enhanced by

utilizing the opposition number-based optimization concept in three levels, namely, population initialization, generation jumping, and local improvement of the population's best member.

[31] proposed a hybridization of DE with the neighborhood search (NS), which appears as a main strategy underpinning evolutionary programming (EP) [32]. The resulting algorithm, known as NSDE, performs mutation by adding a normally distributed random value to each target-vector component in the following way:

$$\vec{V}_{i,G} = \vec{X}_{r_1^i,G} + \begin{cases} \vec{d}_{i,G}.N(0.5, 0.5), & \text{if } rand_i(0,1) < 0.5 \\ \vec{d}_{i,G}.\delta, & \text{otherwise} \end{cases} \tag{13}$$

where $\vec{d}_{i,G} = \vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}$ is the usual difference vector, $N(0.5, 0.5)$ denotes a Gaussian random number with mean 0.5 and standard deviation 0.5, and δ denotes a Cauchy random variable with scale parameter $t = 1$. Recently [33] used a self-adaptive NSDE in the cooperative coevolution framework that is capable of optimizing large-scale nonseparable problems (up to 1000 dimensions). They proposed a random grouping scheme and adaptive weighting for problem decomposition and coevolution. Somewhat similar in spirit to the present paper is the study by [34] on self-adaptive differential evolution with neighborhood search (SaNSDE). SaNSDE incorporates self-adaptation ideas from the SaDE [22] and proposes three self-adaptive strategies: self-adaptive choice of the mutation strategy between two alternatives, self-adaptation of the scale factor F, and self-adaptation of the crossover rate Cr. We would like to point out here that in contrast to Yang *et al.*'s works on NSDE and SaNSDE, we keep the scale factor nonrandom and use a ring-shaped neighborhood topology (inspired by PSO [37]), defined on the index graph of the parameter vectors, in order to derive a local neighborhood-based mutation model. Also instead of F and Cr, the weight factor that unifies two kinds of mutation models have been made self-adaptive in one of the variants of DE/target-to-best/1 scheme, proposed by us. Section IV describes these issues in sufficient details.

Noman and Iba [35], [36] proposed the *Fittest Individual Refinement* (FIR); a crossover-based local search method for DE. The FIR scheme accelerates DE by enhancing its search capability through exploration of the neighborhood of the best solution in successive generations.

As will be evident from Section IV, the proposed method differs significantly from the works described in the last couple of paragraphs. It draws inspiration from the neighborhood topologies used in PSO [37]. Similar to DE, PSO has also emerged as a powerful real parameter optimization technique during the late 1990s. It emulates the swarm behavior of insects, animals herding, birds flocking, and fish schooling, where these swarms search for food in a collaborative manner. A number of significantly improved variants of basic PSO have been proposed in the recent past to solve both benchmark and real-world optimization problems, for example, see [38], [39]. Earlier attempts to hybridize DE with different operators of the PSO algorithm may be traced to [40] and [41].

IV. DE WITH A NEIGHBORHOOD-BASED MUTATION OPERATOR

A. DE/target-to-best/1—A Few Drawbacks

Most of the population-based search algorithms try to balance between two contradictory aspects of their performance: *exploration* and *exploitation*. The first one means the ability of the algorithm to "explore" or search every region of the feasible search space, while the second denotes the ability to converge to the near-optimal solutions as quickly as possible. The DE variant known as DE/target-to-best/1 (5) uses the best vector of the population to generate donor vectors. By "best" we mean the vector that corresponds to the best fitness (e.g., the lowest objective function value for a minimization problem) in the entire population at a particular generation. The scheme promotes exploitation since all the vectors/genomes are attracted towards the same best position (pointed to by the "best" vector) on the fitness landscape through iterations, thereby converging faster to that point. But as a result of such exploitative tendency, in many cases, the population may lose its global exploration abilities within a relatively small number of generations, thereafter getting trapped to some locally optimal point in the search space.

In addition, DE employs a greedy selection strategy (the better between the target and the trial vectors is selected) and uses a fixed scale factor F (typically in [0.4, 1]). Thus if the difference vector $\vec{X}_{r_1,G} - \vec{X}_{r_2,G}$ used for perturbation is small (this is usually the case when the vectors come very close to each other and the population converges to a small domain), the vectors may not be able to explore any better region of the search space, thereby finding it difficult to escape large plateaus or suboptimal peaks/valleys. Mezura-Montes *et al.,* while comparing the different variants of DE for global optimization in [17], have noted that DE/target-to-best/1 shows a poor performance and remains inefficient in exploring the search space, especially for multimodal functions. The same conclusions were reached by Price *et al.* [3, p.156].

B. Motivations for the Neighborhood-Based Mutation

A proper tradeoff between exploration and exploitation is necessary for the efficient and effective operation of a population-based stochastic search technique like DE, PSO, etc. The DE/target-to-best/1, in its present form, favors exploitation only, since all the vectors are attracted by the same best position found so far by the entire population, thereby converging faster towards the same point.

In this context we propose two kinds of neighborhood models for DE. The first one is called the *local neighborhood model*, where each vector is mutated using the best position found so far in a small neighborhood of it and *not* in the entire population. On the other hand, the second one, referred to as the *global mutation model*, takes into account the globally best vector $\vec{X}_{best,G}$ of the entire population at current generation G for mutating a population member. Note that DE/target-to-best/1 employs only the global mutation strategy.

A vector's neighborhood is the set of other parameter vectors that it is connected to; it considers their experience when updating its position. The graph of interconnections is

called the neighborhood structure. Generally, neighborhood connections are independent of the positions pointed to by the vectors. In the *local* model, whenever a parameter vector points to a good region of the search space, it only directly influences its immediate neighbors. Its second degree neighbors will only be influenced after those directly connected to them become highly successful themselves. Thus, there is a delay in the information spread through the population regarding the best position of each neighborhood. Therefore, the attraction to specific points is weaker, which prevents the population from getting trapped in local minima. We would like to mention here that vectors belonging to a *local neighborhood* are not necessarily local in the sense of their geographical nearness or similar fitness values. As will be seen in the next section, the overlapping neighborhoods have been created in DE according to the order of the indices of the population members, following the neighborhood models in PSO.

Finally, we combine the local and the global model using a *weight factor* that appears as a new parameter in the algorithm. The weight factor may be tuned in many different ways. In what follows we describe these issues in sufficient details. Note that the neighborhoods of different vectors were chosen randomly and not according to their fitness values or geographical locations on the fitness landscape, following the PSO philosophy [37]. This preserves the diversity of the vectors belonging to the same neighborhood.

C. Local and Global Neighborhood-Based Mutations in DE

Suppose we have a DE population $P_G = \left[\vec{X}_{1,G}, \vec{X}_{2,G}, \ldots, \vec{X}_{NP,G}\right]$ where each $\vec{X}_{i,G}$ $(i = 1, 2, \ldots, NP)$ is a D-dimensional parameter vector. The vector indices are sorted only randomly (as obtained during initialization) in order to preserve the diversity of each neighborhood. Now, for every vector $\vec{X}_{i,G}$ we define a neighborhood of radius k (where k is a nonzero integer from 0 to $(NP-1)/2$, as the neighborhood size must be smaller than the population size, i.e. $2k+1 \leq NP$), consisting of vectors $\vec{X}_{i-k,G}, \ldots, \vec{X}_{i,G}, \ldots, \vec{X}_{i+k,G}$. We assume the vectors to be organized on a ring topology with respect to their indices, such that vectors $\vec{X}_{NP,G}$ and $\vec{X}_{2,G}$ are the two immediate neighbors of vector $\vec{X}_{1,G}$. The concept of local neighborhood is schematically illustrated in Fig. 2. Note that the neighborhood topology is static and has been defined on the set of indices of the vectors. Although various neighborhood topologies (like star, wheel, pyramid, 4-clusters, and circular) have been proposed in the literature for the PSO algorithms [42], after some initial experimentation over numerical benchmarks, we find that in the case of DE (where the population size is usually larger than in the case of PSO) the circular or ring topology provides best performance compared to other salient neighborhood structures.

For each member of the population, a local donor vector is created by employing the best (fittest) vector in the neighborhood of that member and any two other vectors chosen from the same neighborhood. The model may be expressed as

$$\vec{L}_{i,G} = \vec{X}_{i,G} + \alpha \cdot (\vec{X}_{n\_best_i,G} - \vec{X}_{i,G}) + \beta \cdot (\vec{X}_{p,G} - \vec{X}_{q,G}) \quad (14)$$

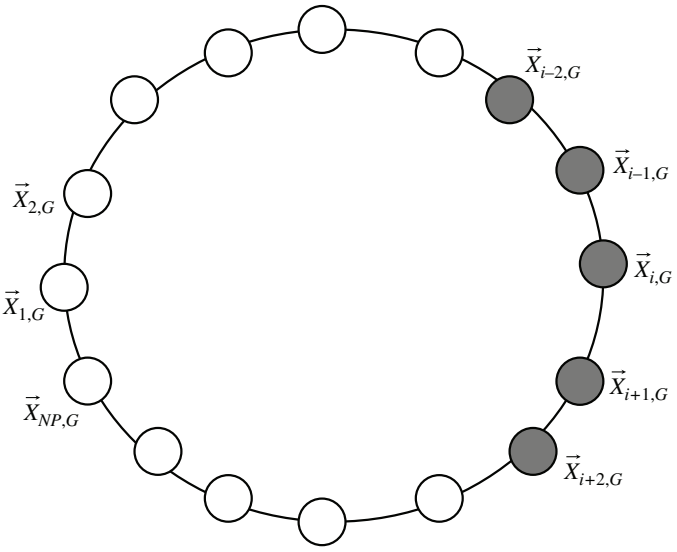


Fig. 2. Ring topology of neighborhood in DE. The dark spheres indicate a neighborhood of radius 2 of the ith population member where $i = 9$.

where the subscript n_best_i indicates the best vector in the neighborhood of $\vec{X}_{i,G}$ and $p, q \in [i-k, i+k]$ with $p \neq q \neq i$. Similarly, the global donor vector is created as

$$\vec{g}_{i,G} = \vec{X}_{i,G} + \alpha \cdot (\vec{X}_{g\_best,G} - \vec{X}_{i,G}) + \beta \cdot (\vec{X}_{r_1,G} - \vec{X}_{r_2,G}) \quad (15)$$

where the subscript g_best indicates the best vector in the entire population at generation G and $r_1, r_2 \in [1, NP]$ with $r_1 \neq r_2 \neq i$. α and β are the scaling factors.

Note that in (14) and (15), the first perturbation term on the right-hand side (the one multiplied by α) is an arithmetical recombination operation, while the second term (the one multiplied by β) is the differential mutation. Thus in both the global and local mutation models, we basically generate mutated recombinants, not pure mutants.

Now we combine the local and global donor vectors using a scalar weight $w \in (0, 1)$ to form the actual donor vector of the proposed algorithm

$$\vec{V}_{i,G} = w.\vec{g}_{i,G} + (1-w).\vec{L}_{i,G}. \quad (16)$$

Clearly, if $w = 1$ and in addition $\alpha = \beta = F$, the donor vector generation scheme in (16) reduces to that of DE/target-to-best/1. Hence the latter may be considered as a special case of this more general strategy involving both global and local neighborhood of each vector synergistically. From now on, we shall refer to this version as DEGL (DE with global and local neighborhoods). The rest of the algorithm is exactly similar to DE/rand/1/bin. DEGL uses a binomial crossover scheme and follows the pseudo-code given in Section III.

Note that in each generation, the vectors belonging to a DE population are perturbed sequentially. If a target vector $\vec{X}_{i,G}$ is replaced with the corresponding trial vector $\vec{U}_{i,G}$, the neighborhood-best $\vec{X}_{n\_best_i,G}$ and the globally best vector $\vec{X}_{g\_best,G}$ may also be updated by $\vec{U}_{i,G}$, provided the latter yields a lower value of the objective function. In Section IV-E, we discuss the additional computational complexity of updating the neighborhood-best vectors in DEGL after the replacement of each target vector in a generation.

D. Control Parameters in DEGL

DEGL introduces four new parameters: α, β, w, and the neighborhood radius k. Among them α and β are playing the same role as the constant F in (5). Thus, in order to reduce the number of parameters further, we take $\alpha = \beta = F$. The most crucial parameter in DEGL is perhaps the weight factor w, which controls the balance between the exploration and exploitation capabilities. Small values of w (close to 0) in (16) favor the local neighborhood component, thereby resulting in better exploration. On the other hand, large values (close to 1) favor the global variant component, promoting exploitation. Therefore, values of w around the middle point, 0.5, of the range [0, 1] result in the most balanced DEGL versions. However, such balanced versions do not take full advantage of any special structure of the problem at hand (e.g., unimodality, convexity, etc.). In such cases, weight factors that are biased towards 0 or 1 may exhibit better performance. Moreover, on-line adaptation of w during the execution of the algorithm can enhance its performance. Optimal values of the weight factor will always depend on the problem at hand. We considered three different schemes for the selection and adaptation of w to gain intuition regarding DEGL performance and we describe them in the following paragraphs.

1) Increasing Weight Factor: All vectors have the same weight factor which is initialized to 0 and is increased up to 1 during the execution of the algorithm. Thus, exploration is favored in the first stages of the algorithm's execution (since $w = 0$ corresponds to the local neighborhood model) and exploitation is promoted at the final stages, when w assumes higher values. Let G denote the generation number, w_G the weight factor at generation G, and $G_{\max}$ the maximum number of generations. We considered two different increasing schedules in our study.

a) Linear increment: w is linearly increased from 0 to 1

$$w_G = \frac{G}{G_{\max}}. \tag{17}$$

b) Exponential increment: The weight factor increases from 0 to 1 in an exponential fashion as follows:

$$w_G = \exp\left(\frac{G}{G_{\max}}.\ln(2)\right) - 1 \tag{18}$$

This scheme results in slow transition from exploration to exploitation in the early stages of the algorithm's execution, but exhibits faster transition in the later stages.

2) Random Weight Factor: In this scheme the weight factor of each vector is made to vary as a uniformly distributed random number in (0, 1) i. e. $w_{i,G} \sim rand(0, 1)$. Such a choice may decrease the convergence speed (by introducing more diversity).

3) Self-Adaptive Weight Factor: In this scheme, each vector has its own weight factor. The factor is incorporated in the vector as an additional variable, augmenting the dimension of the problem. Thus, a generation now consists of vectors $\vec{a}_{i,G} = \{\vec{X}_{i,G}, \vec{S}_{i,G}\}$, where $\vec{S}_{i,G} = \{w_{i,G}\}$ and $w_{i,G}$ is the weight factor for vector $\vec{X}_{i,G}$. During the initialization phase of DE, $w_{i,G}$ is randomly initialized in (0.0, 1.0). Next, while evolving a vector $\vec{a}_{i,G}$, at first local and global mutant vectors $\vec{L}_{i,G}$ and $\vec{g}_{i,G}$ are formed for $\vec{X}_{i,G}$ following (14) and (15). The sub-vector $\vec{S}$ undergoes global mutation only and weight factors perturbing $\vec{S}$ come from the same population members $\vec{a}_{r_1,G}$ and $\vec{a}_{r_2,G}$, which were also used to form $\vec{g}_{i,G}$. The mutation of $w_{i,G}$ leads to the formation of a new trial weight factor $w'_{i,G}$ according to the following equation

$$w'_{i,G} = w_{i,G} + F.(w_{g\_best,G} - w_{i,G}) + F.(w_{r_1,G} - w_{r_2,G}) \tag{19}$$

where, $w_{g\_best,G}$ is the weight factor associated with the best parameter vector $\vec{X}_{g\_best,G}$. The value of the newly formed $w'_{i,G}$ is restricted to the range [0.05, 0.95] in the following way

$$\begin{aligned} &\text{if} && w'_{i,G} > 0.95, && w'_{i,G} = 0.95; \\ &\text{else if} && w'_{i,G} < 0.05, && w'_{i,G} = 0.05 \end{aligned} \tag{20}$$

$w'_{i,G}$ is then used to combine $\vec{L}_{i,G}$ and $\vec{g}_{i,G}$ according to (16) and this leads to the formation of the new donor parameter vector $\vec{V}_{i,G}$. The donor vector thus formed exchanges its components with $\vec{X}_{i,G}$ following the binomial crossover and results in the production of the trial vector $\vec{U}_{i,G}$. Note that the weight factor does not undergo crossover. Now, the newly formed weight factor is promoted to the next generation only if $\vec{U}_{i,G}$ yields an equal or lower objective function value as compared to $\vec{X}_{i,G}$: i. e.,

$$\begin{aligned} \vec{a}_{i,G+1} &= \{\vec{X}_{i,G+1} \\ &= \vec{U}_{i,G}, \vec{S}_{i,G+1} \\ &= \{w'_{i,G}\}\}, \quad \text{if } f(\vec{U}_{i,G}) \le f(\vec{X}_{i,G}) \\ \vec{a}_{i,G+1} &= \{\vec{X}_{i,G+1} \\ &= \vec{X}_{i,G}, \vec{S}_{i,G+1} \\ &= \{w_{i,G}\}\}, \quad \text{otherwise} \end{aligned} \tag{21}$$

the process is repeated sequentially for each vector in a generation. Note that the weight factors associated with the neighborhood-best and globally best vectors are not updated every time a trial vector replaces the corresponding target. The weight factor for a parameter vector is changed only once according to (19) and (20) in each generation. According to the self-adaptation scheme, the dynamics of DEGL are allowed to determine the optimal $w_{i,G}$ for each vector, individually, capturing any special structure of the problem at hand.

Finally, we would like to point out that a proper selection of the neighborhood size affects the tradeoff between exploration and exploitation. However, there are no general rules regarding the selection of neighborhood size, and it is usually based on the experience of the user. The effect of neighborhood size on the performance of DEGL has been further investigated in Section VI-E.

