Enhancing Differential Evolution Utilizing Proximity-based Mutation Operators

M.G. Epitropakis, *Member, IEEE,* D.K. Tasoulis, *Member, IEEE,* N.G. Pavlidis, V.P. Plagianakos, and M.N. Vrahatis

***Abstract*—Differential Evolution is a very popular optimization algorithm and considerable research has been devoted to the development of efficient search operators. Motivated by the different manner in which various search operators behave, we propose a novel framework based on the proximity characteristics among the individual solutions as they evolve. Our framework incorporates information of neighboring individuals, in an attempt to efficiently guide the evolution of the population towards the global optimum, without sacrificing the search capabilities of the algorithm. More specifically, the random selection of parents during mutation is modified, by assigning to each individual a probability of selection that is inversely proportional to its distance from the mutated individual. The proposed framework can be applied to any mutation strategy with minimal changes. In this paper, we incorporate this framework in the original Differential Evolution algorithm, as well as other recently proposed Differential Evolution variants. Through an extensive experimental study, we show that the proposed framework results in enhanced performance for the majority of the benchmark problems studied.**

***Index Terms*—Differential Evolution, Mutation Operator, Affinity Matrix, Nearest Neighbors**

I. Introduction

EVOLUTIONARY Algorithms (EAs) are stochastic search methods that mimic evolutionary processes encountered in nature. The common conceptual base of these methods is to evolve a population of candidate solutions by simulating the main processes involved in the evolution of genetic material of organism populations, such as natural selection and biological evolution. EAs can be characterized as global optimization algorithms. Their population-based nature, allows them to avoid getting trapped in a local optimum and consequently provides a great chance to find global optimal solutions. EAs have been successfully applied to a wide range of optimization problems, such as image processing, pattern recognition, scheduling, and engineering design [1], [2]. The most prominent EAs proposed in the literature are: Genetic Algorithms [1], Evolutionary Programming [3], Evolution Strategies [4], Genetic Programming [5], Particle Swarm Optimization (PSO) [6], and Differential Evolution [7], [8].

In general, every EA starts by initializing a population of candidate solutions (individuals). The quality of each solution is evaluated using a fitness function, which represents the problem at hand. A selection process is applied at each iteration of the EA to produce a new set of solutions (population). The selection process is biased toward the most promising traits of the current population of solutions to increase their chances of being included in the new population. At each iteration (generation), the individuals are evolved through a predefined set of operators, like *mutation* and *recombination*. This procedure is repeated until convergence is reached. The best solution found by this procedure is expected to be a near-optimum solution [2], [9].

Mutation and *recombination* are the two most frequently used operators and are referred to as *evolutionary* operators. The role of *mutation* is to modify an individual by small random changes to generate a new individual [2], [9]. Its main objective is to increase diversity by introducing new genetic material into the population, and thus avoid local optima. The *recombination* (or crossover) operator combines two, or more, individuals to generate new promising candidate solutions [2], [9]. The main objective of the recombination operator is to explore new areas of the search space [2], [10].

In this paper, we study the Differential Evolution (DE) algorithm, proposed by Storn and Price [7], [8]. This method has been successfully applied in a plethora of optimization problems [7], [11]–[19]. Without loss of generality, we only consider minimization problems. In this case, the objective is to locate a global minimizer of a function f (objective function).

Definition 1: A *global minimizer* $x^\star \in \mathbb{R}^D$ of the real–valued function $f\colon \mathcal{E} \to \mathbb{R}$ is defined as:

$$f(x^\star) \leqslant f(x), \quad \forall\, x \in \mathcal{E},$$

where the compact set $\mathcal{E} \subseteq \mathbb{R}^D$ is a D–dimensional scaled translation of the unit hypercube.

A main issue in the application of EAs to a given optimization problem, is to determine the values of the control parameters of the algorithm that will allow the efficient exploration of the search space, as well as its effective exploitation. Exploration enables the identification of regions of the search space in which good solutions are located. On the other hand, exploitation accelerates the convergence to the optimum solution. Inappropriate choice of the parameter values can cause the algorithm to become greedy or very explorative and

M.G. Epitropakis and M.N. Vrahatis are with the Department of Mathematics, University of Patras, Greece.

D.K. Tasoulis is with the Department of Mathematics, Imperial College London, United Kingdom.

N.G. Pavlidis is with Institute for Mathematical Sciences, Imperial College London, United Kingdom.

V.P. Plagianakos is with the Department of Computer Science and Biomedical Informatics, University of Central Greece, Greece.

All authors are members of Computational Intelligence Laboratory (CILab), Department of Mathematics, University of Patras, Greece.

Manuscript received November 30, 2009; revised September 12, 2010.

consequently the search of the optimum can be hindered. For example, a high mutation rate will result in much of the space being explored, but there is also a high probability of losing promising solutions; the algorithm has difficulty to converge to an optimum due to insufficient exploitation. Several Evolutionary Computation approaches have been proposed that try to give a satisfactory answer to this *exploration/exploitation dilemma* [20]–[27]. Recent studies of the exploration and exploitation capabilities of different mutation operators have shown that after a number of iterations of the DE algorithm the individuals exhibit the tendency to gather around optimizers of the objective function [21], [22].

Motivated by these findings, we propose an alternative to the uniform random selection of parents during mutation. We advocate a stochastic selection framework in which the probability of selecting an individual to become a parent is inversely proportional to its distance from the individual undergoing mutation. By favoring search in the vicinity of the mutated individual this framework promotes efficient exploitation, without substantially diminishing the exploration capabilities of the mutation operator. The proposed framework can be applied to any mutation strategy and, as shown through extensive experimental evaluation, produces remarkable improvement. We also incorporate this framework to a number of recently proposed DE variants and observe performance gains.

The rest of the paper is organized as follows: Section II describes the original Differential Evolution algorithm. In Section III, we include a short literature review. Section IV illustrates the behavior of different mutation operators, providing the motivation for the proposed framework, which is presented in Section V. Next, in Section VI we present the results of an extensive experimental analysis, and the paper concludes with a discussion in Section VII.

II. The Differential Evolution Algorithm

Differential Evolution [7], [8] is a population–based stochastic parallel direct search method that utilizes concepts borrowed from the broad class of EAs. The method typically requires few control parameters and numerous studies have shown that it has good convergence properties. DE outperforms other well known EAs in a plethora of problems [7], [11]–[13], [15] and has attracted the interest of the research community. Consequently, several variations of the classical DE algorithm have been proposed in the literature [13], [14], [22], [23], [26]–[34]. A detailed description of the DE algorithm and experimental results on hard optimization problems can be found in [12]–[15], [18].

The DE algorithmic schemes can be classified using the notation DE/*base*/*num*/*cross*. The method of selecting the parent that constitutes the base individual is indicated by *base*. For example, DE/rand/*num*/*cross* selects the parent for the base individual randomly, while in DE/best/*num*/*cross* the parent for the base individual is the best individual of the population. The number of differences between individuals that are used to perturb the base individual is indicated by *num*. Finally, *cross* stands for the crossover type utilized by the mutation strategy, i.e. *exp* for exponential and *bin* for binomial. Exponential and binomial crossover will be discussed in subsection II-C. In this study, we always employ binomial crossover, and thus we exclude the *cross* part to simplify the notation.

In DE the central search operator is known as *mutation strategy*. Consequently, a substantial amount of research has been devoted to the development and the analysis of efficient mutation operators and their dynamics [12], [13], [15], [18], [35], [36]. In more detail, for each individual undergoing mutation (*mutated individual*) a set of individual solutions are uniformly selected across the population (*parents*). The parents and the mutated individual are subsequently mixed to construct a new candidate solution (*mutant individual*). The mutation operators prescribe the manner in which this mixing is performed, and the number of parents that will be used. The search operators efficiently shuffle information among the individuals, enabling the search for an optimum to focus on the most promising regions of the solution space. Next, we describe in detail the DE procedures.

A. Initialization

Following the general concept of EAs, the first step of DE is the initialization of a population of NP, D–dimensional potential solutions (*individuals*) over the optimization search space. We shall symbolize each individual by $x_g^i = [x_{g,1}^i, x_{g,2}^i, \ldots, x_{g,D}^i]$, for $i = 1, 2, \ldots, NP$, where $g = 0, 1, \ldots, g_{\max}$ is the current generation and $g_{\max}$ the maximum number of generations. At the first generation ($g = 0$) the population should be sufficiently scaled to cover as much as possible of the optimization search space. Initialization is implemented by using a random number distribution to generate the potential individuals in the optimization search space. The optimization search space can be defined by lower and upper bound values, i.e. $L = [L_1, L_2, \ldots, L_D]$ and $U = [U_1, U_2, \ldots, U_D]$. Hence, we can initialize the j-th dimension of the i-th individual according to:

$$x_{0,j}^i = L_j + \mathrm{rand}_j(0,1) \cdot (U_j - L_j), \tag{1}$$

where $\mathrm{rand}_j(0,1)$ is a uniformly distributed random number confined in the $[0,1]$ range.

B. Mutation Operators

Following initialization, the evolution process begins with the application of the mutation operator. For each individual of the current population a new individual, called the *mutant individual* v_g^i, is derived through the combination of randomly selected and pre-specified individuals. The originally proposed and most frequently used mutation strategies in the literature are:

1) "DE/best/1"

$$v_g^i = x_g^{\mathrm{best}} + F(x_g^{r_1} - x_g^{r_2}), \tag{2}$$

2) "DE/rand/1"

$$v_g^i = x_g^{r_1} + F(x_g^{r_2} - x_g^{r_3}), \tag{3}$$

3) "DE/current-to-best/1"

$$v_g^i = x_g^i + F(x_g^{\text{best}} - x_g^i) + F(x_g^{r_1} - x_g^{r_2}), \quad (4)$$

4) "DE/best/2"

$$v_g^i = x_g^{\text{best}} + F(x_g^{r_1} - x_g^{r_2}) + F(x_g^{r_3} - x_g^{r_4}), \quad (5)$$

5) "DE/rand/2"

$$v_g^i = x_g^{r_1} + F(x_g^{r_2} - x_g^{r_3}) + F(x_g^{r_4} - x_g^{r_5}), \quad (6)$$

6) "DE/current-to-best/2"

$$v_g^i = x_g^i + F(x_g^{\text{best}} - x_g^i) + \\ + F(x_g^{r_1} - x_g^{r_2}) + F(x_g^{r_3} - x_g^{r_4}), \quad (7)$$

where x_g^{best} denotes the best (fittest) individual of the current generation, the indices $r_1, r_2, r_3, r_4, r_5 \in S_r = \{1, 2, \ldots, NP\} \setminus \{i\}$, are uniformly random integers mutually different and distinct from the running index i, $(x_g^{r_x} - x_g^{r_y})$ is a difference vector that mutates the base vector, ($r_x, r_y \in S_r$), and $F > 0$ is a real positive parameter, called *mutation* or *scaling factor*. The mutation factor controls the amplification of the difference between two individuals and is used to prevent the stagnation of the search process. Large values of this parameter amplify the differences and hence promote exploration, while small values favor exploitation. The inappropriate choice of the mutation factor can therefore cause the deceleration of the algorithm and a reduction of population diversity [12], [15], [28]. In the original DE algorithm, the mutation factor F is a fixed and user defined parameter, while in many adaptive DE variants each individual is associated with a different adaptive mutation factor [23], [26]–[31], [33], [37], [38]. Several DE variants that either introduce new mutation strategies or new self-adaptive techniques to tune the control parameters have been recently proposed [12], [15], [18], [22], [25]–[27], [31], [34], [35], [39]–[44]. A detailed discussion about the current state-of-the-art of DE can be found in a recently published survey [13].

In an attempt to rationalize the mutation strategies, Eqs. (2)–(7), we observe that Eq. (3) is similar to the crossover operator employed by some Genetic Algorithms. Eq. (2) is derived from Eq. (3), by substituting the best member of the previous generation, x_g^{best}, with a random individual x_g^{r1}. Eqs. (4), (5), (6) and (7) are modifications obtained by the combination of Eqs. (2) and (3). It is clear that new DE mutation operators can be generated using the above ones as building blocks. Such examples include the trigonometric mutation operator [39], the recently proposed genetically programmed mutation operators [45], or new classes of mutation operators that attempt to combine the explorative and exploitative capabilities of the original ones [21], [22].

C. Crossover or Recombination Operators

Following mutation, the *crossover* or *recombination* operator is applied to further increase the diversity of the population. It is important to note that without the crossover operator, the original DE algorithm performs poorly on multimodal functions [12]. In crossover, the mutant individuals are combined with other predetermined members of the population, called *target individuals*, to produce the *trial individuals*. The most well known and widely used variants of DE utilize two main crossover schemes; the *exponential* and the *binomial* or *uniform crossover* [7], [12], [13], [46]. The exponential crossover scheme was introduced in the original work of Storn and Price [8], but in the subsequent DE literature the binomial variant [7], [13] is mostly used.

The *binomial* or *uniform* crossover is performed on each component j ($j = 1, 2, \ldots, D$) of the mutant individual v_g^i. In detail, for each component of the mutant vector a random real number r in the interval $[0, 1]$ is drawn and compared with the *crossover rate* or *recombination factor*, $CR \in [0, 1]$, which is the second DE control parameter. If $r \leqslant CR$, then we select, as the j–th component of the trial individual u_g^i, the j–th component of the mutant individual v_g^i. Otherwise, the j–th component of the target vector x_g^i becomes the j–th component of the trial vector. The aforementioned procedure can be outlined as:

$$u_{g,j}^i = \begin{cases} v_{g,j}^i, & \text{if } (\text{rand}_{i,j}(0,1) \leqslant CR \text{ or } j = j_{\text{rand}}), \\ x_{g,j}^i, & \text{otherwise}, \end{cases} \quad (8)$$

where the $\text{rand}_{i,j}(0, 1)$ is a uniformly distributed random number in $[0, 1]$, different for every j-th component of every individual, and $j_{\text{rand}} \in \{1, 2, \ldots, D\}$ is a randomly chosen integer which ensures that at least one component of the mutant vector will be assigned to the target vector. It is evident that for values of the recombination factor close to zero the effect of the mutation operator is very small, since the target and the mutant vector become identical.

D. Selection

Finally, the *selection* operator is employed to maintain the most promising trial individuals in the next generation and to retain the population size constant over the evolution process [12]. The original DE adopts a simple monotone selection scheme. It compares the objective values of the target x_g^i and trial u_g^i individuals. If the trial individual reduces the value of the objective function then it is accepted for the next generation; otherwise the target individual is retained in the population. Thus, the *selection* operator can be defined as:

$$x_{g+1}^i = \begin{cases} u_g^i, & \text{if } f(u_g^i) < f(x_g^i), \\ x_g^i, & \text{otherwise.} \end{cases} \quad (9)$$

The original DE algorithm (DE/rand/1/bin) is illustrated in Algorithm 1.

III. Related work

Darwin was the first to realize that populations may exhibit a spatial structure which can influence the population's dynamics. The Evolutionary Computing (EC) literature today utilizes spatial information in populations and the general concept of a neighborhood in several domains. In this section, we briefly discuss how the neighborhood concept has been utilized in the context of the Differential Evolution algorithm.

Algorithm 1 Algorithmic scheme for the original Differential Evolution algorithm (DE/rand/1/bin)

Set the generation counter $g = 0$
/* **Initialize the population** of NP individuals: $P_g = \{x_g^1, x_g^2, \ldots, x_g^{NP}\}$, with $x_g^i = \{x_{1,g}^i, x_{2,g}^i, \ldots, x_{D,g}^i\}$ for $i = 1, 2, \ldots, NP$ uniformly in the optimization search hyper-rectangle $[L, U]$.*/
for $i = 1$ to NP **do**
  for $j = 1$ to D **do**
    $x_{0,j}^i = L_j + \text{rand}_j(0,1) \cdot (U_j - L_j)$
  end for
  Evaluate individual x_0^i
end for
while termination criteria are not satisfied **do**
  Set the generation counter $g = g + 1$
  for $i = 1$ to NP **do**
    /* **Mutation step** */
    Select uniformly random integers $r_1, r_2, r_3 \in S_r = \{1, 2, \ldots, NP\} \setminus \{i\}$
    /* For each target vector x_g^i generate the corresponding mutant vector v_g^i using Eq. (3) */
    for $j = 1$ to D **do**
      $v_{j,g}^i = x_{j,g}^{r_1} + F(x_{j,g}^{r_2} - x_{j,g}^{r_3})$
    end for
    /* **Crossover step:** For each target vector x_g^i generate the corresponding trial vector u_g^i through the **Binomial Crossover** scheme.*/
    j_{rand} = a uniformly distributed random integer $\in \{1, 2, \ldots, D\}$
    for $j = 1$ to D **do**
      $u_{g,j}^i = \begin{cases} v_{g,j}^i, & \text{if } (\text{rand}_{i,j}(0,1) \leqslant CR \text{ or } j = j_{\text{rand}}), \\ x_{g,j}^i, & \text{otherwise,} \end{cases}$
    end for
    /* **Selection step** */
    if $f(u_g^i) < f(x_g^i)$ **then**
      $x_{g+1}^i = u_g^i$
      if $f(u_g^i) < f(x_g^{\text{best}})$ **then**
        $x_g^{\text{best}} = u_g^i$ and $f(x_g^{\text{best}}) = f(u_g^i)$
      end if
    else
      $x_{g+1}^i = x_g^i$
    end if
  end for
end while

A. Neighborhood concepts in structured EAs

In structured EAs the population is decentralized into sub-populations which can interact and may have different evolutionary roles. Two of the most prominent structured EAs are cellular Evolutionary Algorithms (cEAs) [47] and distributed Evolutionary Algorithms (dEAs) [48], [49]. A comprehensive classification and presentation can be found in [50], [51]. Generally, in cEAs, the sub-populations are created according to a neighborhood criterion and thus each sub-population has both an explorative and an exploitative role for a different region of the search space. On the other hand, in dEAs, distinct sub-populations (islands) explore in parallel the entire search space. In biological terms, dEAs resemble distinct semi-isolated populations in which evolution takes place independently. The migration operator in dEAs controls the exchange of individuals between subpopulations. This operator defines the topology, the migration rate, the migration frequency, and the migration policy [49], [52], [53]. These additional degrees of freedom make dEAs more flexible and capable of tackling harder optimization tasks.

