An Evolutionary Algorithm That Makes Decision Based on the Entire Previous Search History

Chi Kin Chow and Shiu Yin Yuen, *Member, IEEE*

***Abstract*—In this paper, we report a novel evolutionary algorithm that enhances its performance by utilizing the entire previous search history. The proposed algorithm, namely history driven evolutionary algorithm (HdEA), employs a binary space partitioning tree structure to memorize the positions and the fitness values of the evaluated solutions. Benefiting from the space partitioning scheme, a fast fitness function approximation using the archive is obtained. The approximation is used to improve the mutation strategy in HdEA. The resultant mutation operator is parameter-less, anisotropic, and adaptive. Moreover, the mutation operator naturally avoids the generation of out-of-bound solutions. The performance of HdEA is tested on 34 benchmark functions with dimensions ranging from 2 to 40. We also provide a performance comparison of HdEA with eight benchmark evolutionary algorithms, including a real coded genetic algorithm, differential evolution, two improved differential evolution, covariance matrix adaptation evolution strategy, two improved particle swarm optimization, and an estimation of distribution algorithm. Seen from the experimental results, HdEA outperforms the other algorithms for multimodal function optimization.**

***Index Terms*—Benchmarking with other evolutionary algorithms, evolutionary algorithm using search history, fitness function approximation, parameter-less anisotropic adaptive mutation.**

I. Introduction

SEARCH HISTORY, including the performed operations, the positions of the evaluated solutions, and the fitness values of the solutions, is valuable information to enhance the performance of an evolutionary algorithm (EA). Intuitively, it can be used to maintain diversity. It can also guide the search direction or suggest promising regions of interest. In addition, when the same optimum reappears in the search history (i.e., a revisit occurs), it can warn that the search may have been trapped in a local optimum. In expensive objective function optimization, one may use the history to approximate the objective function and pre-evaluate the potential optimum on this approximated function, thus saving computation cost.

Several search algorithms [1]–[5], such as particle swarm optimization (PSO), genetic algorithm (GA), differential evolution (DE), and estimation of distribution algorithm (EDA), employ search history in the form of memory to guide the search strategies. Search histories, in different EAs, are in different forms and contribute to search strategy in different manners. In PSO, each candidate solution is modeled as a particle that moves in a multidimensional search space. The corresponding search history is in terms of its best-visited position $\mathbf{l}$ and the best visited position $\mathbf{g}$ amongst the swarm. The search strategy of PSO, referred to as particle velocity, is adjusted according to the vector difference between its position $\mathbf{x}$ and $\mathbf{l}$, and the vector difference between $\mathbf{x}$ and $\mathbf{g}$.

Canonical GA and DE presume that fitter population contains more promising information about the position of global optimum. Thus, the search strategies of GA and DE refer to the offspring generation by the parent population based genetic operators, such as mutation, crossover, and selection. Comparing with PSO, the search strategies of GA and DE involve shorter search histories.

Recently, the non-revisiting genetic algorithm (NrGA) is proposed by Yuen and Chow [6], [7]. It memorizes all evaluated solutions by a binary partitioning tree (BSP) archive. The non-revisiting scheme of NrGA uses this archive to prevent solution re-evaluation. More importantly, this scheme acts as a parameter-less adaptive mutation operator. The adaptation of this scheme to other search methods is reported in [8], [9].

Owing to the fact that survived individuals have better fitness values, EDA presume that the distribution of the evaluated solutions (the survival individuals) correlates with the probability distribution of global optimum. The more survived individuals in a region, the higher probability the global optimum can be found in there. Thus, EDA memorizes the distribution of the evaluated solutions and obtains new candidate solution by sampling the distribution.

Tabu search (TS) [10] uses a short term memory, called Tabu list, to memorize the solutions that have been visited in the recent past. Different from PSO, GA, DE, and EDA of which their search histories infer promising regions for the prospective search, TS refuses to repeat the search history and prevents revisiting the solutions stored in the Tabu list.

Search history can also be used to build an empirical model that approximates the fitness function. The approximation is then used to predict promising candidate solutions at a smaller evaluation cost than the original problem. A comprehensive review of different approximation methods is provided in [11]–[14].

Besides of mechanism and motivation, the discussed EAs are different in the sense that they use different memory archive structures to memorize search history. In PSO, GA,

Manuscript received April 15, 2009; revised August 21, 2009 and November 26, 2009; accepted December 9, 2009. Date of publication September 22, 2011; date of current version December 1, 2011. This work was supported by CityU under Grant 7002304.

The authors are with the Department of Electronic Engineering, City University of Hong Kong, Kowloon Tong, Hong Kong, China (e-mail: chowchi@cityu.edu.hk; kelviny.ee@cityu.edu.hk).

Digital Object Identifier 10.1109/TEVC.2010.2040180

DE, and TS, the search histories are stored in a list structure archive. For NrGA, the evaluated solutions are memorized by a tree-structure archive which can manage spatial data well. For EDA, the distribution of the evaluated solution is commonly represented by a parametric model [15]. Recently, Liou and Chen [16] presented a multidimensional space discretization method for encoding continuous decision variables with discrete codes by discretizing the continuous domain, which is named multidimensional split-on-demand (mSod). For the EAs that predict promising candidate solutions by the approximated fitness function, search histories are in terms of different approximation models such as low-order polynomial regression, radial basis function network (RBFN), extreme learning machine (ELM) [17], and Gaussian process model (GPM) [13]. A comparison of the models can be found in [12], [18], and [19].

Apart from the employed archive structure, these EAs are different in terms of the usage of search history. The Tabu list in TS is a *negative* search guidance as the list provides the positions of which TS should *not* visit. In contrast, the search histories of GA, PSO, DE, and EDA somehow represent the positions which the EAs should go to.

More importantly, they use different portions of search history to guide the search. GA, DE and PSO only use partial search histories—that is, only part of the information gained from the search is retained and the rest is discarded. TS stores only recently visited solutions in the Tabu list and discards solutions once the list is full. EDA abstracts the search history by a predefined parametric model and proceeds to discard the raw search data. On the other hand, NrGA advocates storing all visited solutions in memory, then organize them in some ways to guide the search. At first glance, there seems a tradeoff between the portion of search history and the corresponding contribution to the search. On the one hand, a larger portion of search history provides a more detailed description of the objective function. On the other hand, it needs a larger amount of memory to record the history. In fact, the amount of memory required is reasonable and acceptable. First, consider applications which have expensive and/or time consuming fitness evaluations; the fitness evaluation cost is significantly higher than the solution generation cost. There are many such applications in engineering, artificial intelligence, and robotics. For such problems, the total number of evaluations that can be made by an EA cannot be too large; and it is culpable to throw away any information gained from fitness evaluations. Second, currently, the memory that is becoming available due to advance in computer technology is increasing drastically (Moore's law [20]). Algorithms that are previously considered memory demanding, e.g., z buffer in computer graphics, are now standard provisions. Third, even in applications whose solution generation costs are significant, it has recently been shown that using the entire search history may still give performance gains in spite of the memory overhead (see [21] for our study on a nondeterministic polynomial (NP) hard problem). Finally, EA are seldom used alone. Memetic algorithms [22], [23], the idea of hybridizing EA with another algorithm which encodes domain specific knowledge, often results in the best algorithm. In such a scenario, the total number of evaluations that the EA is subject to is limited. In conclusion, in many cases, it is preferred to use the entire previous search history to adaptively guide the search strategy.

In this paper, we report a novel evolutionary algorithm, namely History driven Evolutionary Algorithm (HdEA). It is shown that when a BSP tree archive stores the positions and the fitness values of all evaluated solutions, this archive, namely *fitness tree*, can be treated as an approximation of the fitness function. Moreover, we incorporate the *fitness tree* to govern the EA search, through which a new parameter-less and adaptive mutation scheme, namely guided anisotropic search (GAS), is obtained.

HdEA is superior to the methods of [12], [18], and [19] in the sense that the *fitness tree* uses less computation to re-approximate the fitness function. Given an evaluated solution, the *fitness tree* rapidly responds to the update of the approximated fitness function by inserting a tree node. On the other hand, the ELM, RBFN, and GPM in [12], [18], and [19] respond to the update by model re-training, which is more computational expensive. This superiority is more obvious when the dimension of the search space increases. HdEA is also superior to EDA in terms of the robustness of their memory archive structures. EDA suffers from the problem of model parameter selection, but HdEA does not as the *fitness tree* is a nonparametric approximation model.

There are several conceptual differences between TS and HdEA: 1) TS only stores recently visited solutions while HdEA stores all visited solutions; that is, TS is not non-revisiting but HdEA is (though revisit seldom occurs in continuous search space); 2) TS only passively uses the Tabu list to prevent revisits while HdEA organizes the entire previous search history and actively uses it to guide the search; 3) the BSP tree employed in HdEA is a more advanced data structure than the Tabu list; it encodes landscape information rather than local neighborhood information; and 4) HdEA does not introduce any additional control parameter whilst TS does.

Though HdEA and NrGA share the same nature in that they both memorize all evaluated solutions to perform adaptive mutation, HdEA and NrGA are quite different algorithms. As aforementioned, NrGA is applied on discrete search space where the precision of its optima depends on the axis resolution. On the other hand, HdEA deals with continuous search space to which the precision of its optima is up to the precision of the computer. Moreover, the usage of the memory archive in HdEA is different from that in NrGA. Though both use nonparametric representation, NrGA uses the archive to estimate the density of the evaluated solutions, while HdEA uses the archive to approximate the fitness landscape.

In summary, HdEA has the following remarkable features.

1) HdEA guides the search strategy using the entire search history.
2) HdEA naturally avoids generating any out-of-bound solutions and hence the extra solution-repair operator is not necessary.
3) HdEA uses a tree structure archive, which is nonparametric, to memorize the search history and hence no parametric model and its attendant model parameter selection is needed.

4) The fitness landscape re-approximation process in HdEA is achieved by a standard tree node insertion, which is viewed as an incremental learning.
5) GAS computes not only the mutation step size but also the mutation direction.

The rest of this paper is organized as follows. Section II presents the mechanism of HdEA. Section III reports the structure and the distinct features of the memory archive in HdEA. Section IV presents the details of GAS. Section V reports the experimental results and Section VI gives the conclusion.

II. History Driven Evolutionary Algorithm

History driven evolutionary algorithm is a real coded evolutionary algorithm. For a D-dimensional search space $S \subset \Re^D$, an individual $\mathbf{x}$ of HdEA is a 1 by D real valued vector, i.e., $\mathbf{x} \in \Re^D$. Evolutionary algorithm searches for the optimum by four main steps, namely population initialization, crossover, mutation, and selection. HdEA focuses on enhancing the mutation by GAS and the entire previous search history.

HdEA is considered as a generic EA module (EAM) cooperating with an adaptive mutation module (AMM). EAM consists of the standard EA operators such as population initialization, crossover operator, and selection operator; while AMM consists of the *GAS* module and the memory archive, namely *fitness tree*.

HdEA starts with initializing the population $\mathbf{P} = \{\mathbf{x_i}\}$. Afterward, the offspring population $\mathbf{N} = \{\mathbf{n_i}\}$ is generated by genetic operators such as crossover and mutation. The new population is then selected from the parent population and the offspring population. The reproduction process and the selection process are repeated until the number of iterations exceed a pre-determined value N_g. Fig. 1 shows the block diagram of HdEA. The circled numbers represent the step orders within a HdEA iteration. Algorithm A1 shows the pseudo code for HdEA. The code sections: steps 4–7, steps 9–16, and steps 19–21 in **boldface** indicate the newly added codes for HdEA, which will be explained in the following subsections.

Algorithm A1: History Driven Evolutionary Algorithm (HdEA)

Input: 1) fitness function $f(.)$, 2) search space S, 3) population size μ, 4) number of generations N_g

1. Initialize the current population $\mathbf{P} = \{\mathbf{x_i}\}_i = 1, 2, \ldots, \mu$ where $\mathbf{x_i} \in S$
2. Evaluate the population $\mathbf{P}$
3. Find the historical optimal solution $\mathbf{x_0}$

/* *Fitness Tree* Initialization */

4. **Initialize *fitness tree T* to consist of a root node only**
5. **For i := 1 to** μ
6. ***BSPTreeNodeInsert***$([x_i, f(x_i)], T)$
7. **Next *i***

/* End of *Fitness Tree* Initialization */

8. **For** $g := 2$ to N_g

/* Reproduction Strategy of HdEA */

9. **For *i* :=** 1 **to** μ
10. $\mathbf{m_i} := \boldsymbol{GAS}(\mathbf{x_i}, \boldsymbol{T})$
11. **Next *i***
12. **For *i* :=** 1 **to** μ
13. $\mathbf{a} :=$***Random***$(\{\mathbf{m_i}\})$
14. $\mathbf{b} :=$ ***Random***$(\{\mathbf{m_i}\})$ **subject to** $\boldsymbol{b \neq a}$**.**
15. $\mathbf{n_i} =$ ***Crossover***$(\mathbf{a}, \mathbf{b})$
16. **Next *i***

/* End of Reproduction Strategy of HdEA */

17. Evaluate the population $\mathbf{N} = \{\mathbf{n_i}\}_{i=1,2,\ldots,\mu}$
18. Update $\mathbf{x_0}$

/* *Fitness Tree* Update */

19. **For *i* :=** 1 **to** μ
20. ***BSPTreeNodeInsert***$([\mathbf{n_i}, f(\mathbf{n_i})], T)$
21. **Next *i***

/* End of *Fitness Tree* Update */

22. $\mathbf{P} :=$ *Selection*$(\mathbf{P} \cup \mathbf{N})$
23. **Next** g

Output: The optimal solution $\mathbf{x_o}$ found by HdEA

A. Fitness Tree Initialization and Update

In the beginning of EA evolution, the *fitness tree* is initialized to consist of the root node only (step 4 at Algorithm A1). During the evolution, once an individual is evaluated, knowledge to the fitness landscape is gained. *Fitness tree* responds to this gained knowledge by inserting the individual (its position and fitness value) to the tree. The insertion is performed after the population initialization (steps 5–7 at Algorithm A1) and the evaluations of offspring individuals (steps 19–21 at Algorithm A1).

B. Reproduction Strategy of HdEA

In HdEA, offspring pool $\mathbf{N} = \{\mathbf{n_i}\}$ is generated by two genetic operators: mutation and crossover. Given a population pool $\{\mathbf{x_i}\}$, each individual $\mathbf{x_i}$ is mutated by GAS, i.e., $\mathbf{x_i} \xrightarrow{GAS} \mathbf{m_i}$ (The details of GAS will be discussed in Section IV. At the current stage, one can regard it as a black-box mutation operator which maps a solution $\mathbf{x} \in S$ to its mutant $\mathbf{m} \in S$). Afterwards, every offspring individual $\mathbf{n_i}$ is generated by the crossover of two mutants randomly picked from the set $\{\mathbf{m_i}\}$. Steps 9–16 of Algorithm A1 list the implementation of the reproduction strategy of HdEA.

III. Fitness Tree

HdEA uses a BSP tree as an archive which stores the positions and the fitness values of the evaluated solutions $\{[\mathbf{s_i}, f(\mathbf{x_i})]\}$. It partitions the whole search space S according to the distribution of $\{\mathbf{s_i}\}$. A tree node represents a partitioned sub-region of S. Suppose a parent node has two child nodes $\mathbf{l}$ and $\mathbf{r}$. The sub-regions represented by $\mathbf{l}$ and $\mathbf{r}$ are disjoint and their union is the sub-region of the parent, (i.e., the child nodes binary partitions the parent sub-region). As the tree construction depends on the sequence of solutions found by the EA, the BSP tree is a random tree and its topology is different from trial to trial.

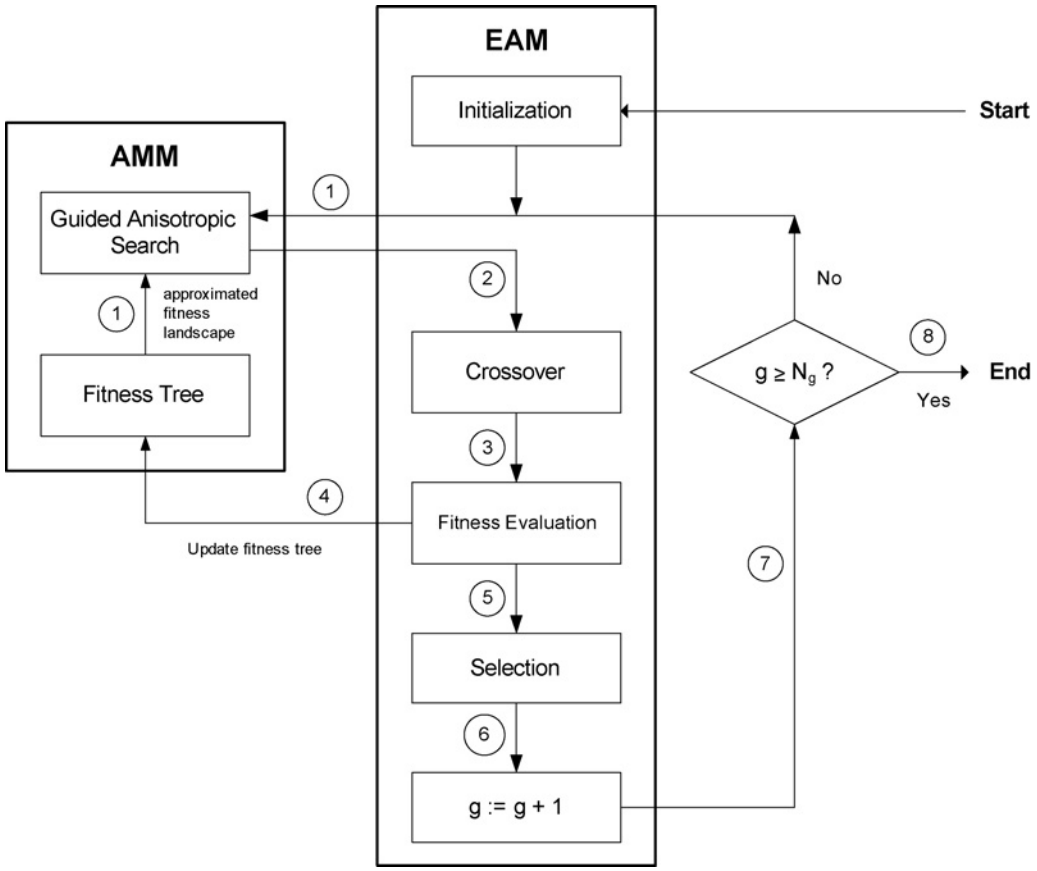


Fig. 1. Block diagram of HdEA.