E. Runtime Complexity of DEGL—A Discussion

Runtime-complexity analysis of the population-based stochastic search techniques like DE, GA, etc. is a critical issue by its own right. Following the works of Zielinski *et al.*, [43] we note that the average runtime of a standard DE algorithm

usually depends on its stopping criterion. While computing the run-time complexity, we usually take into account the fundamental floating-point arithmetic and logical operations performed by an algorithm [44]. We may neglect very simple operations like copy/assignment, etc., as these are merely data-transfer operations between the ALU and/or CPU registers and hardly require any complex digital circuitry like adder, comparator, etc. [44], [45]. Now, in each generation of DE, a loop over NP is conducted, containing a loop over D. Since the mutation and crossover operations are performed at the component level for each DE vector, the number of fundamental operations in DE/rand/1/bin is proportional to the total number of loops conducted until the termination of the algorithm. Thus, if the algorithm is stopped after a fixed number of generations $G_{\max}$, then the runtime complexity is $O(NP \cdot D \cdot G_{\max})$.

For DE/target-to-best/1, runtime complexity of finding the globally best vector depends only on comparing the objective function value against the single best vector's objective function value. Note that the best objective function evaluation value must be upgraded for each newly generated trial vector, if it replaces the target vector. Now that means in the worst possible case (when the target vector is always replaced by the trial vector), this is done $NP \cdot G_{\max}$ times. Thus, the overall runtime remains $O(\max(NP \cdot G_{\max}, NP \cdot D \cdot G_{\max})) = O(NP \cdot D \cdot G_{\max})$.

In DEGL, besides the globally best vector, we have to take into account the best vector of each neighborhood as well. Each individual vector is endowed with a small memory, which can keep track of the best vector in its neighborhood and the corresponding objective function value. At the very onset, once all the vectors are initialized, a search is performed to detect the neighborhood-best for each individual. Note that this search is performed only once at $G = 0$. In subsequent generations, these locally best vectors only need to be updated in the memory of the neighboring vectors. This is just like the updating phase of the globally best vector in DE/target-to-best/1 according to step 2.3 of the DE pseudo-code provided earlier. Now let us try to estimate the cost of the initial search. Note that the neighborhoods in DEGL are actually overlapping in nature (on the index-graph) and this is illustrated in Fig. 3. Any two adjacent vectors (with respect to their indices) will have $2k + 1 + 1 - 2 = 2k$ number of common neighbors.

Suppose $N_k(\vec{X}_{i,G})$ indicates the set of vectors belonging to the immediate neighborhood of radius k for the vector $\vec{X}_{i,G}$. Then evidently the cardinality of both the sets $N_k(\vec{X}_{i,G}) \cap N_k^c(\vec{X}_{i+1,G})$ and $N_k^c(\vec{X}_{i,G}) \cap N_k(\vec{X}_{i+1,G})$ is exactly 1 (where N_k^c stands for complement of the set N_k). We observe that $\vec{X}_{i-k,G} \in N_k(\vec{X}_{i,G}) \cap N_k^c(\vec{X}_{i+1,G})$ and $\vec{X}_{i+k+1,G} \in N_k^c(\vec{X}_{i,G}) \cap N_k(\vec{X}_{i+1,G})$. Now we start by detecting the best vector of the neighborhood of any population member, say $\vec{X}_{i,G}$ and call it $\vec{X}_{n\_best_i,G}$. This is equivalent to finding the lowest entry from an array of $2k + 1$ numbers (objective function values) and requires $2k$ number of comparisons. Next, to calculate the best vector in the neighborhood of $\vec{X}_{i+1,G}$, if $\vec{X}_{n\_best_i,G} \neq \vec{X}_{i-k,G}$ then we simply need to compare the objective function values of $\vec{X}_{i+k+1,G}$ and $\vec{X}_{n\_best_i,G}$ in order to determine $\vec{X}_{n\_best_{i+1},G}$. This requires only one comparison.

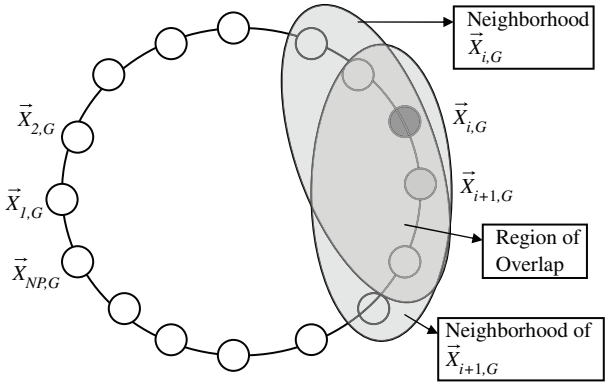


Fig. 3. Overlapping of neighborhoods in DEGL.

But if unfortunately $\vec{X}_{n\_best_i,G} = \vec{X}_{i-k,G}$, we shall have to find the neighborhood best of $\vec{X}_{i+1,G}$ by taking its $2k$ neighbors into account and this requires $O(k)$ runtime. Hence in the worst possible case (when the current neighborhood's best vector is always excluded from the serially next vector's neighborhood) searching the best vectors of all the neighborhoods is completed in $O(NP \cdot k)$ time.

Once the search for all neighborhood-bests is finished, in subsequent generations, the best vector in the neighborhood of $\vec{X}_{i,G}$ is updated only if a newly generated trial vector $\vec{U}_{i,G}$ replaces the target vector $\vec{X}_{i,G}$ and in addition to that $f(\vec{U}_{i,G}) < f(\vec{X}_{n\_best_i,G})$. It is possible that $\vec{X}_{n\_best_i,G}$ differs from $\vec{X}_{n\_best_{i+1},G}$, i.e., two vectors, adjacent on the index graph, may have distinct neighborhood-best vectors. This happens when the best vector in the neighborhood of $\vec{X}_{i+1,G}$ is $\vec{X}_{i+k+1,G}$. Under this condition, it is possible that $\vec{U}_{i,G}$ is better than $\vec{X}_{n\_best_i,G}$ but not better than $\vec{X}_{n\_best_{i+1},G}$. Hence in order to update the best vectors in the memories of all the neighbors of $\vec{X}_{i,G}$ (when $f(\vec{U}_{i,G}) < f(\vec{X}_{n\_best_i,G})$ is satisfied), we have to compare the objective function values of $\vec{U}_{i,G}$ and the neighborhood-bests in the memories of $2k$ neighbors of $\vec{X}_{i,G}$. Thus in the worst possible case, updating of all the local best vectors in the memories of the neighbors of each vector requires $O(NP \cdot k)$ comparisons in each generation. Evidently, over $G_{\max}$ generations, the number of additional comparisons necessary is $O(NP \cdot k \cdot G_{\max})$. This implies that the worst case complexity of DEGL is actually $O(\max(NP \cdot k \cdot G_{\max}, NP \cdot D \cdot G_{\max}))$. Now, the asymptotic order of complexity for DEGL remains $O(NP \cdot D \cdot G_{\max})$ if $k \leq D$. Please note that this condition is usually satisfied when DEGL is applied to the optimization of higher dimensional functions. For example, the usual population size for DE is $NP = 10D$. If the neighborhood size is approximately 10% of the population size (which, as can be seen later, provides reasonably good results with DEGL), we have $2k + 1 = (0.1) \cdot NP = D \Rightarrow k = [(D-1)/2]$ with $D > 1$. Clearly, in this case we have $k \leq D$. Simple algebraic calculations show that this condition holds true if the neighborhood size is below 20% of the population size NP and $D > 1$. Hence, we can say that under such conditions, $O(\max(NP \cdot k \cdot G_{\max}, NP \cdot D \cdot G_{\max})) = O(NP \cdot D \cdot G_{\max})$ and thus DEGL does not impose any serious burden on the runtime complexity of the existing DE variants.

TABLE I
CODE-FUNCTION RUNTIME PROFILES FOR DE/RAND/1/BIN AND DEGL

Algorithm	Total execution time (in milliseconds)	Code-function runtime as % of CPU time						
		init_pop	mutate_vector	Recombine	select _and_update	DE_operator	evaluate_cost	Main
DE/rand/1/bin	9382.703 (1825.335)	0.122 (0.0051)	16.728 (0.628)	29.661 (1.481)	8.726 (7.335)	28.824 (3.561)	13.721 (2.727)	2.018 (0.114)
DEGL	9739.684 (1473.627)	0.109 (0.0046)	15.431 (0.937)	16.362 (2.771)	16.839 (6.361)	36.836 (1.663)	12.954 (1.638)	1.469 (0.118)

In order to validate the arguments made above, we provide in Table I the results of code-function profiling for our implementations of classical DE (DE/rand/1/bin) and DEGL (with random weight factor) using the profiler available with MS Visual C++ 6.0. Both the algorithms were coded in the C language and run on the simple 50-dimensional sphere function (f_1 in the list of benchmarks provided in Table IV). The least complex sphere function was chosen so that most of the CPU time may be spent on the DE operators and not on function evaluations. Here our primary objective is to observe what percentage of the total CPU time is used by the evolutionary operators of DEGL and DE/rand/1/bin. Both algorithms use the same prime modules or code-functions: init_pop (for initializing population), mutate_vector (for performing mutation and creating donor vector), recombine (to perform crossover and create the trial vector), select_and_update (to compare the objective function values of trial and target vectors and in DEGL also to update the neighborhood bests if for the ith vector, the condition $f(\vec{U}_{i,G}) < f(\vec{X}_{n\_best_i,G})$ holds), DE_Operator (module that calls the functions mutate_vector, recombine, and select_and_update for each vector sequentially), evaluate_cost (function that evaluates the objective function for a parameter vector), and the main. The programs were run on a Pentium IV, 2.2-GHz PC, with 512-KB cache and 2 GB of main memory in Windows Server 2003 environment. In Table I we provide the code function profiling results as means (with standard deviations in parentheses) of 1000 runs of the programs, each run continued up to 10^5 cost function evaluations (FEs).

Table I shows that, as expected, the total execution time for DEGL is only marginally higher than that for DE/rand/1/bin. This is because around 16.9% of the total CPU time is consumed by the select_and_update function in DEGL, due to the extra comparisons required for updating the neighborhood-bests. However, if we select a stopping criterion based on a threshold objective function value, instead of the stopping criterion based on maximum number of FEs, DEGL can even take less computation time as compared to DE/rand/1/bin in some cases. This is because DEGL can attain the threshold objective function value much quicker, consuming significantly smaller number of FEs, due to the better tradeoff between exploration and exploitation abilities achieved by its neighborhood-based mutation operators. This fact has been illustrated by providing, in Tables II and III, the mean processor time taken by both the algorithms for both stopping criteria over five most popular benchmark functions used for testing the evolutionary algorithms. Note that both the algorithms start from the same initial population and run under the same software and hardware platforms. All the numerical benchmarks dealt in here are in 25 dimensions, have their true optima at 0.00, and for all of them the target threshold value was set at 1.00e-05 in Tables II and III. A detailed description of these functions can be found in Table IV in the following section. Each result is the average of 50 independent runs.

We would like to point out that, in the evolutionary computing literature, comparison of the computational costs of various evolutionary algorithms is usually performed on the basis of the number of FEs they take to reach a predefined function value. Processor time cannot serve as a reliable metric in this context because first, it is not independent of the hardware and software platforms used, and second, it may provide some unfair advantage to algorithms that use lower computational overheads. In addition, the processor time depends on the style of coding an algorithm [46]. The advantage of measuring the runtime complexity by counting the number of FEs is that the correspondence between this measure and the processor time becomes stronger as the function complexity increases. In Section VI, we compare the computational cost and convergence speed of a number of DE-variants using this measure. The tables included in this section are intended only to provide an approximate feel of the relative time complexities of DEGL and classical DE.

Table II shows that when DEGL and DE/rand/1/bin are run for the same number of FEs (corresponding to the same number of generations for both, as they have the same population size), the processor time required by the former is slightly higher than that of the latter. Table III, however, indicates that DEGL may reach the predefined threshold value with less processor time as compared to DE/rand/1/bin.

V. EXPERIMENTAL SETUP

A. Benchmark Functions

We have used a test bed of 21 traditional numerical benchmarks (Table IV) [47] and three composition functions from the benchmark problems suggested in CEC 2005 [48] to evaluate the performance of the new DE variant. The 21 traditional benchmarks described by Yao *et al.* have been reported in Table IV where D represents the number of dimensions. For $f_1 - f_{13}$ we have tested for $D = 25$ to 100 in

TABLE II
COMPARISON OF ABSOLUTE RUN-TIMES OF DEGL AND DE/RAND/1/BIN, WHEN BOTH THE ALGORITHMS WERE RUN FOR A FIXED NUMBER OF FES

Function	Mean processor time (in milliseconds) and standard deviation (in parentheses)	
	DE/rand/1/bin	DEGL
Step function (f_6)	3692.84 (688.25)	3973.38 (827.51)
Rosenbrock's function (f_5)	6726.57 (1425.53)	7061.48 (1930.51)
Rastrigin's function (f_9)	5883.54 (629.63)	6273.38 (447.23)
Ackley's function (f_{11})	5094.68 (1624.83)	5268.46 (324.68)
Griewank's function (f_{12})	5635.92 (1023.35)	6163.28 (729.46)

TABLE III
COMPARISON OF ABSOLUTE RUN-TIMES OF DEGL AND DE/RAND/1/BIN, WHEN BOTH THE ALGORITHMS WERE RUN UNTIL THEY ATTAIN A PRE-DEFINED OBJECTIVE FUNCTION VALUE

Function	Threshold objective-function value to reach	Mean processor time (in milliseconds) and standard deviation (in parentheses)	
		DE/rand/1/bin	DEGL
Step function (f_6)	1.00-05	3022.84 (271.22)	2873.38 (712.58)
Rosenbrock's function (f_5)	1.00-05	5718.92 (1425.53)	5448.37 (1628.31)
Rastrigin's function (f_9)	1.00-05	2483.56 (442.67)	1682.94 (538.19)
Ackley's function (f_{11})	1.00-05	839.68 (154.41)	692.70 (32.61)
Griewank's function (f_{12})	1.00-05	4836.29 (1023.35)	4667.25 (1416.47)

steps of 25. Among these benchmarks, functions $f_1 - f_{13}$ are multidimensional problems. Functions $f_1 - f_5$ are unimodal (there is some recent evidence [49] that f_5 is multimodal for $D > 3$). Function f_6 is a step function with one minimum and is discontinuous. Function f_7 is a noisy quartic function, where *random* [0, 1) is a uniformly distributed random number in [0, 1).

Functions $f_8 - f_{13}$ are multimodal, with the number of local minima increasing exponentially with the problem dimension [47]. They apparently belong to the most difficult class of problems for many optimization algorithms. Functions $f_{14} - f_{21}$ are low-dimensional functions which have only a few local minima. For unimodal functions, the convergence rates of the DE algorithms are more interesting than the final results of optimization, as there are other methods which are specifically designed to optimize unimodal functions. For multimodal functions, the final results are much more important since they reflect an algorithm's ability of escaping from poor local optima and locating a good near-global optimum. We omitted f_{19} and f_{20} from Yao *et al.*'s study [47] because of difficulties in obtaining the definitions of the constants used in these functions.

The three composition functions $f_{18}(\vec{X})$, $f_{19}(\vec{X})$, and $f_{21}(\vec{X})$, taken from CEC 2005 benchmarking problems [48], are here marked as CF1, CF2, and CF3 respectively. All of them are nonseparable, rotated, and multimodal functions containing a large number of local optima. For all of them, the search range is $\vec{X} \in [-5, 5]^D$. The global optimum of both CF1 and CF2 is $f(\vec{X}^*) = 10$ and that for CF3 is $f(\vec{X}^*) = 360$. The detailed principle of the composite functions is given in [48].

For the generalized penalized functions f_{12} and f_{13}, in Table I, note that

$$u(x_i, a, k, m) \begin{cases} = k(x_i - a)^m, & \text{if } x_i > a \\ = 0, & \text{if } -a \le x_i \le a \\ = k(-x_i - a)^m, & \text{if } x_i < -a \end{cases}$$

and

$$y_i = 1 + \frac{1}{4}(1 + x_i).$$

Values of the other constants used in the expressions of the benchmark functions can be found in [47].

B. Other Optimization Problems Considered

In this section we describe two interesting real-world problems that have been used to test the efficacy of the DEGL family. The problems are selected according to the level of difficulty that they present to the proposed algorithms.