The concept of structured populations has been incorporated in DE. In [23] and [54], distributed DE variants were presented which control adaptively the migration and the DE control parameters according to a genotype diversity criterion. In [55], a distributed DE algorithm is proposed that preserves diversity in the niches in order to solve multimodal optimization problems. In [56] a ring topology distributed DE was proposed with a migration operator that exchanges best performing individuals and replaces random individuals among neighboring sub-populations. In [57], Apolloni et al. proposed a modified version of [56], in which migration is performed through a probabilistic criterion. Modifications of [56] presented in [58]–[60] utilize a locally connected topology, where each node is connected to l other nodes. The recently proposed Distributed Differential Evolution with Explorative—Exploitative Population Families (DDE-EEPF) [24] employs sub-populations which are grouped into two families: explorative and exploitative. Explorative subpopulations have constant size, are arranged according to a ring topology and employ a migration of best performing individuals. On the other hand, exploitative subpopulations have dynamic size, are highly exploitative, and aim to quickly detect fittest solutions. Numerical results show that DDE-EEPF is an efficient and promising distributed DE variant. The Distributed Differential Evolution with Scale factor inheritance mechanism (FACPDE) [61] implements sub-populations arranged in a ring topology. Each sub-population is characterized by its own scale factor and migrates the best individual with its associated scale factor to its neighbors. The distribution of the successful scale factors and the fittest individuals among the subpopulations, enhances the scheme, and its performance substantially.

B. Index neighborhood concepts in Differential Evolution

A popular neighborhood structure in EC is the index-based neighborhood concept, introduced in the PSO algorithm. PSO incorporates an index-based neighborhood structure in its population and not real topological-based neighborhoods. Thus, the neighbors of each potential solution do not necessary lie in the vicinity of its topological region in the search space. Recently, the index neighborhood structures of PSO have also been considered in DE. The Differential Evolution with Global and Local Neighborhoods (DEGL) [13], [25], [42] incorporates concepts of the UPSO algorithm [62], such as the index neighborhoods of each individual, a local and a global scheme to facilitate the exploration and the exploitation of the search space, and a convex combination of these schemes to balance their effect. The Self-adaptive DE (SDE) [63], has been modified by using a ring neighborhood topology

in [64]. The same authors introduced the Barebones Differential Evolution [65] (BBDE). BBDE employs the concept of index neighborhoods in DE and enhances the DE mutation scheme by utilizing as a base vector either a randomly chosen personal best position or a stochastic weighted average of the individual's attractors (e.g. its personal and neighborhood best positions). This mutation scheme tends to explore the search space around the corresponding base vector and thus to exploit the vicinity of the current position.

C. Neighborhood concepts in mutation strategies

Numerous DE variants utilize specialized mutation strategies to exploit population structure. In [66], five mutation strategies have been proposed that produce new vectors in the vicinity of the corresponding base vector. To this end, the weighted difference between two individuals is used in conjunction with an adaptive scaling factor. DE with Parent Centric Crossover (DEPCX) and DE with Probabilistic Parent Centric Crossover (Pro. DEPCX) [67] are inspired by the parent centric crossover operator (PCX) used in GAs [68]. DEPCX utilizes the parent centric approach in the mutation strategy to generate new solution vectors, while Pro. DEPCX stochastically utilizes the parent centric mutation operator along with the basic DE mutation operation. The PCX procedure increases the probability of producing new candidate solution vectors in the vicinity of the parent vectors and thus exploits the neighborhood of parent vectors. In [44] and [69], two modified DE variants called DE with Random Localization (DERL) and DE with Localization using the best vector (DELB) were proposed. Both variants incorporate simple techniques to produce solutions that exhibit a local search effect around the base vector, with global exploration characteristics at the early stages of the algorithm and a local effect in terms of convergence at later stages of the algorithm.

D. Neighborhood concepts through local search

Various DE variants attempt to exploit and refine the position of the best individuals, by incorporating a list of local search procedures. MDE [70] makes use of the Hooke-Jeeves algorithm and a Stochastic Local Searcher adaptively coordinated by a fitness diversity-based measure. The EMDE [16], [17] combines the powerful explorative features of DE with the exploitative features of three local search algorithms employing different pivot rules and neighborhood generating functions, e.g. Hooke Jeeves Algorithm, a Stochastic Local Search, and Simulated Annealing. The Super-Fit Memetic Differential Evolution (SFMDE) [71] employs PSO, the Nelder-Mead algorithm and the Rosenbrock algorithm. SFMDE coordinates the local search algorithms by means of an index that measures the quality of the super-fit individual with respect to the remaining individuals in the population and a probabilistic scheme based on the generalized beta distribution. Noman and Iba [72] recently proposed a Fittest Individual Refinement (FIR), a crossover-based local search DE variant to tackle high dimensional problems. In [73], FIR is enhanced through a local search technique which adaptively adjusts the length of the search, utilizing a hill-climbing heuristic. This approach accelerates DE by enhancing the search capability in the neighborhood of the best solution in successive generations. Additionally, the Scale Factor Local Search Differential Evolution (SFLSDE) [74] is based on the DE/rand/1 mutation strategy and incorporates, within a self-adaptive scheme, two local search algorithms to efficiently adapt the mutation factor during the evolution. The local searchers aim to detect a value of the scale factor that corresponds to a refined offspring and thus tend to correct "weak" individuals.

IV. The Dynamics of DE Mutation Strategies

In this section, we investigate the impact of DE dynamics, i.e. the exploration/exploitation capabilities of the different DE mutation strategies. Our findings suggest that the individuals evolved through some of the original DE mutation strategies sometimes tend to gather around minimizers of the objective function. This motivates our approach, which aims to appropriately select neighboring individuals for incorporation in each mutation strategy. The goal is to efficiently guide the evolution of the population towards a global optimum, without sacrificing the search capabilities of the DE algorithm.

The exploration and exploitation capabilities of different DE mutation strategies were studied in [21], [22], where it was shown that not all DE search operators have the same impact on the exploration/exploitation of the search space. Thus, the choice of the most efficient mutation operator can be cumbersome and problem dependent.

In general, we can distinguish between mutation operators that promote exploration and operators that promote exploitation. An observation of the equations of the mutation operators (Eqs. (2)–(7)), reveals that operators that incorporate the best individual (e.g. DE/best/1, DE/best/2, and DE/current-to-best/1) favor exploitation, since the mutant individuals are strongly attracted around the current best individual. Note that DE/best/2 usually exhibits better exploration than DE/best/1, because it includes one more difference of randomly selected individuals, which adds one more component of random variation in each mutation. In contrast, mutation operators that incorporate either randomly chosen individuals or many differences of randomly chosen individuals (e.g. DE/rand/1 and DE/rand/2) enhance the exploration of the search space, since a high degree of random variability affects each mutation. Again, although DE/current-to-best/2 is based on DE/current-to-best/1 the utilization of a second difference vector further promotes the exploration of the search space [12], [13], [15].

Next, we investigate the impact of the dynamics of different DE mutation strategies on the population. Experimental simulations indicate that DE mutation strategies tend to distribute the individuals of the population in the vicinity of the objective function's minima. Exploitative strategies rapidly gather all the individuals to the basin of attraction of a single minimum, whereas explorative strategies tend to spread the individuals around many minima.

To demonstrate this we employ as a case study the two-dimensional Shekel's Foxholes benchmark function, illustrated in Fig. 1. This function has twenty four distinct local minima and one global minimum $f(-32, 32) = 0.998004$, in the range $[-65.536, 65.536]^2$ [75].

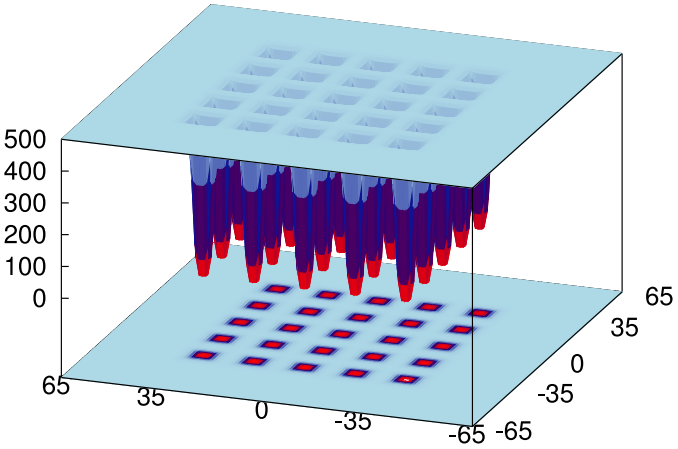

Fig. 1. 3–D Plot of the Shekel's Foxholes function

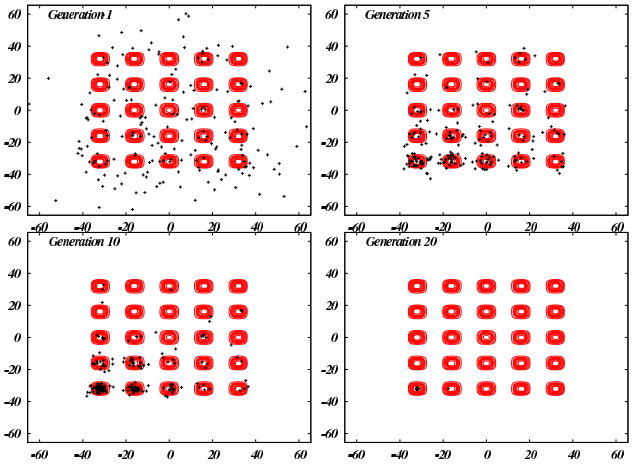

Fig. 2. DE/best/1/bin population after 1, 5, 10, and 20 generations

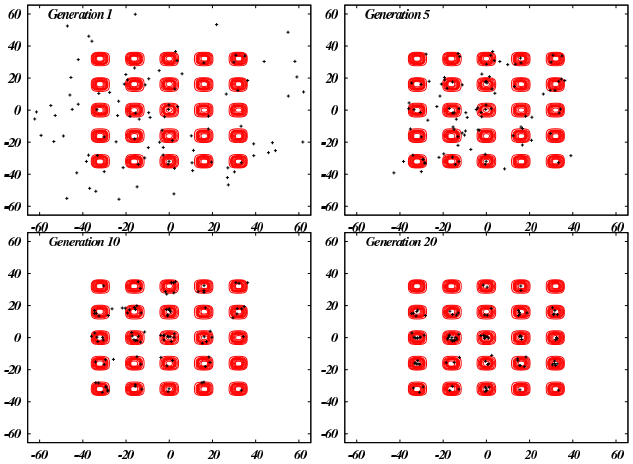

Fig. 3. DE/rand/1/bin population after 1, 5, 10, and 20 generations

We utilize two DE variants with different dynamics; the explorative DE/rand/1 and the exploitative DE/best/1. Contour plots of the Shekel's Foxholes function and the positions of a population of 100 individuals after 1, 5, 10, 20 generations of DE/best/1 and DE/rand/1 are depicted in Figs. 2 and 3, respectively. The two figures show that both DE/best/1 and DE/rand/1 first explore the search space around their initial population positions. The exploitative character of DE/best/1 causes the individuals to gather rapidly around the basin of attraction of the global minimum, (see Fig. 2). On the other hand, DE/rand/1, Fig. 3, spreads the individuals over many minima locations, before gathering them around the global minimum.

To study the *clustering tendency* of different DE mutation strategies we utilize a statistical test called the Hopkins test [76]. Clustering tendency is a well known concept in the cluster analysis literature that deals with the problem of determining the presence or absence of a clustering structure in a data set [77]. The Hopkins test relies on the distances between a number of vectors which are randomly placed in the search space, and the vectors of a data set, $X = \{x_i, i = 1, 2, \ldots, NP\}$, which in our case correspond to the individuals of the population. More specifically, let $Y = \{y_i, i = 1, 2, \ldots, M\}, M \ll NP$, with typically $M = NP/10$, be a set of vectors that are uniformly distributed in the search space. In addition, let $X_1 \subset X$ be a set of M randomly chosen vectors from X. Let d_j be the distance of $y_j \in Y$ to its closest vector in X_1, denoted by x_j, and δ_j be the distance between x_j and its nearest neighbor in $X_1 \setminus \{x_j\}$. The Hopkins statistic involves the l-th powers of d_j and δ_j and is defined as [77]:

$$h = \frac{\sum_{j=1}^{M} d_j^l}{\sum_{j=1}^{M} d_j^l + \sum_{j=1}^{M} \delta_j^l}.$$

This statistic compares the nearest neighbor distribution of the points in X_1 with that from the points in Y. When the dataset X contains clusters, the distances between nearest neighbors in X_1 are expected to be small on average, and h assumes relatively large values. Therefore, large values of h indicate the presence of a clustering structure in the dataset, while small values of h indicate the presence of regularly spaced points. A value around 0.5 indicates that the vectors of the dataset X are randomly distributed over the search space.

Due to the stochastic nature of H-measure, for every generation in every simulation we calculate the H-measure value 100 times, by considering different random solutions. Thus, in Fig. 4, we illustrate the mean value of the H-measure at each generation, obtained from 100 independent simulations for the 30-dimensional versions of the Shifted Sphere and Shifted Griewank functions [78]. Error bars around the mean depict the standard deviation of the H-measure. The Shifted Sphere is a simple unimodal function, while the Shifted Griewank is highly multimodal. These benchmarks were chosen to investigate the behavior of the DE mutation operators in two qualitatively different problems.

As shown, all mutation strategies exhibit large H-measure values within the first 100 generations, indicative of a strong clustering structure, even from these initial stages of the evolution. Also, the relative values of the H-measure for the different strategies indicate an ordering with respect to their exploitation tendency. DE/best/1 appears to be the most exploitative operator, and DE/current-to-best/1 behaves similarly. The least exploitative operator is DE/rand/2.

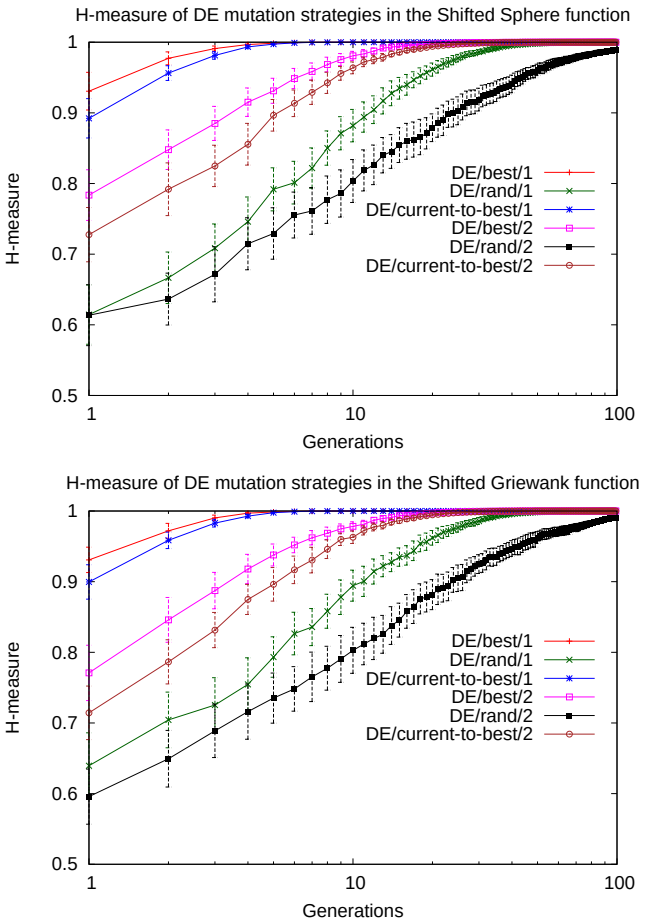

Fig. 4. H-measure of six classic DE mutation strategies on the Shifted Sphere and on the Shifted Griewank

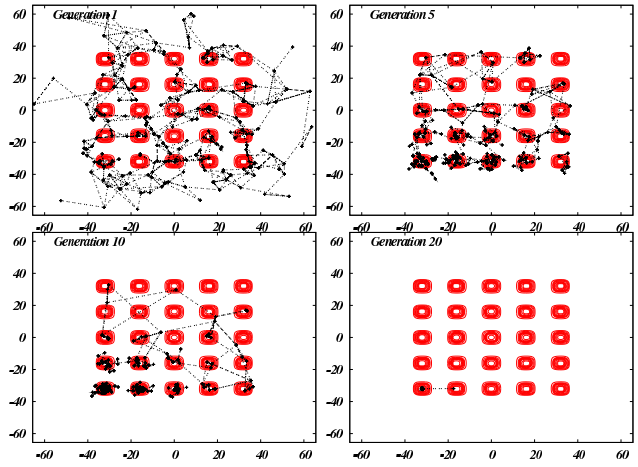

Fig. 5. The 5-Nearest Neighbors graph for the DE/best/1/bin population after 1, 5, 10 and 20 generations

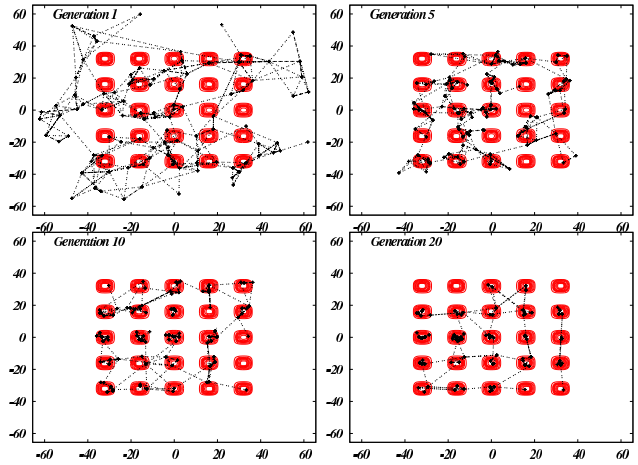

Fig. 6. The 5-Nearest Neighbors graph for the DE/rand/1/bin population after 1, 5, 10 and 20 generations

In this work, we attempt to take advantage of this clustering behavior. To this end, we modify the way that DE mutation strategies choose individuals to form the difference vectors, which are employed to mutate the base vector. More specifically, to generate a mutant individual, we propose to use individuals in the vicinity of the parent vector that probably reside in the same cluster, instead of uniformly random individuals. This has the potential to rapidly exploit the regions of minima, and thus accelerate convergence.