The evaluated solutions in HdEA are recorded by the same procedure used in NrGA. Initially, the tree archive T in HdEA consists of only the root node. Each node of the tree records a distinct previously visited solution $\mathbf{s}$ of the EA. It also represents a sub-region X of the search space. Each sub-region is a hyper-rectangular box of the search space. The tree is organized such that the nodes binary partition the search space using Euclidean metric. Two child nodes $\mathbf{a}$ and $\mathbf{b}$ binary partition the space of its parent node into two disjoint halves A and B, $A \cap B = \phi$, by a hyper-plane at dimension j, chosen such that the difference of x and y along the dimension is the largest amongst all dimensions. In this way, each previous solution generated by the EA is recorded in a node of the tree, and the BSP tree serves as an efficient data structure to query the sub-region of $\mathbf{x}$. Algorithm A2 shows the pseudo code for BSP tree node insertion.

For more details on the tree construction as well as a working example, please refer to [7]. Note that since we are dealing with continuous search space, no node-pruning and backtracking is needed. Thus, the tree construction and management is simpler than that in NrGA [7].

Definition 1: The Sub-Region of $\mathbf{x}$

Suppose $\mathbf{x}$ is a solution in the search space S, i.e., $\mathbf{x} \in S$, and S is partitioned as the sub-region set $H = \cup_i h_i$ by the *fitness tree*, we define the sub-region $h \subseteq H$ as the "*sub-region of* $\mathbf{x}$" if $\mathbf{x} \in h$ and h is represented by a leaf node of the *fitness tree*.

Consider the situation when the fitness values of the evaluated solutions are also stored in the memory archive of HdEA; the archive can be regarded as the approximation $\tilde{f}(\mathbf{x})$ of a fitness function $f(\mathbf{x})$. Given a tree which stores the evaluated solution set $\{[\mathbf{s_i}, y_i = f(\mathbf{s_i})]\}_{i=1,2,\ldots}$ and partitions the search space as sub-region set $H = \cup_i h_i$, the fitness value of an unseen solution $\mathbf{x}$ can be approximated as y_k, i.e., $f(\mathbf{x}) \approx \tilde{f}(\mathbf{x}) = y_k$, if $\mathbf{x}$ is inside the sub-region h_k. Since the fitness of all solutions in the sub-region of a BSP tree node is approximated to the same value, i.e., $\tilde{f}(\mathbf{a}) = \tilde{f}(\mathbf{b}) = y_k$ for all $\mathbf{a}$, $\mathbf{b}$, $\mathbf{s_k} \in h_k$, $\tilde{f}(\mathbf{x})$ is a step-wise function and the landscape of $\tilde{f}(\mathbf{x})$ is of the shape of a "*terraced field*". Because the tree archive is an *approximation model* of the fitness function, we call the tree a *fitness tree*.

As the approximation error of $\tilde{f}(x)$ monotonically decreases with a growing number of evaluated solutions, and the solutions are recorded one by one; this approximation process can be viewed as a simple *incremental learning method* in the field of machine learning.

Comparing between the *fitness tree* and mSoD, they are similar in the sense that both partition the search space into hyper-rectangular and nonoverlapping sub-regions. However, the *fitness tree* binary partitions the space whilst mSoD uses different partitioning schemes for different space dimensionalities, i.e., a quad-tree partitioning for 2-D space; an oct-tree partitioning for 3-D space, and so on. For a 30-dimensional fitness function (as those used in the experiment section), mSoD uses a 2^{30}-tree to represent the space partitioning, and each node of the tree has 2^{30} child nodes. Thus, the structure of the *fitness tree* is much simpler than that of mSoD, and this simplicity in-

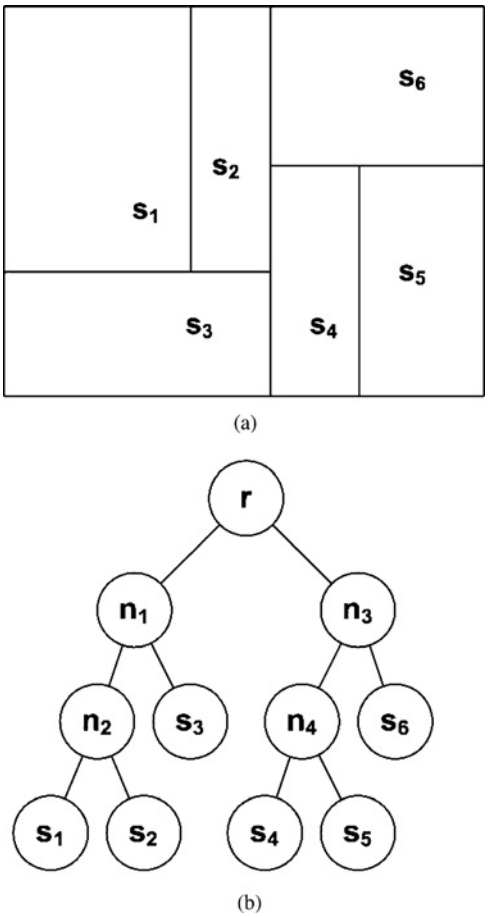


Fig. 2. Content of S: (a) S is partitioned by six evaluated solutions $\{\mathbf{s_1}, \ldots, \mathbf{s_6}\}$. (b) Topology of the *fitness tree*: the leaf nodes labeled $\mathbf{s_1}, \ldots, \mathbf{s_6}$ represent the sub-regions $\mathbf{s_1}, \ldots, \mathbf{s_6}$, respectively, and the node labeled $\mathbf{r}$ represents the root node the *fitness tree*.

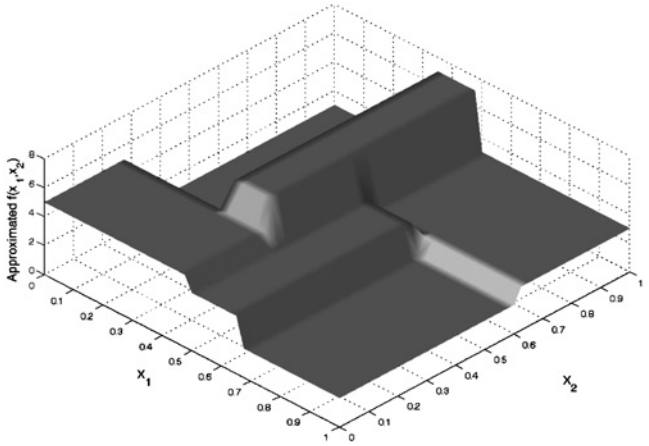


Fig. 3. Approximated fitness landscape by the *fitness tree* shown in Fig. 2(b).

creases along with the space dimensionality. HdEA constantly partitions the search space during the whole evolution. In contrast, the split rate of mSoD increases along with iteration, so the corresponding space partitioning is much more frequent in the latter part of the evolution. The partitioning method is also different. In HdEA, the partition boundary is defined by a rule (aiming at reducing the maximum size of the hyper-rectangular regions), whilst the split point in mSoD is randomly picked. The search space partitioning of mSoD is different from trial to trial even when the sequence of solution generation is kept the same.

Algorithm A2: BSP Tree Node Insertion

Input: 1) An individual $\mathbf{x}$ and it fitness value f, 2) BSP tree T

1. *Curr_node* := root node of T
2. **While** (*Curr_node* has two child nodes: **a** and **b**)
3. Comparing dimension j :=arg $\max_{k\in[1,D]} |\mathbf{a}(k) - \mathbf{b}(k)|$
4. **If** ($|\mathbf{a}(j)\text{-}\mathbf{x}(j)| \leq |\mathbf{b}(j)\text{-}\mathbf{x}(j)|$)
5. *Curr_node* := child node **a**
6. Else
7. *Curr_node* := child node **b**
8. End
9. **Loop**
10. Insert a child node to *Curr_node* that records $\mathbf{x}$ and f

Comparing between HdEA and NrGA, the memory archives of HdEA and NrGA have four differences in terms of the represented information and the tree operation.

1) The memory archive of HdEA stores the fitness values of the evaluated solutions but the memory archive of NrGA does not.
2) The archive of HdEA approximates the fitness landscape; whilst the archive of NrGA represents the density of the evaluated solutions.
3) Since the search space of HdEA is continuous, the number of possible solutions in a sub-region is infinite and no node-pruning is performed.
4) The number of nodes in the *fitness tree* is exactly the same as the number of evaluated solutions, whilst this is in general not true in NrGA due to pruning.

Example: Suppose $S = [0, 1]^2$ is the search space, and EA randomly generates the individual sequence ($\mathbf{s_1}$, $\mathbf{s_5}$, $\mathbf{s_3}$, $\mathbf{s_2}$, $\mathbf{s_6}$, $\mathbf{s_4}$). The corresponding space partitioning is shown in Fig. 2(a). s_i are the real nodes and n_i are the "virtual" nodes. For real nodes, s_i represents both the solution $\mathbf{s_i}$ and the corresponding sub-region h_i. Fig. 2(b) shows the topology of the *fitness tree* T, which is a BSP tree. Please refer to [7] for details on how this tree is constructed.

The leaf nodes labeled $\mathbf{s_1}$, $\mathbf{s_2}$, .., $\mathbf{s_6}$ represent the sub-regions h_1, h_2, $\ldots$, h_6, respectively; the sub-region of node $\mathbf{n_2}$ is $N_2 = h_1 \cup h_2$; the sub-region of node $\mathbf{n_1}$ is $N_1 = h_1 \cup h_2 \cup h_3$; the sub-region of node $\mathbf{n_4}$ is $N_4 = h_4 \cup h_5$; the sub-region of node $\mathbf{n_3}$ is $N_3 = h_4 \cup h_5 \cup h_6$; and node $\mathbf{r}$ represents the entire search space S.

Given the fitness values of the six individuals are: $f(\mathbf{s_1}) = 1, f(\mathbf{s_5}) = 2, f(\mathbf{s_3}) = 5, f(\mathbf{s_2}) = 7, f(\mathbf{s_6}) = 3$, and $f(\mathbf{s_4}) = 4$, the approximated function of $f(\mathbf{x})$ by T is expressed as

$$\tilde{f}(\mathbf{x}) = \begin{cases} 1 & [x_1, x_2] \in [0, 0.35) \times [0.3, 1] \\ 7 & [x_1, x_2] \in [0.35, 0.5) \times [0.3, 1] \\ 5 & [x_1, x_2] \in [0, 0.5) \times [0, 0.3) \\ 4 & [x_1, x_2] \in [0.5, 0.65) \times [0, 0.6) \\ 2 & [x_1, x_2] \in [0.65, 1] \times [0, 0.6) \\ 3 & [x_1, x_2] \in [0.5, 1] \times [0.6, 1]. \end{cases}$$

Fig. 3 shows the output landscape of $\tilde{f}(x)$.

TABLE I
ACTIVE NEIGHBORHOODS OF THE LEAF NODES FOR COMPUTING OPTIMAL SUB-REGIONS

Sub-Region	Neighborhood	Optimal Evaluation Solution of the Neighborhood	Optimal Sub-Region	
h_1	$h_1 \cup h_2 \cup h_3$	$\mathbf{s_1}$	Yes	$f(\mathbf{s_1}) \leq f(\mathbf{s_1})$
h_2	$h_1 \cup h_2 \cup h_3$	$\mathbf{s_1}$	No	$f(\mathbf{s_2}) > f(\mathbf{s_1})$
h_3	S	$\mathbf{s_1}$	No	$f(\mathbf{s_3}) > f(\mathbf{s_1})$
h_4	$h_4 \cup h_5 \cup h_6$	$\mathbf{s_5}$	No	$f(\mathbf{s_4}) > f(\mathbf{s_5})$
h_5	$h_4 \cup h_5 \cup h_6$	$\mathbf{s_5}$	Yes	$f(\mathbf{s_5}) \leq f(\mathbf{s_5})$
h_6	S	$\mathbf{s_5}$	No	$f(\mathbf{s_6}) > f(\mathbf{s_5})$

IV. GUIDED ANISOTROPIC SEARCH

In this section, we present a novel mutation operator GAS. GAS is a *randomized gradient descent-like genetic operator*. Its mutation step size is randomly assigned within an adaptively adjusted range, while its search direction is governed by the approximated fitness landscape from the *fitness tree*.

GAS is an adaptive, parameter-less and guided mutation operator. It has the same advantage as the parameterless adaptive mutation operator in NrGA, namely, the mutation step size is adaptively adjusted based on the search history. Moreover, the search direction of a mutant in GAS is guided by the fitness landscape instead of being a random walk as in the mutation of NrGA.

In GAS, an individual $\mathbf{x}$ moves in the direction that the fitness improvement of $\tilde{f}(\mathbf{x})$ at $\mathbf{x}$ is locally maximal. This can be done by assigning the *mutation direction* $\mathbf{v}$ as the direction pointing to the nearest historical optimum $\mathbf{y}$ of $\mathbf{x}$, i.e., $\mathbf{v} = \mathbf{y} - \mathbf{x}$. Recall that $\tilde{f}(x)$ is a step function; its optimum is in the form of a D-dimensional sub-region rather than a D-dimensional point. Also, the topology of the *fitness tree* represents the adjacencies amongst the steps (sub-regions). Therefore the procedure of finding the nearest optimum of $\mathbf{x}$ is equivalent to finding the nearest optimal sub-region of *the sub-region of* $\mathbf{x}$ in $\tilde{f}(x)$.

To balance the exploitative effect of this gradient descent-like direction assignment, the mutation step size α is randomly selected in the interval (0, 1), in which the mutant $\mathbf{m}$ of $\mathbf{x}$ is a linear combination of $\mathbf{x}$ and $\mathbf{y}$, $\mathbf{m} = \mathbf{x} + \alpha\mathbf{v} = (1-\alpha)\mathbf{x} + \alpha\mathbf{y}$. It may happen that $\mathbf{x}$ is a local optimum, i.e., $\mathbf{x} = \mathbf{y}$. For such cases, the mutant $\mathbf{m}$ will be randomly picked from the sub-region of $\mathbf{x}$. Algorithm A3 shows the procedure of GAS with the fitness landscape approximated by the fitness tree T. Note that GAS naturally avoids generating any out-of-bound solutions. This is an advantage over some other methods that may generate out-of-bound solutions and need to define an extra repair operator to change the solutions back to valid ones.

Seen from Algorithm A3, two types of information about $\tilde{f}(x)$ should be provided to GAS: 1) the optimal sub-region set; and 2) distance metric amongst sub-regions for defining the term "*nearest.*" In the following subsections, we present a method that uses the *fitness tree* to estimate the *optimal sub-region*. Then we propose a metric to measure the distance between sub-regions, and hence define the *nearest optimal sub-region* of a sub-region. An illustrative example is provided. The example employs the same search space, *fitness tree* and terminologies used in the example in Section II. Finally, the strengths and the weaknesses of GAS are discussed.

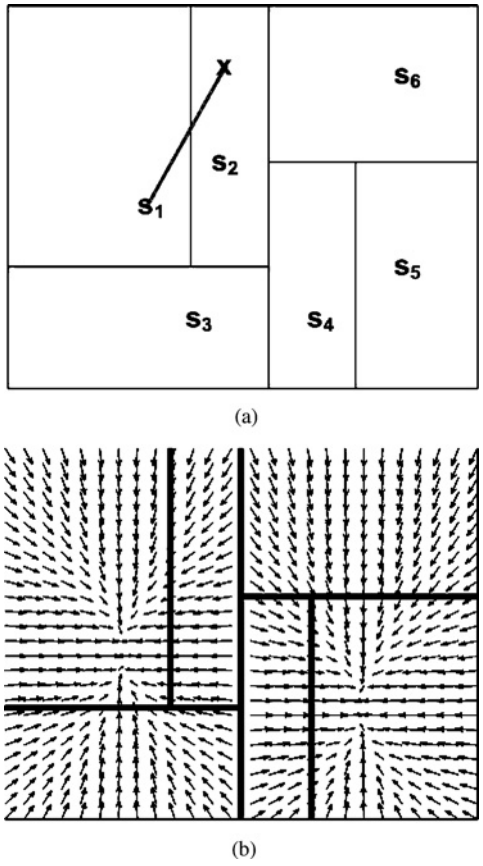


Fig. 4. (a) Gray straight line between $\mathbf{x}$ and $\mathbf{s_1}$ is the range of possible mutants of $\mathbf{x}$ by GAS and (b) mutation directions of $\mathbf{x}$ at different positions of S.

A. Estimation of Optimal Sub-Region

In this subsection, we present a method that estimates the optimal sub-region based on the topology of the *fitness tree*. The sub-region X of node $\mathbf{x}$ is optimal if the fitness value of $\mathbf{x}$ is the smallest amongst the *neighborhood* of X.