1) Spread Spectrum Radar Poly-Phase Code Design Problem: A famous problem of optimal design arises in the field of spread spectrum radar poly-phase codes [50]. Such a problem is very well suited for the application of global optimization algorithms like DE. The problem can be formally stated as

$$\text{Global } \min f(\vec{X}) = \max\{\varphi_1(\vec{X}), \ldots, \varphi_{2m}(\vec{X})\} \tag{22}$$

where

$$\vec{X} = \{(x_1, \ldots, x_D) \in \Re^D | 0 \le x_j \le 2\pi, \quad j = 1, \ldots, D\} \quad \text{and} \quad m = 2 - D - 1$$

with

$$\varphi_{2i-1}(\vec{X}) = \sum_{j=i}^{D} \cos\left(\sum_{k=|2i-j-1|-1}^{j} x_k\right), \qquad i = 1, 2, \ldots, D$$

$$\varphi_{2i}(\vec{X}) = 0.5 + \sum_{j=i+1}^{D} \cos\left(\sum_{k=|2i-j-1|-1}^{j} x_k\right), \qquad i = 1, 2, \ldots, D-1$$

$$\varphi_{m+i}(\vec{X}) = -\varphi_i(\vec{X}), \qquad i = 1, 2, \ldots, m. \tag{23}$$

TABLE IV
TWENTY FIRST TRADITIONAL BENCHMARK FUNCTIONS [47]

Function	D	Search range	Optimum value
$f_1(\vec{X}) = \sum_{i=1}^{D} x_i^2$	25, 50, 75, and 100	$-100 \le x_i \le 100$	$f_1(\vec{0}) = 0$
$f_2(\vec{X}) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} x_i$	25, 50, 75, and 100	$-10 \le x_i \le 10$	$f_2(\vec{0}) = 0$
$f_3(\vec{X}) = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$	25, 50, 75, and 100	$-100 \le x_i \le 100$	$f_3(\vec{0}) = 0$
$f_4(\vec{X}) = \max\lvert x_i \rvert, 1 \le i \le D$	25, 50, 75, and 100	$-100 \le x_i \le 100$	$f_4(\vec{0}) = 0$
$f_5(\vec{X}) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	25, 50, 75, and 100	$-30 \le x_i \le 30$	$f_5(\vec{1}) = 0$
$f_6(\vec{X}) = \sum_{i=1}^{D} (\lfloor x_i + 0.5 \rfloor)^2$	25, 50, 75, and 100	$-100 \le x_i \le 100$	$f_6(\vec{p}) = 0, -\frac{1}{2} \le p_i < \frac{1}{2}$
$f_7(\vec{x}) = \left(\sum_{i=1}^{D} i.x_i^4\right) + rand\,[0, 1)$	25, 50, 75, and 100	$-1.28 \le x_i \le 1.28$	$f_7(\vec{0}) = 0$
$f_8(\vec{X}) = \sum_{i=1}^{D} -x_i . \sin\left(\sqrt{\lvert x_i \rvert}\right)$	25, 50, 75, and 100	$-500 \le x_i \le 500$	$f_8(420.97) = -41898.3$ for $D = 100$
$f_9(x) = \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	25, 50, 75, and 100	$-5.12 \le x_i \le 5.12$	$f_9(\vec{0}) = 0$
$f_{10}(\vec{X}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D} \cos 2\pi x_i\right) + 20 + e$	25, 50, 75, and 100	$-32 \le x_i \le 32$	$f_{10}(\vec{0}) = 0$
$f_{11}(\vec{X}) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$	25, 50, 75, and 100	$-600 \le x_i \le 600$	$f_{11}(\vec{0}) = 0$
$f_{12}(\vec{X}) = \frac{\pi}{D}\{10\sin^2(\pi y_1) + \sum_{i=1}^{D-1} (x_i - 1)^2 . \left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_D - 1)^2\} + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$	25, 50, 75, and 100	$-50 \le x_i \le 50$	$f_{12}(\overrightarrow{-1}) = 0$
$f_{13}(\vec{X}) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{D-1} (x_i - 1)^2 . \left[1 + \sin^2(3\pi x_{i+1})\right] + (x_D - 1)\{1 + \sin^2(2\pi x_n)\}\} + \sum_{i=1}^{D} u(x_i, 5, 100, 4)$	25, 50, 75, and 100	$-50 \le x_i \le 50$	$f_{13}(1, \ldots, 1, -4.76) = -1.1428$
$f_{14}(\vec{X}) = \left(\frac{1}{500} + \sum_{j=1}^{25}\left(j + 1 + \sum_{i=0}^{1} (x_i - a_{ij})\right)^6\right)^{-1}$	2	$-65.54 \le x_i \le 65.54$	$f_{14}(\overrightarrow{-31.95}) = 0.998$
$f_{15}(\vec{X}) = \sum_{i=0}^{10} (a_i - \frac{x_0(b_i^2 + b_i x_1)}{b_i^2 + b_i x_2 + x_3})^2$	4	$-5 \le x_i \le 5$	$f_{15}(0.1928, 0.1908, 0.1231, 0.1358) = 0.0003075$
$f_{16}(\vec{X}) = 4x_0^2 - 2.1x_0^4 + \frac{1}{3}x_0^6 + x_0x_1 - 4x_1^2 + 4x_1^4$	2	$-5 \le x_i \le 5$	$f_{16}(-0.09, 0.71) = -1.0316$
$f_{17}(\vec{X}) = (x_1 - \frac{5.1}{4\pi^2}x_0^2 + \frac{5}{\pi}x_0 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(x_0) + 10$	2	$-5 \le x_i \le 5$	$f_{17}(9.42, 2.47) = 0.398$
$f_{18}(\vec{X}) = \{1 + (x_0 + x_1 + 1)^2(19 - 14x_0 + 3x_0^2 - 14x_1 - 6x_0x_1 + 3x_1^2)\}\{30 + (2x_0 - 3x_1)^2 (18 - 32x_0 + 12x_0^2 + 48x_1 - 36x_0x_1 + 27x_1^2)\}$	2	$-2 \le x_i \le 2$	$f_{18}(1.49e-05, 1.00) = 3$
$f_{19}(\vec{X}) = -\sum_{i=1}^{5} ((\vec{X} - \vec{a}_i)^T(\vec{X} - \vec{a}_i) + c_i)^{-1}$	4	$-10 \le x_i \le 10$	$f_{19}(\vec{4}) = -10.1532$
$f_{20}(\vec{X}) = -\sum_{i=1}^{7} ((\vec{X} - \vec{a}_i)^T(\vec{X} - \vec{a}_i) + c_i)^{-1}$	4	$-10 \le x_i \le 10$	$f_{20}(\vec{4}) = -10.4029$
$f_{21}(\vec{X}) = -\sum_{i=1}^{10} ((\vec{X} - \vec{a}_i)^T(\vec{X} - \vec{a}_i) + c_i)^{-1}$	4	$-10 \le x_i \le 10$	$f_{21}(\vec{4}) = -10.5364$

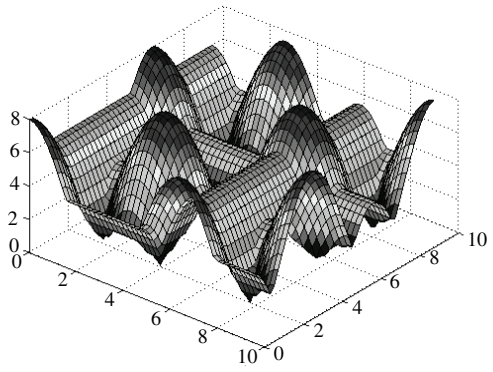


Fig. 4. $f(\vec{X})$ of (22) for $D = 2$.

According to [50] the above problem has no polynomial time solution. The objective function for $D = 2$ is shown in Fig. 4.

2) Application to Parameter Estimation for Frequency-Modulated (FM) Sound Waves: Frequency-modulated (FM) sound synthesis plays an important role in several modern music systems. This section describes an interesting application of the proposed DE algorithms to the optimization of parameters of an FM synthesizer. A few related works that attempt to estimate parameters of the FM synthesizer using the genetic algorithm can be found in [51], [52]. Here we introduce a system that can automatically generate sounds similar to the target sounds. It consists of an FM synthesizer, a DE optimizer, and a feature extractor. The system architecture is shown in Fig. 5. The target sound is a *.wav* file. The DE algorithm initializes a set of parameters and the FM synthesizer generates the corresponding sounds. In the feature extraction step, the dissimilarities of features between the target sound and synthesized sound are used to compute the fitness value. The process continues until synthesized sounds become very similar to the target.

The specific instance of the problem discussed here involves determination of six real parameters. $\vec{X} = \{a_1, \omega_1, a_2, \omega_2, a_3, \omega_3\}$ of the FM sound wave given by (24) for approximating it to the sound wave given in (25) where $\theta = 2\pi/100$. The parameters are defined in the range $[-6.4, +6.35]$. The formula for the estimated sound wave and the target sound wave may be given as

$$y(t) = a_1.\sin(\omega_1.t.\theta + a_2.\sin(\omega_2.t.\theta + a_3.\sin(\omega_3.t.\theta))) \quad (24)$$

$$y_0(t) = 1.0.\sin(5.0.t.\theta - 1.5.\sin(4.8.t.\theta + 2.0.\sin(4.9.t.\theta))). \quad (25)$$

The goal is to minimize the sum of squared errors between the estimated sound and the target sound, as given by (26). This problem involves a highly complex multimodal function having strong epistasis (interrelation among the variables), with optimum value 0.0

$$f(\vec{X}) = \sum_{t=0}^{100} (y(t) - y_0(t))^2. \quad (26)$$

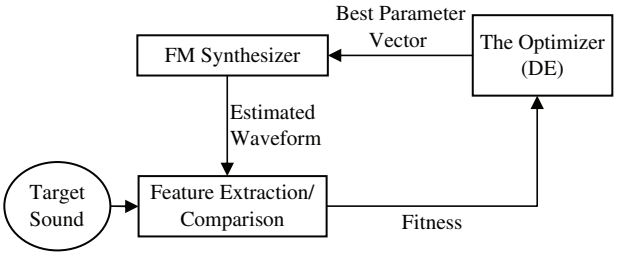


Fig. 5. Architecture of the optimization system.

Owing to the great difficulty of solving this problem with high accuracy without specific operators for continuous optimization (like gradual GAs [52]), we stop the algorithm when the number of function evaluations exceeds 10^5. As in the previous experiments, here also the runs of the competing DE variants start with the same initial population.

C. Algorithms for Comparison

At first, four versions of the proposed DEGL algorithm (with different schedules for changing the weight factor w) are compared with the DE/target-to-best/1/bin. These four versions are referred to as DEGL/LIW (DEGL with linearly increasing weight factor), DEGL/EIW (DEGL with exponential increasing weight factor), DEGL/RandW (DEGL with random weight factor) and DEGL/SAW (DEGL with self-adaptive weight factor). We included a DEGL algorithm with a fixed value of w for all the vectors in this comparative study. For this scheme we choose $w = 0.5$ (which provides equal importance to both local and global mutation schemes and appears to be the best performer as compared to other fixed values of w varying between 0.1 to 1.0 in steps of 0.1). The reason for including this scheme is to illustrate the effectiveness of the time-varying or adaptive weight factor over a fixed weight factor. In order to investigate the effect of the explorative mutation operator, the local-only DEGL (with $w = 0$) was also taken into account in the comparative study.

Simulations were carried out to obtain a comparative performance analysis of DEGL/SAW (that appears to be the best performing algorithm from the first set of experiments) with respect to: 1) DE/rand/1/bin [1], 2) DE/target-to-best/1/bin [19], 3) DE/rand/1/either-or [3], 4) SADE [28], and 5) NSDE [31]. Among the competitors, the first two belong to the classical DE family of Storn and Price. The DE/rand/1/bin algorithm was chosen because of its wide popularity in solving numerical optimization or engineering problems [3].

D. Initial Population and Method of Initialization

For all the contestant algorithms we used the same population size, which is 10 times the dimension D of the problem. To make the comparison fair, the populations for all the DE variants (over all problems tested) were initialized using the same random seeds. Fogel and Beyer [53] have shown that the typical method of symmetric initialization, which is used to compare evolutionary computations, can give false impressions of relative performance. In many comparative experiments, the initial population is uniformly distributed about the entire

TABLE V
AVERAGE AND THE STANDARD DEVIATION OF THE BEST-OF-RUN SOLUTION FOR 50 INDEPENDENT RUNS AND THE SUCCESS RATE TESTED ON FUNCTIONS f_8 TO f_{13} AND COMPOSITE FUNCTIONS CF1 TO CF3

functions	Mean best value (Standard deviation)						
	DE/target-to-best/1/bin	DEGL with fixed $w = 0.5$	DEGL with $w = 0$	DEGL/LI	DEGL/EI	DEGL/RandW	DEGL/SAW ($Cr = 0.9$)
f_8	−3.94382e+04 (5.83e−06)	−3.8756e+04 (7.00e−06)	−3.5621e+04 (8.58e−06)	−4.03634e+04 (3.81e−05)	−4.18436e+04 (5.22e−05)	−4.09039e+04 (8.39e−06)	**−4.18983e+04 (6.98e−06)**
f_9	8.38673e−02 (5.06e−03)	8.35525e−02 (4.96e−02)	5.1215e−03 (3.81e−03)	3.46138e−06 (5.91e−07)	2.90833e−06 (5.91e−06)	8.93821e−21 (5.4342e−18)	**1.7728e−26 (3.88e−25)**
f_{10}	6.76249e−01 (4.27e−01)	6.65735e−01 (7.07e−01)	2.09364e−01 (4.38e−01)	5.48844e−02 (1.68e−01)	3.93270e−04 (3.28e−02)	3.00895e−10 (7.16e−07)	**8.52742e−17 (1.365e−15)**
f_{11}	5.27284e−05 (4.63e−07)	9.07997e−06 (9.02e−05)	6.46925e−06 (3.49e−08)	8.63652e−06 (1.02e−04)	4.82634e−06 (3.63e−06)	8.92369e−12 (6.02e−13)	**4.11464e−15 (6.02e−16)**
f_{12}	5.21919e−02 (2.94e−04)	5.25646e−03 (7.15e−06)	5.25646e−03 (7.15e−06)	4.34325e−04 (3.69e−05)	5.13084e−03 (3.59e−04)	4.74317e−04 (4.05e−05)	**3.00496e−18 (4.82e−17)**
f_{13}	2.30179e+01 (4.38e−01)	1.35424e+01 (3.67e−02)	1.77582e+01 (6.33e−04)	−4.86485e−01 (1.08e−10)	−1.00864e+00 (1.44e−05)	−1.10554e+00 (6.98e−02)	**−1.14282e+00 (9.02e−05)**
CF1	7.35430e+02 (1.546e+02)	7.36630e+02 (4.326e+01)	7.37321e+02 (7.235e+01)	6.98553e+02 (1.236e+02)	6.98661e+02 (2.123e+02)	7.847894e+02 (3.353e+02)	**6.19227e+02 (6.8341e+01)**
CF2	8.65593e+02 (2.541e+02)	8.54723e+02 (2.482e+01)	8.34774e+02 (1.554e+01)	7.82114e+02 (1.231e+02)	6.70442e+02 (1.133e+02)	6.40562e+02 (2.643e+02)	**5.60543e+02 (9.7837e+01)**
CF3	9.73340e+02 (3.221e+02)	9.13774e+02 (5.689e+02)	9.18563e+02 (4.663e+01)	1.12504e+03 (2.236e+02)	8.16728e+02 (2.836e+02)	8.41423e+02 (2.643e+02)	**6.74823e+02 (5.8471e+01)**

search space, which is usually defined to be symmetric about the origin. In addition, many of the test functions are crafted to have optima at or near the origin, including the test functions for this paper. A uniform distribution of initial population members has two potential biases for such functions. In this paper we have adopted an asymmetrical initialization procedure following the work reported in [54]. The procedure limits the initial process to just a portion of the feasible search space (as shown in the third column of Table IV), which is a region defined to be half the distance from the maximum point along each axis back toward the origin. Consequently, as the number of dimensions is increased, the volume of the initialization space in the asymmetric initialization procedure decreases exponentially as compared to that of the symmetric initialization (whose limits are provided in Table IV).

For the spread spectrum radar code design problem, each variable is randomly initialized in the interval $[0, 2\pi]$. The search was kept confined in this region. On the other hand, for the FMS problem, the initialization range of each of the six variables was kept at [0, 6.35], while the search was constricted in the region $[-6.4, 6.35]$ for all the variables.

VI. NUMERICAL RESULTS AND DISCUSSIONS

A. Comparison of Different DEGL Schemes

In this section we compare the performance of six variants of the proposed DEGL algorithm (with different strategies for tuning the weight factor w) and the DE/target-to-best/1 scheme, which uses only a global neighborhood and may be seen as a special case of the DEGL with $w = 1$ and $\alpha = \beta$. All the seven contestant algorithms in this section use the same population size, the same intial population, and the same stopping criterion (i.e., the same number of maximum FEs). Here the results are shown for $D = 100$ and each run of an algorithm is continued upto 5 000 000 FEs. Since all the algorithms have the same population size ($10 \cdot D$), this corresponds to a maximum of approximately 5000 generations for each problem.

In the self-adaptive scheme (DEGL/SAW) for adjusting w, the weight factor of each vector was randomly initialized, using a uniform distribution, and constrained within [0.05, 0.95]. This range gave fairly good results with DEGL/SAW algorithm.

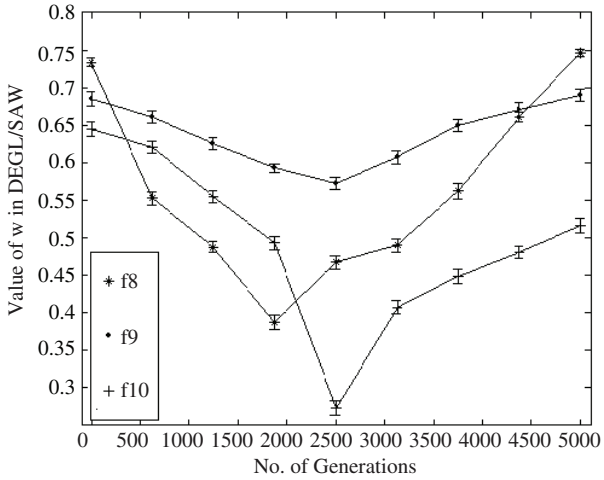
We choose the crossover rate $Cr = 0.9$, and scale factors $\alpha = \beta = F = 0.8$. After some experimentation, we find that a neighborhood size approximately equal to 10% of the population size provides reasonably accurate results for DEGL over nearly all the problems we study here. Hence we stick to a 10% neighborhood size everywhere in this comparative study for DEGL. Section VI-E presents a detailed discussion of the effect of the neighborhood size on DEGL performance.

The mean and the standard deviation (within parentheses) of the best-of-run values for 50 independent runs of each of the five contestant algorithms are presented in Table V for the six hardest benchmark functions functions f_8 to f_{13} (each in 100 dimensions) and also for the three composite functions CF1 to CF3 (each in 10 dimensions), taken from the list of CEC'05 benchmarks [48]. The best solution in each case has been shown in bold. Final accuracy results for all the algorithms studied here have been reported with precision as recorded by the IEEE standard for binary floating-point arithmetic (IEEE 754). Results for relatively easier benchmarks follow a similar trend and have not been included in order to save space.