To illustrate this concept, Figs. 5 and 6 show the 5-nearest neighbors graphs for the DE/best/1 and DE/rand/1 populations of the two-dimensional Shekel's Foxholes function, after 1, 5, 10, and 20 generations, respectively. As shown, selecting individuals amongst the 5-nearest neighbors to produce mutant individuals will achieve our goal of exploiting local information. The occasional connections between individuals clustered around different local minima, suggest that the exploration abilities of the algorithm will not be severely hindered. We further promote exploration by introducing stochasticity into the selection mechanism, instead of just using a prespecified number of nearest neighbors. In particular, we assign a probability of selection to each individual which is inversely proportional to its distance to the parent individual. In the next section, we describe in detail the proposed method.

V. The Proposed Proximity-based Mutation Framework

As shown in the previous section, it is possible to guide the evolution towards a global optimum without compromising the algorithm's search capabilities by incorporating information from neighboring individuals. In this section, we discuss the main concepts behind a Proximity-based Differential Evolution framework (Pro DE). The easiest way to implement the proposed approach would be to select the indices r_1, r_2, r_3 of the individuals involved in mutation, to correspond to the 3-nearest neighbors of the parent individual, rather than being random. However, such an approach could result in an exceedingly exploitative (greedy) algorithm, especially during the first steps of the evolution where such a behavior can be detrimental. Instead, we propose a stochastic selection of $r_i, i \in \{1, 2, 3\}$ in the mutation procedure.

Let us consider a population of NP, D-dimensional individuals $P_g = [x_g^1, x_g^2, \ldots, x_g^{NP}]$. We calculate the affinity matrix, R_d, based on real distances between individuals. Thus, the

Algorithm 2 Pro DE/rand/1: proximity-based mutation algorithmic scheme for DE/rand/1

/* **Mutation step** */
Calculate the probability matrix R_p based on Eq. (10)
Utilize a roulette wheel to select indices $r_1^*, r_2^*, r_3^* \in S_r = \{1, 2, \ldots, NP\} \setminus \{i\}$ based on probability matrix R_p
/* For each target vector x_g^i generate the corresponding mutant vector v_g^i using Eq. (3) */
for $j = 1$ to D **do**
$v_{j,g}^i = x_{j,g}^{r_1^*} + F(x_{j,g}^{r_2^*} - x_{j,g}^{r_3^*})$
end for

$R_d(i, j)$ element of the matrix corresponds to the distance between the i-th and the j-th individuals:

$$R_d = \begin{bmatrix} 0 & \|x_g^1, x_g^2\| & \cdots & \|x_g^1, x_g^{NP}\| \\ \|x_g^2, x_g^1\| & 0 & \cdots & \|x_g^2, x_g^{NP}\| \\ \|x_g^3, x_g^1\| & \|x_g^3, x_g^2\| & 0 & \|x_g^3, x_g^{NP}\| \\ \vdots & \vdots & \ddots & \vdots \\ \|x_g^{NP}, x_g^1\| & \|x_g^{NP}, x_g^2\| & \cdots & 0 \end{bmatrix},$$

where $\|x, y\|$ is a distance measure between the x and y individuals. In the case of decision variables with different search ranges, a scale-invariant distance measure (e.g. the Mahalanobis distance [77]) needs to be used to avoid any dependence on the scale of the variables. It has been shown that a fixed number of points becomes increasingly "sparse" as the dimensionality increases [79]. Therefore, in very high dimensional problems p-norms, with $p \leqslant 1$ can be used [80]. In this paper we use Euclidean distances, since in all the considered problems all the variables have equal ranges.

The affinity matrix is symmetric, due to the symmetric property of the distance. Thus, only the upper triangular part of R_d needs to be calculated. Based on the R_d matrix, we calculate a probability matrix R_p, in which each element $R_p(i, j)$ represents a probability between the i-th and j-th individual with respect to the i-th row. The probability of the i-th individual is inversely proportional to the distance of the j-th individual, i.e. the individual of the row with the minimum distance has the maximum probability:

$$R_p(i, j) = 1 - \frac{R_d(i, j)}{\sum_i R_d(i, j)}, \tag{10}$$

where $i, j = 1, 2, \ldots, NP$. Thus, we incorporate a stochastic selection procedure, in the form of a simple roulette wheel selection without replacement [2], to obtain the indices $r_1^*, r_2^*, r_3^* \in S_r = \{1, 2, \ldots, NP\} \setminus \{i\}$.

A notable observation is that it is not necessary to repeatedly calculate the probability matrix in every generation. As it is previously described, the key role of the proximity framework is to exploit possible clustering structure of the population over the problem's minima and subsequently incorporate that information in the evolution phase of the algorithm. To this end, whenever an individual passes the selection operator its position is altered and the affinity matrix should be updated. Depending either on the computational cost we are willing to pay, or on the characteristics of the DE variant and the considered problem, the R_p matrix can be calculated in every or every few generations. It is evident that when the affinity matrix is not calculated in every generation, it contains errors. Inaccurate information in the affinity matrix may not significantly affect the algorithm's dynamics, due to the desired randomness of indices r_i. In this paper, we propose to update the affinity matrix after each change of an individual's position, which is in essence at every generation.

Some DE variants incorporate operators that rapidly change the position of many individuals either by the greediness of the evolution operator, e.g. the mutation strategies DE/best/1 DE/current-to-best/1, DE/best/2, or due to an extra operator that influences the evolution dynamics, e.g. the population of opposition-based DE [40], [41]. In these cases, we must immediately transfer this information to the proximity framework, and thus update the affinity matrix in every generation.

The proposed proximity-based framework affects only the mutation step, hence it could be directly applied to any DE mutation strategy. The application of this framework for DE/rand/1 is demonstrated in Algorithm 2. We use the notation Pro DE/rand/1 to designate that the proposed proximity-based framework is used.

VI. Experimental Results

In this section, we perform an extensive experimental evaluation of the proposed framework. We employ the CEC 2005 benchmark suite which consists of 25 scalable benchmark functions [78]. Based on their characteristics, the functions of the CEC 2005 benchmark set can be divided into the following four classes. Functions $cf_1 - cf_5$ are unimodal; $cf_6 - cf_{12}$ are basic multimodal functions; cf_{13} and cf_{14} are expanded multimodal functions, and $cf_{15} - cf_{25}$ are hybrid compositions of functions with a huge number of local minima. A thorough description of this test set is provided in [78].

To perform a comprehensive evaluation and highlight the different aspects of the proposed framework, we divide the presentation of the experimental results into four subsections. We first incorporate the proposed proximity framework into the original DE mutation strategies and compare the performance of each strategy with its "Pro DE" variant (Subsection VI-A). Subsequently, we discuss the suitability of the proximity framework for other well-known DE variants (Subsection VI-B). In Subsection VI-C the computational cost of the proposed framework is discussed. Finally, an overall performance comparison among all the considered approaches is provided in Subsection VI-D.

A. The Proximity-based Framework in DE

In this section we incorporate the proposed proximity-based framework in each of the six original DE mutation strategies. To maintain a reliable and fair comparison we employ parameter settings that are extensively used in the literature. In more detail, the parameter settings used are:

(a) Population size, $NP = 100$ [15], [31], [75].
(b) Mutation factor $F = 0.5$ [7], [15], [29], [31].
(c) Recombination factor $CR = 0.9$ [7], [15], [29], [31].

TABLE I
ERROR VALUES OF THE ORIGINAL DE MUTATION STRATEGIES AND THEIR CORRESPONDING PROXIMITY-BASED VARIANTS OVER THE 30–DIMENSIONAL CEC 2005 BENCHMARK SET

	DE/best/1		Pro DE/best/1			DE/rand/1		Pro DE/rand/1			DE/current-to-best/1		Pro DE/current-to-best/1		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	**1.537e+02**	2.477e+02	3.054e+02	2.926e+02	-
cf_2	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	**1.973e+03**	1.338e+03	2.137e+03	1.163e+03	=
cf_3	2.756e+04	1.713e+04	**1.493e+04**	1.042e+04	+	5.077e+05	3.724e+05	**4.096e+05**	2.338e+05	=	**2.689e+06**	2.711e+06	3.130e+06	2.395e+06	=
cf_4	**2.159e+02**	3.773e+02	3.092e+02	6.702e+02	=	2.410e-02	2.700e-02	**1.700e-03**	3.406e-03	+	**3.669e+02**	3.578e+02	4.871e+02	4.515e+02	=
cf_5	**1.555e+03**	1.081e+03	2.184e+03	7.268e+02	-	**1.470e-02**	3.191e-02	1.183e+02	1.372e+02	-	**4.603e+03**	9.657e+02	5.766e+03	1.365e+03	-
cf_6	1.595e+00	1.973e+00	**1.435e+00**	1.933e+00	=	**2.255e+00**	1.406e+00	3.625e+00	2.985e+00	-	**9.147e+06**	1.154e+07	2.884e+07	6.154e+07	-
cf_7	4.764e+03	1.943e+02	**4.696e+03**	1.837e-12	+	**4.696e+03**	7.709e-03	4.696e+03	1.837e-12	-	**5.001e+03**	2.009e+02	5.242e+03	1.685e+02	-
cf_8	**2.095e+01**	6.008e-02	2.101e+01	6.060e-02	-	2.094e+01	4.480e-02	2.094e+01	5.320e-02	=	2.094e+01	5.006e-02	**2.093e+01**	6.071e-02	=
cf_9	1.058e+02	2.711e+01	**9.199e+01**	2.454e+01	+	1.325e+02	2.453e+01	**1.641e+01**	5.282e+00	+	**6.895e+01**	1.639e+01	8.097e+01	1.884e+01	-
cf_{10}	**1.306e+02**	4.933e+01	1.379e+02	3.634e+01	=	1.822e+02	7.871e+00	**3.298e+01**	1.293e+01	+	**8.895e+01**	2.839e+01	1.001e+02	2.865e+01	-
cf_{11}	**2.188e+01**	4.143e+00	2.295e+01	4.283e+00	=	3.903e+01	1.224e+00	**1.180e+01**	4.040e+00	+	**1.447e+01**	2.950e+00	1.753e+01	3.331e+00	-
cf_{12}	5.717e+04	5.796e+04	**1.250e+03**	1.787e+03	+	2.553e+04	2.188e+04	**2.366e+03**	2.147e+03	+	6.172e+04	4.360e+04	**2.441e+04**	1.407e+04	+
cf_{13}	**9.802e+00**	3.429e+00	1.079e+01	3.937e+00	=	1.542e+01	8.584e-01	**2.813e+00**	6.075e-01	+	5.306e+00	3.302e+00	**5.056e+00**	2.991e+00	=
cf_{14}	**1.217e+01**	6.716e-01	1.250e+01	6.501e-01	-	1.356e+01	1.382e-01	**1.315e+01**	2.160e-01	+	1.194e+01	3.418e-01	**1.172e+01**	3.394e-01	+
cf_{15}	5.226e+02	8.110e+01	**4.493e+02**	9.302e+01	+	**2.520e+02**	8.862e+01	3.960e+02	5.330e+01	-	**4.339e+02**	8.339e+01	4.594e+02	1.039e+02	=
cf_{16}	2.825e+02	1.383e+02	**2.476e+02**	1.195e+02	=	2.187e+02	3.637e+01	**5.613e+01**	5.055e+01	+	**2.228e+02**	1.639e+02	2.333e+02	1.672e+02	=
cf_{17}	3.199e+02	1.488e+02	**2.614e+02**	1.284e+02	=	2.461e+02	5.148e+01	**8.541e+01**	5.296e+01	+	2.346e+02	1.639e+02	**2.062e+02**	1.429e+02	=
cf_{18}	**9.292e+02**	3.065e+01	9.477e+02	3.631e+01	-	9.034e+02	4.932e-01	**8.824e+02**	4.422e+01	+	**9.504e+02**	2.106e+01	9.609e+02	3.898e+01	-
cf_{19}	**9.235e+02**	1.746e+01	9.394e+02	4.490e+01	-	9.033e+02	2.236e-01	**8.975e+02**	2.907e+01	+	**9.518e+02**	2.114e+01	9.661e+02	3.295e+01	-
cf_{20}	**9.305e+02**	3.041e+01	9.510e+02	3.363e+01	-	9.033e+02	2.022e-01	**8.952e+02**	3.211e+01	+	**9.411e+02**	2.931e+01	9.582e+02	4.699e+01	-
cf_{21}	8.314e+02	3.085e+02	**6.858e+02**	2.950e+02	=	5.582e+02	1.762e+02	**5.000e+02**	0.000e+00	+	**8.315e+02**	2.839e+02	9.096e+02	2.673e+02	=
cf_{22}	**9.952e+02**	8.255e+01	1.051e+03	5.977e+01	-	**8.591e+02**	1.389e+01	9.031e+02	9.625e+00	-	**9.777e+02**	4.260e+01	9.999e+02	3.521e+01	-
cf_{23}	8.146e+02	3.087e+02	**7.263e+02**	2.973e+02	=	5.697e+02	1.907e+02	**5.060e+02**	4.243e+01	+	**8.596e+02**	2.878e+02	8.808e+02	2.743e+02	=
cf_{24}	9.725e+02	2.424e+02	**3.463e+02**	3.530e+02	+	9.785e+02	1.124e+02	**2.000e+02**	0.000e+00	+	**5.809e+02**	3.556e+02	5.932e+02	3.758e+02	=
cf_{25}	**1.675e+03**	1.595e+01	1.713e+03	1.701e+01	-	1.649e+03	2.918e+00	**1.641e+03**	6.573e+00	+	**1.669e+03**	1.306e+01	1.700e+03	1.108e+01	-
Total number of (+/=/-):				6/11/8					16/4/5					2/11/12	

	DE/best/2		Pro DE/best/2			DE/rand/2		Pro DE/rand/2			DE/current-to-best/2		Pro DE/current-to-best/2		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	4.075e-01	1.397e-01	**0.000e+00**	0.000e+00	+	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_2	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	2.789e+03	6.676e+02	**1.225e+02**	4.465e+01	+	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_3	1.842e+05	9.642e+04	**1.245e+05**	7.092e+04	+	3.793e+07	8.031e+06	**4.471e+06**	1.323e+06	+	8.594e+04	5.361e+04	**5.417e+04**	4.670e+04	+
cf_4	3.477e+02	1.951e+03	**2.000e-05**	1.414e-04	+	6.998e+03	1.553e+03	**8.728e+02**	3.046e+02	+	0000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_5	**4.586e+01**	3.180e+02	6.732e+01	1.132e+02	-	**1.611e+03**	4.637e+02	2.060e+03	2.780e+02	-	**0.000e+00**	0.000e+00	9.150e-02	6.274e-02	-
cf_6	**5.582e-01**	1.397e+00	1.196e+00	1.846e+00	=	3.612e+03	1.890e+03	**1.960e+01**	8.975e-01	+	**1.595e-01**	7.892e-01	2.392e-01	9.565e-01	=
cf_7	**4.609e+03**	1.184e+02	4.696e+03	6.966e-03	-	**4.671e+03**	2.017e+01	4.811e+03	1.310e+01	-	**4.695e+03**	5.519e+00	4.696e+03	1.837e-12	-
cf_8	2.095e+01	4.929e-02	**2.094e+01**	5.294e-02	=	2.095e+01	4.303e-02	**2.095e+01**	5.467e-02	=	2.095e+01	4.476e-02	**2.094e+01**	6.119e-02	=
cf_9	1.725e+02	1.609e+01	**4.493e+01**	1.113e+01	+	2.061e+02	1.248e+01	**1.878e+02**	1.001e+01	+	**1.694e+02**	9.850e+00	1.724e+02	8.759e+00	=
cf_{10}	1.985e+02	1.780e+01	**1.306e+02**	6.338e+01	+	2.321e+02	1.113e+01	**2.061e+02**	1.149e+01	+	**1.898e+02**	1.147e+01	1.899e+02	8.673e+00	=
cf_{11}	3.231e+01	9.431e+00	**3.133e+01**	1.200e+01	=	3.967e+01	1.051e+00	**3.964e+01**	1.050e+00	=	**3.960e+01**	1.126e+00	3.964e+01	1.048e+00	=
cf_{12}	1.487e+05	2.571e+05	**2.233e+03**	3.439e+03	+	9.199e+05	1.270e+05	**2.588e+05**	1.100e+05	+	4.450e+04	7.800e+04	**1.285e+03**	1.542e+03	+
cf_{13}	1.607e+01	1.464e+00	**3.856e+00**	1.783e+00	+	2.360e+01	1.429e+00	**1.738e+01**	9.139e-01	+	1.584e+01	9.103e-01	**1.534e+01**	8.793e-01	+
cf_{14}	**1.309e+01**	2.856e-01	1.329e+01	1.842e-01	-	1.373e+01	1.527e-01	**1.339e+01**	1.586e-01	+	1.343e+01	1.627e-01	**1.329e+01**	1.253e-01	+
cf_{15}	4.284e+02	7.646e+01	**3.556e+02**	1.130e+02	+	4.220e+02	7.917e+01	**4.020e+02**	1.414e+01	=	**3.439e+02**	1.107e+02	3.790e+02	9.150e+01	=
cf_{16}	2.971e+02	9.141e+01	**2.358e+02**	1.330e+02	+	2.793e+02	3.921e+01	**2.317e+02**	1.016e+01	+	2.953e+02	9.467e+01	**2.633e+02**	7.366e+01	=
cf_{17}	3.334e+02	1.004e+02	**3.122e+02**	1.191e+02	=	3.068e+02	3.873e+01	**2.565e+02**	1.290e+01	+	3.024e+02	8.764e+01	**2.779e+02**	8.574e+01	=
cf_{18}	9.071e+02	3.466e+00	**9.004e+02**	3.010e+01	+	**9.063e+02**	1.813e-01	9.096e+02	1.215e+00	-	9.055e+02	1.686e+00	**8.911e+02**	3.720e+01	=
cf_{19}	9.117e+02	2.165e+01	**8.985e+02**	3.329e+01	+	**9.062e+02**	1.883e-01	9.096e+02	1.102e+00	-	9.054e+02	1.552e+00	**8.784e+02**	4.699e+01	=
cf_{20}	9.078e+02	4.287e+00	**8.936e+02**	3.825e+01	=	**9.062e+02**	2.183e-01	9.096e+02	1.019e+00	-	9.053e+02	1.561e+00	**8.888e+02**	3.918e+01	=
cf_{21}	1.030e+03	1.833e+02	**5.603e+02**	1.218e+02	+	8.957e+02	2.830e+02	**5.000e+02**	0.000e+00	+	9.477e+02	2.542e+02	**5.300e+02**	9.091e+01	+
cf_{22}	**8.980e+02**	3.467e+01	9.277e+02	1.944e+01	-	**8.553e+02**	1.974e+01	9.459e+02	7.421e+00	-	**8.754e+02**	2.075e+01	9.124e+02	1.066e+01	-
cf_{23}	1.025e+03	1.808e+02	**5.528e+02**	1.233e+02	+	8.680e+02	2.891e+02	**5.000e+02**	0.000e+00	+	1.004e+03	2.055e+02	**5.180e+02**	7.197e+01	+
cf_{24}	9.185e+02	9.400e+01	**2.000e+02**	0.000e+00	+	9.814e+02	2.377e+01	**2.000e+02**	0.000e+00	+	9.913e+02	1.666e+01	**2.000e+02**	0.000e+00	+
cf_{25}	**1.644e+03**	1.286e+01	1.659e+03	1.112e+01	-	**1.651e+03**	2.052e+00	1.688e+03	3.247e+00	-	**1.653e+03**	5.448e+00	1.672e+03	3.710e+00	-
Total number of (+/=/-):				13/7/5					15/3/7					7/14/4	

The population for all DE variants, over all the benchmark functions, was initialized using a uniform random number distribution with the same random seeds.