Definition 2: Neighborhood of a Sub-region

The sub-region Y of node $\mathbf{y}$ is the neighborhood of the sub-region X of leaf node $\mathbf{x}$ if $X \subseteq Y$.

Remark: Because the BSP tree binary partitions the search space and hierarchically represents the partitioned sub-regions, node $\mathbf{x}$ must be the *descendant* of node $\mathbf{y}$. For example, the sub-region of node $\mathbf{n_2}$ and the sub-region of node $\mathbf{n_1}$ are the neighborhoods of the sub-region of node $\mathbf{s_1}$.

Every sub-region can be regarded as an optimum up to a certain neighborhood size. For example, the sub-region h_1

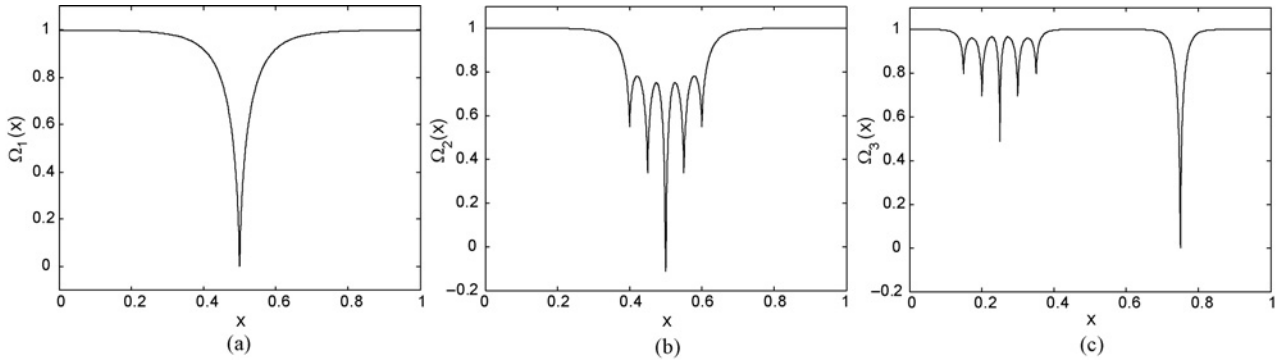


Fig. 5. Sample fitness landscapes for illustrating the effectiveness of GAS. (a) Ω_1. (b) Ω_2. (c) Ω_3.

Fitness function	f_1		f_2		f_3		f_4		f_5		f_6		f_7		f_8		f_9	
D	30	40	30	40	30	40	30	40	30	40	30	40	30	40	30	40	30	40
HdEA				4	8	8	4	6	2	4	6	7						
RCGA-UNDX																		
CMA-ES																		
DE																		
ODE																		
DEahcSPX																		
DPSO																		
SEPSO																		
EDA																		

Fitness function	f_{10}		f_{11}	f_{12}	f_{13}	f_{14}	f_{15}		f_{16}		f_{17}		f_{18}		f_{19}		f_{20}	
D	30	40	2	2	2	2	30	40	30	40	30	40	30	40	30	40	30	40
HdEA					3	5					5	5			7	7		2
RCGA-UNDX																		
CMA-ES																		
DE																		
ODE																		
DEahcSPX																		
DPSO																		
SEPSO																		
EDA																		

Fitness function	f_{21}		f_{22}		f_{23}		f_{24}		f_{25}		f_{26}		f_{27}		f_{28}		f_{29}	
D	30	40	30	40	30	40	30	40	30	40	30	40	30	40	30	40	30	40
HdEA	3	3	2	2					2	2	4	4	7	7			2	2
RCGA-UNDX																		
CMA-ES																		
DE																		
ODE																		
DEahcSPX																		
DPSO																		
SEPSO																		
EDA																		

Fitness function	f_{30}		f_{31}		f_{32}		f_{33}		f_{34}	
D	30	40	30	40	30	40	30	40	30	40
HdEA					8	7				2
RCGA-UNDX										
CMA-ES										
DE										
ODE										
DEahcSPX										
DPSO										
SEPSO										
EDA										

Fig. 6. Indicators of the best test algorithm in the experiments. The cells with gray color represents that the corresponding test algorithm outperforms the others for a particular function and a particular function dimension.

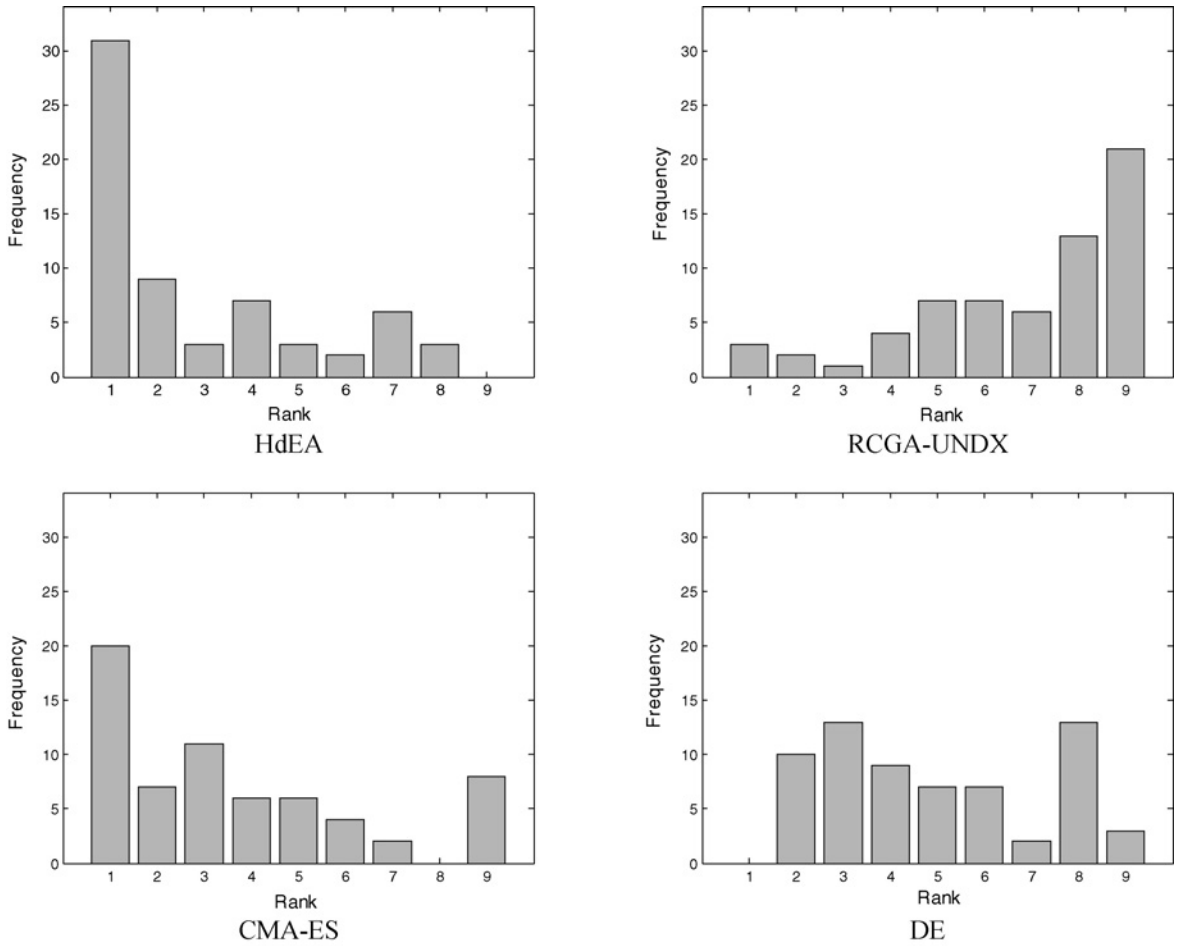


Fig. 7. Histograms of the ranks of HdEA, RCGA-UNDX, CMA-ES, DE, ODE, DEahcSPX, DPSO, SEPSO, and EDA on the 64 test cases.

Algorithm A3: GAS

Input: 1) An individual $\mathbf{x}$, 2) fitness tree T

1. $\{\mathbf{s_i}\}$:= The evaluated solution set stored in T
2. $H = \cup_i h_i$:= The partitioned sub-region set where h_i is the sub-region of $\mathbf{s_i}$

   /* Search for the nearest optimal sub-region $\mathbf{y}$ of $\mathbf{x}$ */
3. Search for the sub-region $h \subseteq H$ of $\mathbf{x}$
4. Find the nearest optimal sub-region of h
5. $\mathbf{y}$:= The evaluated solution inside h

   /* End of Search for the nearest optimal sub-region $\mathbf{y}$ of $\mathbf{x}$ */
6. **If** $(\mathbf{x} = \mathbf{y})$
7. $\mathbf{m} := Random(h)$
8. **Else**
9. $\alpha := Random((0, 1))$
10. $\mathbf{m} := \alpha\mathbf{x} + (1 - \alpha)\mathbf{y}$
11. **End**

Output: The mutated individual $\mathbf{m}$

shown in Fig. 2(a) is optimal when its neighborhood is the whole search space; the sub-region h_5 is an optimum if h_4 is the only neighbor of h_5; in the extreme case, h_2 is an optimum if the corresponding neighborhood is itself. Commonly, the term "*neighborhood size*" refers to the number of neighbor points concerned. Due to the topology of the *fitness tree*, the number of neighbor points (evaluated solutions) of a node decreases with its node depth. For example, when comparing node $\mathbf{n_2}$ and its parent node $\mathbf{n_1}$, the corresponding neighbor point sets increase from $\{\mathbf{s_2}\}$ to $\{\mathbf{s_2}, \mathbf{s_3}\}$. Similarly, while considering node $\mathbf{n_1}$ and its parent node $\mathbf{r}$, the corresponding neighbor point set further increases from $\{\mathbf{s_2}, \mathbf{s_3}\}$ to $\{\mathbf{s_2}, \mathbf{s_3}, \mathbf{s_4}, \mathbf{s_5}, \mathbf{s_6}\}$. Therefore, we represent "*neighborhood size*" by the tree node depth difference:

Definition 3: Neighborhood Size of a Sub-region

Suppose the sub-region Y of node $\mathbf{y}$ is the neighborhood of the sub-region X of node $\mathbf{x}$, the neighborhood size of Y related to X is defined as the depth difference of $\mathbf{y}$ and $\mathbf{x}$.

We denote by l the neighborhood size for estimating the optimal sub-region: Suppose the sub-region Y of node $\mathbf{y}$ is the neighborhood of the sub-region X of node $\mathbf{x}$; and the neighborhood size of Y related to X is l; X is estimated as an optimal sub-region if the fitness value of $\mathbf{x}$ is the smallest amongst all evaluated solutions in Y. When l is chosen to be the depth of the *fitness tree*, it is assumed that the approximated fitness function consists of only one optimum. All individuals are enforced to approach to the best found optimum. On the other hand, if l equals zero, every sub-region is regarded as a local optimum; the corresponding GAS is equivalent to

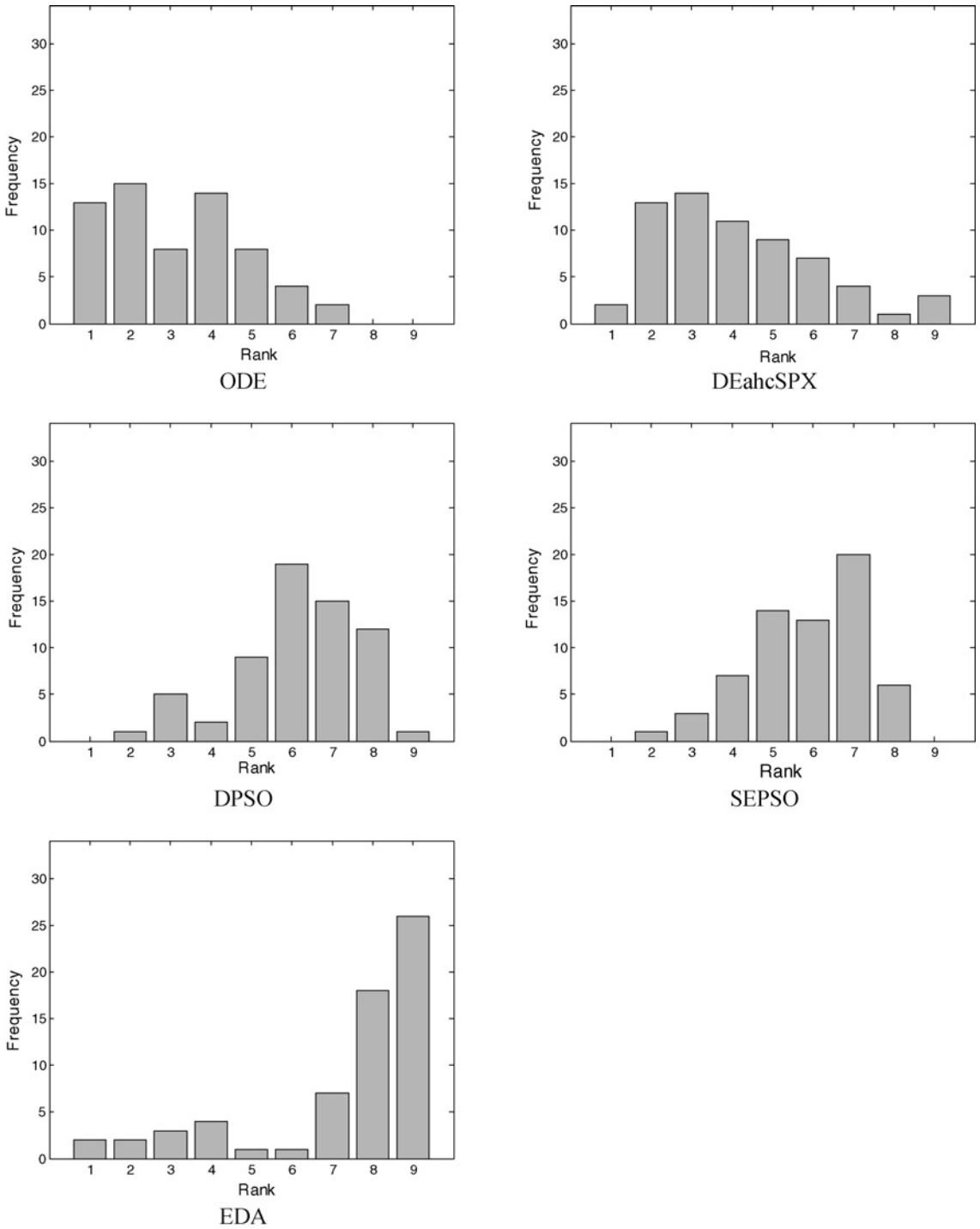


Fig. 7 (*Continued*)

TABLE II
DISTANCES $D(\mathbf{x}, \mathbf{y})$ OF THE SUB-REGIONS AS SHOWN IN FIG. 2(a)

$X \setminus Y$	h_1	h_2	h_3	h_4	h_5	h_6	N_1	N_2	N_3	N_4
h_1	0	1	2	3	3	3	2	1	3	3
h_2	1	0	2	3	3	3	2	1	3	3
h_3	1	1	0	2	2	2	1	1	2	2
h_4	3	3	3	0	1	2	3	3	2	1
h_5	3	3	3	1	0	2	3	3	2	1
h_6	2	2	2	1	1	0	2	2	1	1
N_1	0	0	0	1	1	1	0	0	1	1
N_2	0	0	1	2	2	2	1	0	2	2
N_3	1	1	1	0	0	0	1	1	0	0
N_4	2	2	2	0	0	1	2	2	1	0

The highlighted cell indicates the nearest optimal sub-region of a particular sub-region. For example, the nearest optimal sub-region of h_2 is h_1; and the nearest optimal sub-region of h_4 is h_5.

the adaptive mutation in NrGA, i.e., the UIS, in which an individual is randomly mutated within its sub-region. To make the approximated fitness function locally guide the search in GAS, l should be slightly larger than zero but much smaller than the depth of the *fitness tree*. In this paper, l is chosen to be 2. In the special case that $\mathbf{x}$ is the immediate descendant of $\mathbf{r}$, then $l = 1$.

Table I lists the active neighborhoods of all sub-regions that appear in Fig. 2. Seen from the table, the sub-regions of nodes $\mathbf{s_1}$ and $\mathbf{s_2}$ are regarded as the optimal sub-regions.

B. The Nearest Optimal Sub-Region

After locating the optimal sub-regions of the approximated fitness landscape, the mutation direction of a sub-region X is defined as the direction pointing to its nearest optimal sub-region. Suppose Z is the optimal sub-region set in S. The sub-region $Y \subseteq Z$ is the *nearest optimal sub-region* of the sub-region X of node $\mathbf{x}$ if the metric $D(.)$ from X to Y is the minimum amongst the metrics from X to all optimal sub-regions, i.e., $Y = \arg\min_{A \subseteq Z} D(X, A)$. The following metric measures the distance between sub-regions:

Definition 4: Sub-Region Distance $D(X, Y)$

Suppose X and Y are the sub-regions of nodes $\mathbf{x}$ and $\mathbf{y}$, respectively; let the sub-region P of node $\mathbf{p}$ be the smallest sub-region which contains both X and Y, i.e., $X, Y \subseteq P$, then the sub-region distance from X to Y is defined as the depth difference between $\mathbf{x}$ and $\mathbf{p}$.

Remark: *Sub-region distance* is not commutative; unless $\mathbf{x}$ and $\mathbf{y}$ are at the same tree level, the distance from X to Y may not be equivalent to that from Y to X.