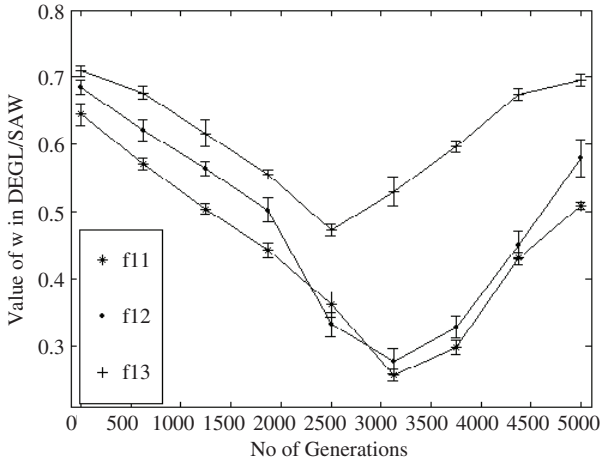
From Table V, it is interesting to see that there are always one or more versions of DEGL that outperform the standard DE/target-to-best/1/bin scheme. This reflects the effectiveness of the incorporation of the hybrid mutation operator in DE. We also note that in all the cases the time-varying weight factors outperform the schemes with fixed weight factor. It is interesting to see that DEGL with a fixed w for all vectors yields final accuracies very close to that produced by the DE/target-to-best/1/bin scheme. However, performance of the *local-only* DEGL with $w = 0$ remains comparable to DEGL with $w = 0.5$ but poorer than the three other DEGL schemes with time-varying weight factor. Most of the runs of DEGL with $w = 0$ fail to converge very near to the global optima within the prescribed number of FEs due to its sluggish behavior during the final stages of the search. This suggests that a judicious tradeoff between the explorative and the exploitative mutation operators is the key to the success of the search-dynamics of DEGL. The self-adaptive DEGL/SAW scheme exhibited very good performance over all the test problems, indicating the ability of DEGL to capture the dynamics of the problem under test and determine the proper weight factor. In Fig. 6 the evolution of the weight factor over successive generations has been shown for the best vector of the median run of DEGL/SAW over functions f_8–f_{13}. The standard deviations have also been plotted at the sampled generations in the same figure.

Very interestingly, Fig. 6 indicates that the general tendency of the evolutionary learning is at first a decrease of the weight factor (favoring exploration at earlier stages) and then increasing the weight factor towards a high value (favoring exploitation at later stages of the search).

In the following sections we report results of comparison between DEGL/SAW and other state-of-the-art DE variants. We exclude the other variants of DEGL to save space and also considering the fact that DEGL/SAW outperformed all other schemes of controlling the weight factor over the selected test suite.



(a) Variation of w for DEGL/SAW over functions f_8 to f_{10}



(b) Variation of w for DEGL/SAW over functions f_{11} to f_{13}

Fig. 6. Self-adaptation characteristics of the best vector of median run for the DEGL/SAW scheme.

B. Comparison of DEGL/SAW With State-of-the-Art DE Variants

In this section, we compare DEGL/SAW with five other DE variants mentioned in Section V-C. The comparative study focuses on four important aspects of all the competitor algorithms: 1) The quality of the final solutions produced by each algorithm, irrespective of the computational time it consumes; 2) The speed of convergence measured in terms of the number of FEs required by an algorithm to reach a predefined threshold value of the objective function; 3) the frequency of hitting the optima (or *success rate*) measured in terms of the number of runs of an algorithm that converge to a threshold value within a predetermined number of FEs; and 4) the issue of scalability, i.e., how the performance of an algorithm changes with the growth of the search-space dimensionality.

The parametric setup for DEGL was kept same as before. For DE/rand/1/bin and DE/target-to-best/1/bin we have taken $F = 0.8, Cr = 0.9$, and $NP = 10 \cdot D$. In the case of DE/rand/1/either-or, we took $p_F = 0.4$ [3]. For NSDE and SADE, the best set of parameters was employed from the relevant literature [31] and [28], respectively. Once set, the same parameters were used over all the tested problems and no further hand tuning was allowed for any of the algorithms.

1) Comparison of Quality of the Final Solution: To judge the accuracy of different DE variants, we first let each of them run for a very long time over every benchmark function, until the number of FEs exceeds a given upper limit (which was fixed depending on the complexity of the problem). The mean and the standard deviation (within parentheses) of the best-of-run values for 50 independent runs of each of the six algorithms are presented in Tables VI, VII, and VIII. Missing standard deviation values in any result table in this paper indicate zero standard deviation. Although the experiments were conducted for $D = 25$, 50, 75, and 100 for functions f_1 to f_{13}, we report here results for 25 and 100 dimensions in order to save space. Please note that the omitted results follow a similar trend as those reported in Tables VI, VII, and VIII.

Since all the algorithms start with the same initial population over each problem instance, we used paired t-tests to compare

TABLE VI
AVERAGE AND THE STANDARD DEVIATION OF THE BEST-OF-RUN SOLUTION FOR 50 INDEPENDENT RUNS AND THE SUCCESS RATE TESTED ON f_1 TO f_8

Function	Dim	Max FEs	Mean best value (Standard deviation)						
			DE/rand/1/bin	DE/target-to-best/1/bin	DE/rand/1/either-or	SADE [27]	NSDE [30]	DEGL/SAW	Statistical significance
f_1	25	5×10^5	6.8594e−29 (4.984e−23)	5.7093e−25 (2.109e−19)	7.3294e−36 (5.394e−34)	4.0398e−35 (3.905e−32)	9.5462e−35 (3.009e−34)	**8.7845e−37 (3.823e−35)**	·
	100	5×10^6	8.4783e−24 (4.664e−22)	2.5693e−23 (3.746e−21)	**4.9382e−26 (4.9382e−25)**	5.8472e−24 (3.8271e−23)	8.3812e−23 (3.925e−25)	3.6712e−25 (4.736e−23)	·
f_2	25	5×10^5	7.5462e−29 (6.731e−29)	5.7362e−25 (4.837e−10	7.4723e−31 (2.736e−34)	8.3392e−26 (4.837e−28)	8.9437e−30 (1.003e−30)	**4.9392e−36 (3.928e−34)**	+
	100	5×10^6	1.6687e−09 (6.77e−10)	3.5273e−06 (1.68e−08)	6.2827e−13 (1.91e−15)	2.6595e−12 (3.36e−14)	9.1395e−10 (3.36e−10)	**6.9982e−14 (1.34e−16)**	+
f_3	25	5×10^5	4.9283e−11 (2.03e−11)	6.2713e−09 (4.82e−10)	5.8463e−24 (4.737e−24)	4.2761e−14 (3.87e−14)	3.0610e−09 (4.22e−10)	**1.2094e−26 (3.827e−25)**	+
	100	5×10^6	6.5712e−10 (2.91e−10)	5.6125e−10 (3.22e−12)	3.4315e−11 (5.07e−12)	4.5641e−10 (5.29e−13)	7.3412e−10 (6.12e−10)	**5.8832e−13 (3.06e−16)**	+
f_4	25	5×10^5	8.3611e−14 (6.37e−13)	5.3711e−10 (9.03e−09)	1.6281e−14 (3.42e−13)	3.0229e−14 (1.37e−15)	2.0936e−11 (1.09e−08)	**4.9932e−15 (1.18e−14)**	+
	100	5×10^6	3.0095e−12 (3.26e−11)	3.0005e−08 (3.69e−09)	9.4442e−13 (3.29e−14)	3.7001e−11 (1.08e−13)	6.0927e−09 (4.45e−08)	**3.5677e−14 (4.55e−13)**	+
f_5	25	5×10^5	9.8372e−23 (4.837e−24)	3.0345e−10 (3.69e−09)	4.9372e−25 (3.726e−21)	**5.6472e−26 (9.367e−24)**	2.6473e−25 (4.536e−25)	6.8948e−25 (4.361e−26)	·
	100	5×10^6	8.4511e−05 (2.748e−05)	2.6183e−01 (1.329e−03)	8.5462e−23 (4.635e−23)	8.6471e−25 (3.782e−24)	5.9208e−08 (2.03e−09)	**1.5463e−25 (7.301e−22)**	·
f_6	25	5×10^5	6.0938e−32 (9.362e−40)	7.6473e−41 (3.827e−37)	2.6839e−45 (3.837e−43)	1.6729e−36 (2.637e−32)	4.0361e−28 (2.949e−34)	**9.5627e−48 (2.732e−45)**	+
	100	5×10^6	3.2387e−14 (2.67e−09)	4.0102e−12 (3.85e−13)	8.3026e−15 (5.51e−16)	6.4897e−21 (3.938e−19)	5.8924e−15 (6.00e−13)	**9.4826e−22 (7.483e−24)**	+
f_7	25	5×10^5	4.9391e−03 (5.92e−04)	9.0982e−03 (2.08e−04)	6.9207e−04 (4.26e−06)	3.7552e−02 (9.02e−03)	4.3482e−03 (6.50e−04)	**1.0549e−07 (2.33e−06)**	+
	100	5×10^6	2.8731e−02 (2.33e−02)	3.3921e−02 (3.32e−02)	4.3332e−03 (5.76e−02)	5.9281e−02 (4.31e−03)	9.8263e−02 (2.90e−03)	**6.9921e−06 (4.56e−05)**	+
f_8	25	5×10^5	−1.0182e+04 (2.83e−04)	−1.0236e+04 (3.81e−05)	**−1.0475e+04 (2.27e−06)**	**−1.0475e+04 (2.27e−06)**	−1.1472e+04 (2.91e−03)	**−1.0475e+04 (3.77e−03)**	NA
	100	5×10^6	−4.18315e+04 (2.83e−04)	−3.9382e+04 (5.83e−06)	−4.18445e+04 (5.22e−05)	−4.18091e+04 (2.49e−06)	−4.18091e+04 (2.49e−06)	**−4.18983e+04 (6.98e−06)**	·

the means of the results produced by best and the second best algorithms (with respect to their final accuracies). The t-tests are quite popular among researchers in evolutionary computing and they are fairly robust to violations of a Gaussian distribution with large number of samples like 50 [55]. In the 10th columns of Tables VI, VII, and VIII we report the statistical significance level of the difference of the means of best two algorithms. Note that here '+' indicates the t value of 49 degrees of freedom is significant at a 0.05 level of significance by two-tailed test, '.' means the difference of means is not statistically significant and 'NA' stands for Not Applicable, covering cases for which two or more algorithms achieve the best accuracy results.

A close inspection of Tables VI–VIII indicates that the performance of the proposed DEGL/SAW algorithm has remained clearly and consistently superior to that of the two classical DE schemes (DE/rand/1/bin and DE/target-to-best/1/bin) as well as the three state-of-the-art DE variants. One may note from Tables VI and VII that for a few relatively simpler test-functions like the Sphere (f_1), Schwefel's problem 2.22 (f_2), 25-dimensional Step function (f_6), generalized Rastrigin's function (f_9), generalized Griewank's function (f_{11}), and the Shekel's family function f_{22}, most of the algorithms end up with almost equal accuracy. Substantial performance differences however, are noticed for the rest of the more challenging benchmark functions and especially for functions with higher dimensions like 100. In the case of the multimodal functions f_8 to f_{13}, the three state-of-the-art DE variants (DE/rand/1/either-or, SADE, and NSDE) and DEGL/SAW outperformed the two classical DE algorithms: DE/rand/1/bin and DE/target-to-best/1/bin. The quality of the solutions produced by the SADE, DE/target-to-best/1/bin, and NSDE algorithm is close to that of the DEGL in a few cases (e.g., the 25-dimensional f_{12}, f_{14}, and the 2-dimensional f_{16} and f_{18} functions).

It is interesting to see that out of the 34 benchmark instances, in 25 cases DEGL outperforms its nearest competitor in a statistically significant fashion. In three cases (f_1 with $D = 100$, f_8 with $D = 25$, f_9 with $D = 100$, and f_{12} with $D = 25$) DE/rand/1/either-or achieved best average accuracy beating DEGL, which remained the second best algorithm. Paired t-tests, however, confirm that the difference

TABLE VII

AVERAGE AND THE STANDARD DEVIATION OF THE BEST-OF-RUN SOLUTION FOR 50 INDEPENDENT RUNS TESTED ON f_9 TO f_{21}

Func	D	Max FEs	Mean best value (Standard deviation)						
			DE/rand/1/bin	DE/target-to-best/1/bin	DE/rand/1/either-or	SADE [27]	NSDE [30]	DEGL/SAW	Statistical significance
f_9	25	5×10^5	1.0453e−03 (8.04e−02)	9.5278e−01 (4.72e−01)	1.7109e−23 (2.726e−24)	6.7381e−24 (3.728e−21)	4.8392e−21 (8.872e−20)	**5.8492e−25 (5.333e−27)**	·
	100	5×10^6	2.1121e−02 (4.86e−03)	6.76249e−01 (4.27e−01)	**8.4719e−23 (9.36e−22)**	5.8824e−21 (4.83e−20)	5.5732e−05 (5.93e−04)	1.7728e−22 (3.88e−20)	·
f_{10}	25	5×10^5	4.1902e−08 (3.36e−08)	9.8035e−03 (6.80e−03)	6.9437e−15 (4.86e−15)	7.8343e−15 (2.85e−15)	5.9749e−10 (3.2231e−04)	**5.9825e−23 (1.00e−22)**	+
	100	5×10^6	7.6687e−05 (6.767e−05)	6.76249e−01 (4.237e−01)	6.9398e−13 (4.852e−13)	3.0665e−12 (5.125e−13)	4.1232e−05 (7.496e−06)	**8.52742e−17 (1.365e−15)**	+
f_{11}	25	5×10^5	6.8318e−22 (3.837e−25)	7.94504e−07 (8.03e−08)	3.0905e−34 (7.462e−34)	1.8274e−28 (7.682e−29)	7.9318e−26 (3.774e−28)	**2.9931e−36 (4.736e−35)**	+
	100	5×10^6	2.1962e−10 (8.45e−11)	5.27284e−05 (4.63e−07)	3.2928e−12 (2.77e−13)	8.9569e−13 (1.02e−14)	5.0392e−10 (4.29e−08)	**4.11464e−15 (6.02e−16)**	+
f_{12}	25	5×10^5	7.0931e−16 (6.22e−15)	2.8962e−13 (2.25e−10)	**5.1469e−32 (4.22e−29)**	9.3718e−24 (6.193e−28)	5.8471e−21 (3.728e−21)	7.2094e−27 (4.838e−28)	+
	100	5×10^6	4.2455e−10 (2.96e−09)	5.21919e−02 (2.94e−04)	2.9137e−15 (4.30e−16)	2.8417e−15 (1.45e−14)	4.8923e−12 (8.45e−13)	**3.00496e−18 (4.82e−17)**	+
f_{13}	25	5×10^5	−1.12836e+00 (4.46e−08)	−4.86485e−01 (1.08e−10)	−1.1382e+00 (3.29e−10)	−1.14280e+00 (3.85e−07)	−1.14276e+00 (3.44e−09)	**−1.14282e+00 (5.81e−06)**	+
	100	5×10^6	2.0621e−02 (5.58e−03)	5.81493e−01 (1.08e−02)	2.19321e+00 (3.32e−01)	−1.1014e+00 (6.98e−03)	−1.10266e+00 (7.84e−05)	**−1.14282e+00 (9.02e−05)**	+
f_{14}	2	5×10^5	9.9813292e−01 (5.42e−10)	9.9860553e−01 (4.26e−03)	**9.9800390e−01 (1.13e−16)**	9.9800884e−01 (1.93e−18)	9.9860346e−01 (1.07e−02)	**9.9800390e−01 (1.15e−18)**	NA
f_{15}	4	5×10^5	4.0361420e−04 (2.81e−04)	4.8242655e−04 (6.41e−05)	3.6734442e−04 (5.13e−05)	3.7044472e−04 (9.82e−07)	3.7320963e−04 (4.33e−03)	**3.7041849e−04 (2.11e−09)**	+
f_{16}	2	5×10^5	−1.029922e+00 (1.82e−08)	−1.031149e+00 (2.44e−08)	−1.031242e+00 (4.98e−06)	**−1.031630e+00 (9.73e−12)**	**−1.031630e+00 (3.33e−10)**	**−1.031630e+00 (4.28e−10)**	NA
f_{17}	2	5×10^5	3.9788959e−01 (6.39e−06)	3.9789793e−01 (6.28e−07)	3.9788915e−01 (6.82e−06)	3.9788783e−01 (2.68e−06)	3.9788392e−01 (4.09e−06)	**3.9788170e−01 (8. 54e−04)**	·
f_{18}	2	5×10^5	3.0834435e+00 (4.73e−01)	3.146090e+00 (5.83e−01)	**3.000000e+00**	**3.000000e+00**	**3.000000e+00**	**3.000000e+00**	NA
f_{19}	2	5×10^5	−1.0042985e+01 (4.32e−05)	−6.840054e+00 (3.87e+00)	−1.010974e+01 (2.67e−05)	−1.015050e+01 (4.59e−04)	−1.014876e+01 (3.57e−03)	**−1.015323e+01 (7.34e−08)**	+
f_{20}	2	5×10^5	−1.0400382e+01 (8.54e−10)	−1.040073e+01 (4.53e−08)	−1.040068e+01 (9.24e−10)	−1.040189e+01 (6.94e−05)	−1.040089e+01 (3.00e−08)	**−1.040295e+01 (5.93e−04)**	+
f_{21}	2	5×10^5	−1.0536082e+01 (2.87e−03)	−7.023436e+01 (4.78e−05)	−1.0474381e+01 (6.88e−03)	−1.0536234e+01 (2.46e−06)	−1.023436e+01 (2.72e−02)	**−1.053641e+01 (3.90e−08)**	+

TABLE VIII

AVERAGE AND THE STANDARD DEVIATION OF THE BEST-OF-RUN SOLUTION FOR 50 INDEPENDENT RUNS TESTED ON COMPOSITE FUNCTIONS CF1 TO CF3 TAKEN FROM THE CEC'05 BENCHMARKS

Func	D	Max FEs	Mean best value (Standard deviation)							
			DE/rand/1/bin	DE/target-to-best/1/bin	DE/rand/1/either-or	SADE [27]	NSDE [30]	DEGL/SAW ($Cr = 0.9$)	DEGL/SAW ($Cr = 1$)	Statistical Significance
CF1	10	5×10^6	6.400300e+02 (2.3428e+02)	7.92834e+02 (3.0922e+02)	6.280932e+02 (2.0703e+02)	5.334983e+02 (3.9672e+01)	6.230469e+02 (4.5297e+01)	6.19227e+02 (6.8341e+01)	**5.03826e+02 (4.0995e+01)**	+
CF2	10	5×10^6	6.340356e+02 (2.6635e+02)	7.993241e+02 (4.6723e+02)	6.157323e+02 (9.8836e+01)	5.15284e+02 (2.0784e+02)	7.198302e+02 (4.8735e+02)	7.60543e+02 (9.7837e+01)	**4.18542e+02 (8.9984e+01)**	+
CF3	10	5×10^6	8.56392e+02 (9.4863e+01)	1.12873e+03 (6.7394e+01)	7.48427e+02 (5.8473e+01)	7.88492e+02 (4.4342e+01)	8.93824e+02 (3.8764e+01)	6.74823e+02 (5.8471e+01)	**4.76239e+02 (3.7842e+01)**	+

of their means is not statistically significant for f_1 and f_9 in 100 dimensions.