To evaluate the performance of the algorithms we will use the *solution error measure*, defined as $f(x') - f(x^\star)$, where $x^\star$ is the global optimum of the benchmark function and x' is the best solution achieved after $10^4 \cdot D$ function evaluations [78], where D is the dimensionality of the problem at hand. Each algorithm was executed independently 100 times, to obtain an estimate of the mean solution error and its standard deviation. For each pair of original mutation strategy and its proximity-based variant, we use boldface font to indicate the best performance in terms of mean solution error. To evaluate the statistical significance of the observed performance differences we apply a two-sided Wilcoxon rank sum test between the original mutation strategies and their proximity-based variants. The null hypothesis in each test is that the samples compared are independent samples from identical continuous distributions with equal medians. We mark with "+" the cases when the null hypothesis is rejected at the 5% significance level and the proximity-based variant exhibits superior performance, with "-" when the null hypothesis is rejected at the same level of significance and the proximity-based variant exhibits inferior performance and with "=" when the performance difference is not statistically significant. At the bottom of each table, for each pair, we also show the total number of the aforementioned statistical significant cases (+/=/-). Finally, we underline the algorithm that exhibits the best result in each benchmark function.

TABLE II
ERROR VALUES OF THE ORIGINAL DE MUTATION STRATEGIES AND THEIR CORRESPONDING PROXIMITY-BASED VARIANTS OVER THE 50–DIMENSIONAL CEC 2005 BENCHMARK SET

	DE/best/1		Pro DE/best/1			DE/rand/1		Pro DE/rand/1			DE/current-to-best/1		Pro DE/current-to-best/1		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	**2.198e+02**	1.792e+02	5.241e+02	3.542e+02	-
cf_2	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	3.960e+03	9.307e+02	**3.254e+02**	1.093e+02	+	**2.136e+03**	1.128e+03	2.662e+03	1.368e+03	-
cf_3	3.270e+05	1.439e+05	**1.440e+05**	7.123e+04	+	5.404e+07	1.310e+07	**7.509e+06**	1.766e+06	+	**1.089e+07**	6.857e+06	1.196e+07	6.456e+06	=
cf_4	3.473e+03	3.649e+03	**1.133e+03**	1.312e+03	+	1.180e+04	3.332e+03	**2.476e+03**	6.914e+02	+	**1.489e+03**	1.068e+03	1.532e+03	1.042e+03	=
cf_5	4.674e+03	1.098e+03	**4.608e+03**	1.038e+03	=	**1.709e+03**	6.938e+02	2.192e+03	2.949e+02	-	**7.462e+03**	1.326e+03	7.982e+03	1.078e+03	-
cf_6	**8.771e-01**	1.668e+00	1.116e+00	1.808e+00	=	4.231e+01	1.182e+01	**3.855e+01**	1.776e+01	+	**1.078e+07**	1.237e+07	3.382e+07	3.099e+07	-
cf_7	6.235e+03	1.902e+02	**6.195e+03**	4.594e-12	+	**6.195e+03**	4.594e-12	6.199e+03	5.281e-01	-	**6.669e+03**	1.795e+02	6.771e+03	1.407e+02	-
cf_8	2.113e+01	3.904e-02	**2.113e+01**	3.087e-02	=	2.114e+01	3.330e-02	**2.114e+01**	4.345e-02	=	2.113e+01	4.841e-02	**2.112e+01**	3.798e-02	=
cf_9	2.091e+02	4.272e+01	**1.951e+02**	3.987e+01	=	3.468e+02	1.199e+01	**1.382e+02**	1.366e+01	+	**1.406e+02**	2.939e+01	1.554e+02	2.977e+01	-
cf_{10}	**2.378e+02**	5.911e+01	2.617e+02	6.366e+01	=	3.763e+02	1.578e+01	**3.529e+02**	1.482e+01	+	**1.717e+02**	4.524e+01	2.107e+02	4.353e+01	-
cf_{11}	4.269e+01	7.379e+00	**4.210e+01**	5.234e+00	=	7.264e+01	1.212e+00	**7.263e+01**	1.614e+00	=	**2.950e+01**	4.451e+00	3.141e+01	5.123e+00	-
cf_{12}	2.749e+05	2.925e+05	**6.740e+03**	6.345e+03	+	2.049e+06	5.887e+05	**9.192e+03**	7.919e+03	+	2.505e+05	1.137e+05	**7.933e+04**	3.736e+04	+
cf_{13}	2.281e+01	7.498e+00	**2.107e+01**	7.419e+00	=	3.296e+01	1.446e+00	**2.238e+01**	2.494e+00	+	**1.412e+01**	8.942e+00	1.546e+01	9.144e+00	=
cf_{14}	2.179e+01	5.072e-01	**2.108e+01**	7.160e-01	+	2.339e+01	1.486e-01	**2.304e+01**	1.389e-01	+	2.186e+01	4.091e-01	**2.183e+01**	5.069e-01	=
cf_{15}	4.990e+02	7.981e+01	**4.248e+02**	5.845e+01	+	**2.047e+02**	2.819e+01	4.000e+02	0.000e+00	-	4.489e+02	5.001e+01	**4.298e+02**	4.158e+01	=
cf_{16}	2.520e+02	1.058e+02	**2.391e+02**	1.044e+02	=	2.715e+02	1.470e+01	**2.479e+02**	9.706e+00	+	1.812e+02	1.157e+02	**1.806e+02**	1.001e+02	=
cf_{17}	**2.616e+02**	9.805e+01	2.772e+02	1.097e+02	=	3.049e+02	2.457e+01	**2.735e+02**	1.049e+01	+	**1.724e+02**	8.798e+01	1.840e+02	1.064e+02	=
cf_{18}	**9.519e+02**	2.242e+01	9.958e+02	2.375e+01	-	9.151e+02	6.997e-01	**8.928e+02**	4.981e+01	+	**9.745e+02**	1.704e+01	9.930e+02	1.688e+01	-
cf_{19}	**9.489e+02**	1.704e+01	9.903e+02	2.706e+01	-	9.154e+02	5.033e-01	**8.836e+02**	5.534e+01	+	**9.753e+02**	1.585e+01	9.914e+02	1.853e+01	-
cf_{20}	**9.509e+02**	2.111e+01	9.868e+02	2.346e+01	-	9.153e+02	5.553e-01	**9.001e+02**	4.417e+01	+	**9.736e+02**	2.018e+01	9.962e+02	1.766e+01	-
cf_{21}	1.042e+03	2.088e+01	**6.970e+02**	3.033e+02	+	1.004e+03	1.101e+00	**5.000e+02**	0.000e+00	+	**7.930e+02**	2.516e+02	9.818e+02	2.708e+02	-
cf_{22}	**9.837e+02**	4.562e+01	1.071e+03	4.943e+01	-	**9.061e+02**	3.577e+00	9.586e+02	1.018e+01	-	**1.020e+03**	3.024e+01	1.058e+03	2.456e+01	-
cf_{23}	1.005e+03	1.366e+02	**6.700e+02**	2.821e+02	+	1.003e+03	1.029e+00	**5.000e+02**	0.000e+00	+	**7.381e+02**	2.312e+02	8.923e+02	2.823e+02	-
cf_{24}	**1.103e+03**	7.326e+01	1.126e+03	3.130e+02	-	1.038e+03	1.717e+00	**2.000e+02**	0.000e+00	+	**1.022e+03**	3.010e+02	1.180e+03	1.258e+02	-
cf_{25}	**1.715e+03**	1.764e+01	1.777e+03	1.898e+01	-	**1.688e+03**	2.591e+00	1.709e+03	3.603e+00	-	**1.717e+03**	1.094e+01	1.755e+03	1.220e+01	-
Total number of (+/=/-):				8/11/6					17/3/5					1/8/16	

	DE/best/2		Pro DE/best/2			DE/rand/2		Pro DE/rand/2			DE/current-to-best/2		Pro DE/current-to-best/2		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	6.899e+03	8.880e+02	**1.217e+02**	3.143e+01	+	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_2	6.836e+01	8.537e+01	**5.139e+00**	2.709e+00	+	9.715e+04	8.252e+03	**5.834e+04**	6.223e+03	+	5.934e+02	1.328e+02	**1.028e+02**	2.798e+01	+
cf_3	4.328e+06	1.598e+06	**2.641e+06**	9.875e+05	+	4.859e+08	6.865e+07	**2.687e+08**	4.238e+07	+	1.451e+07	2.936e+06	**6.597e+06**	1.459e+06	+
cf_4	5.346e+03	6.011e+03	**1.523e+03**	8.763e+02	+	1.390e+05	1.502e+04	**7.970e+04**	8.841e+03	+	4.268e+03	9.739e+02	**2.107e+03**	5.709e+02	+
cf_5	2.746e+03	1.769e+03	**2.740e+03**	5.324e+02	=	2.126e+04	1.720e+03	**1.785e+04**	9.111e+02	+	**1.035e+03**	1.155e+03	2.287e+03	5.298e+02	-
cf_6	1.439e+01	1.028e+01	**3.434e+00**	2.767e+00	+	6.040e+08	1.238e+08	**4.587e+05**	1.577e+05	+	2.583e+01	1.463e+01	**1.622e+01**	1.196e+01	+
cf_7	**6.205e+03**	5.419e+01	6.322e+03	3.314e+01	-	**6.201e+03**	2.003e+00	8.391e+03	1.181e+02	-	**6.195e+03**	4.594e-12	6.236e+03	6.720e+00	-
cf_8	2.114e+01	3.572e-02	**2.114e+01**	3.384e-02	=	2.114e+01	3.143e-02	**2.113e+01**	4.171e-02	=	2.113e+01	3.968e-02	**2.113e+01**	3.402e-02	=
cf_9	3.806e+02	3.573e+01	**2.777e+02**	1.001e+02	+	4.596e+02	1.477e+01	**4.413e+02**	1.935e+01	+	**3.653e+02**	1.772e+01	3.933e+02	1.873e+01	-
cf_{10}	4.223e+02	2.514e+01	**4.072e+02**	2.742e+01	+	5.344e+02	1.467e+01	**4.671e+02**	1.751e+01	+	**3.988e+02**	1.363e+01	4.067e+02	2.008e+01	-
cf_{11}	**7.050e+01**	7.783e+00	7.286e+01	1.513e+00	=	**7.253e+01**	1.553e+00	7.287e+01	1.267e+00	=	7.309e+01	1.275e+00	**7.249e+01**	1.681e+00	=
cf_{12}	3.813e+05	3.877e+05	**6.504e+03**	6.901e+03	+	4.879e+06	3.996e+05	**2.425e+06**	1.833e+05	+	1.538e+05	2.758e+05	**1.147e+05**	1.969e+05	+
cf_{13}	3.396e+01	2.146e+00	**3.088e+01**	2.132e+00	+	2.873e+05	1.109e+05	**4.247e+01**	1.876e+00	+	3.366e+01	1.824e+00	**3.293e+01**	1.321e+00	+
cf_{14}	2.314e+01	2.062e-01	**2.306e+01**	1.928e-01	=	2.360e+01	1.387e-01	**2.318e+01**	1.702e-01	+	2.328e+01	1.458e-01	**2.305e+01**	1.407e-01	+
cf_{15}	3.927e+02	5.897e+01	**2.791e+02**	8.862e+01	+	9.125e+02	1.553e+01	**4.453e+02**	2.914e+00	+	**2.760e+02**	8.419e+01	3.240e+02	1.079e+02	=
cf_{16}	3.278e+02	4.836e+01	**3.120e+02**	4.674e+01	+	3.805e+02	1.884e+01	**3.251e+02**	1.166e+01	+	3.225e+02	4.977e+01	**3.001e+02**	3.625e+01	=
cf_{17}	3.666e+02	5.547e+01	**3.562e+02**	5.005e+01	=	4.378e+02	2.531e+01	**3.772e+02**	1.603e+01	+	3.428e+02	4.725e+01	**3.329e+02**	3.840e+01	=
cf_{18}	9.202e+02	8.482e+00	**8.871e+02**	1.268e+02	+	**9.412e+02**	6.037e+00	1.001e+03	6.688e+00	-	9.157e+02	1.964e+00	**8.088e+02**	2.127e+02	+
cf_{19}	9.193e+02	6.276e+00	**9.172e+02**	3.634e+01	+	**9.398e+02**	5.601e+00	1.000e+03	6.091e+00	-	9.156e+02	8.459e-01	**8.610e+02**	1.703e+02	+
cf_{20}	9.192e+02	6.324e+00	**8.786e+02**	1.520e+02	+	**9.408e+02**	5.730e+00	1.000e+03	6.548e+00	-	9.154e+02	1.449e+00	**8.687e+02**	1.274e+02	+
cf_{21}	1.011e+03	3.265e+01	**5.240e+02**	8.221e+01	+	1.028e+03	2.190e+00	**5.307e+02**	7.809e+00	+	1.005e+03	3.089e+00	**5.060e+02**	4.243e+01	+
cf_{22}	**9.445e+02**	3.246e+01	9.878e+02	1.545e+01	-	**9.253e+02**	1.030e+01	1.106e+03	1.330e+01	-	**9.193e+02**	1.592e+01	9.804e+02	1.410e+01	-
cf_{23}	1.011e+03	3.223e+01	**5.000e+02**	0.000e+00	+	1.028e+03	2.243e+00	**5.291e+02**	6.513e+00	+	1.007e+03	7.759e+00	**5.180e+02**	7.197e+01	+
cf_{24}	1.039e+03	1.774e+01	**2.000e+02**	0.000e+00	+	1.043e+03	7.243e+00	**3.313e+02**	3.470e+01	+	1.041e+03	4.154e+00	**2.000e+02**	0.000e+00	+
cf_{25}	**1.685e+03**	8.853e+00	1.722e+03	6.387e+00	-	**1.697e+03**	2.288e+00	1.798e+03	5.523e+00	-	**1.692e+03**	3.666e+00	1.732e+03	3.349e+00	-
Total number of (+/=/-):				16/6/3					17/6/2					13/6/6	

Table I reports the results on the 30–dimensional version of the CEC 2005 benchmark set. We observe that for the explorative mutation strategies, DE/rand/1 and DE/rand/2, the incorporation of the proximity-based framework yields significant performance, with the best results obtained for DE/rand/1. For DE/rand/2, it exhibits substantial performance improvement in most of the unimodal functions ($cf_1 - cf_6$) with the exception of cf_5. Furthermore, there are 5 hybrid composition multimodal functions in which the proposed framework deteriorates performance slightly ($cf_{18} - cf_{20}, cf_{22}$ and cf_{25}). The framework, however, yields a significant improvement in the other 5 hybrid functions ($cf_{16}, cf_{17}, cf_{21}, cf_{23}$ and cf_{24}). For DE/current-to-best/2 although the mean error is smaller in most cases the improvement is significant in 7 cases. In this strategy, the proposed framework does not hinder the algorithm's performance on the hybrid multimodal functions.

For the two exploitative strategies, DE/best/1, DE/current-to-best/1, the proximity-based framework does not yield similar performance improvement. DE/best/1 in most of the unimodal and multimodal functions exhibits either marginal improvement ($cf_3, cf_7, cf_9, cf_{12}, cf_{15}$ and cf_{24}) or an equal performance, while in five hybrid functions the proposed framework deteriorates performance slightly ($cf_{18} - cf_{20}, cf_{22}$ and cf_{25}). DE/current-to-best/1 is not improved by the proximity-based framework. In general, this strategy produces the largest errors, which indicate its inability to locate global minimizers. This is more evident for the unimodal functions cf_1–cf_5. This behavior also explains the inability of the proposed approach to improve it. DE/current-to-best/1 is so exploitative that it has difficulty in locating the minimizers. This implies that it

is highly unlikely for this strategy to produce a local structure that could be exploited from the proximity framework. Note also that this strategy utilizes only two random individuals to generate an offspring, whereas the similar and also exploitative DE/current-to-best/2 strategy uses four. Finally, despite the exploitative character of DE/best/2 the proximity framework enhances its performance in most multimodal and hybrid functions. The original DE/best/2 exhibits superior performance in five cases only ($cf_5, cf_7, cf_{14}, cf_{22}$ and cf_{25}). It must be noted that qualitatively similar results were also obtained for the YAO benchmark function set [75], but due to space limitations, we do not present them here.

We further evaluate the proposed framework on the 50–dimensional version of the CEC 2005 set of benchmark functions. Higher dimensional problems are typically harder to solve and a common practice is to employ a larger population size. At present we increased the population size to 200, but we did not attempt to fine tune this parameter to obtain optimal performance. In this set of experiments algorithms terminated after performing $500,000$ function evaluations [78]. The results summarized in Table II indicate that the behavior on the 50–dimensional benchmark function set is very similar to that on the 30–dimensional benchmark. The main difference is that the improvement of using the proximity-based approach is now statistically significant in the majority of the test functions. Despite the exploitative character of the DE/current-to-best/2 strategy its proximity-based modification is superior in most of the unimodal, multimodal and hybrid composition functions in this benchmark function set. On the other hand, the proximity framework does not improve the exploitative operator DE/current-to-best/1 strategy, while there is a marginal improvement for DE/best/1 in two unimodal and six multimodal functions.