Remark 2: If node $\mathbf{x}$ is a *sibling* of node $\mathbf{y}$, the distance from X to any partitioned sub-region A except X and Y is equivalent to the distance from Y to A, i.e., $D(X, A) = D(Y, A)$ where $A \neq X, Y$.

Referring to the example in Fig. 2, the distances amongst the sub-regions h_1–h_6 and N_1–N_4 are listed in Table II. The highlighted cells indicate the nearest optimal sub-regions of the sub-regions. For example, the nearest optimal sub-region of h_2 is h_1; and the nearest optimal sub-region of h_4 is h_5.

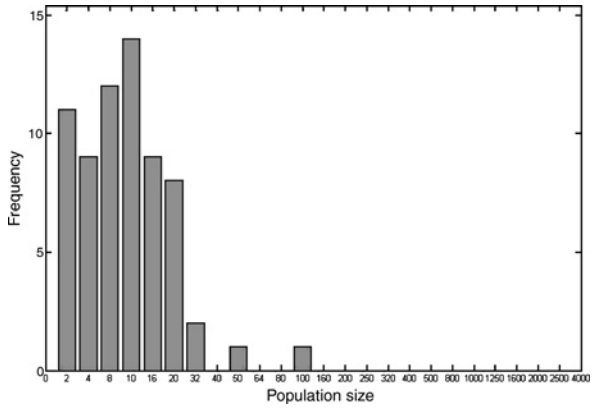


Fig. 8. Histogram of optimal population sizes for f_1–f_{10} and f_{15}–f_{34}.

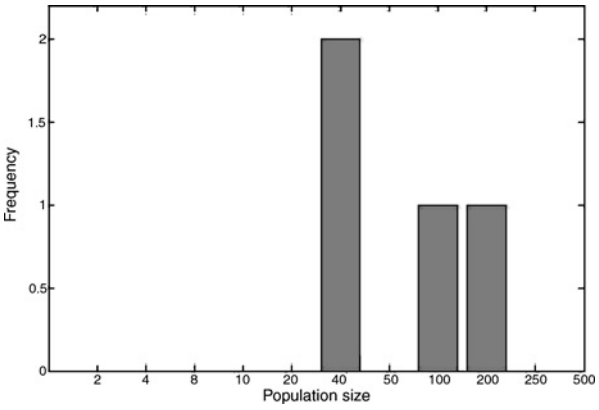


Fig. 9. Histogram of empirical optimal population sizes for f_{11}–f_{14}.

C. Example

In this example, the search space S and the individual sequence (i.e., hence the *fitness tree*) are the same as the example that appeared in Section III. Suppose $\mathbf{x}$ is an individual to be mutated by GAS, the corresponding mutation starts by searching the sub-region $\mathbf{x}$. In actual programming, the result is obtained by tree node search starting at the root node. Suppose the search is terminated at node $\mathbf{s_2}$ and h_2 is the sub-region of $\mathbf{x}$. Afterwards, we compute the distances from h_2 to all optimal sub-regions. According to Table I, the search space consists of two optimal sub-regions: h_1 and h_5. The nearest optimal sub-region of h_2 is h_1 because $D(h_2, h_1) = 1 < D(h_2, h_5) = 3$ (as listed in Table II). Thus, the mutant of $\mathbf{x}$ is suggested as one of the points lying on the straight line between $\mathbf{x}$ and $\mathbf{s_1}$. The gray line shown in Fig. 4(a) indicates the range of possible mutants. Fig. 4(b) shows the mutation directions at different positions in S. Seen from the figure, the directions can be considered as a discrete approximation of the gradient flow of the fitness landscape. Thus, GAS is analogous to *the steepest gradient descent with a random learning rate*.

D. Strengths and Weaknesses of GAS

In this section, we analyze the strengths and the weaknesses of GAS while minimizing three typical forms of fitness landscapes shown in Fig. 5. When searching the fitness landscape Ω_1 with an unusual spike [Fig. 5(a)], since GAS mutates

TABLE III
AVERAGED DIFFERENCE OF THE COMPUTATION TIME (IN SECONDS) BETWEEN HdEA AND RCGA-UNDX, DE, ODE, DEAHCSPX, DPSO AND SEPSO: f_1–f_{14}

Test Function	D	RCGA-UNDX	DE	ODE	DEahcSPX	DPSO	SEPSO
f_1	30	−14.162	1.019	1.013	0.986	0.965	−1.201
	40	−27.764	0.967	0.965	0.923	0.893	−1.994
f_2	30	−14.167	0.997	0.996	0.990	0.970	−1.228
	40	−27.791	0.911	0.910	0.904	0.873	−2.058
f_3	30	−14.676	0.477	0.477	0.460	0.455	−1.741
	40	−28.047	0.644	0.636	0.626	0.614	−2.307
f_4	30	−14.841	0.222	0.222	0.220	0.199	−1.995
	40	−28.306	0.260	0.259	0.258	0.227	−2.698
f_5	30	−14.547	0.602	0.593	0.595	0.591	−1.517
	40	−27.974	0.718	0.706	0.716	0.708	−2.101
f_6	30	−14.429	0.722	0.722	0.716	0.696	−1.474
	40	−27.731	0.959	0.951	0.953	0.924	−1.972
f_7	30	−14.326	0.702	0.699	0.668	0.663	−1.502
	40	−27.835	0.618	0.616	0.587	0.564	−2.319
f_8	30	−13.927	1.103	1.093	1.085	1.077	−1.116
	40	−27.471	0.978	0.974	0.959	0.943	−1.977
f_9	30	−14.484	0.543	0.539	0.538	0.522	−1.672
	40	−27.845	0.602	0.597	0.596	0.574	−2.349
f_{10}	30	−14.032	0.881	0.874	0.852	0.857	−1.331
	40	−27.489	0.817	0.804	0.790	0.785	−2.135
f_{11}	30	0.003	0.003	0.003	0.002	0.012	0.012
f_{12}	40	0.000	0.002	0.002	0.001	0.003	0.003
f_{13}	30	−0.001	0.001	0.001	0.001	0.002	0.002
f_{14}	40	0.000	0.002	0.002	0.001	0.003	0.003

individuals based on the local structure of the fitness function, the mutants always move toward the global optimum. Different from the traditional steepest gradient descent method which may either converge slowly when a small step size is used; or oscillates around the spike when a large step size is used; GAS adaptively computes the mutation step size and so HdEA preserves a fast convergence rate no matter how unusual the spike is.

In the case of fitness landscape Ω_2 with dense and clustered optimums [Fig. 5(b)], such as the Shekel's Foxholes function [24], the crossover operator in HdEA diversifies individuals and helps them to escape from local optima.

For the fitness landscape Ω_3 shown in Fig. 5(c), the success rate of HdEA depends on the selection scheme instead of GAS. Note that Ω_3 is a mixture of Ω_2 and Ω_3. If the search space is divided into two sub-regions (the left and the right sub-regions), and two independent HdEAs are performed on each of the sub-regions, the global optima of Ω_3 should be obtained. However, when only one HdEA is performed, the selection scheme enforces a large portion of population to search the left sub-region, and the search effort on the global optima basin located at the right sub-region is naturally weakened. To tackle this problem, increasing diversity is a possible solution. In other words, given prior knowledge that the structure of fitness function is known to be Ω_3, GAS is still helpful to the search when a selection scheme with lower selection pressure is employed.

HdEA mainly derives benefit from the local structure of the approximated fitness function. GAS speeds up the convergence of HdEA as it rapidly reaches the nearest optimum. However, similar to the steepest gradient descent method, GAS sacrifices diversity to exchange for a faster convergence rate. While dealing with the optimization of noisy and/or highly oscillated function landscapes that consists of densely and/or uniformly distributed local optimas, the corresponding search strategy should concern large-scale structures instead of local structures. Thus, the local optimas may interfere with GAS and occasionally traps the search in a local optimum.

V. EXPERIMENTAL RESULTS

A. Test Function Set

A real valued function set $\mathbf{F} = \{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, \mathbf{f_{34}}(\mathbf{x})\}$ consisting of 34 well-known benchmark functions is employed to evaluate the performance of HdEA. These 34 test functions are as follows.

1. Sphere function [24].
2. Schwefel's problem 2.22 [24].
3. Schwefel's problem 1.2 [24].
4. Schwefel's problem 2.21 [24].
5. Generalized Rosenbrock function [24].
6. Quartic function [24].
7. Generalized Rastrigin function [24].
8. Generalized Griewank function [24].
9. Generalized Schwefel's problem 2.26 [24].
10. Ackley function [24].
11. Shekel's Foxholes function [24].
12. Six-Hump Camel–Back function [24].
13. Branin function [24].
14. Goldstein–Price function [24].
15. High conditioned elliptic function [25].
16. Weierstrass's function [25].
17. Hybrid composition function [25].

TABLE IV
AVERAGED DIFFERENCE OF THE COMPUTATION TIME (IN SECONDS) BETWEEN HdEA AND RCGA-UNDX, DE, ODE, DEAHCSPX, DPSO, AND SEPSO: f_{15}–f_{24}

Test Function	D	RCGA-UNDX	DE	ODE	DEahcSPX	DPSO	SEPSO
f_{15}	30	−13.813	1.176	1.162	1.185	1.176	−1.015
	40	−27.312	1.083	1.077	1.100	1.085	−1.930
f_{16}	30	−11.355	3.145	2.910	3.037	3.342	1.218
	40	−23.882	3.985	3.873	3.924	4.329	1.388
f_{17}	30	−10.323	7.177	6.624	3.446	4.254	4.602
	40	−24.476	7.308	6.884	2.574	3.429	3.853
f_{18}	30	−14.051	0.962	0.957	0.962	0.934	−1.209
	40	−27.501	0.926	0.925	0.931	0.890	−1.965
f_{19}	30	−14.467	0.553	0.553	0.543	0.534	−1.660
	40	−27.635	0.807	0.806	0.794	0.777	−2.145
f_{20}	30	−14.463	0.568	0.567	0.556	0.544	−1.650
	40	−27.803	0.649	0.648	0.637	0.617	−2.307
f_{21}	30	−14.298	0.659	0.656	0.709	0.497	−1.610
	40	−27.565	0.802	0.779	0.887	0.589	−2.219
f_{22}	30	−14.448	0.592	0.583	0.514	0.500	−1.633
	40	−27.687	0.752	0.741	0.657	0.626	−2.216
f_{23}	30	−14.042	1.008	1.003	0.964	0.955	−1.217
	40	−27.535	0.939	0.934	0.885	0.869	−2.024
f_{24}	30	−14.385	0.606	0.585	0.605	0.615	−1.588
	40	−27.648	0.752	0.734	0.756	0.757	−2.173

TABLE V
AVERAGED DIFFERENCE OF THE COMPUTATION TIME (IN SECONDS) BETWEEN HdEA AND RCGA-UNDX, DE, ODE, DEAHCSPX, DPSO, AND SEPSO: f_{25}–f_{34}

Test Function	D	RCGA-UNDX	DE	ODE	DEahcSPX	DPSO	SEPSO
f_{25}	30	−14.504	0.502	0.498	0.500	0.470	−1.662
	40	−27.812	0.601	0.599	0.606	0.562	−2.279
f_{26}	30	−14.482	0.491	0.486	0.500	0.463	−1.698
	40	−27.756	0.619	0.612	0.637	0.584	−2.296
f_{27}	30	−14.243	0.730	0.723	0.749	0.732	−1.454
	40	−27.445	0.928	0.922	0.955	0.932	−1.979
f_{28}	30	−14.026	1.535	1.533	1.400	1.483	−0.539
	40	−27.109	2.005	1.999	1.765	1.966	−0.723
f_{29}	30	−14.555	0.426	0.415	0.399	0.379	−1.808
	40	−27.872	0.514	0.503	0.479	0.451	−2.461
f_{30}	30	−14.582	0.468	0.465	0.425	0.413	−1.747
	40	−27.937	0.538	0.528	0.485	0.465	−2.412
f_{31}	30	−17.417	0.412	0.409	0.383	0.358	−2.209
	40	−33.438	0.422	0.421	0.418	0.380	−3.034
f_{32}	30	−14.508	0.527	0.525	0.509	0.505	−1.674
	40	−27.862	0.587	0.581	0.568	0.557	−2.347
f_{33}	30	−14.282	0.589	0.567	0.605	0.701	−1.492
	40	−27.464	0.780	0.752	0.811	0.929	−1.990
f_{34}	30	−12.080	0.438	0.338	0.831	2.880	−0.363
	40	−22.730	0.649	0.251	2.406	5.621	0.871

18. Levy function [1].
19. Zakharov function [1].
20. Alpine function [1].
21. Pathological function [1].
22. Inverted cosine wave function (Masters) [1].
23. Inverted cosine mixture problem [26].
24. Epistatic Michalewicz problem [26].
25. Levy and Montalvo 2 problem [26].
26. Neumaier 3 problem [26].
27. Odd square problem [26].
28. Paviani problem [26].
29. Periodic problem [26].
30. Salomon problem [26].
31. Shubert problem [26].
32. Sinusoidal problem [26].
33. Michalewicz function [1].
34. Whitely's function [24].

The mathematical forms of these functions are given in Appendix I.

Seven of them are uni-modal functions; the remaining 27 are multimodal functions designed with a considerable amount of local minima. Additionally, the function f_6 is a noisy function and the function f_{17} is a hybrid composition function. The dimensions of the first ten and the last 20 functions are adjustable while the dimensions of f_{11}–f_{14} are fixed at two, as they are 2-D functions as defined in the original references.

TABLE VI
CONSIDERED COMBINATIONS OF POPULATION SIZE AND THE NUMBER OF GENERATIONS FOR f_1–f_{10} and f_{15}–f_{34}

μ	N_g	μ	N_g	μ	N_g
2	20 000	80	500	1250	32
4	10 000	100	400	1600	25
5	8000	125	320	2000	20
8	5000	160	250	2500	16
10	4000	200	200	4000	10
16	2500	250	160	5000	8
20	2000	320	125	8000	5
25	1600	400	100	10 000	4
32	1250	500	80	20 000	2
40	1000	625	64		
50	800	800	50		
64	625	1000	40		

TABLE VII
CONSIDERED COMBINATIONS OF POPULATION SIZE AND THE NUMBER OF GENERATIONS FOR f_{11}–f_{14}

μ	N_g	μ	N_g	μ	N_g
2	500	20	50	125	8
4	250	25	40	200	5
5	200	40	25	250	4
8	125	50	20	500	2
10	100	100	10		

The optimal points of the functions f_{24}, f_{27}, f_{28}, f_{31}, f_{33}, and f_{34} are not known. Simulations are carried out to find the global minimum of each function.

B. Algorithms for Comparison

To evaluate the impact of the proposed algorithm, we compare the optimal fitness found by HdEA with eight benchmark evolutionary algorithms. The designs and settings of HdEA and the algorithms for comparison are summarized below. Simulation settings are given in Section C.

1) *Test Algorithm 1—HdEA:* HdEA, proposed in this paper, is an evolutionary algorithm that uses the entire search history to improve its mutation strategy. It uses the fitness function approximated from the search history to perform mutation. Since the proposed mutation operator is adaptive and parameter-less, HdEA has only three control parameters: neighborhood size, population size, and crossover rate. In this section, random population initialization, uniform crossover operator and (μ+μ) elitism selection are used in EAM of HdEA. Note that HdEA is not limited to these operator settings. Other crossover operators such as *arithmetic crossover* and *n-point crossover*, and any selection scheme such as *proportional selection* and *tournament selection* are also applicable. The neighborhood size is constantly assigned to be two throughout this section. The source code of HdEA is available at http://www.ee.cityu.edu.hk/~syyuen/Public/Code.html

2) *Test Algorithm 2—Real Coded GA With Uni-Modal Normal Distribution Crossover (RCGA-UNDX):* RCGA-UNDX [27] is a real coded GA that deals with continuous search spaces. It applies the uni-modal normal distribution crossover (UNDX) to preserve the statistics of the population. UNDX is a multiparent genetic operator in which the distribution of the corresponding offspring follows the distribution of the parents.

3) *Test Algorithm 3—Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [28]:* CMA-ES is an evolution strategy that adapts the full covariance matrix of a normal search (mutation) distribution. An important property of CMA-ES is its invariance against linear transformations of the search space. The underlying idea is to gather information about successful search steps to modify the covariance matrix of the mutation distribution in a de-randomized, goal directed fashion. Changes to the covariance matrix are such that variances in directions of the search space that have previously been successful are increased, while those in other directions decrease passively. The accumulation of information over a number of search steps makes it possible to reliably adapt the covariance matrix even when using small populations. CMA-ES is designed with the emphasis that the same parameters are used in all applications in order to be "parameter-less." The source code of CMA-ES is taken from [28] (Aug. 2007 version).

4) *Test Algorithm 4—DE:* Differential evolution [4], [5] is a stochastic search algorithm. The basic idea behind DE is a scheme that generates trial parameter vectors. DE adds the weighted difference between two population vectors to a mutant vector, and the trial vector is the crossover between the mutant vector and the parent vector. By doing so, no separate probability distribution is used, which makes the scheme completely self-organizing.

5) *Test Algorithm 5—Opposition-Based Differential Evolution (ODE) [29]:* Opposition-based differential evolution utilizes the concept of opposition-based learning (OBL) [30] to accelerate the convergence rate of DE. The main idea behind OBL is the simultaneous considerations of a solution and its corresponding opposite solution. ODE considers the evaluations of the opposite solution in a generation depending on a *jumping rate*.