As long as $Cr < 1$, DEGL will not be rotationally invariant, i.e., its performance will depend on the orientation of the coordinate system in which vectors are evaluated [3]. Since the composite functions CF1, CF2, and CF3 are rotated in nature, we also solve them using DEGL/SAW with $Cr = 1$. Table V shows that this rotationally invariant version of

TABLE IX

NO. OF SUCCESSFUL RUNS, MEAN NO. OF FES AND STANDARD DEVIATION (IN PARENTHESES) REQUIRED TO CONVERGE TO THE CUT-OFF FITNESS OVER THE SUCCESSFUL RUNS FOR FUNCTIONS f_8 TO f_{11}

Function	D	Threshold objective function value	No. of successful runs, mean no. of FEs, and (standard deviation) required to converge to the prescribed threshold fitness					
			DE/rand/1/bin	DE/target-to-best/1/bin	DE/rand/1/either-or	SADE [28]	NSDE [31]	DEGL/SAW
f_1	25	1.00e−20	50, 109 372.5 (4773.28)	50, 376 421.20 (10 983.46)	50, 98 204.24 (2942.87)	50, 104 982.64 (5182.67)	50, 105 727.80 (3427.57)	**50, 91 935.40 (3888.45)**
	100	1.00e−20	50, 687 322.24 (12 153.67)	50, 1033 567.40 (58 391.56)	**50, 403 922.56 (3814.25)**	50, 738 720.84 (28 731.88)	50, 565 382.24 (2827.56)	50, 498 521.54 (10 832.41)
f_2	25	1.00e−20	50, 266 371.40 (31 923.45)	50, 417 382.80 (23 221.45)	50, 198 342.22 (3421.68)	50, 306 742.28 (18 534.55)	50, 300 371.48 (9034.26)	**50, 157 234.76 (4451.72)**
	100	1.00e−20	13, 2034 583.46 (18 235.48)	6, 2935 411.45 (21 893.56)	28, 1062 744.69 (44 583.41)	23, 1257 362.57 (3417.34)	20, 1782 336.10 (36 710.05)	**34, 978 357.83 (23 727.45)**
f_3	25	1.00e−20	12, 298 341.67 (24 376.27)	5, 378 392.20 (34 621.22)	50, 123 682.54 (63 827.06)	16, 296 473.93 (27 268.45)	7, 363 986.82 (52 741.78)	**50, 110 528.68 (13 873.51)**
	100	1.00e−20	13, 2638 224.33 (57 398.21)	10, 4562 312.70 (17 372.68)	15, 2745 218.47 (37 123.69)	14, 2696 359.51 (14 225.47)	13, 2671 982.93 (46 188.26)	**18, 2063 728.48 (27 351.57)**
f_4	25	1.00e−20	16, 376 291.47 (12 836.48)	8, 467 262.25 (26 111.78)	19, 309 309.52 (17 829.46)	17, 292 478.83 (8372.58)	11, 408 291.79 (26 721.77)	**21, 294 812.82 (36 173.52)**
	100	1.00e−20	19, 3174 782.17 (17 283.49)	3, 4453 782.67 (18 253.58)	22, 3228 379.27 (4824.81)	17, 3139 382.38 (33 728.42)	5, 4140 835.40 (22 338.86)	**25, 2263 976.44 (28 371.46)**
f_5	25	1.00e−20	50, 356 253.38 (82 732.33)	17, 478 290.91 (57 263.72)	50, 315 633.92 (47 192.57)	**50, 267 319.74 (23 556.24)**	50, 299 831.26 (48 382.57)	50, 338 279.08 (28 846.37)
	100	1.00e−20	1, 3398272	0	50, 3067 263.78 (56 723.83)	50, 2844 738.62 (66 729.38)	3, 4563 742.33 (128 123.57)	**50, 2709 313.82 (12 338.11)**
f_6	25	1.00e−20	50, 189 367.38 (83 412.84)	50, 132 676.28 (6769.48)	50, 122 845.64 (7378.36)	50, 173 490.18 (7638.46)	50, 235 177.72 (13 223.94)	**50, 96 832.24 (4631.66)**
	100	1.00e−20	18, 2357 827.59 (33 253.68)	16, 3098 277.26 (83 921.47)	20, 2299 868.50 (27 632.58)	47, 1824 359.69 (27 733.61)	25, 3622 719.24 (47 378.19)	**50, 1238 461.98 (36 278.64)**
f_7	25	1.00e−20	0	0	2, 467 236.50 (43 827.83)	0	0	**4, 417 823.25 (27 192.82)**
	100	1.00e−20	0	0	1, 3689 267.48	0	0	**3, 3163 563.67 (78 282.58)**
f_8	25	−1.0410e+04	12, 19 817.50 (8723.837)	17, 13 039.65 (336.378)	50, 12 410.04 (1201.278)	50, 9887.50 (822.281)	32, 37 847.82 (4431.90)	**50, 9492.64 (871. 76)**
	100	−4.1800e+04	3, 359 834.33 (4353.825)	1, 51729	13, 133 282.73 (5362.366)	25, 363 291.80 (2338.944)	20, 2178 283.50 (24 332.78)	**35, 39 928.45 (231.627)**
f_9	25	1.00e−20	19, 345 328.18 (41 128.91)	13, 46 843.92 (34 521.372)	50, 330 272.74 (3642.289)	50, 195 823.88 (4249.392)	44, 345 654.73 (326.84)	**50, 87 148.34 (1325.72)**
	100	1.00e−20	5, 1840 322.80 (3852.196)	2, 2022 275.50 (27 327.24)	50, 838 932.48 (23 677.66)	50, 744 938.28 (34 147.928)	16, 3290 384.57 (53 209.58)	**50, 539 282.72 (26 547.09)**
f_{10}	25	1.00e−20	14, 226 816.89 (44 721.76)	4, 412 675.25 (16 834.37)	34, 238 372.74 (32 325.67)	32, 236 290.86 (15 533.08)	26, 287 812.83 (14 039.54)	**50, 224 883.78 (13 212.87)**
	100	1.00e−20	13, 1873 625.56 (29 123.902)	2, 4486 372.50 (98 273.57)	15, 1782 210.66 (72 233.371)	13, 1065 920.64 (24 383.71)	7, 2082 983.84 (81 744.84)	**27, 925 628.73 (7823.28)**
f_{11}	25	1.00e−20	50, 333 948.52 (12 314.821)	6, 356 061.52 (11 300.97)	50, 225 092.84 (12 123.19)	50, 316 382.04 (35 338.83)	50, 369 283.71 (45 478.88)	**50, 196 258.22 (14 235.83)**
	100	1.00e−20	26, 1887 635.65 (44 612.34)	12, 2833 416.96 (17 218.06)	29, 2633 782.74 (10 217.26)	34, 1936 287.62 (14 235.37)	27, 2235 653.56 (30 362.67)	**43, 1627 092.58 (11 217.31)**

TABLE X

NO. OF SUCCESSFUL RUNS, MEAN NO. OF FES AND STANDARD DEVIATION (IN PARENTHESES) REQUIRED TO CONVERGE TO THE CUT-OFF FITNESS OVER THE SUCCESSFUL RUNS FOR FUNCTIONS f_{12} TO f_{21}

Func	D	Threshold objective function value	No. of successful runs, mean no. of FEs, and (standard deviation) required to converge to the prescribed threshold fitness					
			DE/rand/1/bin	DE/target-to-best/1/bin	DE/rand/1/either-or	SADE [28]	NSDE [31]	DEGL/SAW
f_{12}	25	1.00e−20	35, 294 584.44 (22 563.378)	30, 3472 185.67 (13 382.229)	42, 209 372.87 (12 742.03)	50, 126 574.64 (16 833.89)	46, 478 732.05, (3884.04)	**50, 150 039.62 (4831.28)**
	100	1.00e−20	8, 3122 658.25 (62 922.84)	5, 3908 138.80 (13 937.383)	23, 2664 722.53 (47 212.38)	20, 1637 409.40 (18 219.526)	10, 2673 864.70 (53 121.65)	**27, 1436 190.89 (13627. 82)**
f_{13}	25	−1.1428e+00	13, 230 372.52 (7313.297)	3, 428 023.33 (84 517.371)	26, 237 639.09 (14 573.96)	42, 213 739.78 (12 347.391)	24, 738742. 34 (24 322.82)	**48, 121 940.72 (33 398.90)**
	100	−1.1428e+00	1, 3328426	0	0	14, 1702 654.85 (21 743.57)	25, 1283 665.44 (9487.37)	**28, 398 493.74 (25 134.38)**
f_{14}	2	9.9800390e−01	19, 94 233.57 (2312.57)	14, 89 371.53 (1409.26)	46, 68 392.37 (5231.48)	27, 84 032.58 (3842.53)	15, 77 362.94 (4437.28)	**47, 67 823.84 (3725.36)**
f_{15}	4	3.705e−04	0	0	13, 58 935.28 (3822.72)	33, 68 293.46 (2219.58)	20, 73 821.05 (6319.48)	**41, 65 783.38 (1749.51)**
f_{16}	2	−1.03170e+00	32, 83 920.68 (2124.56)	37, 98 529.61 (1098.59)	27, 83 782.79 (1271.47)	50, 77 129.34 (3731.63)	50, 71 036.28 (1211.48)	**50, 67 382.39 (1726.49)**
f_{17}	2	3.980e−01	41, 103 273.57 (2231.68)	43, 79 382.42 (907.31)	43, 75 823.45 (3281.68)	38, 78 939.37 (1325.46)	47, 84 983.94 (2258.10)	**49, 73 727.83 (4308.58)**
f_{18}	2	3.00e+00	21, 67 392.59 (3381.62)	23, 77 539.42 (4839.86)	**50, 89 482.78 (3238.56)**	**50, 79 035.28 (3381.98)**	**50, 80 382.70 (419.49)**	**50, 69 837.62 (1724.08)**
f_{19}	2	−1.01550e+01	23, 109 372.48 (3341.67)	34, 98 922.93 (3212.68)	44, 68 672.70 (1332.67)	41, 67 478.37 (2001.83)	37, 79 820.42 (1692.78)	**46, 58 372.96 (3827.58)**
f_{20}	2	−1.04500e+01	35, 84 721.07 (3412.39)	42, 107 482.69 (10 824.57)	48, 58 373.47 (2221.680	47, 48 372.83 (2294.83)	44, 85 933.58 (3329.74)	**50, 56 098.08 (3187.44)**
f_{21}	2	−1.05500e+01	26, 86 743.93 (6983.07)	30, 85 999.67 (2901.83)	32, 84 892.66 (2319.59)	46, 68 492.69 (2326.09)	23, 100 232.67 (3721.78)	**49, 67 583.93 (3317.58)**

TABLE XI

NO. OF SUCCESSFUL RUNS, MEAN NO. OF FES AND STANDARD DEVIATION (IN PARENTHESES) REQUIRED TO CONVERGE TO THE CUT-OFF FITNESS OVER THE SUCCESSFUL RUNS FOR COMPOSITE FUNCTIONS CF1 TO CF3

Func	D	Threshold objective function value	No. of successful runs, mean no. of FEs, and (standard deviation) required to converge to the prescribed threshold fitness						
			DE/rand/1 /bin	DE/target-to-best/1 /bin	DE/rand/1/ either-or	SADE [28]	NSDE [31]	DEGL/SAW ($Cr = 0.9$)	DEGL/SAW ($Cr = 1$)
CF1	10	8.10e+02	34, 2683 073.04 (45 214.48)	19, 3835 238.75 (18 183.95)	42, 637 222.35 (39357. 23)	50, 1823 847.64 (52 932.821)	12, 624 732.56 (35 330.493)	36, 1707 873.04 (13 434.482)	**50, 1645 938.75 (18 843.905)**
CF2	10	8.10e+02	33, 530 857.85 (13 439.09)	21, 2539 841.89 (87 438.490)	25, 942 325.40 (3173.74)	25, 818 472.16 (7384.492)	37, 510 932.79 (3438.473)	36, 1230 857.85 (13 139.409)	**39, 83 401.86 (5438.46)**
CF3	10	1.20e+03	17, 3645 817.50 (95 823.83)	17, 4834 039.65 (35 336.78)	41, 1597 232.03 (37 811.28)	40, 196 887.50 (12 372.28)	24, 3139 492.64 (54 431.26)	45, 149 817.56 (2339.37)	**50, 913 039.68 (3576.78)**

DEGL performs significantly better on the composite test functions as compared to the DEGL with $Cr = 0.9$. However, the performance over the 21 traditional benchmarks (which are unrotated) is nearly the same for both the versions. In order to save space we have not shown the results of DEGL/SAW with $Cr = 1$ in Tables VI and VII.

2) Comparison of the Convergence Speed and Success Rate: In order to compare the speeds of different algorithms, we

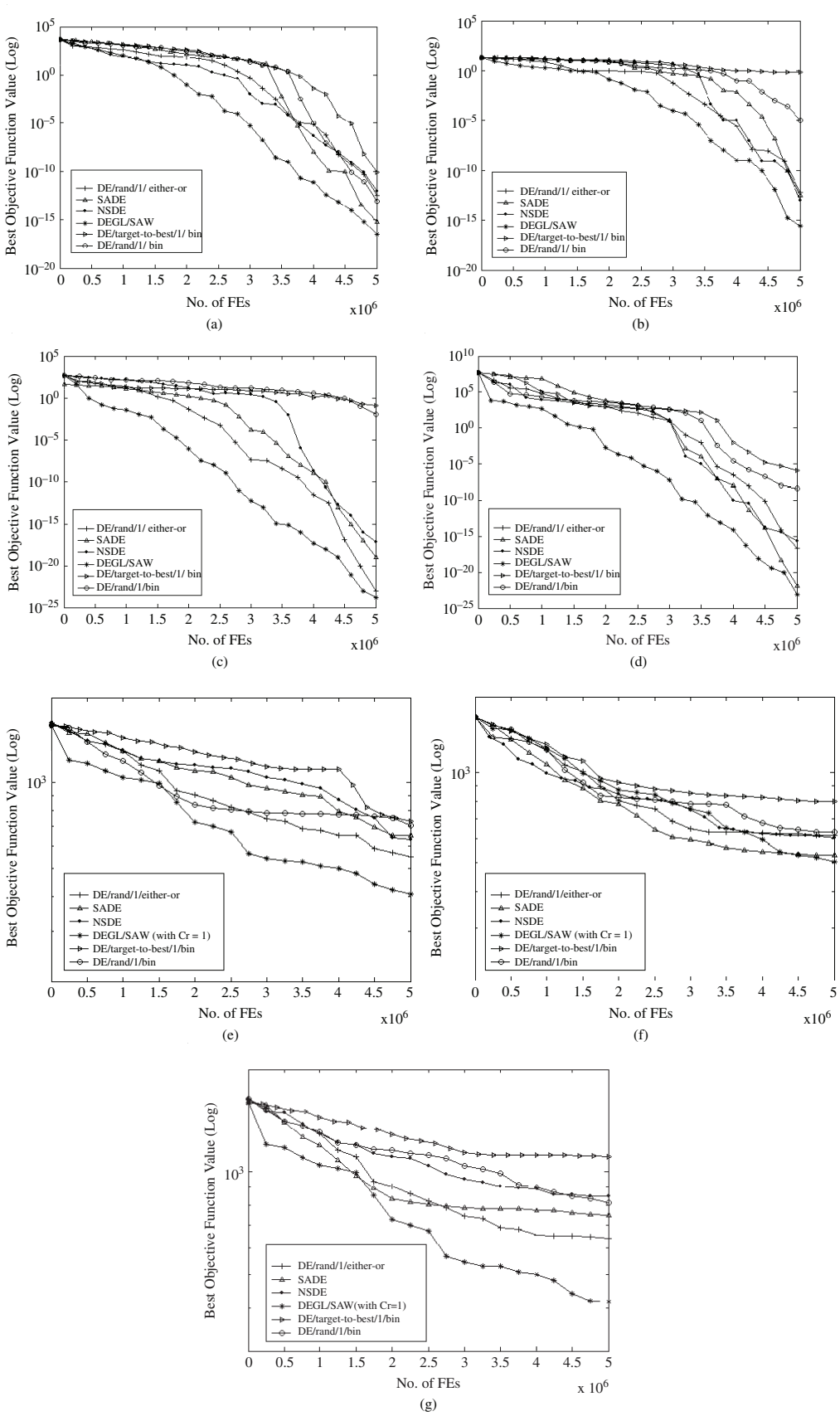


Fig. 7. Progress towards the optimum solution for median run of six algorithms over seven difficult test functions. (a) Generalized Ackley's function (f_{10}). (b) Generalized Griewnk's function (f_{11}). (c) Generalized Rastrigin's Function (f_9). (d) Generalized Rosenbrock's function (f_5). (e) Composite function CF1. (f) Composite function CF2. (g) Composite function CF3.

select a threshold value of the objective function for each benchmark problem. For functions with minima at zero, this threshold is at 10^{-20}. To obtain an unbiased comparative performance, for other functions, this value is chosen to be somewhat larger than the minimum objective function value found by each algorithm in Tables VI, VII, and VIII. We run each algorithm on a function and stop as soon as the best fitness value determined by the algorithm falls below the predefined threshold. Then we note the number of FEs the algorithm takes. A lower number of FEs corresponds to a faster algorithm. Tables IX, X, and XI report the number of runs (out of 50) that managed to find the optimum solution (within the given tolerance) as well as the mean number of FEs and standard deviations (within parenthesis) required by the algorithms to converge within the prescribed threshold value. Entries marked as 0 indicate that no runs of the corresponding algorithm converged below the threshold objective function value. Missing values of standard deviation in these tables also indicate a zero standard deviation.