Overall the comparison of each of the original DE mutation strategies with its proximity-based variant indicates that the proposed framework significantly improves the explorative strategies. Exploitative strategies are not improved when the original strategy is already too greedy and on some hard highly multimodal functions. Note however that in relatively few of the latter cases the proximity-based framework deteriorates performance significantly.

B. Comparison Against Other DE Variants

In this subsection we apply the proximity-based framework on eight well known and widely used DE variants. Specifically, we implement the proximity framework on: i) the Trigonometric Differential Evolution (TDE) [39], ii) the Opposition based Differential Evolution (ODE) [40], [41], iii) the Differential Evolution with Global and Local Neighborhoods (DEGL) [25], [42], iv) the Balanced Differential Evolution (BDE) [22], v) the Self-Adaptive Control Parameters in DE algorithm (jDE) [31], vi) the Adaptive Differential Evolution with optional external archive algorithm (JADE) [18], [26], vii) the Differential Evolution algorithm with Strategy Adaptation (SaDE) [27], [43], and viii) the Differential Evolution Algorithm with Random Localization (DERL) [44].

We evaluate the performance of the eight DE variants and their corresponding proximity-based modifications over the 30–dimensional version of the CEC 2005 function set. Table III reports the experimental results for the first six DE variants, TDE, ODE, BDE, jDE JADE and SaDE. The results show that the proximity framework influences substantially the performance of TDE, jDE and ODE. Specifically, in nine functions the performance of TDE is not significantly different from that of Pro TDE. In 11 of the 25 functions Pro TDE achieves a significantly better performance. The benefit from the proximity framework is evident in the unimodal function cf_5, most of the basic multimodal functions, the two expanded functions (cf_{13} and cf_{14}), and in most of the hybrid composition functions ($cf_{16} - cf_{20}$ and cf_{24}). TDE is significantly better than Pro TDE in only four functions ($cf_4, cf_{15}, cf_{22},$ and cf_{25}). Overall therefore, TDE is substantially enhanced through the proximity framework. Note that TDE is based on DE/rand/1 and the proximity-based framework has been shown to substantially improve this strategy.

For the Opposition-based DE, we observe that the proximity framework efficiently exploits the population structure and guides the evolution towards more promising solutions. As Table III indicates, Pro ODE outperforms ODE in fourteen cases and exhibits similar performance in seven functions. Particularly, in four out of five unimodal functions Pro ODE produces lower mean error values. The performance difference is statistically significant in cf_1 and cf_5, while in cf_2 and cf_4 it is not. Moreover, in basic multimodal and expanded functions Pro ODE performs either as well as ODE (cf_7, cf_8, cf_{10} and cf_{13}) or significantly better ($cf_6, cf_9, cf_{11}, cf_{12}$ and cf_{14}). On the other hand, ODE is significantly superior only in four test functions ($cf_3, cf_{15}, cf_{22},$ and cf_{25}). Furthermore, the proximity framework produces substantial improvement in the optimization of hybrid composition functions which are characterized by a huge number of local minima. Pro ODE significantly outperforms ODE in seven out of eleven hybrid composition functions, ($cf_{16}, cf_{17}, cf_{19} - cf_{21}, cf_{23},$ and cf_{24}). Note that although the population in ODE changes rapidly, due to the opposition mechanism, the proximity approach efficiently exploits the population structure and guides the evolution process successfully towards more promising solutions.

Pro BDE either enhances BDE or performs equally well. In more detail, BDE is enhanced by the proximity framework in nine functions (three unimodal and six multimodal), while the performance of the two is not statistically different in the majority of functions (thirteen of the twenty five functions). The impact of the proximity framework is evident in the expanded and hybrid composition functions ($cf_{13}, cf_{15}, cf_{18} - cf_{20}$ and cf_{22}). Moreover, BDE significantly outperforms Pro BDE only in three functions (cf_3, cf_{14} and cf_{23}).

jDE is substantially enhanced by the proximity framework. Pro jDE exhibits either significantly better or similar performance in 23 of the 25 functions. Only in cf_3 and cf_{25} jDE significantly outperforms Pro jDE. More specifically, in the unimodal functions Pro jDE is significantly better in cf_3 and cf_4 and exhibits similar performance in cf_1 and cf_2. In the basic multimodal functions, Pro jDE generally produces smaller or equal mean error to jDE ($cf_6 - cf_{12}$ except for cf_9) and a significant enhancement in $cf_6, cf_{10},$ and cf_{11}. In

TABLE III
ERROR VALUES OF THE ORIGINAL TDE, ODE, BDE, JDE, JADE, SADE ALGORITHMS AND THEIR CORRESPONDING PROXIMITY-BASED VARIANTS OVER THE 30–DIMENSIONAL CEC 2005 BENCHMARK SET

	TDE		Pro TDE			ODE		Pro ODE			BDE		Pro BDE		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	3.970e-02	2.269e-01	**0.000e+00**	0.000e+00	+	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_2	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	4.200e-03	1.807e-02	**0.000e+00**	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_3	**5.003e+05**	2.907e+05	5.462e+05	2.061e+05	=	**3.836e+05**	1.538e+05	5.639e+05	2.855e+05	-	**4.219e+04**	3.647e+04	6.101e+04	2.946e+04	-
cf_4	**6.200e-04**	3.149e-03	3.340e-03	3.008e-03	-	2.314e-02	5.476e-02	**1.352e-02**	2.317e-02	=	1.443e+00	1.020e+01	**0.000e+00**	0.000e+00	+
cf_5	1.159e+03	5.486e+02	**8.294e+02**	2.541e+02	+	3.889e+02	3.428e+02	**1.663e+02**	1.741e+02	+	3.288e+02	3.649e+02	**5.469e+00**	1.676e+01	+
cf_6	3.449e+02	2.024e+03	**2.942e+01**	1.750e+01	+	1.359e+06	6.516e+06	**5.773e+01**	4.148e+01	+	1.834e+00	2.007e+00	**1.276e+00**	1.879e+00	=
cf_7	4.696e+03	1.837e-12	4.696e+03	1.837e-12	=	4.696e+03	1.837e-12	4.696e+03	1.837e-12	=	4.711e+03	9.901e+01	**4.623e+03**	1.469e+02	+
cf_8	2.095e+01	4.631e-02	**2.094e+01**	5.047e-02	=	2.095e+01	5.900e-02	**2.095e+01**	5.184e-02	=	**2.093e+01**	5.278e-02	2.095e+01	4.741e-02	=
cf_9	1.524e+01	1.117e+01	**1.386e+01**	3.761e+00	=	1.933e+01	7.090e+00	**1.605e+01**	3.897e+00	+	5.415e+01	3.564e+01	**5.095e+01**	1.720e+01	=
cf_{10}	1.657e+02	9.457e+00	**1.642e+02**	9.791e+00	=	**3.737e+01**	1.559e+01	3.763e+01	1.277e+01	=	8.273e+01	6.000e+01	**5.573e+01**	2.685e+01	=
cf_{11}	3.938e+01	1.149e+00	**3.844e+01**	2.311e+00	+	1.739e+01	7.264e+00	**7.848e+00**	3.340e+00	+	2.839e+01	1.077e+01	**2.610e+01**	1.158e+01	=
cf_{12}	2.819e+04	3.015e+04	**4.222e+03**	4.758e+03	+	2.258e+04	2.525e+04	**3.071e+03**	2.391e+03	+	**4.237e+04**	9.659e+04	4.344e+04	6.796e+04	=
cf_{13}	1.278e+01	1.747e+00	**3.730e+00**	2.095e+00	+	2.953e+00	5.820e-01	**2.903e+00**	6.246e-01	=	6.280e+00	4.015e+00	**4.523e+00**	3.235e+00	+
cf_{14}	1.333e+01	2.021e-01	**1.321e+01**	1.943e-01	+	1.326e+01	2.522e-01	**1.283e+01**	3.995e-01	+	**1.274e+01**	5.181e-01	1.297e+01	4.022e-01	-
cf_{15}	**3.087e+02**	1.012e+02	3.860e+02	5.718e+01	-	**3.353e+02**	1.058e+02	4.201e+02	5.727e+01	-	4.086e+02	7.225e+01	**3.738e+02**	9.279e+01	+
cf_{16}	2.261e+02	6.978e+01	**1.744e+02**	8.096e+01	+	9.672e+01	7.248e+01	**5.408e+01**	1.965e+01	+	**2.383e+02**	1.574e+02	2.629e+02	1.751e+02	=
cf_{17}	2.609e+02	8.430e+01	**1.948e+02**	1.212e+01	+	9.586e+01	7.529e+01	**7.417e+01**	4.456e+01	+	**2.323e+02**	1.583e+02	2.436e+02	1.653e+02	=
cf_{18}	9.052e+02	1.992e+00	**8.897e+02**	3.960e+01	+	9.044e+02	9.994e-01	**8.762e+02**	4.800e+01	=	9.128e+02	1.057e+01	**9.085e+02**	4.961e+00	+
cf_{19}	9.050e+02	1.434e+00	**8.916e+02**	3.740e+01	+	9.045e+02	8.907e-01	**8.872e+02**	4.131e+01	+	9.167e+02	1.841e+01	**9.080e+02**	2.881e+00	+
cf_{20}	9.054e+02	1.782e+00	**8.920e+02**	3.755e+01	+	9.045e+02	1.137e+00	**8.849e+02**	4.292e+01	+	9.128e+02	7.249e+00	**9.085e+02**	3.227e+00	+
cf_{21}	5.000e+02	2.622e-01	**5.000e+02**	0.000e+00	=	5.659e+02	1.822e+02	**5.060e+02**	4.243e+01	+	**6.761e+02**	2.593e+02	7.878e+02	2.966e+02	=
cf_{22}	**8.667e+02**	1.598e+01	9.055e+02	8.301e+00	-	**8.703e+02**	2.011e+01	9.031e+02	1.054e+01	-	8.988e+02	2.739e+01	**8.854e+02**	2.141e+01	+
cf_{23}	5.000e+02	9.496e-02	5.000e+02	0.000e+00	=	5.825e+02	2.059e+02	**5.120e+02**	5.938e+01	+	**6.359e+02**	2.517e+02	7.579e+02	2.874e+02	-
cf_{24}	4.688e+02	3.784e+02	**2.000e+02**	0.000e+00	+	6.252e+02	3.965e+02	**2.000e+02**	0.000e+00	+	**7.420e+02**	3.589e+02	8.825e+02	2.561e+02	=
cf_{25}	**1.620e+03**	7.223e+00	1.637e+03	8.744e+00	-	**1.631e+03**	1.115e+01	1.650e+03	7.670e+00	-	**1.636e+03**	1.029e+01	1.639e+03	1.025e+01	=
Total number of (+/=/-):					12/9/4					14/7/4					9/13/3

	jDE		Pro jDE			JADE		Pro JADE			SaDE		Pro SaDE		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_2	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_3	**2.026e+05**	1.062e+05	3.981e+05	2.102e+05	-	**9.209e+03**	6.153e+03	1.849e+04	1.437e+04	-	**2.129e+06**	9.490e+05	2.280e+06	9.066e+05	=
cf_4	3.440e-03	2.304e-02	**1.980e-03**	4.048e-03	+	3.805e+00	1.914e+01	**0.000e+00**	0.000e+00	+	2.000e-05	1.414e-04	2.000e-05	1.414e-04	=
cf_5	6.614e+02	3.056e+02	**1.230e+02**	1.151e+02	+	1.963e+02	5.146e+02	**5.896e+01**	1.073e+02	+	3.935e+02	2.847e+02	**5.505e+01**	1.186e+02	+
cf_6	3.196e+01	2.660e+01	**3.352e+00**	2.626e+00	+	2.669e+01	6.501e+01	**1.886e+01**	3.138e+01	=	1.754e+00	1.999e+00	**1.356e+00**	1.908e+00	=
cf_7	4.696e+03	1.837e-12	4.696e+03	1.837e-12	=	**4.648e+03**	3.002e+01	4.696e+03	1.837e-12	-	4.696e+03	1.837e-12	4.696e+03	1.837e-12	=
cf_8	2.095e+01	4.434e-02	2.095e+01	4.420e-02	=	2.095e+01	5.331e-02	**2.086e+01**	2.927e-01	=	**2.094e+01**	6.041e-02	2.095e+01	5.220e-02	=
cf_9	**1.540e+01**	3.587e+00	1.707e+01	4.947e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	4.179e-01	1.156e+00	**0.000e+00**	0.000e+00	+
cf_{10}	1.075e+02	6.888e+01	**3.578e+01**	1.258e+01	+	**6.315e+01**	1.052e+01	8.180e+01	1.261e+01	-	**9.844e+01**	9.514e+00	1.005e+02	3.254e+01	-
cf_{11}	3.931e+01	1.269e+00	**1.263e+01**	5.744e+00	+	**2.974e+01**	1.727e+00	3.009e+01	1.567e+00	=	**3.204e+01**	3.037e+00	3.374e+01	1.434e+00	-
cf_{12}	1.947e+03	1.930e+03	**1.849e+03**	1.926e+03	=	2.896e+04	1.125e+04	**2.634e+04**	1.160e+04	=	2.322e+03	5.755e+03	**1.478e+03**	1.801e+03	=
cf_{13}	4.404e+00	3.329e+00	**2.750e+00**	6.322e-01	=	**2.481e+00**	2.885e-01	3.324e+00	2.646e-01	-	3.705e+00	2.470e+00	**2.949e+00**	2.189e+00	=
cf_{14}	1.329e+01	1.709e-01	**1.313e+01**	2.069e-01	+	1.296e+01	2.398e-01	**1.291e+01**	2.298e-01	=	**1.295e+01**	2.393e-01	1.306e+01	1.949e-01	-
cf_{15}	3.982e+02	7.427e+01	**3.960e+02**	2.828e+01	=	**3.042e+02**	1.431e+02	3.707e+02	1.027e+02	-	**3.698e+02**	6.718e+01	3.864e+02	6.377e+01	=
cf_{16}	1.204e+02	8.210e+01	**6.048e+01**	5.013e+01	+	1.509e+02	1.264e+02	**1.135e+02**	5.004e+01	=	1.248e+02	8.672e+01	**6.973e+01**	3.617e+01	+
cf_{17}	2.336e+02	6.407e+01	**8.945e+01**	5.773e+01	+	1.872e+02	1.166e+02	**1.454e+02**	5.668e+01	=	1.340e+02	4.715e+01	**7.203e+01**	4.306e+01	+
cf_{18}	8.955e+02	3.579e+01	**8.870e+02**	4.120e+01	+	9.058e+02	1.692e+00	**8.600e+02**	5.592e+01	=	**8.481e+02**	5.721e+01	8.555e+02	5.610e+01	=
cf_{19}	8.968e+02	3.265e+01	**8.825e+02**	4.428e+01	+	9.053e+02	1.377e+00	**8.896e+02**	4.534e+01	+	8.787e+02	5.459e+01	**8.668e+02**	5.513e+01	+
cf_{20}	8.906e+02	4.000e+01	**8.824e+02**	4.424e+01	+	9.056e+02	1.503e+00	**8.959e+02**	3.915e+01	+	8.694e+02	5.746e+01	**8.515e+02**	5.638e+01	+
cf_{21}	5.240e+02	8.221e+01	**5.000e+02**	0.000e+00	+	5.250e+02	1.080e+02	**5.060e+02**	4.243e+01	=	5.317e+02	1.330e+02	**5.000e+02**	0.000e+00	=
cf_{22}	9.060e+02	9.828e+00	**9.008e+02**	9.866e+00	+	**8.723e+02**	2.491e+01	8.952e+02	2.224e+01	-	9.138e+02	1.285e+01	**9.091e+02**	8.996e+00	+
cf_{23}	5.180e+02	7.197e+01	**5.060e+02**	4.243e+01	=	5.359e+02	1.153e+02	**5.000e+02**	0.000e+00	+	5.000e+02	0.000e+00	5.000e+02	0.000e+00	=
cf_{24}	2.000e+02	0.000e+00	2.000e+02	0.000e+00	=	2.624e+02	2.137e+02	**2.000e+02**	0.000e+00	+	2.000e+02	0.000e+00	2.000e+02	0.000e+00	=
cf_{25}	**1.634e+03**	1.126e+01	1.642e+03	7.715e+00	-	**1.642e+03**	2.921e+00	1.667e+03	2.929e+00	-	1.633e+03	6.407e+00	**1.632e+03**	6.282e+00	=
Total number of (+/=/-):					13/10/2					6/12/7					7/15/3

cf_{12} and $cf_7 - cf_9$ the performance of the two algorithms is not statistically different. In the next two functions (cf_{13} and cf_{14}), Pro jDE exhibits lower mean error and in cf_{14} the difference is statistically significant. Finally, in most of the hybrid composition functions ($cf_{15} - cf_{25}$) Pro jDE clearly outperforms jDE. The only case where jDE appears superior is cf_{25}. Recall that jDE utilizes the DE/rand/1 mutation strategy, which is greatly improved by the proximity framework.

Pro JADE exhibits either similar or better performance in 18 out of the 25 functions. Specifically, Pro JADE achieves significantly better performance on two unimodal functions (cf_4 and cf_5) and four of the hybrid composition functions ($cf_{19}, cf_{20}, cf_{23},$ and cf_{24}). JADE outperforms Pro JADE in seven functions, $cf_3, cf_7, cf_{10}, cf_{13}, cf_{15}, cf_{22},$ and cf_{25}, most of which are multimodal.

Pro SaDE demonstrates either similar or significantly better performance in 22 functions ($cf_5, cf_9, cf_{16}, cf_{17}, cf_{19}, cf_{20},$ and cf_{22}). As for the previous algorithms, the impact of the proximity framework is evident mostly in hybrid composition functions. In five of these functions, Pro SaDE attains a statistically significant performance improvement. SaDE significantly outperforms its proximity variant only in three functions ($cf_{10}, cf_{11},$ and cf_{14}). The obtained results show that the proximity framework rarely hinders the performance of the efficient self adaptive algorithms, such as JADE and SaDE. Incorporating the proposed framework typically yields algorithms with similar or better performance, especially in functions with a multitude of local minima, like the hybrid composition functions.