6) *Test Algorithm 6—Differential Evolution With Adaptive Hill-Climbing Simplex Crossover (DEahcSPX):* DEahcSPX [31] attempts to accelerate the classic DE by a local search strategy, named adaptive hill-climbing crossover-based local search. It adopts the simplex crossover operation (SPX) [32] to generate offspring individual for hill-climbing.

7) *Test Algorithm 7—Dissipative Particle Swarm Optimization (DPSO):* DPSO [2] is a modified PSO which introduces random mutation that helps particles to escape from local minima. Its formula is described as follows:

$$\text{If } \eta_3 < C_v \text{ then } v_i = \eta_4 \times V_{\max}/C_m$$

where η_3 and η_4 are uniformly distributed random variables in the range [0, 1], C_v is the mutation rate to control the velocity, C_m is a constant to control the extent of mutation, and $V_{\max}$ is the maximum velocity.

8) *Test Algorithm 8—PSO With Spatial Particle Extension (SEPSO):* SEPSO [33] is another modified PSO which introduces the spatial particle extension model to increase the diversity. When particles start to cluster and collide, they bounce off by adjusting their velocities.

9) *Test Algorithm 9—EDA:* The tested EDA is based on undirected graphical model and Bayesian network. The source code of the EDA is taken from [34] (Feb. 2009 version). The implementation is conceived to allow the user different combinations of selection, learning, sampling, and local search procedures [35].

C. Simulation Settings

The parameter settings of the test algorithms for all conducted experiments are as follows.

For HdEA, the crossover rate is set to 0.1. The population sizes are chosen to be 20.

For the rest of the algorithms, we use the same parameter settings as that recommended in the original literature

For RCGA-UNDX, the crossover rate is set to 0.1. The UNDX operator is in the three-parent mode. For uni-modal test functions (i.e., f_1–f_6), the population size is chosen to be 50. For multimodal test functions (i.e., f_7–f_{34}), the population size is chosen to be 300. These parameters (i.e., the number of parents for UNDX and the population size of RCGA-UNDX) are suggested in [27].

For CMA-ES, the population size λ is chosen by the suggested setting in [28] (i.e., $\lambda = 4 + \lfloor 3 \ln D \rfloor$). The remaining parameters used in CMA-ES are chosen to be the predefined values in [28].

For DE and ODE, the crossover rate and the differential amplification factor are set to 0.95 and 0.5, respectively. These values have been used in literature [5], [36]–[38], [39]. The mutation strategy is DE/rand/1/bin (classic version of DE) [5], [38], [40], [41]. The jumping rate constant of ODE is chosen to be 0.3 [29]. The population sizes of DE and ODE are assigned as 100 [29] for all test functions.

For DEahcSPX, the crossover rate and the differential amplification factor are selected as 0.9 and 0.9, respectively [31]. The population size λ of DEahcSPX is assigned as the dimension of test function, i.e., $\lambda = D$. The number of parents participating in SPX, as suggested in [32], is chosen to be 3.

For DPSO and SEPSO, the values of c_1, c_2 are set to 2. The inertia w is linearly decreasing from 0.9 to 0.4. The swarm sizes are set to 20. Suppose $S = \prod_{i=1}^{D}[L_i, U_i]$ is the search space of a D-dimensional test function, where L_i and U_i are the lower and upper limits in the ith dimension, the maximum velocity $V_{\max}$ is set to $0.1R$ where $R = \max_{i \in [1,D]} (U_i - L_i)$. The parameters C_v and C_m of DPSO are chosen to be 0.001 and 0.002, respectively. For SEPSO, a simple velocity line bouncing with bouncing factor -1 is used. These parameter settings are recommended in the original works [2], [33].

For EDA, the selection scheme is $(\mu + \mu)$ elitism selection. The offspring population is sampled by probabilistic logic sampling. The parameters of these operators are chosen to be the values suggested in the source code.

To provide a fair comparison amongst the test algorithms, the total number of function evaluations of all algorithms is kept a constant: For functions f_1–f_{10} and f_{15}–f_{34}, the total number of fitness evaluations of all test algorithms is fixed at 40 000. For functions f_{11}–f_{14}, the total number of fitness evaluations is fixed at 1000.

All test functions with the exception of f_{11}–f_{14}, which are 2-D, are tested with dimensions 30 and 40. Since the test algorithms are stochastic, their performances on each test function are evaluated based on statistics obtained from 100 independent runs. All simulations are performed on a PC with 3.2 GHz central processing unit and 1 GB memory. All test algorithms except CMA-ES and EDA (HdEA, RCGA-UNDX, DE, ODE, DEahcSPX, DPSO, and SEPSO) are implemented in C language. CMA-ES uses source code in [28] and MATLAB version 6.1. EDA uses source code in [34] and MATLAB version 6.1.

D. Simulation Results

In this section, we first observe the performance in terms of accuracy (quality of the averaged best found fitness) of HdEA in comparison with RCGA-UNDX, CMA-ES, DE, ODE, DEahcSPX, DPSO, SEPSO, and EDA. To be a practical solution to real world problems, the processing time of HdEA should be within a reasonable range. Thus, the average processing time is also observed. Finally, the influence of population size to the performance of HdEA is observed.

The average and the standard deviation (inside brackets) of the optimal fitness for 100 trials are presented in Tables VIII–XV. In Tables III–V, the averaged differences of the computation time between HdEA and RCGA-UNDX, DE, ODE, DEahcSPX, DPSO, and SEPSO are presented. In Fig. 10, the performance of HdEA with various population sizes is presented.

1) *Accuracy:* Fig. 6 presents a summary of the results. The shaded cells in Fig. 6 indicate that the corresponding test algorithm is the best algorithm on a particular test function at a particular function dimension. The values inside the table cells for HdEA indicate the ranking of HdEA on a particular test function at a particular function dimension when it is not the best algorithm. It can be observed that HdEA outperforms the other test algorithms: HdEA ranks first (or joint first) in 31 and is second in nine out of a total of 64 cases. Fig. 7 shows the histogram of the ranking amongst the test algorithms in the 64 test cases.

The detailed simulation results (mean and standard deviation) are listed in Tables VIII–XV in Appendix II. It lists the average and the standard deviation (inside brackets) of the optimal fitness for 100 independent trials. A value in **boldface** indicates that the corresponding algorithm is the best amongst the test algorithms on a particular test function at a particular function dimension. To illustrate the significance of the indicator in Fig. 6, the confidence levels C (in terms of %) for t-tests comparing the averaged optimal fitness values of HdEA with other algorithms are listed. In the tables, a larger confidence level infers a higher significance of a comparison result.

The significance of the rank of HdEA (i.e., HdEA ranks first (or joint first) in 31 and is second in nine out of a total of 64 cases) is summarized as follows. We are 99.95% confident that HdEA performs the best amongst all test algorithms in

21 test cases. For the remaining ten test cases that HdEA ranks or jointly ranks first, two out of them are with 99% confidence level, one out of them is with 97.5% confidence level, three out of them are with 95% confidence level, two out of them are with 80% confidence level, and the remaining two cases are with less than 50% confidence level. For the nine test cases that HdEA ranks second, eight out of them are with 99.95% confidence level and the remaining one is with 99.75% confidence level. From the above, we conclude that the comparisons of HdEA with other algorithms are statistically significant.

Fig. 7 shows the histograms of the ranks of the test algorithms on the 64 test cases. Seen from figure, nearly 80% of HdEA results are in the first four ranks. It shows the consistent and superior performance of HdEA amongst the test algorithms. For ODE, its performance is also consistent but not as good as HdEA. On the other hand, for CMA-ES which is the second best (in terms of the number of test cases with first rank) test algorithm, around 69% of CMA-ES results are ranked lower or equal to fourth. In addition, it performs the worst amongst the test algorithms in eight out 64 test cases. Thus, though the number of first ranked test cases of CMA-ES is just slightly lower than that of HdEA, the number of worst ranked test cases of CMA-ES is much more than that of HdEA (i.e., eight for CMA-ES and none for HdEA). We conclude that HdEA is superior to CM-ES in terms of consistency.

2) *Processing Time:* During the search process, an algorithm spends its computation resources on either solution generation or function evaluation. Different algorithms use different strategies to generate trial solutions, so the corresponding processing times are different. HdEA spends most computation on recording the evaluated solutions (i.e., BSP tree node insertion) and performing GAS. RGA-UNDX and DEahcSPX spend computations on their specific crossover operators. Comparing with DE that employs simple vector addition to generate trial vectors, ODE involves additional processes such as triggering the opposition-based learning and the generation of opposite solutions. CMA-ES generates new candidate solutions according to the covariance matrix of the multivariate normal mutation distribution. DPSO checks if any particle should be re-initialized. For SEPSO, path tracking is required for detecting the possible collisions amongst the swarm. EDA estimates the probability distribution of the evaluated solutions and generates new candidate solutions based on it. As a practical solution to real world applications, the processing time of HdEA should be within an acceptable range. In this section, the computation load of an algorithm, in terms of processing time, is studied.

Tables III–V list the averaged differences of the computation time (in second) between HdEA and RCGA-UNDX, DE, ODE, DEahcSPX, DPSO, and SEPSO for f_1–f_{34}. The comparisons with CMA-ES and EDA are not made since they are implemented in MATLAB. A negative value indicates that the corresponding test algorithm is slower than HdEA for a particular function at a particular dimension. Seen from the tables, HdEA is faster than RCGA-UNDX for all test cases except that it is just 0.00312 seconds and 0.00015 seconds slower than RCGA-UNDX for f_{11} and f_{12}, respectively. Comparing with SEPSO, HdEA is faster at all high-dimensional test cases (i.e., $D = 30$ and 40) except for f_{16}, f_{17}, and f_{34} which are relatively time consuming. For the low-dimensional test cases (i.e., $D = 2$), the overheads of HdEA are in a small range from 0.0022 seconds to 0.01219 seconds. Though HdEA spends more computation time than DE ODE, DEahcSPX, and DPSO for all test cases; the corresponding overheads range from 0.00095 seconds to 7.3084 seconds; but its performance is better than those of DE, ODE, DEahcSPX, and DPSO in 60, 38, 43, and 52 out of 64 test cases, respectively. Moreover, it should be emphasized that, for real world applications involving expensive and/or time consuming fitness evaluations, such as surface registration [42], optimized design and energy management of heating, ventilating and air conditioning systems [43]–[45], function evaluations are much more expensive and/or time consuming than solution generation. For such applications, the overhead of HdEA, of maximum a few more seconds, is insignificant.

3) *Influence of Population Size:* In this experiment, we examine the performance of HdEA for different population sizes but fixed number of fitness evaluations. For the test functions f_1–f_{10} and f_{15}–f_{34}, the population size is varied from 2 to 20 000. Table VI lists the combinations of population size μ and the number of generations N_g. The total numbers of fitness evaluations for these functions are fixed at 40 000.

For the test functions f_{11}–f_{14}, the population size is varied from 2 to 500. Table VII lists the combinations of μ and N_g. The total numbers of fitness evaluations are 1000.

The remaining parameter settings for this experiment are as follows.

1) Crossover operator = uniform crossover.
2) Crossover rate = 0.1.
3) Selection scheme = $(\mu + \mu)$ elitism selection.
4) Dimensions D of the test functions: $D = 30$ for f_1–f_{10} and f_{15}–f_{34}, and $D = 2$ for f_{11}–f_{14}.

Fig. 8 shows the frequencies of the empirical optimal population sizes for f_1–f_{10} and f_{15}–f_{34}. Fig. 9 shows the frequencies of the optimal population sizes for f_{11}–f_{14}. Seen from the figures, we should employ a small population size ranging from for 2 to 100 for both low and high-dimensional fitness functions when the number of fitness evaluations is fixed.

Fig. 10 in Appendix III shows the averaged best fitness found by HdEA against population size for f_1–f_{34}. The x-axis is in log scale. Seen from the figure, the relation between population size and averaged best fitness can be divided into three cases. For the first case, the averaged best fitness monotonically increases along with population size. The corresponding curve vaguely resembles a sigmoid function. The averaged best fitness of HdEA is insensitive to the population size μ in the range between 2 and 100. When μ further increases beyond this range, the corresponding fitness value monotonically increases and converges to a certain value. Twenty seven out of the 34 test functions

(f_1–f_{10}, f_{14}–f_{18}, f_{20}–f_{26}, f_{28}–f_{30}, and f_{33}–f_{34}) have similar scenarios.

For the second case, the relation between the averaged best fitness and population size is in the form of a convex function. Six test functions (f_{11}–f_{13}, f_{27}, f_{31}, and f_{32}) belong to this scenario and the corresponding optimal population sizes are in a small range from 50 to 200.

For the remaining test function f_{19}, the proper selection of population size is critical to the accuracy of HdEA. The curve of f_{19} shown in Fig. 10 is oscillating: it is neither monotonically increasing, monotonically decreasing, concave, nor convex.

VI. Conclusion

In this paper, we propose a new evolutionary algorithm that adaptively guides mutation by the entire previous search history. This evolutionary search, namely HdEA, integrates an evolutionary algorithm with a binary space partitioning tree that encodes the search history. Moreover, we employ the tree to approximate the fitness function, and use the approximated function to derive a parameter-less, adaptive, and guided mutation. This mutation somewhat resembles the gradient descent in classical optimization methods, but the step size is random and interesting, the maximum step size is parameter-less and is a function of the entire search history. HdEA has the following properties.

1) Because of the partitioning scheme, the neighborhoods of the evaluated solutions follow the topology of *fitness tree*; a fast fitness function approximation is obtained.
2) The proposed adaptive mutation in HdEA suggests both the direction and the magnitude of the mutation vector in a parameter-less manner.
3) The adaptive mutation naturally avoids the out-of-bound solution problem in bounded real valued optimization algorithms.
4) It is empirically suggested that HdEA performs well with small population size (i.e., 40 to 100 individuals for a 40-dimensional function).

In the experiment section, we examine HdEA on 34 benchmark problems, including both uni-modal and multi-modal functions. The dimensions of the test functions are from 2 to 40. We compare the performance of HdEA with eight benchmark real coded evolutionary algorithms. It is found that for multimodal functions, HdEA outperforms all the other algorithms, while it does not perform as well for uni-modal functions. This suggests that HdEA should be used in the optimization of multimodal functions, which represents the harder and more challenging application problems.

Possible directions for future work include applying other function approximation methods (e.g., parametric model fitting and neural network) and extending the usage of the whole search history (e.g., cooperative mutation strategy and parameter controls such as adaptive crossover rate and adaptive selection pressure).

Appendix I

1. Sphere function [24]

$$f_1(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$$

where $\mathbf{x} \in [-100, 100]^D$

$$\min_{\mathbf{x}} f(\mathbf{x}) = f_1([0, 0, \cdots, 0]) = 0.$$

2. Schwefel's problem 2.22 [24]

$$f_2(\mathbf{x}) = \sum_{i=1}^{D} |x_i| + \prod_{i=1}^{D} |x_i|$$

where $\mathbf{x} \in [-10, 10]^D$

$$\min_{\mathbf{x}} f_2(\mathbf{x}) = f_2([0, 0, \cdots, 0]) = 0.$$

3. Schwefel's problem 1.2 [24]

$$f_3(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2$$

where $\mathbf{x} \in [-100, 100]^D$

$$\min_{\mathbf{x}} f_3(\mathbf{x}) = \mathbf{f_3}([\mathbf{0}, \mathbf{0}, \cdots, \mathbf{0}]) = \mathbf{0}.$$

4. Schwefel's problem 2.21 [24]

$$f_4(\mathbf{x}) = \max_{i \in [1, D]} |x_i|$$

where $\mathbf{x} \in [-100, 100]^D$

$$\min_{\mathbf{x}} f_4(\mathbf{x}) = \mathbf{f_4}([\mathbf{0}, \mathbf{0}, \cdots, \mathbf{0}]) = \mathbf{0}.$$

5. Generalized Rosenbrock function [24]

$$f_5(\mathbf{x}) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right]$$

where $\mathbf{x} \in [-29, 31]^D$

$$\min_{\mathbf{x}} f_5(\mathbf{x}) = f_5([1, 1, \cdots, 1]) = 0.$$

6. Quartic function [24]

$$f_6(\mathbf{x}) = \sum_{i=1}^{D} i x^4 + random[0, 1]$$

where $\mathbf{x} \in [-1.28, 1.28]^D$

$$\min_{\mathbf{x}} f_6(\mathbf{x}) = f_6([0, 0, \cdots, 0]) = 0.$$

Note: This is a noisy fitness function. There is a random measurement noise in each fitness evaluation.

7. Generalized Rastrigin function [24]

$$f_7(\mathbf{x}) = \sum_{i=1}^{D} \left[x_i^2 - 10 \cos(2\pi x_i) + 10 \right]$$

where $\mathbf{x} \in [-5.12, 5.12]^D$

$$\min_{\mathbf{x}} f_7(\mathbf{x}) = f_7([0, 0, \cdots, 0]) = 0.$$

8. Generalized Griewank function [24]

$$f_8(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos \frac{x_i}{\sqrt{i}} + 1$$

where $\mathbf{x} \in [-600, 600]^D$

$$\min_{\mathbf{x}} f_{\mathbf{x}}(\vec{x}) = f_8([0, 0, \cdots, 0]) = 0.$$

9. Generalized Schwefel's problem 2.26 [24]

$$f_9(\mathbf{x}) = -\sum_{i=1}^{D} x_i \sin\sqrt{|x_i|}$$

where $\mathbf{x} \in [-500, 500]^D$

$$\min_{\mathbf{x}} f_9(\mathbf{x}) = f_9([420.9687, \cdots, 420.9687]) = -418.9829D.$$

10. Ackley function [24]

$$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x_i\right) + 20 + e$$

where $\mathbf{x} \in [-32, 32]^D$ and $\min_{\mathbf{x}} f_{10}(\mathbf{x}) = f_{10}([0, 0, \cdots, 0]) = 0$.