Tables VI and IX indicate that, not only does DEGL/SAW yield the most accurate results for nearly all the benchmark problems, but it does so consuming the least amount of computational time. In addition, the number of runs that converge below a prespecified cut-off value is also greatest for DEGL over most of the benchmark problems covered here. This indicates the higher robustness (i.e., the ability to produce similar results over repeated runs on a single problem) of the algorithm as compared to its other four competitors. Usually in the community of stochastic search algorithms, robust search is weighted over the highest possible convergence rate [56], [57].

The convergence characteristics of seven difficult test functions are shown in Fig. 7 in terms of the fitness value of the median run of each algorithm. All the graphs except for the composite functions CF1 to CF3 have been drawn for $D = 100$ dimensions. Convergence graphs for the composite functions appear for $D = 10$ dimensions.

3) Scalability Comparison: Performance of most of the evolutionary algorithms (including DE and PSO) deteriorates with the growth of the dimensionality of the search space. Increase of dimensions implies a rapid growth of the hypervolume of the search space and this in turn slows down the convergence speed of most of the global optimizers. Here we show how the performance of the six DE variants scale against the growth of dimensions from 25 to 100. Fig. 8 shows the scalability of the six algorithms over four difficult test functions - how the average computational cost (measured in number of FEs required to yield a threshold fitness value) to find the solution varies with an increase in the dimensionality of the search space.

We note that the computational cost of both DEGL/SAW and SADE (to yield a given accuracy) increases most sluggishly with the search space dimensionality for the following test-functions: f_5, f_{10}, f_{11}, and f_9.

C. Comparison With Other State-of-the-Art Evolutionary Techniques

In this section we compare the performance of DEGL/SAW with that of four state-of-the-art evolutionary and swarm-based optimization techniques, well-known as CPSO-H [38], IPOP-CMA-ES [58], MA-S2 [59], and G3 with PCX [60]. Below we briefly describe each of these algorithms.

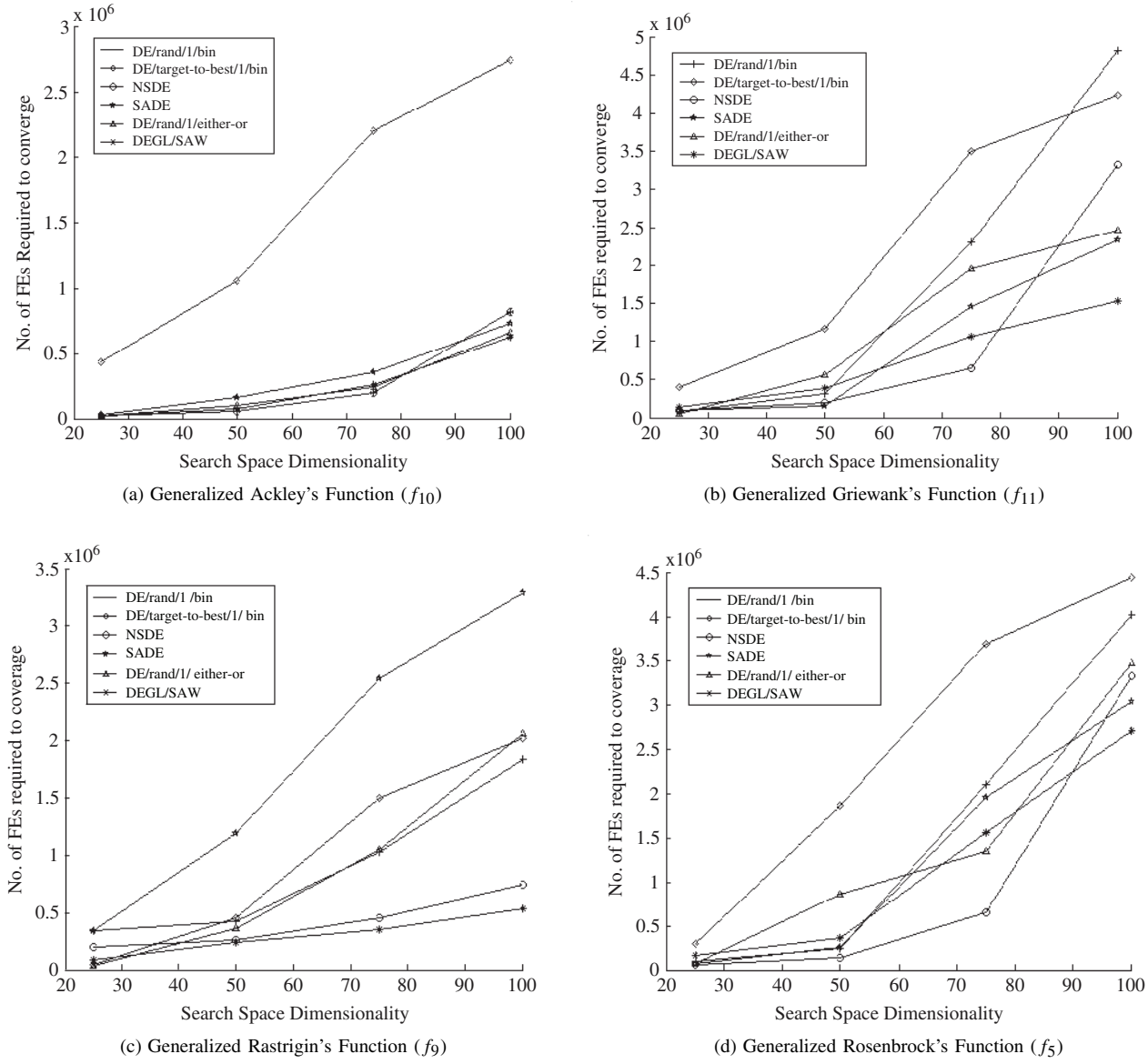
1) CPSO-H: van den Bergh and Engelbrecht proposed a cooperative particle swarm optimizer (CPSO) in [36]. Although CPSO uses one-dimensional (1-D) swarms to search each dimension separately, the results of these searches are integrated by a global swarm to significantly improve the performance of the original PSO on multimodal problems. The CPSO-H algorithm uses a hybrid swarm, consisting of a maximally split cooperative swarm (D one-dimensional swarms for one D-dimensional parameter vector) and a plain swarm. Both components employ identical values for the acceleration coefficients ($C_1 = C_2 = 1.49$) and the inertial factor ω decreasing linearly with time. They use a maximum velocity $\vec{V}_{\max}$ clamped to the search domain [38].

2) IPOP-CMA-ES: Covariance matrix adaptation evolution strategy (CMA-ES) [61], [62] is an evolutionary strategy that uses informed mutation based on local structural information, but does not directly bias its search motion toward other individuals of the population. Auger and Hansen have recently proposed a restart CMA-ES [58], where the population size is increased (IPOP) for each restart. By increasing the population size, the search characteristic becomes more global after each restart. This variant is named IPOP-CMA-ES.

3) MA-S2: Memetic algorithms (MAs) [63], [64] are based on the hybridization of genetic algorithm (GA) with local search (LS) techniques. In this paper, MA-S2 [59] stands for an adaptive Meta-Lamarckian learning-based MA that employs a stochastic approach (the biased roulette wheel strategy) making use of the knowledge gained online to select a suitable local method with the GA.

4) G3 with PCX: The main research effort in the field of real parameter GA is more or less focussed on the design of efficient recombination operators used to create offspring from parent solutions. Deb *et al.* [60] proposed a generic parent-centric recombination scheme (PCX) and integrated it with a steady state, elite preserving, scalable, and computationally fast population alteration model of the GA, which they named the G3 (generalized generation gap) model. Their results indicate that the G3 model with PCX can outperform many other existing GA models when tested on the standard benchmark functions.

We employ the best parametric set-up for all these algorithms as prescribed in their respective sources. The mean and the standard deviation (within parentheses) of the best-of-run values of 50 independent runs for each algorithm have been presented in Tables XII and XIII. In order to save space, we report only the hardest problem instances (multidimensional functions with $D = 100$) in these tables. The algorithms compared in this section have different population sizes and also differ in their initial population structure. Thus to test the statistical significance of the results, we used two-tailed unpaired t tests between the two best algorithms. The results of t test have been indicated in the 9-th column of Table XII and 10-th column of Table XIII. Note that here '+' indicates the t value of 98 degrees of freedom is significant within a 95% confidence interval by two-tailed test, '.' means the difference



(a) Generalized Ackley's Function (f_{10})

(b) Generalized Griewank's Function (f_{11})

(c) Generalized Rastrigin's Function (f_9)

(d) Generalized Rosenbrock's Function (f_5)

Fig. 8. Variation of mean number of FEs required for convergence to predefined threshold accuracy with increase in dimensionality of the search space.

of means is not statistically significant and 'NA' stands for Not Applicable, covering cases in which two or more algorithms achieve the best accuracy results.

These simulation results show that DEGL/SAW is superior to all the other algorithms in terms of the average final accuracy over 12 cases reported in Table XII and two cases in Table XIII. DEGL/SAW yields results comparable to two or more algorithms for six cases in Table XII. It is interesting to see that out of the 12 cases in Table XII, where DEGL/SAW was able to beat all its contestant algorithms, for nine instances the difference between the means of DEGL/SAW and its nearest competitor is statistically significant. From Table XII, we find that CPSO-H was able to outperform DEGL/SAW (and all the other contestants) over the 100-dimensional Schwefel's problem 1.2 (f_3) and IPOP-CMA-ES alone achieved the greatest accuracy for the 100-dimensional generalized penalized function (f_{12}) beating DEGL/SAW. For f_3, DEGL/SAW remained the third best algorithm (after CPSO-H and G3 with PCX) while for the f_{12} function, it secured the second place in terms of final accuracy. However, the last column of Table XII shows the difference of means of DEGL/SAW and IPOP-CMA-ES is *not* statistically significant in the case of the f_{12} function.

For lower dimensional multimodal functions f_{14} to f_{21}, almost all the algorithms end up with nearly equal levels of final accuracy, f_{19} being an exception where DEGL/SAW appeared to perform significantly better as compared to all other algorithms. For the higher dimensional and multimodal functions f_8 to f_{13}, however, CPSO-H and IPOP-CMA-ES remained as the toughest competitor of DEGL/SAW. Note that over these functions DEGL/SAW remained statistically better as compared to the MA-S2 algorithm, which also employs local search strategies in an adaptive fashion with GA. Except for the generalized penalized function f_{12}, DEGL/SAW met or beat the IPOP-CMA-ES over all other multimodal functions in 100 dimensions. The final accuracy provided by DEGL/SAW

TABLE XII

AVERAGE AND STANDARD DEVIATION OF THE BEST-OF-RUN SOLUTIONS FOR 50 INDEPENDENT RUNS AND THE SUCCESS RATE TESTED ON f_1 TO f_{21}

Func	D	Max FEs	Mean best value (Standard deviation)					
			CPSO-H	IPOP-CMA-ES	MA-S2	G3 with PCX	DEGL/SAW	Statistical significance
f_1	100	5×10^6	6.5635e−22 (7.234e−28)	9.6853e−23 (7.232e−26)	7.5364e−22 (3.454e−25)	2.8002e−20 (6.467e−14)	**8.3812e−23 (3.925e−25)**	·
f_2	100	5×10^6	7.4164e−08 (6.225e−07)	2.7429e−03 (1.648e−07)	6.2899e−04 (1.91e−15)	2.6595e−06 (3.36e−10)	**9.1395e−10 (3.36e−10)**	+
f_3	100	5×10^6	**3.5712e−23 (7.239e−22)**	2.5358e−08 (1.923e−09)	8.0005e−07 (8.947e−05)	3.7659e−10 (2.596e−10)	9.7852e−10 (6.132e−08)	+
f_4	100	5×10^6	6.5132e−13 (1.795e−16)	1. 7685e−12 (4.949e−06)	4.8865e−12 (2.209e−13)	7.4823e−13 (3.773e−09)	**3.7068e−14 (1.08e−12)**	·
f_5	100	5×10^6	1.5041e−01 (9.423e−01)	6.0499e−22 (8.345e−24)	1.5639e−20 (2.700e−20)	5.7778e−18 (2.233e−19)	**1.5463e−25 (7.301e−22)**	+
f_6	100	5×10^6	1.4532e−15 (1.713e−16)	2.1052e−20 (8.691e−21)	1.4455e−13 (3.938e−11)	7.0054e−17 (2.644e−14)	**8.6493e−22 (8.483e−23)**	+
f_7	100	5×10^6	8.5829e−13 (1.492e−03)	2.9890e−03 (7.086e−01)	9.6648e−05 (2.331e−09)	1.7984e−02 (6.834e−03)	**6.9921e−06 (4.56e−05)**	+
f_8	100	5×10^6	−4.0572e+04 (9.481e−06)	−4.18783e+04 (1.129e−04)	−4.18774e+04 (4.227e−05)	−4.03386e+04 (2.349e−05)	**−4.18983e+04 (6.98e−06)**	+
f_9	100	5×10^6	1.7382e−01 (4.093e−02)	9.24702e−21 (4.324e−21)	7.32562e−04 (2.781e−05)	5.92381e−03 (3.779e−04)	**1.7728e−22 (3.838e−23)**	+
f_{10}	100	5×10^6	1.7725e−12 (2.489e−13)	8.85280e−17 (7.638e−14)	3.71596e−09 (9.328e−08)	3.47432e−10 (7.146e−09)	**3.52742e−17 (1.365e−15)**	+
f_{11}	100	5×10^6	2.5361e−02 (7.2281e−03)	3.67528e−14 (6.932e−14)	1.56794e−13 (3.6433e−09)	8.92369e−11 (8.157e−15)	**4.11464e−15 (6.02e−16)**	+
f_{12}	100	5×10^6	4.2042e−10 (6.955e−11)	**4.45366e−19 (3.634e−16)**	2.75934e−09 (8.359e−06)	6.86492e−04 (8.035e−03)	8.00496e−19 (4.82e−17)	·
f_{13}	100	5×10^6	**−1.142822e+00 (9.472e−06)**	**−1.142822e+00 (1.342e−03)**	−1.00864e+00 (1.44e−05)	−1.10967e+00 (8.345e−01)	**−1.142823e+00 (9.032e−05)**	NA
f_{14}	2	5×10^6	**9.9800390e−01 (7.228e−16)**	**9.9800390e−01 (2.673e−16)**	9.9800400e−01 (9.373e−09)	**9.9800390e−01 (1.138e−16)**	**9.9800390e−01 (1.15e−18)**	NA
f_{15}	4	5×10^6	3.706461e−04 (1.551e−06)	**3.7041849e−04 (4.837e−10)**	3.706851e−04 (2.558e−05)	4.156548e−04 (2.981e−04)	**3.7041849e−04 (2.11e−09)**	NA
f_{16}	2	5×10^6	**−1.031630e+00 (7.236e−11)**	**−1.031630e+00 (3.668e−11)**	−1.031628e+00 (4.538e−08)	**−1.031630e+00 (2.548e−09)**	**−1.031630e+00 (1.749e−10)**	NA
f_{17}	2	5×10^6	3.9788231e−01 (2.683e−06)	**3.9788170e−01 (1.260e−08)**	3.9788794e−01 (7.638e−06)	3.9788396e−01 (6.039e−06)	**3.9788170e−01 (8. 544e−04)**	NA
f_{18}	2	5×10^6	**3.000000e+00**	**3.000000e+00**	**3.000000e+00**	**3.000000e+00**	**3.000000e+00**	NA
f_{19}	2	5×10^6	−1.015306e+00 (2.453e−06)	−1.015314e+01 (8.071e−07)	−1.015058e+01 (1.593e−06)	−1.014888e+01 (5.568e−01)	**−1.015323e+01 (7.341e−08)**	+
f_{20}	2	5×10^6	−1.040236e+01 (3.116e−06)	−1.040293e+01 (7.974e−10)	−1.040125e+01 (1.944e−05)	−1.040089e+01 (3.00e−08)	**−1.040295e+01 (5.923e−04)**	·
f_{21}	2	5×10^6	−1.053427e+01 (1.593e−08)	**−1.053641e+01 (6.049e−07)**	−1.053669e+01 (1.446e−03)	−1.023386e+01 (9.638e−02)	**−1.053641e+01 (3.90e−08)**	NA

improves significantly as compared to all other algorithms for three hardest unimodal functions: the generalized Rosenbrock's function (f_5), the discontinuous step function (f_6), and the noisy quartic function (f_7).