DEGL is inspired from PSO and incorporates the concept

TABLE IV
ERROR VALUES OF THE ORIGINAL DEGL, DERL ALGORITHMS AND THEIR CORRESPONDING PROXIMITY-BASED VARIANTS OVER THE 30–DIMENSIONAL CEC 2005 BENCHMARK SET

	DEGL		Pro DEGL1			Pro DEGL2			DERL		Pro DERL		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	=	**0.000e+00**	0.000e+00	4.700e-03	2.689e-02	=
cf_2	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_3	4.230e+04	3.707e+04	4.203e+04	2.606e+04	=	**3.856e+04**	2.575e+04	=	**6.926e+04**	4.492e+04	8.777e+04	5.098e+04	-
cf_4	1.361e+01	4.118e+01	**0.000e+00**	0.000e+00	+	**0.000e+00**	0.000e+00	+	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_5	5.069e+02	5.803e+02	**1.907e+01**	6.182e+01	+	**5.904e+01**	3.957e+02	+	**1.731e+02**	2.921e+02	1.579e+03	4.573e+02	-
cf_6	**1.196e+00**	1.846e+00	**1.196e+00**	1.846e+00	=	1.515e+00	1.955e+00	=	**5.928e+02**	1.867e+03	2.568e+06	1.081e+07	-
cf_7	4.696e+03	1.177e+00	**4.689e+03**	8.806e+01	+	**4.692e+03**	4.524e+01	+	**4.696e+03**	3.136e-03	4.696e+03	1.837e-12	=
cf_8	2.095e+01	4.541e-02	**2.094e+01**	6.158e-02	=	2.095e+01	3.346e-02	=	**2.095e+01**	4.599e-02	2.102e+01	4.715e-02	-
cf_9	6.477e+01	1.597e+01	**3.576e+01**	9.749e+00	+	**3.801e+01**	1.516e+01	+	**2.662e+01**	8.886e+00	4.498e+01	1.442e+01	-
cf_{10}	7.791e+01	2.079e+01	**5.182e+01**	1.573e+01	+	**5.473e+01**	4.157e+01	+	6.202e+01	5.406e+01	**5.840e+01**	2.172e+01	+
cf_{11}	**1.785e+01**	3.463e+00	2.008e+01	7.597e+00	=	3.085e+01	1.128e+01	-	3.819e+01	5.133e+00	**2.260e+01**	5.892e+00	+
cf_{12}	2.529e+04	3.543e+04	**2.351e+04**	2.413e+04	=	**2.233e+04**	4.256e+04	=	3.298e+04	3.605e+04	**2.360e+03**	2.489e+03	+
cf_{13}	6.179e+00	2.930e+00	**3.404e+00**	2.100e+00	+	6.743e+00	4.258e+00	=	**2.931e+00**	1.207e+00	4.541e+00	1.509e+00	-
cf_{14}	**1.197e+01**	4.399e-01	1.244e+01	3.270e-01	-	1.280e+01	4.564e-01	-	1.313e+01	2.643e-01	**1.289e+01**	4.687e-01	+
cf_{15}	4.310e+02	8.986e+01	**3.527e+02**	9.868e+01	+	**3.619e+02**	8.926e+01	+	**3.063e+02**	9.588e+01	3.842e+02	6.574e+01	-
cf_{16}	2.703e+02	1.760e+02	**1.761e+02**	1.439e+02	+	**1.490e+02**	1.425e+02	+	1.166e+02	1.065e+02	**1.095e+02**	1.070e+02	=
cf_{17}	2.000e+02	1.428e+02	**1.601e+02**	1.367e+02	+	2.275e+02	1.511e+02	=	2.169e+02	1.276e+02	**1.497e+02**	1.353e+02	+
cf_{18}	9.221e+02	1.696e+01	**9.089e+02**	4.928e+00	+	**9.083e+02**	4.570e+00	+	9.064e+02	2.694e+00	**8.982e+02**	4.366e+01	+
cf_{19}	9.191e+02	1.667e+01	**9.099e+02**	6.037e+00	+	**9.089e+02**	6.152e+00	+	9.065e+02	3.917e+00	**8.885e+02**	5.047e+01	+
cf_{20}	9.197e+02	1.875e+01	**9.098e+02**	5.907e+00	+	**9.077e+02**	2.775e+00	+	9.066e+02	2.320e+00	**9.001e+02**	4.111e+01	+
cf_{21}	7.547e+02	2.958e+02	**6.794e+02**	2.554e+02	=	**6.633e+02**	2.508e+02	=	6.131e+02	2.233e+02	**5.678e+02**	1.586e+02	=
cf_{22}	9.199e+02	3.919e+01	**8.939e+02**	2.596e+01	+	**8.883e+02**	2.518e+01	+	**8.741e+02**	2.081e+01	9.204e+02	1.503e+01	-
cf_{23}	7.494e+02	2.952e+02	**6.771e+02**	2.536e+02	=	**6.685e+02**	2.589e+02	=	**5.530e+02**	1.645e+02	5.700e+02	1.871e+02	=
cf_{24}	**6.554e+02**	3.812e+02	7.766e+02	3.460e+02	=	6.728e+02	3.864e+02	=	7.284e+02	3.663e+02	**2.000e+02**	0.000e+00	+
cf_{25}	1.638e+03	1.150e+01	**1.635e+03**	1.103e+01	=	**1.632e+03**	1.120e+01	+	**1.622e+03**	5.034e+00	1.639e+03	6.131e+00	-
Total number of (+/=/-):				13/11/1			12/11/2					9/7/9	

of index neighborhoods [25], [42]. The DEGL algorithm combines a local and a global mutation model to produce the mutant individual. In the local model, which promotes exploration, a neighborhood based on indices is implemented to select individuals. In the global model, individuals from the entire population can be selected. Therefore, the proximity framework, and thus the concept of "real" neighborhoods, can be incorporated in more than one ways. We denote by Pro DEGL1 the variant of DEGL in which Pro DE is incorporated only in the global model. In this case the global model uses as parents the two individuals closer to the current one, as given by the proximity framework. A second DEGL variant (Pro DEGL2) is considered in which the proximity framework is incorporated in both the local and global models. In this variant, four individuals are selected through the proximity framework. To retain the intuition of DEGL, the two individuals closer to the current one are used in the global model, to promote exploitation, while the other two are utilized in the local model, to promote exploration.

Table IV summarizes the experimental results for DEGL and DERL on the 30–dimensional version of the CEC 2005 function set. Both Pro DEGL1 and Pro DEGL2 significantly outperform DEGL in thirteen and twelve cases, respectively. In more detail, Pro DEGL1 and Pro DEGL2 exhibit similar or significantly better performance in all unimodal functions ($cf_1 - cf_5$) and in most of the basic multimodal functions. In the expanded functions, DEGL outperforms the proximity variants in cf_{14}, while in cf_{13} Pro DEGL1 is superior and Pro DEGL2 is not statistically different. The main effect of the proximity framework is once again observed in the hybrid composition functions. Pro DEGL1 and Pro DEGL2 exhibit better performance in seven hybrid composition functions, while their performance is not statistically different from DEGL in the remaining four. DERL is significantly better than Pro DERL in the unimodal functions cf_3 and cf_5. In the multimodal functions, Pro DERL significantly outperforms DERL in nine functions, while its performance is significantly worse than that of DERL in six functions. The DERL mutation operator is based on DE/rand/1 and utilizes as base vector the best of a set of randomly selected individuals. Thus, introducing the proximity framework could yield an overly exploitative approach. However, as the experimental results show, the proximity framework does not hinder the dynamics of DERL. On the contrary, Pro DERL in most of the functions either enhances DERL by exploiting the resulting population structure (as in DE/rand/1) or exhibits similar performance. In functions where there are no optimization bounds and the global optimum is located outside the initialization range (e.g. cf_7 and cf_{25}), the local characteristics of the proximity-based framework do not appear to enhance performance.

Tables V–VI summarize the experimental results of all the DE variants and their corresponding proximity-based modifications on the 50–dimensional versions of the CEC 2005 function set. As expected, almost all variants exhibit similar behavior with the 30–dimensional versions of the function set. The proximity-based framework clearly enhances TDE, ODE, jDE and DERL in the majority of functions. As previously, Pro BDE either enhances BDE or performs equally well, while Pro SaDE attains an equal performance in most of the functions and only in three cases exhibits a statistically significant performance improvement (cf_2, cf_9 and cf_{17}). On the other hand, JADE outperforms the proximity variant in nine functions, most of which are hybrid composition functions. Pro JADE on the other hand, demonstrates superior performance in three multimodal and two hybrid composition functions ($cf_9, cf_{11}, cf_{12}, cf_{14}$ and cf_{21}, cf_{23}, respectively). Pro DEGL1 exhibits a statistically significant better performance in three cases (cf_3, cf_4 and cf_{24}) and attains similar performance in the

TABLE V
ERROR VALUES OF THE ORIGINAL TDE, ODE, BDE, JDE, JADE, SADE ALGORITHMS AND THEIR CORRESPONDING PROXIMITY-BASED VARIANTS OVER THE 50–DIMENSIONAL CEC 2005 BENCHMARK SET

	TDE		Pro TDE			ODE		Pro ODE			BDE		Pro BDE		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_2	5.940e+03	1.444e+03	**3.557e+02**	1.047e+02	+	7.460e+03	2.530e+03	**5.941e+02**	2.662e+02	+	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_3	9.855e+07	1.957e+07	**5.295e+06**	1.308e+06	+	8.271e+07	1.869e+07	**1.116e+07**	2.407e+06	+	6.150e+05	2.114e+05	**5.689e+05**	2.085e+05	=
cf_4	1.513e+04	3.408e+03	**2.280e+03**	6.947e+02	+	1.960e+04	3.996e+03	**3.989e+03**	1.512e+03	+	**1.488e+01**	3.475e+01	3.547e+01	1.015e+02	=
cf_5	**1.478e+03**	5.389e+02	2.330e+03	2.324e+02	-	**2.104e+03**	7.673e+02	2.322e+03	4.177e+02	-	**1.603e+03**	7.517e+02	1.887e+03	8.151e+02	=
cf_6	**3.549e+01**	6.305e-01	3.783e+01	1.624e+01	-	**6.609e+01**	2.913e+01	7.917e+01	3.460e+01	-	**3.987e-01**	1.208e+00	1.037e+00	1.767e+00	-
cf_7	**6.195e+03**	4.594e-12	6.196e+03	4.036e-02	-	**6.195e+03**	4.594e-12	6.213e+03	2.356e+00	-	6.195e+03	1.282e-02	**6.194e+03**	6.550e+00	=
cf_8	2.113e+01	3.215e-02	**2.113e+01**	4.420e-02	=	2.114e+01	3.628e-02	2.114e+01	3.377e-02	=	2.114e+01	3.483e-02	**2.114e+01**	3.557e-02	=
cf_9	3.348e+02	1.260e+01	**1.850e+02**	2.245e+01	+	2.231e+02	2.711e+01	**1.767e+02**	1.806e+01	+	2.455e+02	1.099e+02	**1.028e+02**	4.449e+01	+
cf_{10}	3.599e+02	1.226e+01	**3.477e+02**	1.316e+01	+	**1.331e+02**	1.109e+02	1.900e+02	1.322e+02	-	3.501e+02	1.945e+01	**1.484e+02**	1.177e+02	+
cf_{11}	7.315e+01	1.217e+00	**7.273e+01**	1.291e+00	=	4.370e+01	2.283e+01	**1.154e+01**	3.315e+00	+	7.278e+01	1.301e+00	**6.139e+01**	1.966e+01	=
cf_{12}	5.022e+05	5.360e+05	**1.428e+04**	8.611e+03	+	8.331e+05	7.064e+05	**1.126e+04**	8.078e+03	+	1.873e+05	3.140e+05	**1.306e+05**	1.905e+05	=
cf_{13}	3.178e+01	1.524e+00	**2.657e+01**	1.212e+00	+	2.348e+01	1.955e+00	**1.789e+01**	2.686e+00	+	2.820e+01	2.372e+00	**9.629e+00**	7.790e+00	+
cf_{14}	2.331e+01	1.420e-01	**2.296e+01**	1.663e-01	+	2.320e+01	1.884e-01	**2.286e+01**	2.150e-01	+	2.318e+01	2.144e-01	**2.308e+01**	2.586e-01	=
cf_{15}	**2.000e+02**	2.230e-03	3.840e+02	5.481e+01	-	**2.282e+02**	7.003e+01	3.880e+02	4.799e+01	-	**3.254e+02**	6.195e+01	3.517e+02	7.057e+01	-
cf_{16}	2.724e+02	2.770e+01	**2.447e+02**	8.253e+00	+	**1.315e+02**	8.142e+01	1.354e+02	9.116e+01	=	2.890e+02	5.498e+01	**1.659e+02**	1.079e+02	+
cf_{17}	2.887e+02	2.024e+01	**2.685e+02**	1.105e+01	+	**1.994e+02**	7.025e+01	2.609e+02	4.309e+01	-	3.127e+02	6.154e+01	**2.425e+02**	1.287e+02	+
cf_{18}	9.134e+02	1.552e+00	**8.908e+02**	9.505e+01	+	9.156e+02	5.774e-01	**8.859e+02**	5.414e+01	+	**9.215e+02**	5.931e+00	9.254e+02	8.857e+00	-
cf_{19}	9.133e+02	1.512e+00	**8.956e+02**	4.831e+01	+	9.156e+02	5.922e-01	**8.813e+02**	5.637e+01	+	9.217e+02	1.165e+01	**9.215e+02**	8.476e+00	=
cf_{20}	9.129e+02	1.429e+00	**9.030e+02**	4.199e+01	+	9.157e+02	4.992e-01	**8.907e+02**	5.148e+01	+	9.228e+02	6.646e+00	**9.223e+02**	5.959e+00	=
cf_{21}	1.002e+03	7.508e-01	**5.000e+02**	0.000e+00	+	1.005e+03	1.342e+00	**5.060e+02**	4.243e+01	+	**1.008e+03**	3.070e+01	1.009e+03	5.803e+01	-
cf_{22}	**9.024e+02**	2.782e+00	9.561e+02	1.176e+01	-	**9.078e+02**	2.700e+00	9.637e+02	1.068e+01	-	9.286e+02	2.880e+01	**9.243e+02**	2.344e+01	=
cf_{23}	1.002e+03	8.619e-01	**5.000e+02**	0.000e+00	+	1.005e+03	1.214e+00	**5.000e+02**	0.000e+00	+	1.013e+03	8.372e+00	**9.921e+02**	8.810e+01	=
cf_{24}	1.036e+03	1.393e+00	**2.000e+02**	0.000e+00	+	9.692e+02	2.292e+02	**2.000e+02**	0.000e+00	+	8.690e+02	3.379e+02	**8.193e+02**	3.572e+02	+
cf_{25}	**1.684e+03**	3.518e+00	1.702e+03	4.503e+00	-	**1.690e+03**	1.776e+00	1.714e+03	3.084e+00	-	1.676e+03	1.051e+01	**1.674e+03**	1.149e+01	=
Total number of (+/=/-):					16/3/6					14/3/8					6/15/4

	jDE		Pro jDE			JADE		Pro JADE			SaDE		Pro SaDE		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_2	5.202e+03	1.486e+03	**3.205e+02**	1.146e+02	+	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	2.280e-03	8.545e-03	**7.400e-04**	1.426e-03	+
cf_3	2.977e+07	5.744e+06	**8.036e+06**	2.123e+06	+	4.436e+04	1.485e+04	**4.297e+04**	1.864e+04	=	**7.179e+05**	1.007e+06	7.824e+05	1.043e+06	=
cf_4	1.654e+04	3.218e+03	**2.437e+03**	8.128e+02	+	**3.160e-01**	4.134e-01	3.186e-01	4.411e-01	=	9.778e+01	9.835e+01	**6.641e+01**	5.384e+01	=
cf_5	4.206e+03	5.088e+02	**2.225e+03**	3.025e+02	+	**1.055e+03**	5.485e+02	1.829e+03	4.126e+02	-	1.992e+03	4.256e+02	**1.949e+03**	5.185e+02	=
cf_6	**4.178e+01**	8.910e+00	4.230e+01	2.131e+01	-	**4.692e+00**	1.736e+01	1.039e+01	3.460e+01	=	**1.137e+01**	1.044e+01	1.148e+01	1.390e+01	=
cf_7	6.311e+03	1.596e+01	**6.199e+03**	5.581e-01	+	**6.193e+03**	1.840e+00	6.195e+03	2.840e-02	-	6.195e+03	4.594e-12	6.195e+03	4.594e-12	=
cf_8	**2.113e+01**	3.807e-02	2.114e+01	3.461e-02	=	2.114e+01	3.251e-02	**2.099e+01**	3.929e-01	=	2.113e+01	3.458e-02	2.113e+01	3.974e-02	=
cf_9	3.716e+02	1.409e+01	**1.433e+02**	1.523e+01	+	3.352e+01	2.591e+00	**2.771e+01**	2.209e+00	+	6.148e+00	1.266e+01	**6.610e-01**	3.443e+00	+
cf_{10}	3.843e+02	1.600e+01	**3.528e+02**	1.360e+01	+	**1.935e+02**	2.060e+01	1.992e+02	1.795e+01	=	6.342e+01	1.287e+01	**6.226e+01**	1.204e+01	=
cf_{11}	7.330e+01	1.008e+00	**7.245e+01**	1.500e+00	+	6.208e+01	1.777e+00	**6.029e+01**	1.733e+00	+	6.634e+01	1.485e+00	**6.613e+01**	2.047e+00	=
cf_{12}	1.473e+05	1.928e+05	**9.893e+03**	7.099e+03	+	1.768e+05	7.105e+04	**9.446e+04**	5.969e+04	+	8.781e+03	7.092e+03	**7.336e+03**	7.223e+03	=
cf_{13}	3.260e+01	1.322e+00	**2.237e+01**	2.333e+00	+	9.211e+00	4.784e-01	**9.142e+00**	5.252e-01	=	8.571e+00	4.416e+00	**6.900e+00**	3.452e+00	=
cf_{14}	2.309e+01	1.410e-01	**2.307e+01**	1.778e-01	=	2.284e+01	2.983e-01	**2.263e+01**	2.552e-01	+	2.284e+01	1.803e-01	**2.281e+01**	1.746e-01	=
cf_{15}	4.000e+02	0.000e+00	**3.960e+02**	2.828e+01	=	**2.569e+02**	8.661e+01	3.800e+02	6.061e+01	-	**3.881e+02**	4.800e+01	3.961e+02	2.830e+01	=
cf_{16}	2.716e+02	7.979e+00	**2.485e+02**	9.086e+00	+	**1.437e+02**	4.007e+01	1.437e+02	1.738e+01	-	4.912e+01	1.003e+01	**4.846e+01**	8.343e+00	=
cf_{17}	3.059e+02	1.163e+01	**2.723e+02**	9.956e+00	+	**1.896e+02**	3.737e+01	1.918e+02	3.403e+01	=	1.241e+02	6.634e+01	**9.361e+01**	5.921e+01	+
cf_{18}	9.145e+02	3.417e+01	**8.855e+02**	5.390e+01	+	**9.206e+02**	2.983e+00	9.264e+02	4.370e+01	-	**9.041e+02**	5.208e+01	9.050e+02	5.392e+01	=
cf_{19}	9.141e+02	3.402e+01	**8.904e+02**	5.133e+01	+	**9.211e+02**	5.326e+00	9.318e+02	2.823e+01	-	9.084e+02	4.736e+01	**8.973e+02**	9.973e+01	=
cf_{20}	9.167e+02	2.982e+01	**8.876e+02**	9.554e+01	+	**9.207e+02**	2.755e+00	9.325e+02	3.630e+01	-	9.152e+02	4.183e+01	**9.105e+02**	4.935e+01	=
cf_{21}	5.000e+02	0.000e+00	5.000e+02	0.000e+00	=	8.523e+02	2.330e+02	**5.000e+02**	0.000e+00	+	5.000e+02	0.000e+00	5.000e+02	0.000e+00	=
cf_{22}	9.796e+02	1.055e+01	**9.568e+02**	1.117e+01	+	**8.987e+02**	9.075e+00	9.486e+02	1.398e+01	-	9.608e+02	6.429e+00	**9.603e+02**	6.543e+00	=
cf_{23}	5.000e+02	0.000e+00	5.000e+02	0.000e+00	=	8.103e+02	2.455e+02	**5.000e+02**	0.000e+00	+	**5.000e+02**	0.000e+00	5.060e+02	4.243e+01	=
cf_{24}	2.000e+02	0.000e+00	2.000e+02	0.000e+00	=	2.000e+02	0.000e+00	2.000e+02	0.000e+00	=	2.000e+02	0.000e+00	2.000e+02	0.000e+00	=
cf_{25}	1.728e+03	2.883e+00	**1.709e+03**	2.859e+00	+	**1.684e+03**	2.182e+00	1.711e+03	4.235e+00	-	**1.687e+03**	3.898e+00	1.687e+03	4.170e+00	=
Total number of (+/=/-):					17/7/1					6/10/9					3/22/0