11. Shekel's Foxholes function [24]

$$f_{11}(\mathbf{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i - a_{i,j})^6}\right]^{-1}$$

where $\mathbf{x} \in [-98, 34]^2$ and

$$\{a_{i,j}\} = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{bmatrix}$$

$$\min_{\mathbf{x}} f_{11}(\mathbf{x}) = f_{11}([-32, -32]) \approx 1.$$

12. Six-hump Camel–Back function [24]

$$f_{12}(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

where $\mathbf{x} \in [-4.91017, 5.0893] \times [-5.7126, 4.2874]$ and

$$\min_{\mathbf{x}} f_{12}(\mathbf{x}) = f_{12}([0.08983, -0.7126]) = f_{12}([-0.08983, 0.7126]) = -1.0316285.$$

13. Branin function [24]

$$f_{13}(\mathbf{x}) = (x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos x_1 + 10$$

where $\mathbf{x} \in [-8.142, 6.858] \times [-12.275, 2.725]$ and

$$\min_{\mathbf{x}} f_{13}(\mathbf{x}) = f_{13}([-3.142, 12.275]) = f_{13}([3.142, 2.275]) = f_{13}([9.425, 2.425]) = 0.398.$$

14. Goldstein–Price function [24]

$f_{14}(\mathbf{x}) = g(\mathbf{x}) \times h(\mathbf{x})$

where

$$g(\mathbf{x}) = 1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)$$

$$h(\mathbf{x}) = 30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)$$

$\mathbf{x} \in [-2, 2] \times [-3, 1]$ and $\min_{\mathbf{x}} f_{14}(\mathbf{x}) = f_{14}([0, -1]) = 3$.

15. High conditioned elliptic function [25]

$$f_{15}(\mathbf{x}) = \sum_{i=1}^{D} 10^{\frac{6(i-1)}{D-1}} z_i^2$$

where $\mathbf{x} \in [-100, 100]^D$ and

$$\min_{\mathbf{x}} f_{15}(\mathbf{x}) = f_{15}([-100, -100, ..., -100]) = 0.$$

16. Weierstrass's function [25]

$$f_{16}(\mathbf{x}) = \sum_{i=1}^{D}\sum_{k=0}^{D} a^k \cos(2\pi b^k(z_i - 0.5)) - D\sum_{i=1}^{D} a^k \cos(\pi b^k)$$

where $\mathbf{x} \in [-0.5, 0.5]^D$ and $\min_{\mathbf{x}} f_{16}(\mathbf{x}) = f_{16}([0, \ldots, 0]) = 0$.

17. Hybrid composition function [25]

$$f_{17}(\mathbf{x}) = \sum_{i=1}^{10} w_i(g_i(\lambda_i(\mathbf{x} - \mathbf{o_i})) + b_i)$$

where

$$\mathbf{o_i} = \{o_{i,j}\}$$

$$g_i(\lambda_i\mathbf{x}) = C \times \frac{h_i(\lambda_i\mathbf{x})}{h_i(\lambda_i[5, 5, \ldots, 5])},$$

$$h_{1-2}(\mathbf{x}) = f_7(\mathbf{x})$$

$$h_{3-4}(\mathbf{x}) = f_{16}(\mathbf{x})$$

$$h_{5-6}(\mathbf{x}) = f_8(\mathbf{x})$$

$$h_{7-8}(\mathbf{x}) = f_{10}(\mathbf{x})$$

$$h_{9-10}(\mathbf{x}) = f_1(\mathbf{x})$$

$$w_i = \exp\left(-\sum_{j=1}^{D}\frac{(x_j - o_{i,j})^2}{2D\sigma_j^2}\right)$$

$\sigma_j = 1$ for $j = 1, 2, \ldots, D$

$\lambda_i = [1, 1, 0.1, 0.1, 12, 12, 6.4, 6.4, 20, 20]$

$b_i = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900]$

$\mathbf{o_i} = [u_i, u_i, \ldots, u_i]$ where $u_i = 0.5 \times floor(i/2)$

$C = 2000$

$\mathbf{x} \in [-5, 5]^D$ and $\min_{\mathbf{x}} f_{17}(\mathbf{x}) = f_{17}([0, \ldots, 0]) = 4500$.

18. Levy function [1]

$$f_{18}(\mathbf{x}) = \sin^2(\pi x_1) + \textstyle\sum_{i=1}^{D-1}(x_i - 1)^2(1 + 10\sin^2(\pi x_{i+1})) + (x_n - 1)^2(1 + 10\sin^2(2\pi x_n))$$

where $y_i = 1 + \frac{x_i - 1}{4}$ and $\mathbf{x} \in [-10, 10]^D$ and $\min_{\mathbf{x}} f_{18}(\mathbf{x}) = f_{18}([1, \ldots, 1]) = 0$.

19. Zakharov function [1]

$$f_{19}(\mathbf{x}) = \sum_{i=1}^{D} x_i^2 + \left(\sum_{i=1}^{D} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{D} 0.5ix_i\right)^4$$

where $\mathbf{x} \in [-5, 10]^D$ and $\min_{\mathbf{x}} f_{19}(\mathbf{x}) = f_{19}([0, \ldots, 0]) = 0$.

20. Alpine function [1]

$$f_{20}(\mathbf{x}) = \sum_{i=1}^{D} |x_i \sin(x_i) + 0.1x_i|$$

where $\mathbf{x} \in [-10, 10]^D$ and $\min_{\mathbf{x}} f_{20}(\mathbf{x}) = f_{20}([0, \ldots, 0]) = 0$.

21. Pathological function [1]

$$f_{21}(\mathbf{x}) = \sum_{i=1}^{D-1}\left(0.5 + \frac{\sin^2\sqrt{100x_i^2 + x_{i+1}^2} - 0.5}{1 + 0.001\left(x_i^2 - 2x_ix_{i+1} + x_{i+1}^2\right)^2}\right)$$

where $\mathbf{x} \in [-100, 100]^D$ and $\min_{\mathbf{x}} f_{21}(\mathbf{x}) = f_{21}([0, \ldots, 0]) = 0$.

22. Inverted cosine wave function (Masters) [1]

$$f_{22}(\mathbf{x}) = -\sum_{i=1}^{D-1} \left(\exp\left(\frac{-\left(x_i^2+x_{i+1}^2+0.5x_ix_{i+1}\right)}{8}\right) \times \cos\left(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_ix_{i+1}}\right)\right)$$

where $\mathbf{x} \in [-5, 5]^D$ and $\min_{\mathbf{x}} f_{22}(\mathbf{x}) = f_{22}([0, \ldots, 0]) = -D+1$.

23. Inverted cosine mixture problem [26]

$$f_{23}(\mathbf{x}) = 0.1D - \left(0.1\sum_{i=1}^{D}\cos(5\pi x_i) - \sum_{i=1}^{D} x_i^2\right)$$

where $\mathbf{x} \in [-1, 1]^D$ and $\min_{\mathbf{x}} f_{23}(\mathbf{x}) \approx f_{23}([0, \ldots, 0]) = 0$.

24. Epistatic Michalewicz problem [26]

$$f_{24}(\mathbf{x}) = -\sum_{i=1}^{D}\sin(y_i^2)\sin^{2m}\left(\frac{iy_i^2}{\pi}\right)$$

where $y_i = \begin{cases} x_i\cos\theta - x_{i+1}\sin\theta & i = 1, 3, 5... < n \\ x_i\sin\theta + x_{i+1}\cos\theta & i = 2, 4, 6... < n \\ x_i & i = n \end{cases}$

m =10, $\theta = \pi/6$ and $\mathbf{x} \in [0, \pi]^D$.

25. Levy and Montalvo 2 problem [26]

$$f_{25}(\mathbf{x}) = 0.1\left(\sin^2(3\pi x_1) + \sum_{i=1}^{D-1}(x_i-1)^2(1+10\sin^2(3\pi x_{i+1})) + (x_n-1)^2(1+10\sin^2(2\pi x_n))\right)$$

where $\mathbf{x} \in [-5, 5]^D$ and $\min_{\mathbf{x}} f_{25}(\mathbf{x}) = f_{25}([1, \ldots, 1]) = 0$.

26. Neumaier 3 problem [26]

$$f_{26}(\mathbf{x}) = \sum_{i=1}^{D}(x_i-1)^2 - \sum_{i=2}^{D} x_ix_{i-1}$$

where $\mathbf{x} \in [-\mathrm{D}^2, D^2]D^2$ and $\min_{\mathbf{x}} f_{26}(\mathbf{x}) = f_{26}(\mathbf{x_0}) = -\frac{D(D+4)(D-1)}{6}$, $x_{0,\ i} = i(D+1-i)$.

27. Odd square problem [26]

$$f_{27}(\mathbf{x}) = -\left(1 + \frac{0.2n}{N+0.1}\right)\cos(N\pi)\, e^{-\frac{N}{2\pi}}$$

where

$$n = \sqrt{\sum_{i=1}^{D}(x_i-b_i)^2},\ N = \sqrt{D}\max_{i\in[1,D]}|x_i - b_i|$$

$\mathbf{b} = [1, 1.3, 0.8, -0.4, -1.3, 1.6, -2, -6, 0.5, 1.4, 1, 1.3, \ldots]$ and $\mathbf{x} \in [-15, 15]^D$.

28. Paviani problem [26]

$$f_{28}(\mathbf{x}) = \sum_{i=1}^{D}\left[(\ln(x_i-2))^2 + (\ln(10-x_i))^2\right] - \left(\prod_{i=1}^{D} x_i\right)^{0.2}$$

where $\mathbf{x} \in [2, 10]^D$.

29. Periodic problem [26]

$$f_{29}(\mathbf{x}) = 1 + \sum_{i=1}^{D}(\sin x_i)^2 - 0.1\prod_{i=1}^{D}\exp\left(-x_i^2\right)$$

where $\mathbf{x} \in [-10, 10]^D$ and $\min_{\mathbf{x}} f_{29}(x) = f_{29}([0, \ldots, 0]) = 0.9$.

30. Salomon problem [26]

$$f_{30}(\mathbf{x}) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{D} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{D} x_i^2}$$

where $\mathbf{x} \in [-100, 100]^D$ and $\min_{\mathbf{x}} f_{30}(\mathbf{x}) = f_{30}([0, \ldots, 0]) = 0$.

31. Shubert problem [26]

$$f_{31}(\mathbf{x}) = \prod_{i=1}^{D}\left(\sum_{j=1}^{5} j\cos((j+1)x_i + j)\right)$$

where $\mathbf{x} \in [-10, 10]^D$ and $\min_{\mathbf{x}} f_{31}(\mathbf{x}) \approx -186.7309$.

32. Sinusoidal problem [26]

$$f_{32}(\mathbf{x}) = -\left[A\prod_{i=1}^{D}\sin\left(\frac{(x_i-z)\pi}{180}\right) + \prod_{i=1}^{D}\sin\left(\frac{B(x_i-z)\pi}{180}\right)\right]$$

where $A = 2.5$, $B = 5$, $z = 30$, $\mathbf{x} \in [0, 180]^D$ and $\min_{\mathbf{x}} f_{32}(\mathbf{x}) = f_{32}([90+z, \ldots, 90+z]) = -(A+1)$.

33. Michalewicz function [1]

$$f_{33}(\mathbf{x}) = -\sum_{i=1}^{D}\left(\sin(x_i^2)\sin^{2m}\left(\frac{ix_i^2}{\pi}\right)\right)$$

where $m = 10$ and $\mathbf{x} \in [0, \pi]^D$.

34. Whitely's function [24]

$$f_{34}(\mathbf{x}) = \sum_{j=1}^{D}\sum_{i=1}^{D}\left(\frac{y_{i,j}}{4000} - \cos(y_{i,j}) + 1\right)$$

where $y_{i,j} = 100\left(x_j - x_i^2\right)^2 + (1-x_i)^2$ and $\mathbf{x} \in [-100, 100]^D$.

APPENDIX II

TABLE VIII

AVERAGE, STANDARD DEVIATION, AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY HdEA, RCGA-UNDX, CMA-ES, DE, ODE, DEAHCSPX, DPSO, SEPSO, AND EDA: f_1–f_4

Fitness Function		f_1		f_2		f_3		f_4	
D		30	40	30	40	30	40	30	40
HdEA	Average	**0.00**	**0.00**	**0.00**	0.0034	16920.23	34006.13	10.8802	22.6509
	Std. dev.	**(0.00)**	**(0.00)**	**(0.00)**	(0.0013)	(2818.22)	(4440.54)	(1.3212)	(1.8335)
RCGA-UNDX	Average	**0.00**	**0.00**	**0.00**	0.0005	2811.5869	27142.12	91.8556	99.9358
	Std. dev.	**(0.00)**	**(0.00)**	**(0.00)**	(0.0002)	(1668.95)	(7377.41)	(20.8748)	(0.352)
	C (t-test)	< **50%**	< **50%**	< **50%**	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	Average	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	Std. dev.	**(0.00)**	**(0.00)**	**(0.00)**	**(0.00)**	**(0.00)**	**(0.00)**	**(0.00)**	**(0.00)**
	C (t-test)	< **50%**	< **50%**	< **50%**	**99.95%**	**99.95%**	**99.95%**	**99.95%**	**99.95%**
DE	Average	0.0221	0.167	0.3594	1.1491	26073.04	45895.90	49.5082	60.7734
	Std. dev.	(0.0048)	(0.0279)	(0.0371)	(0.1101)	(3339.06)	(4761.74)	(4.0114)	(3.8266)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	Average	0.000006	0.001	0.0953	0.3459	78.1691	409.988	0.0111	0.1528
	Std. dev.	(0.00)	(0.0009)	(0.0486)	(0.1754)	(45.1045)	(230.1221)	(0.0808)	(0.4848)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
DEahcSPX	Average	0.1075	0.0071	0.0322	0.0011	65.9908	310.1678	15.5882	15.8102
	Std. dev.	(0.399)	(0.0389)	(0.1423)	(0.0045)	(65.222)	(161.2674)	(3.5293)	(3.6463)
	C (t-test)	99.50%	95%	97.50%	99.95%	99.95%	99.95%	99.95%	99.95%
DPSO	Average	3.3462	6.9563	8.8458	19.4899	1955.2153	3986.0922	10.539	13.2989
	Std. dev.	(0.9949)	(1.2491)	(1.1698)	(5.3605)	(24.5853)	(35.7403)	(1.3743)	(1.4465)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	95.00%	99.95%
SEPSO	Average	2.8527	6.8806	8.8184	21.4078	2984.5912	6057.7894	13.867	16.9939
	Std. dev.	(0.9324)	(1.5195)	(1.398)	(5.7894)	(29.6756)	(41.8452)	(1.7487)	(1.663)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
EDA	Average	3439.532	5203.446	22.252	34.532	3749.133	7174.429	21.142	25.884
	Std. dev.	(1221.21)	(1828.79)	(5.433797)	(7.02534)	(1294.74)	(3081.31)	(5.932983)	(6.132852)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

TABLE IX

AVERAGE, STANDARD DEVIATION, AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY HdEA, RCGA-UNDX, CMA-ES, DE, ODE, DEAHCSPX, DPSO, SEPSO, AND EDA: f_5–f_8

Fitness Function		f_5		f_6		f_7		f_8	
D		30	40	30	40	30	40	30	40
HdEA	Average	21.1276	93.6767	10.4615	16.1244	**0.00**	**0.0082**	**0.00**	**0.001**
	Std. dev.	(13.7111)	(27.2255)	(0.6028)	(0.8751)	**(0.00)**	**(0.0048)**	**(0.00)**	**(0.0026)**
RCGA-UNDX	Average	71.1275	78.3633	8.8836	13.2587	219.8173	340.1991	1.6946	11.3365
	Std. dev.	(67.2881)	(72.0327)	(0.4382)	(0.5691)	(12.6701)	(14.534)	(0.1151)	(1.5104)
	C (t-test)	99.95%	95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	Average	**0.00**	**0.00**	**0.2303**	**0.2659**	53.6481	76.1739	0.0014	0.0012
	Std. dev.	**(0.00)**	**(0.00)**	**(0.0893)**	**(0.0963)**	(14.1662)	(17.1046)	(0.0036)	(0.0031)
	C (t-test)	**99.95%**	**99.95%**	**99.95%**	**99.95%**	99.95%	99.95%	99.95%	< 50%
DE	Average	614.4588	2023.166	10.9846	16.3383	25.7105	57.3944	0.9931	1.1542
	Std. dev.	(112.1748)	(405.6226)	(0.5641)	(0.6334)	(2.65)	(5.3769)	(0.0301)	(0.0259)
	C (t-test)	99.95%	99.95%	99.95%	95.00%	99.95%	99.95%	99.95%	99.95%
ODE	Average	27.0344	42.2877	8.5199	12.4331	102.3277	170.7193	0.0258	0.2427
	Std. dev.	(0.7121)	(13.7948)	(0.4121)	(0.5279)	(37.9763)	(55.5588)	(0.0326)	(0.1488)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
DEahcSPX	Average	3160.5891	2287.669	8.7536	13.0129	40.807	189.1699	0.1613	0.039
	Std. dev.	(9387.60)	(9595.24)	(0.52)	(0.5081)	(26.7653)	(47.6562)	(0.2685)	(0.0604)
	C (t-test)	99.90%	97.50%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
DPSO	Average	12789.49	31765.02	9.9202	14.6241	125.4958	201.5571	4.0567	7.0049
	Std. dev.	(78.0953)	(110.4379)	(0.8192)	(0.8821)	(3.9027)	(4.6532)	(0.8773)	(1.1996)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	Average	10464.07	41826.01	10.6244	15.6301	112.9555	181.948	3.6995	7.433
	Std. dev.	(80.0175)	(141.3864)	(0.8755)	(1.0337)	(4.5789)	(4.8482)	(0.9511)	(1.2926)
	C (t-test)	99.95%	99.95%	90.00%	99.95%	99.95%	99.95%	99.95%	99.95%
EDA	Average	30214.733	61973.272	100.069	353.440	188.384	272.859	30.504	49.551
	Std. dev.	(12162.08)	(25541.42)	(52.002)	(142.018)	(20.555)	(29.016)	(10.987)	(17.855)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