The convergence characteristics of the contestant algorithms over the six hardest test functions have been shown in Fig. 9 in terms of the objective function value of the median run of each algorithm. For the step function, characterized by plateaus and discontinuity, DEGL/SAW maintained a steady convergence rate that finally finished at the lowest objective function value, while the local search-based MA-S2 showed a much slower convergence. Usually a local search method that relies on geographical neighborhoods performs poorly on the step function because the algorithm mainly searches in a relatively small local neighborhood. On the other hand, DEGL employs a geographically randomized neighborhood structure (*local* only in the sense of vector indices), and the individuals can make longer jumps enabling them to move from one plateau to a lower one with relative ease.

Fig. 9 reveals that for Ackley (f_{10}), Rastrigin (f_9), and Griewank (f_{11}), as well as harder composite functions CF1 and CF2, initially CPSO-H and IPOP-CMA-ES converge at the quickest rate among all the algorithms. However, in the neighborhood of the global optima, DEGL/SAW overtakes both of them, attaining greater final accuracy. The composite function CF1 appears as an exception to this trend (that is also exhibited by the convergence graphs of other functions, which were omitted to save space), where the convergence

TABLE XIII
AVERAGE AND STANDARD DEVIATION OF THE BEST-OF-RUN SOLUTIONS FOR 50 INDEPENDENT RUNS TESTED ON COMPOSITE FUNCTIONS CF1 TO CF3 TAKEN FROM THE CEC'05 BENCHMARKS

Func	D	Max FEs	Mean best value (Standard deviation)						
			CPSO-H	IPOP-CMA-ES	MA-S2	G3 with PCX	DEGL/SAW ($Cr = 0.9$)	DEGL/SAW ($Cr = 1$)	Statistical Significance
CF1	10	5×10^6	5.24167e+02 (1.046e+01)	**3.83592e+02 (1.236e+02)**	1.98661e+03 (2.123e+02)	1.847894e+03 (3.353e+02)	6.19227e+02 (6.8341e+01)	5.03826e+02 (4.0995e+01)	+
CF2	10	5×10^6	9.23762e+02 (6.718e+01)	6.82114e+02 (1.8469e+01)	1.53459e+03 (1.133e+02)	1.49463e+04 (7.846e+02)	7.60543e+02 (9.7837e+01)	**4.18542e+02 (8.9984e+01)**	+
CF3	10	5×10^6	7.58269e+02 (9.462e+02)	5.12504e+02 (2.586e+02)	7.16728e+02 (2.836e+02)	1.91423e+03 (2.643e+02)	6.74823e+02 (5.8471e+01)	**4.76239e+02 (3.7842e+01)**	+

rate of CMA-ES remained higher than DEGL/SAW until the maximum number of FEs were reached.

D. Comparative Performance Over Real-Life Optimization Problems

This section investigates the performance of the six competitive DE-variants over two real-world optimization problems, viz., the spread spectrum radar poly-phase code design problem and the sound frequency modulator synthesis problem. Both problems have been briefly described earlier in Section V-B.

In Table XIV, we show the mean and the standard deviation (within parentheses) of the best-of-run values for 30 independent runs of each of the six algorithms over the two most difficult instances of the radar poly-phase code design problem (for dimensions $D = 19$ and $D = 20$). Table XV reports the results of the same experiments performed over the FM synthesizer problem. Figs. 9 and 10 graphically present the rate of convergence of the DE-variants for these two problems (graphs in Fig. 9 have been shown for 20 dimensions for the radar code design problem). The 8-th column in Table XIV and the 7-th column in Table XV indicate the statistical significance level obtained from a paired t test between the best and the next-to-best performing algorithms in each case.

Tables XIV and XV show that DEGL/SAW outperforms all the other DE-variants in terms of final accuracy over two instances of the radar poly-phase code design problem as well as the FMS problem.

E. Selection of the Neighborhood Size

The proper selection of the neighborhood's size (equal to $2k + 1$, where k is the neighborhood radius) in DEGL affects the tradeoff between exploitation and exploration. For solving any given optimization problem, this selection remains an open problem. In practice, it is up to the practitioner and is based solely on his/her experience. Some empirical guidelines may, however, be provided based on the fact that if the neighborhood size is large (near the population size), then because of the overlapping of the neighborhoods of successive vectors, neighborhood-best of a number of vectors can be similar to the globally best vector in the entire population. This again increases the attraction of most of the vectors towards a specific point in the search space and results in loss of the explorative power of the algorithm. Our experiments suggest that a neighborhood size that is above 40% of the population size makes the performance of DEGL comparable to that of the DE/target-to-best/1/bin. Again, too small a neighborhood runs the risk of losing diversity of the population, as the difference vector in the local mutation model [(14)] may become too small. This is due to the fact that the vectors belonging to a small neighborhood may quickly become very similar to each other. We empirically observe that for $NP = 10 \cdot D$, the overall performance of the algorithm is not very sensitive to the neighborhood size varying between 10% and 20% of NP. Other choices for the population size NP and the corresponding radius of the neighborhood are topics of future research.

Below we provide the overall success rate of the DEGL/SAW algorithm for neighborhood size varying from 5–70% of NP, over 100-dimensional multimodal functions f_{10} and f_{11}. Since both the functions have their optima at the origin (0), we plot the percentage of runs that successfully yielded a final accuracy below 10^{-15} for different neighborhood sizes. We relaxed the threshold objective function value from 10^{-20}, so that at least one run of DEGL for all neighborhood sizes may converge below the threshold value.

Thorough experimentation with all the test problems shows that a neighborhood size of around 10% provides reasonably accurate results with high success rates over most of the benchmark problems covered here.

F. Correlation Between the Neighborhood Size and Weight Factor

Both the neighborhood size and the weight factor w are related to the balancing of the explorative and exploitative tendencies of DEGL. Establishment of any theoretical correlation between these two parameters remains an interesting problem for future research. In this section we provide a discussion on such correlation, based on our empirical results on the benchmark functions.

If we keep w constant throughout, then for neighborhood sizes ($2k + 1$, where k is the neighborhood radius) varying between approximately 15–25% of NP, reasonably good accuracy is achieved with $0.45 < w < 0.55$ over most of the uni- and multimodal benchmarks. Larger values of

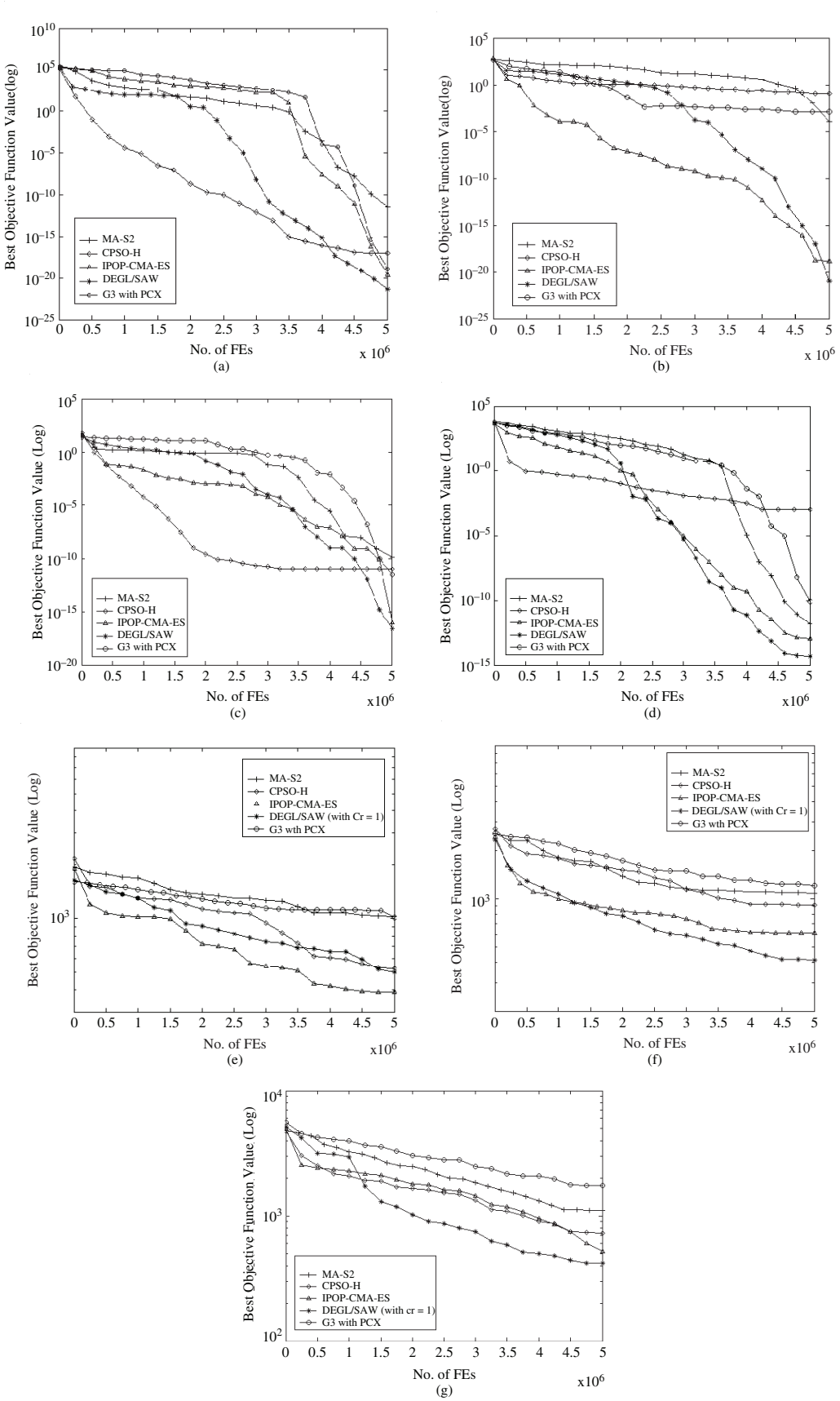


Fig. 9. Convergence characteristics for median run of five algorithms over seven difficult benchmark functions. (a) Step Function (f_6), (b) Generalized Rastrigin's Function (f_9), (c) Generalized Ackley's Function (f_{10}), (d) Generalized Griewnk's Function (f_{11}), (e) Composite Function CF1, (f) Composite Function CF2, (g) Composite Function CF3.

TABLE XIV

AVERAGE AND STANDARD DEVIATION (IN PARENTHESES) OF THE BEST-OF-RUN SOLUTIONS FOR 30 RUNS OVER THE SPREAD SPECTRUM RADAR POLY-PHASE CODE DESIGN PROBLEM (NUMBER OF DIMENSIONS $D = 19$ AND $D = 30$). FOR ALL CASES EACH ALGORITHM WAS RUN UP TO 5×10^6 FES

D	Mean best-of-run solution (Std Dev)						
	DE/rand/1/bin	DE/target-to-best/1/bin	DE/rand/1/either-or	SADE [28]	NSDE [31]	DEGL/SAW	Statistical Significance
19	7.4849e−01 (8.93e−03)	7.6535e−01 (5.93e−04)	7.5834e−01 (9.56e−04)	7.5932e−01 (3.88e−05)	7.6094e+01 (4.72e−03)	**7.4439e−01 (5.84e−04)**	·
20	8.5746e−01 (4.83e−03)	9.3534e−01 (4.55e−02)	8.3982e−01 (3.98e−03)	8.3453e−01 (6.53e−04)	8.4283e−01 (3.44e−02)	**8.0304e−01 (2.73e−03)**	+

TABLE XV

AVERAGE AND STANDARD DEVIATION (IN PARENTHESES) OF THE BEST-OF-RUN SOLUTIONS FOR 50 RUNS OF SIX ALGORITHMS ON THE FREQUENCY MODULATOR SYNTHESIS PROBLEM. EACH ALGORITHM WAS RUN FOR 10^5 FES

Mean best-of-run solution (Std Deviation)						
DE/rand/1/bin	DE/target-to-best/1/bin	DE/rand/1/either-or	SADE	NSDE	DEGL/SAW	Statistical Significance
1.7484e−01 (4.268e−02)	1.8255e+00 (1.158e−01)	3.8523e−04 (2.995e−04)	7.8354e−02 (5.8254e−03)	9.4559e−03 (6.924e−01)	**4.8152e−09 (6.2639e−08)**	+

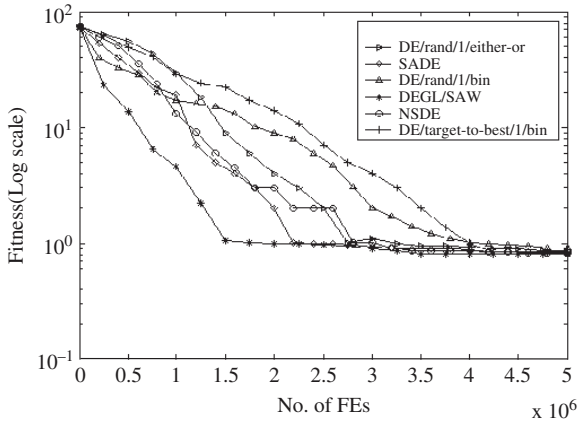


Fig. 10. Progress to the optimum solution for spread spectrum radar poly-phase code problem ($D = 20$).

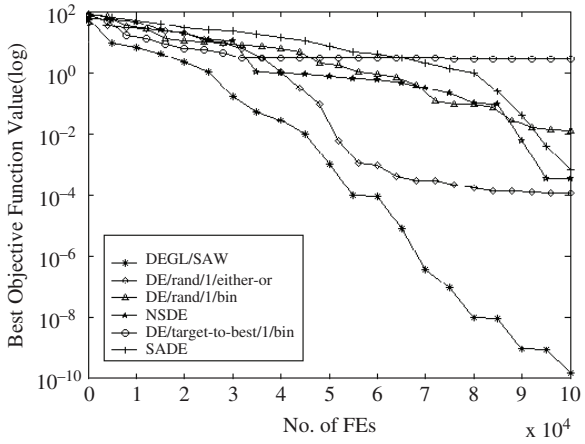


Fig. 11. Progress to the optimum solution for the FMS problem.

w in [0.7, 1.0], result in marginally better results compared to DE/target-to-best/1/bin but comparable or worse than one or more DE-variants tested here. However, for still smaller neighborhood size varying between 5 and 15% of NP, the optimal range of w for best accuracy is observed in [0.6, 0.75]. For neighborhood sizes roughly above 65% of the population size NP none of the time-varying weight factor schemes (described in Section IV-D) provided significant improvement of DEGL over DE/target-to-best/1/bin. This is expected because when the neighborhood size approaches the population size, the global and local mutation models do not differ significantly with respect to their best vectors and the role of weight factor becomes less prominent.

In the case when w is made self-adaptive, if the neighborhood-size is below 30% of NP, DEGL exhibits an evolutionary learning strategy that initially promotes exploration of the feasible search volume, but during the later stages of search favors exploitation and thus aids quick convergence to the global optimum. This trend has also been shown in Fig. 6 for various benchmark functions. However, we observe that if the neighborhood size is increased beyond 30%, the evolutionary learning gradually becomes erratic and for neighborhood sizes beyond 60% of NP, the self-adaptive characteristics of w become almost random over generations for most of the benchmarks. This tendency has been shown in Fig. 13 for the generalized Ackley's function f_{10}. This figure indicates that if the neighborhood size approaches NP, the adaptation mechanisms of w can hardly guide the search. We intend to investigate these facts more thoroughly in a future communication.

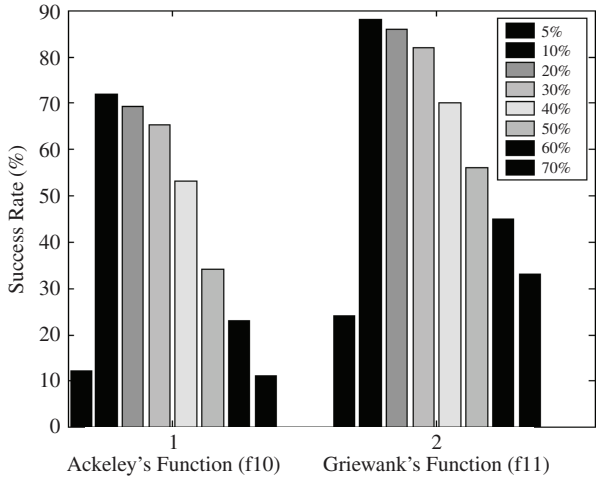


Fig. 12. Variation of the overall success rate of DEGL/SAW with increasing neighborhood size (for 100-dimensional functions f_{10} and f_{11}). Neighborhood sizes are indicated in the legend.

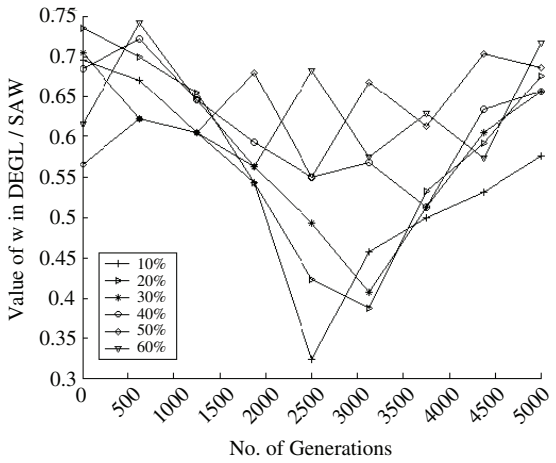


Fig. 13. Self-adaptation characteristics of the best vector of the DEGL/SAW scheme on the generalized Ackley's function (f_{10}) for different neighborhood sizes.

VII. Conclusion and Future Work

In this paper, we proposed a hybrid DE-type mutation/recombination operator that is a linear combination of two other mutation/recombination operators (an explorative and an exploitive operator), in an attempt to balance their effects. The new operator depends on a user-defined weight factor w. To circumvent the problem of determining a proper value of w for each problem, we proposed six different schemes for selecting and tuning this parameter. Among these, the self-adaptive weight scheme performed best on most of the benchmark functions tested.