rest of the function set. Finally, Pro DEGL2 exhibits improved performance in four functions (the unimodal cf_4, cf_5 and the hybrid compositions cf_{19}, cf_{22}), while DEGL outperforms the second proximity-based framework in four multimodal and two hybrid functions ($cf_{10}, cf_{11}, cf_{13}, cf_{14}$ and cf_{16}, cf_{17} respectively).

Finally, in Fig. 7 we present convergence graphs for six of the 50–dimensional CEC 2005 benchmark functions, namely, $cf_3, cf_4, cf_9, cf_{11}, cf_{12}$ and cf_{13}. The graphs illustrate median solution error value curves for all DE variants considered in this section obtained from 100 independent simulations. As previously mentioned the graphs indicate that in most cases the proximity-based framework either enhances the convergence of a strategy or behaves similarly to it. There are relatively few cases where the proximity-based framework significantly deteriorates performance.

C. Computational Cost of the proposed framework

Several real-world problems implement computer based simulations that demand resource-intensive evaluations of the objective function, e.g. large-scale finite element analysis (FEA), computational fluid dynamics (CFD), engineering design problems, or demanding industrial applications [81]. Such simulations can be computationally expensive requiring from minutes to hours to evaluate a candidate solution.

In the proposed framework, individuals are evolved using information contained in the Affinity Matrix. The computational complexity of the proximity framework depends on the update of this matrix. In the worst case where all individuals in the current population have been evolved, a situation that

TABLE VI
ERROR VALUES OF THE ORIGINAL DEGL, DERL ALGORITHMS AND THEIR CORRESPONDING PROXIMITY-BASED VARIANTS OVER THE 50–DIMENSIONAL CEC 2005 BENCHMARK SET

	DEGL		Pro DEGL1			Pro DEGL2			DERL		Pro DERL		
cf_i	Mean	St.D.	Mean	St.D.		Mean	St.D.		Mean	St.D.	Mean	St.D.	
cf_1	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=
cf_2	0.000e+00	0.000e+00	0.000e+00	0.000e+00	=	0.000e+00	0.000e+00	=	8.963e+01	3.880e+01	**6.240e+00**	3.064e+00	+
cf_3	2.311e+05	1.032e+05	**1.930e+05**	1.254e+05	+	2.779e+05	1.146e+05	=	9.126e+06	2.847e+06	**2.669e+06**	9.487e+05	+
cf_4	1.574e+00	9.501e+00	**1.080e-01**	2.747e-01	+	**6.760e-03**	2.000e-02	+	1.096e+03	4.822e+02	**2.769e+02**	1.767e+02	+
cf_5	2.093e+03	6.840e+02	2.231e+03	7.426e+02	=	**1.625e+03**	5.690e+02	+	**5.091e+02**	5.252e+02	1.413e+03	3.605e+02	-
cf_6	1.356e+00	1.908e+00	**8.771e-01**	1.668e+00	=	**1.116e+00**	1.808e+00	=	3.104e+01	2.123e+01	**2.337e+01**	1.838e+01	+
cf_7	6.195e+03	4.594e-12	6.195e+03	4.594e-12	=	6.195e+03	4.594e-12	=	6.195e+03	4.594e-12	6.195e+03	1.769e-02	+
cf_8	**2.113e+01**	3.917e-02	2.114e+01	2.969e-02	=	**2.113e+01**	4.002e-02	=	2.114e+01	3.786e-02	**2.113e+01**	3.547e-02	=
cf_9	**7.620e+01**	1.706e+01	7.937e+01	2.040e+01	=	1.176e+02	8.655e+01	=	3.356e+02	1.219e+01	**4.334e+01**	9.757e+00	+
cf_{10}	1.031e+02	6.612e+01	**9.239e+01**	2.767e+01	=	2.974e+02	8.587e+01	-	3.655e+02	1.137e+01	**3.174e+02**	6.714e+01	+
cf_{11}	6.290e+01	1.360e+01	**6.138e+01**	1.363e+01	=	6.994e+01	1.013e+01	-	7.270e+01	1.510e+00	**7.138e+01**	2.010e+00	+
cf_{12}	**5.781e+04**	4.566e+04	6.316e+04	6.160e+04	=	6.091e+04	8.453e+04	=	1.058e+05	1.012e+05	**6.119e+03**	6.169e+03	+
cf_{13}	6.063e+00	4.361e+00	**5.413e+00**	1.426e+00	=	2.473e+01	5.094e+00	-	3.130e+01	1.342e+00	**4.939e+00**	1.142e+00	+
cf_{14}	2.262e+01	3.180e-01	**2.255e+01**	3.143e-01	=	2.296e+01	2.719e-01	-	2.336e+01	1.936e-01	**2.304e+01**	1.517e-01	+
cf_{15}	3.443e+02	7.724e+01	3.443e+02	5.944e+01	=	**3.180e+02**	6.749e+01	=	**2.040e+02**	2.828e+01	3.800e+02	6.061e+01	-
cf_{16}	**1.264e+02**	1.061e+02	1.630e+02	1.388e+02	=	2.327e+02	7.620e+01	-	2.649e+02	1.951e+01	**2.202e+02**	4.425e+01	+
cf_{17}	**1.629e+02**	1.271e+02	1.943e+02	1.344e+02	=	3.024e+02	5.909e+01	-	2.921e+02	2.843e+01	**2.650e+02**	1.144e+01	+
cf_{18}	9.267e+02	1.172e+01	9.278e+02	7.720e+00	=	**9.238e+02**	8.560e+00	=	9.121e+02	9.224e-01	**9.027e+02**	4.188e+01	+
cf_{19}	9.282e+02	7.646e+00	**9.277e+02**	7.849e+00	=	**9.229e+02**	9.567e+00	+	9.119e+02	4.105e-01	**8.706e+02**	1.271e+02	+
cf_{20}	9.278e+02	9.662e+00	9.291e+02	8.160e+00	=	**9.233e+02**	1.250e+01	=	9.120e+02	7.366e-01	**8.954e+02**	4.819e+01	+
cf_{21}	9.791e+02	1.317e+02	**9.497e+02**	1.747e+02	=	9.909e+02	1.060e+02	=	1.002e+03	1.215e+00	**5.000e+02**	0.000e+00	+
cf_{22}	9.329e+02	2.361e+01	**9.277e+02**	2.429e+01	=	**9.216e+02**	2.083e+01	+	**9.037e+02**	3.273e+00	9.495e+02	1.364e+01	-
cf_{23}	9.869e+02	1.088e+02	**9.347e+02**	1.765e+02	=	9.951e+02	8.296e+01	=	1.002e+03	1.029e+00	**5.000e+02**	0.000e+00	+
cf_{24}	7.521e+02	4.004e+02	**6.808e+02**	4.118e+02	+	8.364e+02	3.613e+02	=	1.036e+03	1.759e+00	**2.000e+02**	0.000e+00	+
cf_{25}	1.671e+03	8.121e+00	**1.669e+03**	6.051e+00	=	**1.669e+03**	8.034e+00	=	**1.682e+03**	6.184e+00	1.693e+03	4.987e+00	-
Total number of (+/=/-):				3/22/0				4/15/6					19/2/4

rarely occurs, the computational cost amounts to computing $\left(\frac{NP^2-2\cdot NP}{2}\right)$ distances between individuals. This is due to the symmetric property of the distance measure.

Strictly speaking, in a pre-specified D–dimensional problem f, let the computational cost of a function evaluation be equal to c units of real computation time, while the cost of computing a distance between two individuals be $d = \kappa \cdot c$ units of real computational time. Thus, the computational cost per generation of an original DE strategy is: $\mathrm{Cost}_{\mathrm{DE}} = NP \cdot c$, while the worst case scenario for the computational cost of the corresponding proximity variant yields: $\mathrm{Cost}_{\mathrm{ProDE}} = NP \cdot c + \frac{NP^2-2\cdot NP}{2} \cdot \kappa \cdot c$. In a real case, the number of distances that have to be computed depends on the successful mutations of the algorithm (selection rate), which in turn depends on the phase of the evolution process and on the problem at hand. One can estimate the ratio $\mathrm{Cost}_{\mathrm{ProDE}}/\mathrm{Cost}_{\mathrm{DE}}$ to obtain an estimate of the computational overhead of the proximity framework.

In this study, we employed the CEC 2005 benchmark function set. To quantify the overhead of the proximity framework on these functions we compute $\mathrm{Cost}_{\mathrm{DE}}$ and $\mathrm{Cost}_{\mathrm{ProDE}}$ using the worst case scenario, in which each update of the affinity matrix involves the computation of all of its elements. The computed median value of the ratio for the CEC 2005 benchmarks is approximately 1.0834. The nature of the functions in the CEC 2005 benchmark set is such that the computational cost of DE algorithms is mostly determined by function evaluations. In such cases the implementation of the proximity framework is highly recommended, because it can yield significant improvements in the quality of the solutions, with a relatively small computational overhead. The overhead is reversed when the cost of a function evaluation is small relative to the cost of computing the affinity matrix. To demonstrate this behavior, we have computed the ratio $\mathrm{Cost}_{\mathrm{ProDE}}/\mathrm{Cost}_{\mathrm{DE}}$ for the functions in the YAO benchmark set [75]. The ratio is very high, with the median value approximately equal to 9.5351. In such cases, the proximity framework can only be justified if the improvement in the quality of the solutions is highly valued by the user.

D. Overall Performance

We conclude the presentation of the experimental results, by providing a summarizing comparison over all the benchmark functions. To this end, we utilize the Empirical Cumulative probability Distribution Function ($ECDF$) of the Normalized Mean Error (NME).

The NME measure attempts to capture the relative performance of an algorithm against the best performing algorithm on a particular function. Specifically, for an algorithm A on a function f is computed as the ratio of the Mean Error (ME) achieved by A on function f, over the lowest ME on f achieved by any of the considered algorithms (denoted as ME_{best}):

$$NME_{A,f} = \frac{ME}{ME_{\mathrm{best}} + \epsilon},$$

where $\epsilon = 1$ is a small real constant number used to avoid zero values in the denominator. Therefore, smaller values of NME correspond to better performance.

The $ECDF$ of $NMEs$ for a number of algorithms n_A and a number of functions n_f is a cumulative probability distribution function defined as:

$$ECDF(x) = \frac{1}{n_A \times n_f} \sum_{i=1}^{n_A} \sum_{j=1}^{n_f} I(NME_{i,j} \leqslant x),$$

where $I(\cdot)$ is the indicator function. In other words, the $ECDF$ measure captures the empirical probability of observing an NME value smaller or equal to x.

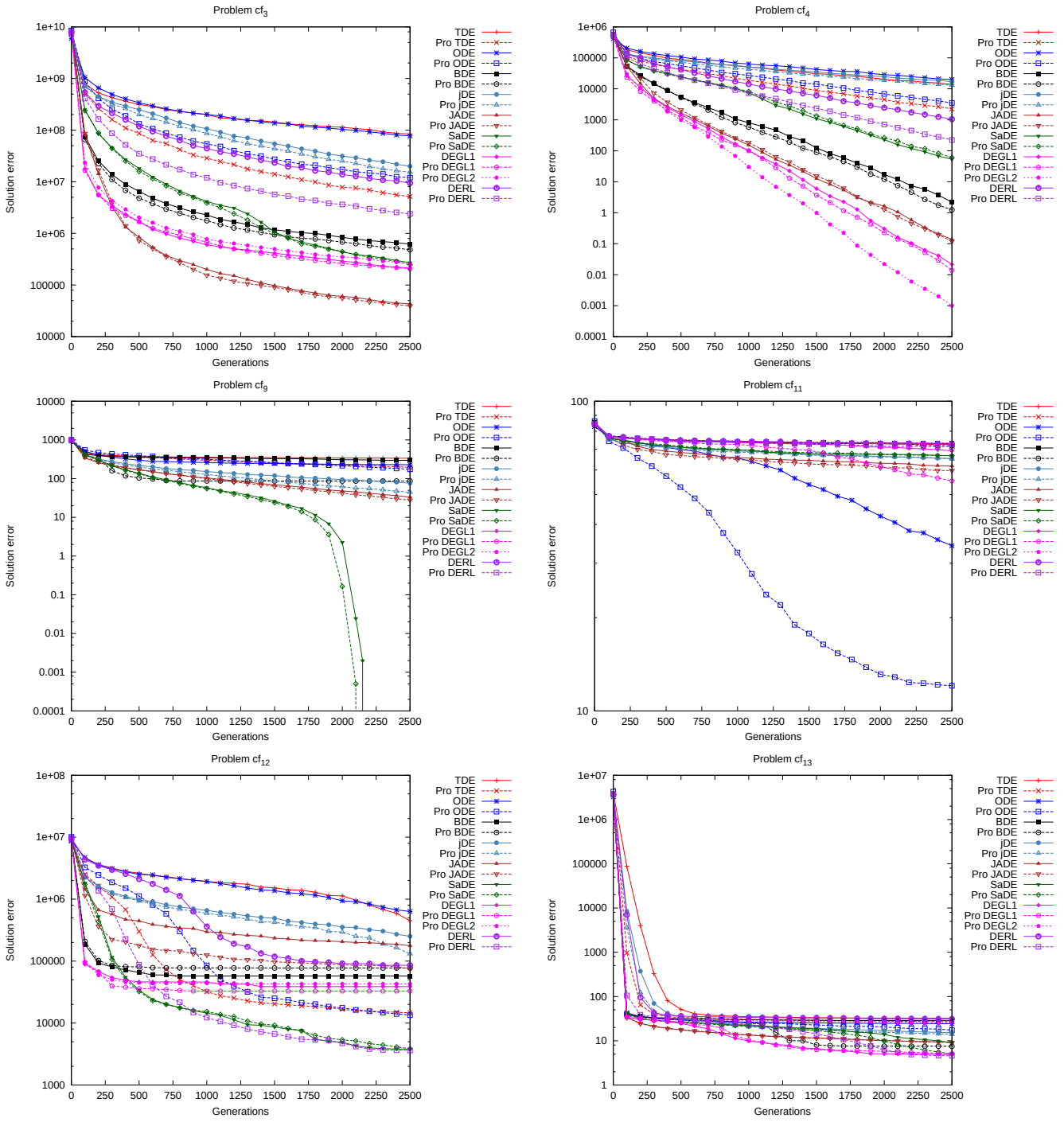

Fig. 7. Convergence graph (median curves) for the state-of-the-art DE variants over the 50–dimensional $cf_3, cf_4, cf_9, cf_{11}, cf_{12}$ and cf_{13} CEC 2005 benchmark functions. The horizontal axis illustrates the number of generations, and the vertical axis illustrates the median of solution error values over 100 independent simulations.

First, we compute the *NME* for all considered algorithms over all the functions. We then separate the algorithms into two sets, the original DE algorithms and the Pro DE variants, and compute the *ECDF* for each set. This enables a summarizing comparison of the algorithms in the two sets, as larger values of *ECDF* for the same argument correspond to better performance.