TABLE X

AVERAGE, STANDARD DEVIATION, AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY HdEA, RCGA-UNDX, CMA-ES, DE, ODE, DEAHCSPX, DPSO, SEPSO AND EDA: f_9–f_{14}

Fitness Function		f_9		f_{10}		f_{11}	f_{12}	f_{13}	f_{14}
D		30	40	30	40	2	2	2	2
HdEA	Average	**−13780.72**	**−18374.62**	**0.00**	**0.005**	**1.2082**	**−1.0316**	0.401	4.4101
	Std. dev.	**(25.3953)**	**(18.2885)**	**(0.00)**	**(0.002)**	**(0.7333)**	**(0.0001)**	(0.0264)	(4.0821)
RCGA-UNDX	Average	−5941.04	−6580.023	20.7175	20.8336	6.5167	−0.6587	0.4596	56.0632
	Std. dev.	(486.7029)	(470.1085)	(0.0904)	(0.1001)	(3.1656)	(0.3117)	(0.0565)	(29.5616)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	Average	−5406.98	−7187.4	21.3439	21.495	13.5224	−1.0235	**0.3979**	7.32
	Std. dev.	(95.6)	(184.1)	(0.4316)	(0.1658)	(5.3565)	(0.0816)	**(0.00)**	(16.6041)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	80%	**80.00%**	95%
DE	Average	−12801.84	−15568.76	1.8269	3.047	13.9738	−0.6695	1.5219	14.8393
	Std. dev.	(156.2624)	(256.1296)	(0.3319)	(0.2579)	(21.6527)	(0.3211)	(1.348)	(10.4039)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	Average	−5475.53	−6559.977	0.0099	0.1181	2.4491	−1.0214	0.425	3.5207
	Std. dev.	(506.0389)	(839.0191)	(0.0117)	(0.1698)	(1.4992)	(0.0109)	(0.0277)	(0.4767)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	97.50%
DEahcSPX	Average	−10285.82	−9462.16	2.8201	4.6082	19.077	−0.4882	6.683	21.806
	Std. dev.	(692.3803)	(1390.58)	(4.0335)	(5.1296)	(61.8988)	(3.2941)	(19.2551)	(85.0168)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.50%	90%	99.90%	97.50%
DPSO	Average	−5185.645	−6196.144	6.0714	6.9424	1.491	−1.0229	1.4407	3.1429
	Std. dev.	(25.7171)	(27.7505)	(0.821)	(0.8013)	(0.84)	(0.1171)	(1.6188)	(0.6474)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.00%	75%	99.95%	99.75%
SEPSO	Average	−7702.762	−9108.41	6.4363	7.7494	2.2467	−1.0252	0.4096	**3.0618**
	Std. dev.	(27.2504)	(30.4254)	(1.0018)	(0.9357)	(1.1537)	(0.104)	(0.1366)	**(0.2934)**
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	$< 50\%$	$< 50\%$	**99.90%**
EDA	Average	−4679.949	−5510.891	10.177	11.035	2.821	−1.031	**0.398**	**3.000**
	Std. dev.	(703.073)	(742.236)	(1.294046)	(1.321336)	(1.73705)	(0.001204)	**(0)**	**(0.000002)**
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	**80%**	**99.95%**

TABLE XI

AVERAGE, STANDARD DEVIATION, AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY HdEA, RCGA-UNDX, CMA-ES, DE, ODE, DEAHCSPX, DPSO, SEPSO, AND EDA: f_{15}–f_{18}

Fitness Function		f_{15}		f_{16}		f_{17}		f_{18}	
D		30	40	30	40	30	40	30	40
HdEA	Average	**0.00**	**1.6503**	**0.0047**	**0.1451**	8814.832	14179.81	**0.00**	**0.00**
	Std. dev.	**(0.00)**	**(2.7983)**	**(0.0014)**	**(0.0286)**	(2547.73)	(4285.01)	**(0.00)**	**(0.00)**
RCGA-UNDX	Average	69184345	1.67E+8	30.1414	65.2566	27770.55	34420.03	0.3329	8.5853
	Std. dev.	(1.2E+7)	(2.8E+7)	(6.7271)	(2.2689)	(4571.35)	(6050.13)	(0.0954)	(1.8914)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	Average	41.4371	116566.8	7.1621	10.5753	6410.02	6698.98	0.0115	**0.00**
	Std. dev.	(100.7638)	(76286.01)	(12.9456)	(18.654)	(362.67)	(324.05)	(0.1153)	**(0.00)**
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	80%	**$< 50\%$**
DE	Average	4659.844	37580.67	1.485	3.3923	2335.616	2454.875	0.0032	0.0278
	Std. dev.	(926.189)	(6525.303)	(0.119)	(0.2111)	(83.3354)	(77.024)	(0.0008)	(0.0059)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	Average	1.1579	254.327	2.1745	4.2087	**1851.785**	**2003.177**	0.000015	0.0006
	Std. dev.	(1.2544)	(302.6105)	(0.4339)	(0.8853)	**(164.0192)**	**(163.5926)**	(0.00)	(0.0005)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	**99.95%**	**99.95%**	99.95%	99.95%
DEahcSPX	Average	38175.4	2085.96	0.7444	0.64	2637.816	2665.809	0.0598	0.0616
	Std. dev.	(200625.0)	(10954.24)	(0.6961)	(0.6938)	(113.3806)	(93.9438)	(0.1701)	(0.2015)
	C (t-test)	95%	95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.75%
DPSO	Average	8 264 438	15 086 215	13.0929	19.525	10 080 636	11 335 533	4.1547	7.0132
	Std. dev.	(1925.56)	(2359.02)	(1.2306)	(1.3333)	(1168.58)	(1009.98)	(1.6574)	(1.8273)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	Average	6 281 119	13 793 165	14.0028	20.9514	234 963	365663.1	3.6763	7.4596
	Std. dev.	(1550.51)	(2260.61)	(1.3528)	(1.5315)	(277.2721)	(398.9803)	(1.4536)	(1.886)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
EDA	Average	6751507.8	23 576 245	16.726	24.507	74236.525	2.599E+9	5.980	12.734
	Std. dev.	(4 130 691)	(1.19E+7)	(2.452092)	(3.438128)	(55975.6)	(2.5E+10)	(3.278065)	(6.62116)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	80%	99.95%	99.95%

TABLE XII

AVERAGE, STANDARD DEVIATION, AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY HdEA, RCGA-UNDX, CMA-ES, DE, ODE, DEAHCSPX, DPSO, SEPSO, AND EDA: f_{19}–f_{22}

Fitness Function		f_{19}		f_{20}		f_{21}		f_{22}	
D		30	40	30	40	30	40	30	40
HdEA	Average	261.3953	395.2277	**0.0004**	0.0057	4.8663	7.6184	−24.9443	−31.9794
	Std. dev.	(34.998)	(42.3205)	**(0.0002)**	(0.001)	(0.3451)	(0.399)	(0.9411)	(0.9206)
RCGA-UNDX	Average	293.5157	496.9959	10.2837	27.5203	7.61	11.0298	−6.633	−7.12
	Std. dev.	(36.933)	(45.4164)	(1.3909)	(1.9434)	(0.5734)	(0.7434)	(0.452)	(0.6458)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	Average	**0.00**	**0.00**	0.0025	0.0097	13.7823	18.9196	−0.9678	−1.0698
	Std. dev.	**(0.00)**	**(0.00)**	(0.0028)	(0.0079)	(0.2792)	(0.2544)	(0.732)	(0.7926)
	C (t-test)	**99.95%**	**99.95%**	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
DE	Average	274.0171	425.7634	0.1641	1.1909	5.3987	8.2231	−18.8816	−22.652
	Std. dev.	(30.0291)	(42.4568)	(0.0486)	(0.2507)	(0.5198)	(0.5673)	(0.556)	(0.7497)
	C (t-test)	99.50%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	Average	24.2815	164.5583	0.0299	0.0865	**0.0237**	**0.1127**	**−27.8856**	**−37.6543**
	Std. dev.	(8.0691)	(38.622)	(0.0099)	(0.0887)	**(0.1431)**	**(0.3907)**	**(1.8404)**	**(2.359)**
	C (t-test)	99.95%	99.95%	99.95%	99.95%	**99.95%**	**99.95%**	**99.95%**	**99.95%**
DEahcSPX	Average	2.2143	30.1464	0.0013	**0.0001**	4.6963	6.1197	−14.6745	−11.075
	Std. dev.	(3.8524)	(13.0009)	(0.0078)	**(0.0003)**	(1.3226)	(2.1082)	(4.0927)	(1.9585)
	C (t-test)	99.95%	99.95%	80.00%	**99.95%**	80.00%	99.95%	99.95%	99.95%
DPSO	Average	134.6547	278.6536	5.758	9.024	11.8132	16.3968	−15.4114	−18.3188
	Std. dev.	(7.912)	(10.7637)	(1.3801)	(1.5181)	(0.521)	(0.5582)	(1.2455)	(1.382)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	Average	71.7805	155.882	9.5052	14.9113	12.0147	16.7829	−16.9436	−20.5577
	Std. dev.	(4.8726)	(6.205)	(1.7957)	(1.9475)	(0.532)	(0.5337)	(1.2194)	(1.3924)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
EDA	Average	69.740	94.707	12.235	20.463	12.309	17.025	−18.728	−22.469
	Std. dev.	(29.0711)	(32.8311)	(3.110696)	(4.22087)	(0.17794)	(0.212107)	(4.185106)	(4.879363)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

TABLE XIII

AVERAGE, STANDARD DEVIATION, AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY HdEA, RCGA-UNDX, CMA-ES, DE, ODE, DEAHCSPX, DPSO, SEPSO, AND EDA: f_{23}–f_{26}

Fitness Function		f_{23}		f_{24}		f_{25}		f_{26}	
D		30	40	30	40	30	40	30	40
HdEA	Average	**0.00**	**0.00**	**−25.3678**	**−32.2103**	0.1626	2.4995	8025.425	73996.06
	Std. dev.	**(0.00)**	**(0.00)**	**(0.572)**	**(0.6295)**	(0.4616)	(4.3733)	(4773.20)	(24998.18)
RCGA-UNDX	Average	0.3566	2.4797	−8.8451	−10.32	152.9672	566.6314	62837.03	296115.4
	Std. dev.	(0.073)	(0.1908)	(0.5691)	(0.5004)	(17.6976)	(81.3621)	(3012.70)	(17576.71)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	Average	0.4493	0.6355	−19.1834	−24.0149	**0.023**	**0.0219**	**−2428.19**	**−413.708**
	Std. dev.	(0.258)	(0.2534)	(1.8797)	(1.8759)	**(0.0472)**	**(0.0465)**	**(0.00)**	**(0.00)**
	C (t-test)	99.95%	99.95%	99.95%	99.95%	**99.75%**	**99.95%**	**99.95%**	**99.95%**
DE	Average	0.0027	0.0189	−18.3183	−22.3111	60.0966	131.245	122598.2	886175.2
	Std. dev.	(0.0006)	(0.0031)	(0.6445)	(0.6373)	(10.9953)	(10.733)	(25422.81)	(139048.9)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	Average	0.000027	0.000962	−12.5543	−14.4373	26.0994	47.3521	−4930	11440
	Std. dev.	(0.00)	(0.0008)	(1.2739)	(1.3917)	(12.9456)	(15.3637)	(1162.05)	(3206.49)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
DEahcSPX	Average	0.1752	0.1894	−12.9365	−13.2637	37.1675	52.3789	1911.297	8562.511
	Std. dev.	(0.1499)	(0.1578)	(2.0401)	(1.3542)	(17.6322)	(19.5292)	(4085.18)	(9956.65)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
DPSO	Average	0.6795	1.1519	−10.3292	−12.0257	135.0221	245.9962	26342.66	154356.6
	Std. dev.	(0.4754)	(0.5166)	(0.8904)	(0.9585)	(6.6864)	(8.3421)	(86.3998)	(205.9114)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	Average	0.7684	1.3428	−10.9954	−12.5813	152.5561	259.8431	30572.09	165457.5
	Std. dev.	(0.5383)	(0.5792)	(0.9763)	(1.0432)	(7.1358)	(8.0873)	(100.8033)	(229.2899)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
EDA	Average	1.885	2.979	−9.361	−11.071	10.527	18.692	141156.77	835826.06
	Std. dev.	(0.444738)	(0.553565)	(0.75983)	(0.888224)	(4.54527)	(5.99804)	(65517.19)	(326503.5)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

TABLE XIV

AVERAGE, STANDARD DEVIATION, AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY HdEA, RCGA-UNDX, CMA-ES, DE, ODE, DEAHCSPX, DPSO, SEPSO, AND EDA: f_{27}–f_{30}

Fitness function		f_{27}		f_{28}		f_{29}		f_{30}	
D		30	40	30	40	30	40	30	40
HdEA	Average	−0.0004	−0.0001	**−997867**	**−1E+8**	1.0004	1.0046	1.2051	2.1347
	Std. dev.	(0.001)	(0.0006)	**(0.0271)**	**(0.00)**	(0.0002)	(0.0012)	(0.1365)	(0.2002)
RCGA-UNDX	Average	−0.0001	0.00	−330 341	−509 252	6.9638	9.9936	2.3127	5.0289
	Std. dev.	(0.00)	(0.00)	(0.00)	(0.00)	(0.3607)	(0.4529)	(0.1817)	(0.397)
	C (t-test)	99.75%	95.00%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	Average	−0.0107	−0.0032	−951 804	−9.8E+7	8.142	13.0446	1.1979	1.4629
	Std. dev.	(0.0195)	(0.0064)	(187570.7)	(4 361 415)	(5.7645)	(7.2567)	(0.247)	(0.2635)
	C (t-test)	99.95%	99.95%	99%	99.95%	99.95%	99.95%	$<$ 50%	99.95%
DE	Average	−0.0001	0.00	−958 473	−8.8E+07	1.4523	2.1431	3.6819	5.476
	Std. dev.	(0.0001)	(0.00)	(6695.09)	(1 605 337)	(0.076)	(0.1473)	(0.2572)	(0.3649)
	C (t-test)	99.75%	95.00%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	Average	**−0.8233**	**−0.7227**	−610 112	−3.8E+07	**0.9107**	**0.9137**	**0.4718**	0.9017
	Std. dev.	**(0.058)**	**(0.1171)**	(37311.6)	(3 450 073)	**(0.0418)**	**(0.0581)**	**(0.1056)**	(0.1632)
	C (t-test)	**99.95%**	**99.95%**	99.95%	99.95%	**99.95%**	**99.95%**	**99.95%**	99.95%
DEahcSPX	Average	−0.0416	−0.0265	−996 116	−9.5E+07	2.4177	5.8216	0.4953	**0.5465**
	Std. dev.	(0.0311)	(0.0235)	(5578.90)	(1 488 854)	(0.8642)	(1.16)	(0.1756)	**(0.1399)**
	C (t-test)	99.95%	99.95%	99.75%	99.95%	99.95%	99.95%	99.95%	**99.95%**
DPSO	Average	−0.051	−0.0192	−342 933	−1.7E+07	4.8472	7.7492	2.9157	3.8791
	Std. dev.	(0.1507)	(0.094)	(217.3297)	(2125.222)	(0.8796)	(0.9589)	(0.6078)	(0.629)
	C (t-test)	99.90%	98%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	Average	−0.0272	−0.0101	−965 431	−8.8E+07	3.3681	5.5601	3.3948	4.6228
	Std. dev.	(0.1234)	(0.0912)	(24.4007)	(3225.70)	(0.7988)	(0.9266)	(0.7648)	(0.7211)
	C (t-test)	97.50%	80%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
EDA	Average	−0.060	−0.018	−286765.1	−14 874 856	7.629	11.228	5.186	6.780
	Std. dev.	(0.04363)	(0.014031)	(36881.35)	(2 468 007)	(0.544293)	(0.552775)	(1.072429)	(1.362282)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

TABLE XV

AVERAGE, STANDARD DEVIATION, AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY HdEA, RCGA-UNDX, CMA-ES, DE, ODE, DEAHCSPX, DPSO, SEPSO, AND EDA: f_{31}–f_{34}