The neighborhood-based DE mutation, equipped with self-adaptive weight factor, attempts to make a balanced use of the exploration and exploitation abilities of the search mechanism and is therefore more likely to avoid false or premature convergence in many cases. An extensive performance comparison with five significant DE variants and four other state-of-the-art evolutionary optimization techniques indicated that the proposed approaches enhance DE ability to accurately locate solutions in the search space. The use of the self-adaptive mutation scheme can lead to reliable optimization since it alleviates the problems generated by poor tradeoff between the explorative and exploitative tendencies of the algorithm, such as decreased rate of convergence, or even divergence and premature saturation.

This, however, does not lead us to claim that the DEGL family of algorithms may outperform their contestants over every possible objective function since it is impossible to model all possible complexities of real-life optimization problems with the limited test-suite that we used for testing the algorithms. In addition, the performance of the competitor DE variants may also be improved by blending other mutation strategies with judicious parameter tuning, a topic of future research. The conclusion we can draw at this point is that DE with the suggested modifications can serve as an attractive alternative for optimizing a wide variety of objective functions.

The present paper can be extended in several directions. Future research may focus on providing some empirical or theoretical guidelines for selecting the neighborhood size over different types of optimization problems. The effect of other neighborhood topologies (star-shaped, wheel-shaped, fully connected, etc.) on the performance of DEGL should be investigated theoretically. It would be interesting to study the performance of the DEGL family when the various control parameters (NP, F, and Cr) are self-adapted following the ideas presented in [22], [28].

References

[1] R. Storn and K. V. Price, "Differential evolution–A simple and efficient adaptive scheme for global optimization over continuous spaces," Institute of Company Secretaries of India, Chennai, Tamil Nadu. Tech. Report TR-95-012, 1995.

[2] R. Storn and K. V. 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.

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

[4] T. Rogalsky, R. W. Derksen, and S. Kocabiyik, "Differential evolution in aerodynamic optimization," in *Proc. 46th Annu. Conf. Can. Aeronautics Space Inst.*, 1999, pp. 29–36.

[5] R. Joshi and A. C. Sanderson, "Minimal representation multi-sensor fusion using differential evolution," *IEEE Trans. Syst., Man, Cybern., Part A*, vol. 29, no. 1, pp. 63–76, Jan. 1999.

[6] S. Das and A. Konar, "Design of two dimensional IIR filters with modern search heuristics: A comparative study," *Int. J. Comput. Intell. Applicat.*, vol. 6, no. 3, pp. 329–355, 2006.

[7] F.-S. Wang and H.-J. Jang, "Parameter estimation of a bio-reaction model by hybrid differential evolution," in *Proc. IEEE Congr. Evol. Comput. 2000*, vol. 1. Piscataway, NJ: IEEE Press, pp. 410–417.

[8] J. Lampinen. (1999). A bibliography of differential evolution algorithm. Lappeenranta University of Technology. Department of Information Technology, Laboratory of Information Processing, Tech. Report [Online]. Available: http://www.lut.fi/jlampine/debiblio.htm

[9] M. Omran, A. P. Engelbrecht, and A. Salman, "Differential evolution methods for unsupervised image classification," in *Proc. 7th Congr. Evol. Comput. (CEC-2005)*, vol. 2. Piscataway, NJ: IEEE Press, pp. 966–973.

[10] S. Das, A. Abraham, and A. Konar, "Adaptive clustering using improved differential evolution algorithm," *IEEE Trans. Syst., Man, Cybern. A*, vol. 38, no. 1, pp. 218–237, Jan. 2008.

[11] J. H. Holland, *Adaptation Natural, and Artificial Syst.*. Ann Arbor, MI: Univ. Michigan Press, 1975.

[12] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[13] J. Vesterstrøm and R. Thomson, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proc. 6th Congr. Evol. Comput. (CEC-2004)*, vol. 2. Piscataway, NJ: IEEE Press, Jun. 2004, pp. 1980–1987.

[14] U. K. Chakraborty, *Advances Differential Evolution*, Heidelberg, Germany: Springer-Verlag, 2008.

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

[16] J. Ronkkonen, S. Kukkonen, and K. V. Price, "Real parameter optimization with differential evolution," in *Proc. IEEE Congr. Evol. Comput. (CEC-2005)*, vol. 1. Piscataway, NJ: IEEE Press, pp. 506–513.

[17] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. C. Coello, "A comparative study of differential evolution variants for global optimization," in *Proc. Genetic Evol. Comput. Conf. (GECCO 2006)*, pp. 485–492.

[18] U. K. Chakraborty, S. Das, and A. Konar, "Differential evolution with local neighborhood," in *Proc. IEEE Congr. Evol. Comput. (CEC-2006)*, Piscataway, NJ: IEEE Press, pp. 7395–7402.

[19] K. V. Price, "An introduction to differential evolution," in *New Ideas Optimization*. London, U.K.: McGraw-Hill, 1999, pp. 293–298.

[20] R. Gamperle, S. D. Muller, and A. Koumoutsakos, "Parameter study for differential evolution," in *Proc. WSEAS NNA-FSFS-EC 2002*, Interlaken, Switzerland, Feb. 2002, pp. 293–298.

[21] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Computing–Fusion Found., Methodologies Applicat.*, vol. 9, no. 6, pp. 448–462, 2005.

[22] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, Apr. 2009.

[23] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," in *Proc. MENDEL 2003, 9th Int. Conf. Soft Computing*, Brno, Czech Republic, Jun. 2003, pp. 41–46.

[24] D. Zaharie and D. Petcu, "Adaptive pareto differential evolution and its parallelization," in *Proc. 5th Int. Conf. Parallel Process. Appl. Math.*, LNCS vol. 3019. Czestochowa, Poland, Sep. 2003, pp. 261–268.

[25] H. Abbass, "The self-adaptive pareto differential evolution algorithm," in *Proc. 2002 Congr. Evol. Comput.*, vol. 1. Honolulu, HI, May 2002, pp. 831–836.

[26] M. Omran, A. Salman, and A. P. Engelbrecht, "Self-adaptive differential evolution," *Int. Conf. comput. intell. security, Pt. 1, Springer Lecture Notes in Artificial Intelligence 3801*, pp. 192–199, 2005.

[27] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," in *Soft Computing–Fusion Found., Methodologies Applicat.*, vol. 10, no. 8, pp. 673–686, Jun. 2006.

[28] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 646–657, Dec. 2006.

[29] S. Das, A. Konar, and U. K. Chakraborty, "Two improved differential evolution schemes for faster global search," in *Proc. ACM-SIGEVO GECCO*, Washington, D.C., Jun. 2005, pp. 991–998.

[30] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition-Based Differential Evolution," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 64–79, Feb. 2008.

[31] Z. Yang, J. He, and X. Yao, *Making a Difference to Differential Evolution, in Advances in Metaheuristics for Hard Optimization*. New York: Springer-Verlag, 2007, pp. 415–432.

[32] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*. Berlin, Germany: Springer-Verlag, 1999.

[33] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Inform. Sci.*, vol. 178, no. 15, pp. 2985–2999, 2008.

[34] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *Proc. IEEE Congr. Evol. Comput. (CEC-2008)*, Hong Kong, China, Jun. 2008, pp. 1110–1116.

[35] N. Noman and H. Iba, "Enhancing differential evolution performance with local search for high dimensional function optimization," in *Proc. 2005 Conf. Genetic Evol. Comput.*, Jun. 2005, pp. 967–974.

[36] N. Noman and H. Iba, "Accelerating Differential Evolution Using an Adaptive Local Search," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 107–125, Feb. 2008.

[37] K. E. Parsopoulos and M. N. Vrahatis, "UPSO: A unified particle swarm optimization scheme," in *Proc. Lecture Ser. Computer and Comput. Sci., Proc. Int. Conf. Comput. Meth. Sci. Eng. (ICCMSE 2004)*, vol. 1. Zeist, Netherlands: VSP Int. Sci. Publishers, pp. 868–873.

[38] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 225–239, Jun. 2004.

[39] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 281–295, Jun. 2006.

[40] W.-J. Zhang and X.-F. Xie. (2003) DEPSO: Hybrid particle swarm with differential evolution operator. in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, vol. 4, pp. 3816–3821, [Online]. Available: http://citeseer.ist.psu.edu/635224.html

[41] S. Das, A. Konar, and U. K. Chakraborty, "An improved particle swarm optimization algorithm for faster global search," in *Proc. ACM-SIGEVO Genetic Evol. Comput. Conf. (GECCO-2005)*, Washington, D.C., Jun. 2005.

[42] R. Mendes and J. Kennedy, "The fully informed particle swarm: Simpler, maybe better," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 204–210, Jun. 2004.

[43] K. Zielinski, D. Peters, and R. Laur, "Run time analysis regarding stopping criteria for differential evolution and particle swarm optimization," in *Proc. 1st Int. Conf. Experiments/Process/Syst. Modelling/Simulation/Optimization*, Athens, Greece, 2005.

[44] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*. Reading, MA: Addison-Wesley, 1983.

[45] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 1st ed. Cambridge, MA: MIT Press and New York: McGraw-Hill, 1990.

[46] P. Collet, J. Louchet, and E. Lutton, "Issues on the optimization of evolutionary algorithms code," in *Proc. 2002 Congr. Evol. Comput. (CEC '02)*, vol. 2. Honolulu, HI, May 2002, pp. 1103–1108.

[47] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Jul. 1999.

[48] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Nanyang Technol. Univ., Singapore, Tech. Report and IIT, Kanpur, India, KanGAL Report #2005005, May 2005.

[49] Y. W. Shang and Y. H. Qiu, "A note on the extended Rosenbrock function," *Evol. Comput.*, vol. 14, no. 1, pp. 119–126, 2006.

[50] N. Mladenović, J. Petrovic, V. Kovacevic-Vujicic, and M. Cangalovic, "Solving spread-spectrum radar polyphase code design problem by tabu search and variable neighborhood search," *Eur. J. Oper. Res.*, vol. 151, no. 2, pp. 389–399, 1st Dec. 2003.

[51] A. Horner, J. Beauchamp, and L. Haken, "Genetic algorithms and their application to FM matching synthesis," *Comput. Music J.*, vol. 17, no. 4, pp. 17–29, 1993.

[52] F. Herrera and M. Lozano, "Gradual distributed real-coded genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 4, no. 1, pp. 43–62, Apr. 2000.

[53] D. Fogel and H.-G. Beyer, "A note on the empirical evaluation of intermediate recombination," *Evol. Comput.*, vol. 3, no. 4, pp. 491–495, 1995.

[54] P. J. Angeline, "Evolutionary optimization versus particle swarm optimization: Philosophy and the performance difference," in *Proc. 7th Int. Conf. Evol. Programming-—Evol. Programming VII*, LNCS vol. 1447, 1998, pp. 84–89.

[55] B. Flury, *A First Course in Multivariate Statistics*. vol. 28, New York: Springer, 1997.

[56] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 124–141, Jul. 1999.

[57] A. Konar, *Comput. Intelligence: Principles, Techniques, and Applications*. New York: Springer, 2005.

[58] A. Auger and N. Hansen, "A restart CMA evolution strategy with increasing population size," in *Proc. IEEE Congr. Evol. Comput. (CEC 2005)*, vol. 2. Sep. 2005, pp. 1769–1776.

[59] Y.-S. Ong and A. J. Keane, "Meta-lamarckian learning in memetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 99–110, Apr. 2004.

[60] K. Deb, A. Anand, and D. Joshi, "A computationally efficient evolutionary algorithm for real-parameter optimization," *Evol. Comput.*, vol. 10, no. 4, pp. 371–395, 2002.

[61] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, 2001.

[62] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation," in *Proc. 1996 IEEE Conf. Evol. Comput. (ICEC '96)*, Berlin, Germany, May 1996, pp. 312–317.
[63] *Recent Advances in Memetic Algorithms*. Berlin, Heidelberg, Germany: New York: Springer-Verlag, 2004.
[64] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: Model, taxonomy, and design issues," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 474–488, Oct. 2005.

Swagatam Das was born in Kolkata, India, in 1980. He received the B.E. and M.E. degrees in control engineering and the Ph.D. degree in engineering, all from Jadavpur University, Kolkata, India, in 2003, 2005, and 2009, respectively.

He is presently a Lecturer in the Department of Electronics and Telecommunication Engineering, Jadavpur University. His current research interests include evolutionary computing, swarm intelligence, pattern recognition, bioinformatics, control systems engineering, and digital signal processing. He has published more than 60 research articles in peer-reviewed journals and international conferences. He is presently coauthoring a text book on swarm intelligence and a research monograph, both to be published by Springer by mid-2009.

Dr. Das is the recipient of the Best Paper Award at the Sixth International Conference on Intelligent Systems Design and Applications (ISDA2006) held in Jinan, Shandong, China. He serves as an Editorial Board member of the *International Journal of Artificial Intelligence* and *Soft Computing*. He has been a reviewer for journals such as *Pattern Recognition*, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, *Signal Processing*, *Information Sciences*, and *Neurocomputing*.

Ajith Abraham (M'96–SM'07) received the Ph.D. degree from Monash University, Melbourne, Australia, in 2001.

He is currently with the Center for Quantifiable Quality of Service in Communication Systems (Q2S), Center of Excellence, Norwegian University of Science and Technology (NTNU), Trondheim, Norway and Machine Intelligence Research Labs (MIR Labs). He has worldwide academic experience of nearly ten years with formal appointments in Monash University, Australia; Oklahoma State University; Chung-Ang University, Seoul; Jinan University, China; Rovira i Virgili University, Spain; Dalian Maritime University, China; Yonsei University, Seoul; Open University of Catalonia, Spain; and NTNU, Norway. For about two and a half years, he was working under the Institute of Information Technology Advancement Professorship Program funded by the South Korean Government. He was working with three multi-national companies: Keppel Engineering, Singapore; Hyundai Engineering, Korea; and Ashok Leyland Ltd, India where he was involved in different industrial research and development projects for nearly eight years. He has authored or coauthored more than 500 research publications in peer-reviewed reputed journals, book chapters, and conference proceedings. His primary research interests are in advanced computational intelligence, with a focus on using global optimization techniques for designing intelligent systems. Their application areas include Web services, information security, Web intelligence, social networks, financial modeling, multicriteria decision making, data mining, etc. He has given more than 25 plenary lectures and conference tutorials in these areas.

Dr. Abraham Co-chairs the IEEE Systems Man and Cybernetics Society Technical Committee on Soft Computing. He is a regular reviewer of IEEE INTELLIGENT SYSTEMS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, IEEE TRANSACTIONS ON NEURAL NETWORKS, IEEE TRANSACTIONS ON FUZZY SYSTEMS, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, and IEEE TRANSACTIONS ON POWER SYSTEMS. He serves on the Editorial Boards of more than 30 international journals and has also guest-edited 30 special issues on various topics for international journals. Since 2001, he has been actively involved in the hybrid intelligent systems and the Intelligent Systems Design and Applications series of annual international conferences. He was the recipient of five Best Paper Awards.

Amit Konar (M'97) received the B.E. degree from Bengal Engineering and Science University, Shibpur, India, in 1983 and the M.E., M.Phil., and Ph.D. degrees from Jadavpur University, Kolkata, in 1985, 1988, and 1994, respectively.

He is currently a Professor in the Department of Electronics and Telecommunication Engineering, Jadavpur University. He is also the Joint Coordinator of the Jadavpur University Excellence Program in Cognitive Science, and a founding Coordinator of the M.Tech. Course on Intelligent Automation and Robotics. His research areas include the study of computational intelligence algorithms and their applications to the entire domain of electrical engineering and computer science. Specifically, he worked on fuzzy sets and logic, neuro-computing, evolutionary algorithms, Dempster-Shafer theory, and Kalman filtering, and applied the principles of computational intelligence in image understanding, VLSI design, mobile robotics, pattern recognition, bioinformatics, and computational finance. He has supervised 10 Ph.D. theses in different areas of machine intelligence. He has published or presented about 200 papers in international journals and conferences. He is the author of six books, including two popular texts: ARTIFICIAL INTELLIGENCE AND SOFT COMPUTING and BEHAVIORAL AND COGNITIVE MODELING OF THE HUMAN BRAIN (CRC Press) in 2000, and COMPUTATIONAL INTELLIGENCE: PRINCIPLES, TECHNIQUES AND APPLICATIONS (Springer) in 2005. He serves as the Editor-in-Chief of the INTERNATIONAL JOURNAL OF ARTIFICIAL INTELLIGENCE AND SOFT COMPUTING, and is also a member of the Editorial Board of five other international journals, including NEUROCOMPUTING (Elsevier) and Hybrid INTELLIGENT SYSTEMS (IOS Press). He was a recipient of AICTE-accredited 1997–2000 Career Award for Young Teachers for his significant contribution in teaching and research. He was a Visiting Professor for the Summer Courses in University of Missouri, St. Louis, in 2006. He is in the Program committee of several international conferences, held in Europe, America, Australia, and Asia.

Uday Kumar Chakraborty obtained the Ph.D. degree on stochastic models of genetic algorithms.

Currently, he is an Associate Professor of Computer Science at the University of Missouri, St. Louis. He has held positions at the CAD Center, Calcutta, CMC Limited (Calcutta and London), Jadavpur University, Calcutta, and the German National Research Center for Computer Science, Bonn. His research interests include evolutionary computation, soft computing, scheduling, and computer graphics. He is the co-author/editor of 3 books and 90 articles in journals and conference proceedings. He is an Area Editor of NEW MATHEMATICS & NATURAL COMPUTATION and an editor of JOURNAL OF COMPUTING AND INFORMATION TECHNOLOGY, and serves on the Editorial Boards of three other journals. He has guest-edited special issues on evolutionary computation of many computer science journals and has served as track chair or program committee member of numerous international conferences.