Fig. 8 illustrates the *ECDF* of *NMEs* for all the original DE mutation operators versus their proximity-based variants for the CEC 2005 function set. The proximity framework exhibits a great potential on the CEC 2005 function set. The proximity DE mutation strategies significantly outperform the corresponding original DE mutation strategies in most cases. Despite the fact that the two very exploitative strategies, DE/current-to-best/1 and DE/best/2 and their Pro DE variants, yield high mean error values, the Pro DE *ECDF* curve is

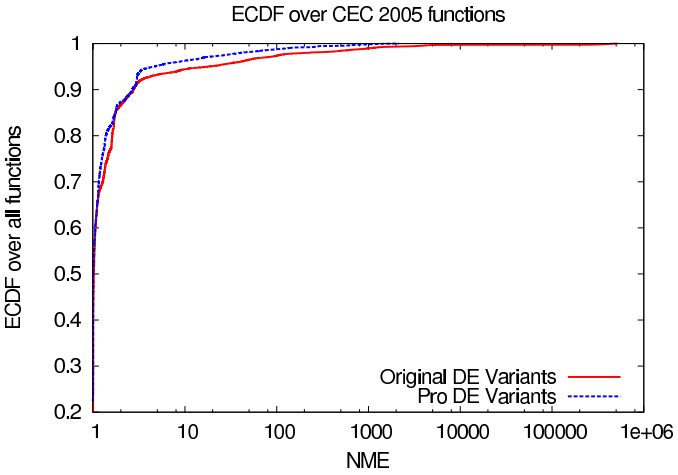

Fig. 8. Empirical cumulative probability distribution of normalized mean error of all DE algorithms against the corresponding proximity-based frameworks over the CEC 2005 benchmark functions.

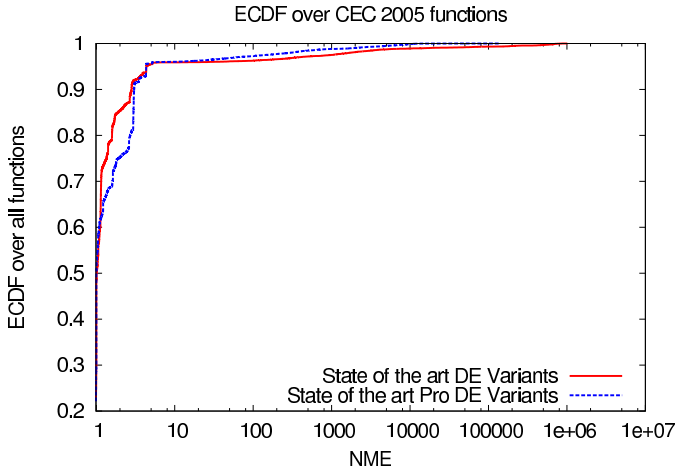

Fig. 9. Empirical cumulative probability distribution of normalized mean error of all state-of-the-art DE variants against the corresponding proximity-based frameworks over the CEC 2005 benchmark functions.

almost always above that of the original DE strategies. In general, Pro DE mutation strategies produce two orders of magnitude less NME than the original DE mutation strategies, i.e. the Pro DE curve reaches unity at approximately $NME \approx 2,000$ while the DE curve at $NME \approx 900,000$.

Fig. 9 demonstrates the $ECDF$ curves of NME for the considered state-of-the-art DE variants and their proximity-based modifications for the CEC 2005 function set. The $ECDF$ curve of the proximity-based modifications of the state-of-the-art DE variants, during the initial stages, is below that of original algorithms' $ECDF$ curve. However, notice that the proximity-based $ECDF$ curve reaches unity in two orders of magnitude less NME than the original state-of-the-art DE variants. Specifically, the proximity-based $ECDF$ curve reaches unity at approximately $NME \approx 10^4$, while the state-of-the-art DE variants curve at $NME \approx 10^6$.

VII. Concluding Remarks

It has been recognized that during the evolutionary process of the Differential Evolution (DE) algorithm a clustering structure of the population of individuals can arise. In this work, we attempt to take advantage of this characteristic behavior to improve the performance of the algorithm. To this end, we substitute the uniformly random selection of parents during mutation. We propose a probabilistic selection scheme that assigns probabilities that are inversely proportional to the distance from the mutated individual. The proposed proximity-based scheme is generic, as it is independent of the mutation strategy. In this work we have applied it to the original DE mutation strategies and a number of state-of-the-art DE variants.

The experimental results show that the proposed framework improves significantly excessively exploratory mutation strategies since it promotes the exploitation of some areas of the search space. For exploitative mutation strategies the proximity scheme does not lead to great benefits. However, even for these strategies, the proximity-based framework very rarely deteriorates their performance. The incorporation of the framework in eight state-of-the-art DE variants with different dynamics exhibited either substantial performance gains, or statistically indistinguishable behavior. Moreover, the main impact of the proposed framework was observed in high dimensional multimodal functions like the hybrid composition functions of the CEC 2005 test set. Finally, the self-adaptive parameter mechanisms of state-of-the-art DE variants are not inhibited by the incorporation of the proximity framework.

This performance improvement comes at an additional computational cost due to the computation of pairwise distances between individuals. This cost can be substantial when the cost of the function evaluation is trivial. In such cases, the utilization of the proximity framework can only be justified, if the improvement in the quality of the obtained solutions is highly valued by the user. On the contrary, when a function evaluation is computationally or otherwise costly, the computational overhead is negligible.

The effect of dimensionality and different population sizes on the performance of the proposed framework is an important aspect which we intend to study further in future work. Another interesting aspect which will be considered is the effect of proximity on structured populations.

Acknowledgment

The authors would like to thank the Guest Editors, and the three anonymous reviewers for their valuable comments and suggestions that helped to improve the content as well as the clarity of the manuscript.

References

[1] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison Wesley, 1989.
[2] T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Oxford University Press, 1997.
[3] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial intelligence through simulated evolution*. Wiley, 1966.
[4] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies a comprehensive introduction," *Natural Computing*, vol. 1, pp. 3–52, 2002.
[5] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[6] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *In Proceedings of the IEEE International Conference on Neural Networks*, vol. IV. Piscataway, NJ: IEEE Service Center, 1995, pp. 1942–1948.

[7] R. Storn and K. Price, "Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[8] ——, "Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces," Technical Report TR-95-012, ICSI, Tech. Rep., 1995. [Online]. Available: ftp://ftp.icsi.berkeley.edu/

[9] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, 1996.

[10] A. A. Salman, "Linkage crossover operator for genetic algorithms," Ph.D. dissertation, Syracuse University, Syracuse, NY, USA, 1999, adviser-Mohan, Chilukuri K.

[11] R. Storn, "System design by constraint adaptation and differential evolution," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 22–34, 1999.

[12] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[13] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, 2010, 10.1109/TEVC.2010.2059031.

[14] V. P. Plagianakos and M. N. Vrahatis, "Parallel evolutionary training algorithms for 'hardware–friendly' neural networks," *Natural Computing*, vol. 1, pp. 307–322, 2002.

[15] U. K. Chakraborty, *Advances in Differential Evolution*. Springer Publishing Company, Incorporated, 2008.

[16] V. Tirronen, F. Neri, T. Kärkkäinen, K. Majava, and T. Rossi, "An enhanced memetic differential evolution in filter design for defect detection in paper production," *Evolutionary Computation*, vol. 16, no. 4, pp. 529–555, 2008.

[17] F. Neri and V. Tirronen, "Memetic differential evolution frameworks in filter design for defect detection in paper production," in *Evolutionary Image Analysis and Signal Processing*, 2009, pp. 113–131.

[18] J. Zhang and A. C. Sanderson, *Adaptive Differential Evolution: A Robust Approach to Multimodal Problem Optimization*, ser. Evolutionary Learning and Optimization. Springer Berlin Heidelberg, 2009, vol. 1.

[19] M. G. Epitropakis, V. P. Plagianakos, and M. N. Vrahatis, "Hardware-friendly Higher-Order neural network training using distributed evolutionary algorithms," *Applied Soft Computing*, vol. 10, no. 2, pp. 398–408, 2010.

[20] W. Macready and D. Wolpert, "Bandit problems and the exploration/exploitation tradeoff," *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 1, pp. 2–22, 1998.

[21] D. K. Tasoulis, V. P. Plagianakos, and M. N. Vrahatis, "Clustering in evolutionary algorithms to efficiently compute simultaneously local and global minima," in *IEEE Congress on Evolutionary Computation, 2005. CEC 2005.*, vol. 2, 2005, pp. 1847–1854.

[22] M. G. Epitropakis, V. P. Plagianakos, and M. N. Vrahatis, "Balancing the exploration and exploitation capabilities of the differential evolution algorithm," in *IEEE Congress on Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence).*, 2008, pp. 2686–2693.

[23] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," in *Proceedings of MENDEL 2003, 9th International Mendel Conference on Soft Computing*, P. O. R. Matouek, Ed., 2003, pp. 41–46.

[24] M. Weber, F. Neri, and V. Tirronen, "Distributed differential evolution with explorative–exploitative population families," *Genetic Programming and Evolvable Machines*, vol. 10, no. 4, pp. 343–371, 2009.

[25] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, "Differential evolution using a Neighborhood-Based mutation operator," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 526–553, 2009.

[26] J. Zhang and A. Sanderson, "JADE: adaptive differential evolution with optional external archive," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 945–958, 2009.

[27] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[28] F. Neri and V. Tirronen, "Recent advances in differential evolution: a survey and experimental analysis," *Artificial Intelligence Review*, vol. 33, no. 1, pp. 61–106, 2010.

[29] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 9, no. 6, pp. 448–462, 2005.

[30] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 10, no. 8, pp. 673–686, 2006.

[31] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer, "Self-Adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.

[32] J. Brest and M. Sepesy Maučec, "Population size reduction for the differential evolution algorithm," *Applied Intelligence*, vol. 29, no. 3, pp. 228–247, 2008.

[33] J. Tvrdík, "Adaptation in differential evolution: A numerical comparison," *Applied Soft Computing*, vol. 9, no. 3, pp. 1149–1155, 2009.

[34] R. Mallipeddi, P. Suganthan, Q. Pan, and M. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," *Applied Soft Computing*, vol. In Press, pp. –, 2010.

[35] V. Feoktistov, *Differential Evolution: In Search of Solutions*, 1st ed. Springer, 2006.

[36] S. Dasgupta, S. Das, A. Biswas, and A. Abraham, "On stability and convergence of the population-dynamics in differential evolution," *AI Communications*, vol. 22, no. 1, pp. 1–20, 2009.

[37] S. Das, A. Konar, and U. K. Chakraborty, "Two improved differential evolution schemes for faster global search," in *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2005, pp. 991–998.

[38] A. Ghosh, A. Chowdhury, R. Giri, S. Das, and S. Das, "A fitness-based adaptation scheme for control parameters in differential evolution," in *GECCO '10: Proceedings of the 12th annual conference comp on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2010, pp. 2075–2076.

[39] H. Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," *Journal of Global Optimization*, vol. 27, pp. 105–129, 2003.

[40] S. Rahnamayan, H. Tizhoosh, and M. Salama, "Opposition-Based differential evolution for optimization of noisy problems," in *IEEE Congress on Evolutionary Computation, 2006. CEC 2006.*, 2006, pp. 1865–1872.

[41] ——, "Opposition-Based differential evolution algorithms," in *IEEE Congress on Evolutionary Computation, 2006. CEC 2006.*, 2006, pp. 2010–2017.

[42] U. Chakraborty, S. Das, and A. Konar, "Differential evolution with local neighborhood," in *IEEE Congress on Evolutionary Computation, 2006. CEC 2006.*, 2006, pp. 2042–2049.

[43] A. K. Qin and P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in *IEEE Congress on Evolutionary Computation, 2005. CEC 2005.*, vol. 2, 2005, pp. 1785–1791 Vol. 2.

[44] P. Kaelo and M. Ali, "A numerical study of some modified differential evolution algorithms," *European Journal of Operational Research*, vol. 169, no. 3, pp. 1176–1184, 2006.

[45] N. G. Pavlidis, D. K. Tasoulis, V. P. Plagianakos, and M. N. Vrahatis, "Human designed vs. genetically programmed differential evolution operators," in *IEEE Congress on Evolutionary Computation, 2006. CEC 2006.*, 2006, pp. 1880–1886.

[46] D. Zaharie, "Influence of crossover on the behavior of differential evolution algorithms," *Applied Soft Computing*, vol. 9, no. 3, pp. 1126–1138, 2009.

[47] B. Dorronsoro and E. Alba, *Cellular Genetic Algorithms*, ser. Operations Research/Computer Science Interfaces Series. Springer Berlin Heidelberg, 2008, vol. 42.

[48] M. Tomassini, "Parallel and distributed evolutionary algorithms: A review," in *Evolutionary Algorithms in Engineering and Computer Science*, K. Miettinen, M. Mäkelä, P. Neittaanmäki, and J. Périaux, Eds., 1999, pp. 113–133.

[49] N. Nedjah, E. Alba, and L. de Macedo Mourelle, *Parallel Evolutionary Computations (Studies in Computational Intelligence)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[50] M. Nowostawski and R. Poli, "Parallel genetic algorithm taxonomy," in *Proceedings of the Third International Conference on Knowledge-Based Intelligent Information Engineering Systems*. IEEE, 1999, pp. 88–92.

[51] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.

[52] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[53] Z. Skolicki and K. D. Jong, "The influence of migration sizes and intervals on island models," in *Proceedings of the 2005 conference on Genetic and evolutionary computation, GECCO '05*. New York, NY, USA: ACM, 2005, pp. 1295–1302.

[54] D. Zaharie and D. Petcu, "Parallel implementation of multi-population differential evolution," in *Proceedings of 2nd Workshop on Concurrent Information Processing and Computing (CIPC'03)*, D. G. et al., Ed., 2003, pp. 223–232.

[55] D. Zaharie, "A multipopulation differential evolution algorithm for multimodal optimization," in *Proc. of Mendel 2004*, P. O. R. Matousek, Ed. 10th International Conference on Soft Computing, 2004.

[56] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution," in *IEEE Congress on Evolutionary Computation, 2004. CEC 2004.*, vol. 2, 2004, pp. 2023–2029.

[57] J. Apolloni, G. Leguizamon, J. Garcia-Nieto, and E. Alba, "Island based distributed differential evolution: An experimental study on hybrid testbeds," in *Eighth International Conference on Hybrid Intelligent Systems, 2008. HIS '08*, 2008, pp. 696–701.

[58] I. De Falco, U. Scafuri, E. Tarantino, and A. Della Cioppa, "A distributed differential evolution approach for mapping in a grid environment," in *15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing, 2007. PDP '07*, 2007, pp. 442 –449.

[59] I. De Falco, A. Della Cioppa, D. Maisto, U. Scafuri, and E. Tarantino, "Satellite image registration by distributed differential evolution," in *Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 251–260.

[60] I. De Falco, D. Maisto, U. Scafuri, E. Tarantino, and A. Della Cioppa, "Distributed differential evolution for the registration of remotely sensed images," in *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP '07*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 358–362.

[61] M. Weber, V. Tirronen, and F. Neri, "Scale factor inheritance mechanism in distributed differential evolution," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 14, pp. 1187–1207, 2010.

[62] K. E. Parsopoulos and M. N. Vrahatis, "UPSO: A unified particle swarm optimization scheme," in *Lecture Series on Computer and Computational Sciences, Proceedings of the International Conference of "Computational Methods in Sciences and Engineering" (ICCMSE 2004)*, vol. 1, 2004, pp. 868–873.

[63] A. Salman, A. P. Engelbrecht, and M. G. Omran, "Empirical analysis of self-adaptive differential evolution," *European Journal of Operational Research*, vol. 183, no. 2, pp. 785–804, 2007.

[64] M. G. Omran, A. P. Engelbrecht, and A. Salman, "Using the ring neighborhood topology with self-adaptive differential evolution," in *Advances in Natural Computation*, 2006, pp. 976–979.

[65] ——, "Bare bones differential evolution," *European Journal of Operational Research*, vol. 196, no. 1, pp. 128–139, 2009.

[66] R. Thangaraj, M. Pant, and A. Abraham, "New mutation schemes for differential evolution algorithm and their application to the optimization of directional over-current relay settings," *Applied Mathematics and Computation*, vol. 216, no. 2, pp. 532–544, 2010.

[67] M. Pant, M. Ali, and V. Singh, "Parent-centric differential evolution algorithm for global optimization problems," *OPSEARCH*, vol. 46, no. 2, pp. 153–168, 2009.

[68] K. Deb, A. Anand, and D. Joshi, "A computationally efficient evolutionary algorithm for Real-Parameter optimization," *Evolutionary Computation*, vol. 10, no. 4, pp. 371–395, 2002.

[69] P. Kaelo and M. Ali, "Differential evolution algorithms using hybrid mutation," *Computational Optimization and Applications*, vol. 37, no. 2, pp. 231–246, 2007.

[70] V. Tirronen, F. Neri, T. Kärkkäinen, K. Majava, and T. Rossi, "A memetic differential evolution in filter design for defect detection in paper production," in *Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 320–329.

[71] A. Caponio, F. Neri, and V. Tirronen, "Super-fit control adaptation in memetic differential evolution frameworks," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 13, no. 8-9, pp. 811–831, 2009.

[72] N. Noman and H. Iba, "Enhancing differential evolution performance with local search for high dimensional function optimization," in *Proceedings of the 2005 conference on Genetic and evolutionary computation, GECCO '05*. New York, NY, USA: ACM, 2005, pp. 967–974.

[73] ——, "Accelerating differential evolution using an adaptive local search," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 107–125, 2008.

[74] F. Neri and V. Tirronen, "Scale factor local search in differential evolution," *Memetic Computing*, vol. 1, no. 2, pp. 153–171, 2009.

[75] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.

[76] B. Hopkins and J. G. Skellam, "A new method for determining the type of distribution of plant individuals," *Annals of Botany*, vol. 18, no. 2, pp. 213–227, 1954.

[77] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Fourth Edition*, 4th ed. Academic Press, 2008.

[78] 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 Technological University and KanGAL Report #2005005, IIT Kanpur, India., Tech. Rep., 2005.

[79] M. Steinbach, L. Ertöz, and V. Kumar, "The challenges of clustering high dimensional data," in *New Vistas in Statistical Physics – Applications in Econophysics, Bioinformatics, and Pattern Recognition*. Springer-Verlag, 2003, ch. 2.

[80] A. Hinneburg, C. C. Aggarwal, and D. A. Keim, "What is the nearest neighbor in high dimensional spaces?" in *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB '00*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 506–515.

[81] Y. Tenne and C. Goh, *Computational Intelligence in Expensive Optimization Problems*, 1st ed., ser. Evolutionary Learning and Optimization. Springer Berlin Heidelberg, 2010, vol. 2.