Fitness Function		f_{31}		f_{32}		f_{33}		f_{34}	
D		30	40	30	40	30	40	30	40
HdEA	Average	**−2E+34**	**−2.7E+45**	−1.521	−1.3187	**−29.559**	**−38.8299**	**20.6012**	694.993
	Std. dev.	**(3.6E+33)**	**(6.7E+44)**	(0.6748)	(0.5164)	**(0.0289)**	**(0.1222)**	**(28.8726)**	(145.0827)
RCGA-UNDX	Average	−1.3E+20	−3.7E+25	−3.3678	−2.4558	−10.2002	−12.0481	7440.466	722794.5
	Std. dev.	(2.1E+20)	(9.9E+25)	(0.0231)	(0.0556)	(0.5695)	(0.5948)	(2193.41)	(204048.1)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	Average	−1.1E+29	−1.5E+36	−2.6016	−2.52	−19.1408	−24.5539	319.3721	**541.3366**
	Std. dev.	(4.4E+29)	(8.1E+36)	(1.5276)	(1.5794)	(2.0299)	(2.7287)	(102.4076)	**(178.4243)**
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	**99.95%**
DE	Average	−9E+29	−1.6E+38	−2.1663	−1.5323	−24.8678	−30.0203	830.2062	1765.99
	Std. dev.	(1.6E+30)	(3.5E+38)	(0.1757)	(0.237)	(0.418)	(0.5516)	(15.6812)	(56.5738)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	Average	−1.2E+24	−3.7E+31	**−3.5**	**−3.4892**	−14.7206	−17.6109	766.9481	1438.332
	Std. dev.	(7.1E+24)	(2.9E+32)	**(0.00)**	**(0.0412)**	(1.0599)	(1.357)	(22.6609)	(26.614)
	C (t-test)	99.95%	99.95%	**99.95%**	**99.95%**	99.95%	99.95%	99.95%	99.95%
DEahcSPX	Average	−1.9E+24	−3.1E+29	−3.3078	−3.479	−16.9775	−15.5823	9536.837	2341.138
	Std. dev.	(1.2E+25)	(3.0E+30)	(0.3708)	(0.1145)	(2.3975)	(1.7578)	(38054.29)	(4702.41)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99%	99.95%
DPSO	Average	−2.7E+24	−9.6E+30	−1.8368	−1.257	−13.6114	−15.8861	11716.56	45903.7
	Std. dev.	(4.1E+12)	(8.0E+15)	(0.7575)	(0.7312)	(0.965)	(1.0123)	(77.5304)	(147.5638)
	C (t-test)	99.95%	99.95%	99.75%	75.00%	99.95%	99.95%	99.95%	99.95%
SEPSO	Average	−2.1E+25	−6.2E+31	−2.3102	−1.8404	−14.0837	−16.6464	11007.86	56578.07
	Std. dev.	(8.6E+12)	(1.8E+16)	(0.8684)	(0.8695)	(1.1034)	(1.1435)	(81.4433)	(176.2171)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
EDA	Average	−1.24E+22	−3.01E+29	−1.222	−0.548	−10.781	−12.962	985056.31	2262390.1
	Std. dev.	(8.9E+22)	(2.4E+30)	(0.287902)	(0.195443)	(0.708519)	(1.195155)	(769131.6)	(1 513 021)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

APPENDIX III

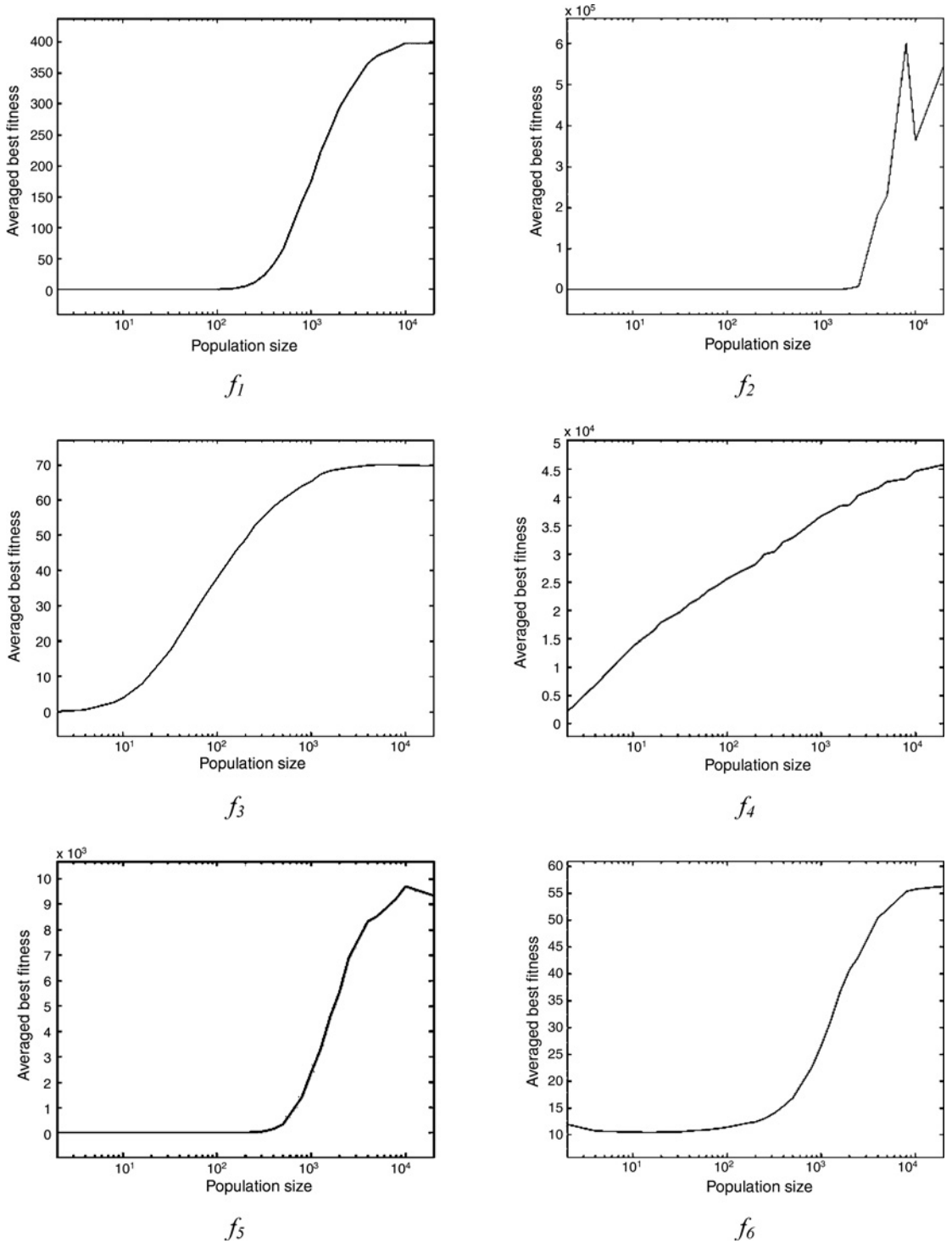


Fig. 10. Averaged best fitness values against population size for the test functions f_1–f_{34}.

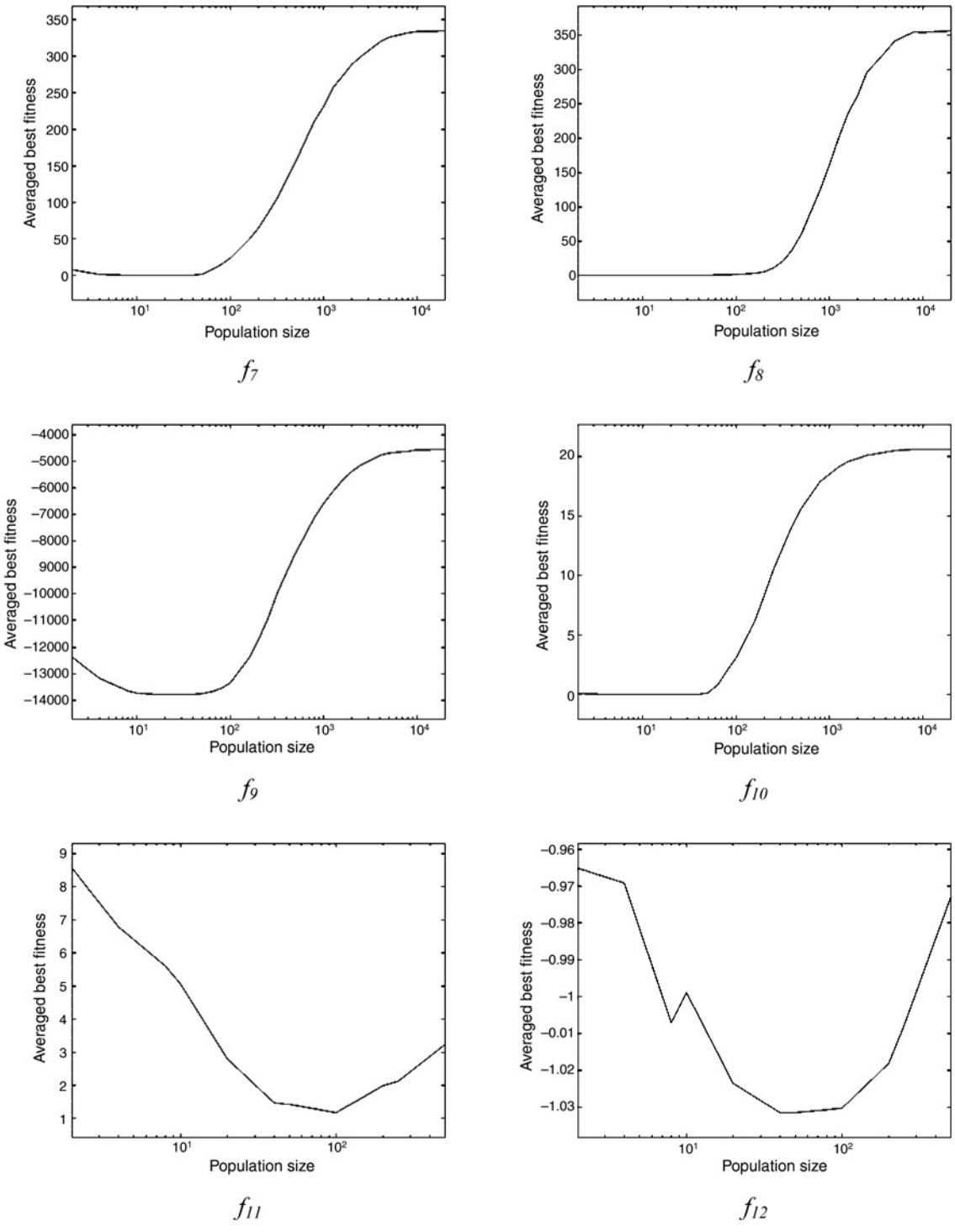


Fig. 10 (*Continued*)

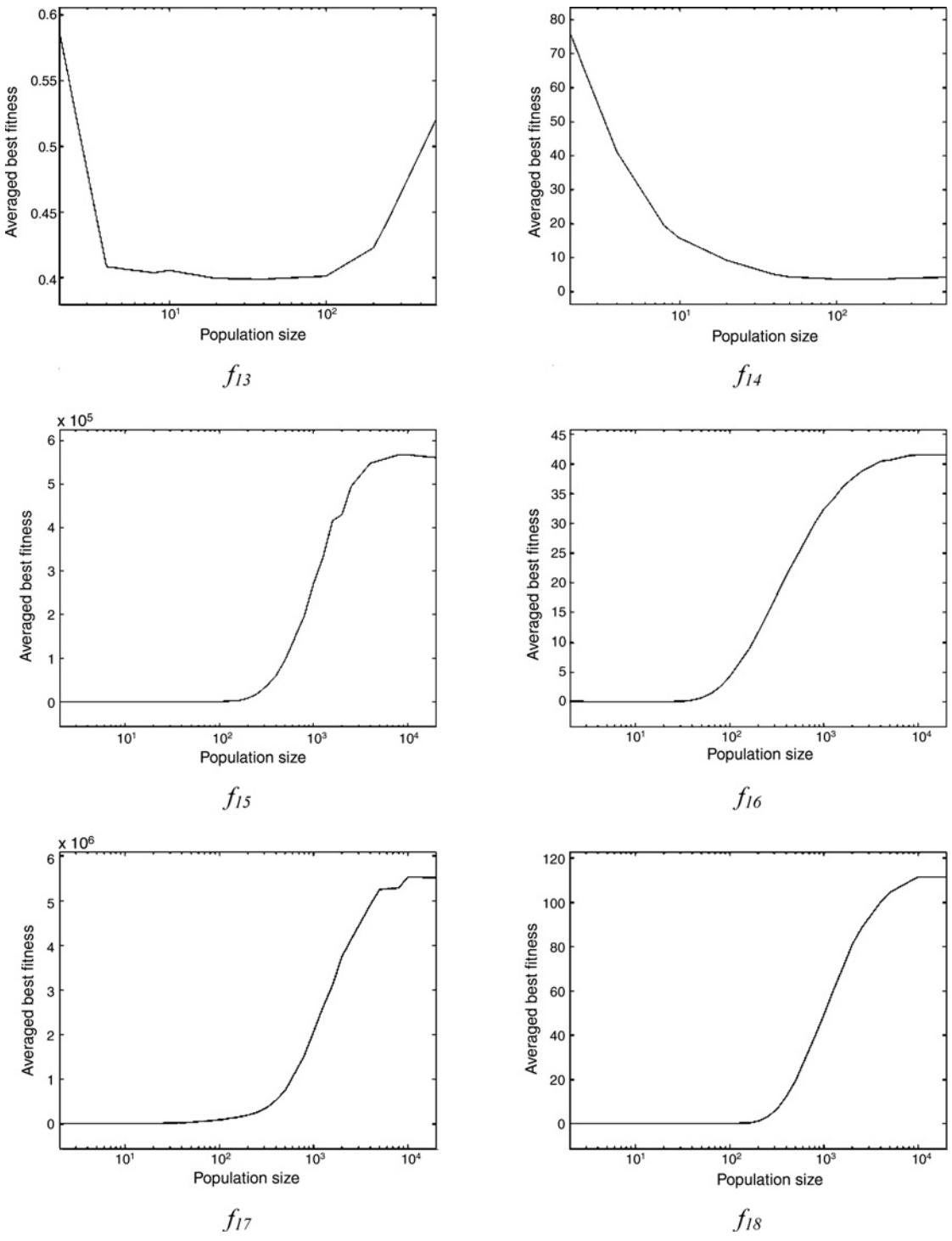


Fig. 10 (*Continued*)

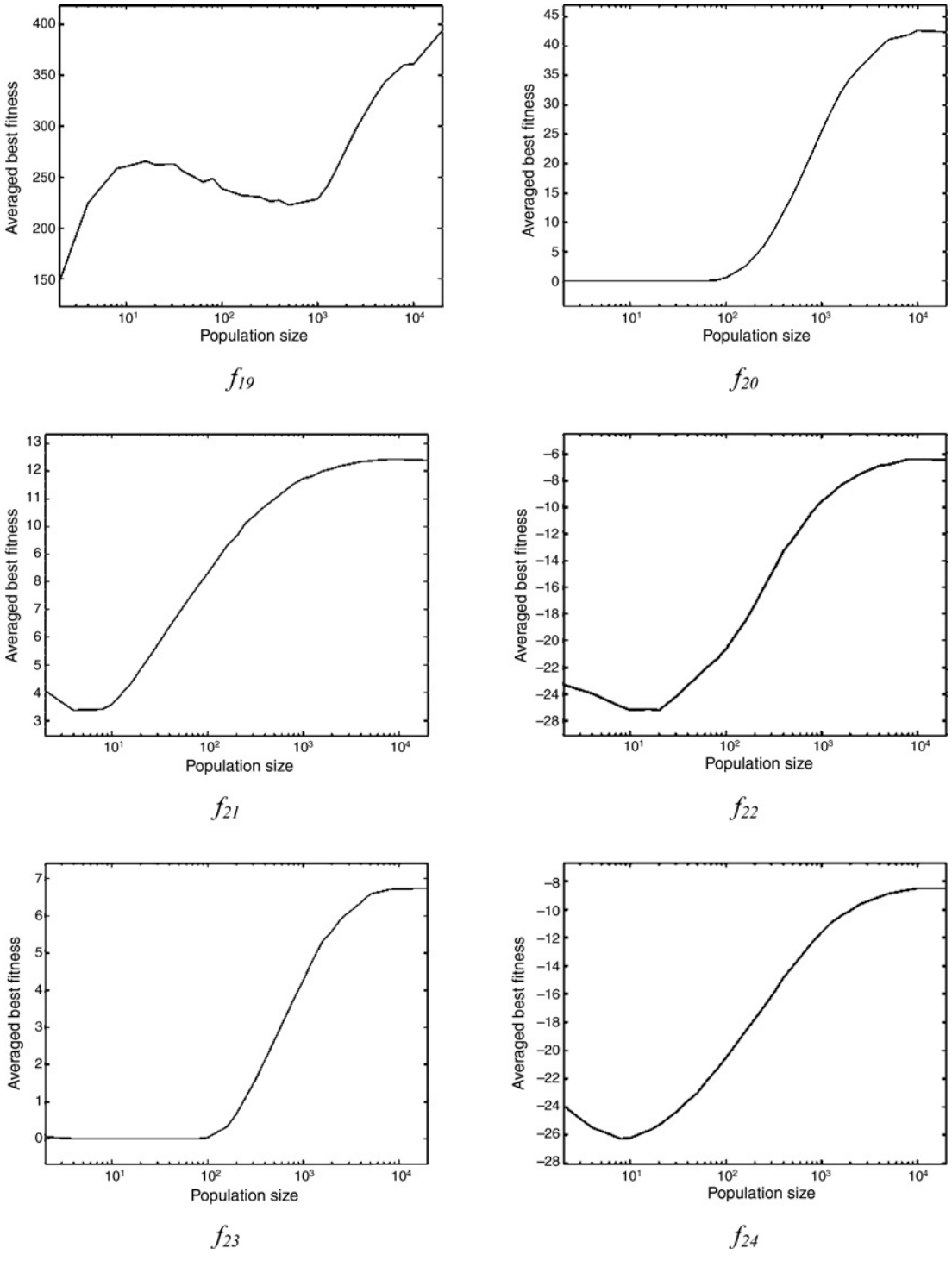


Fig. 10 (*Continued*)

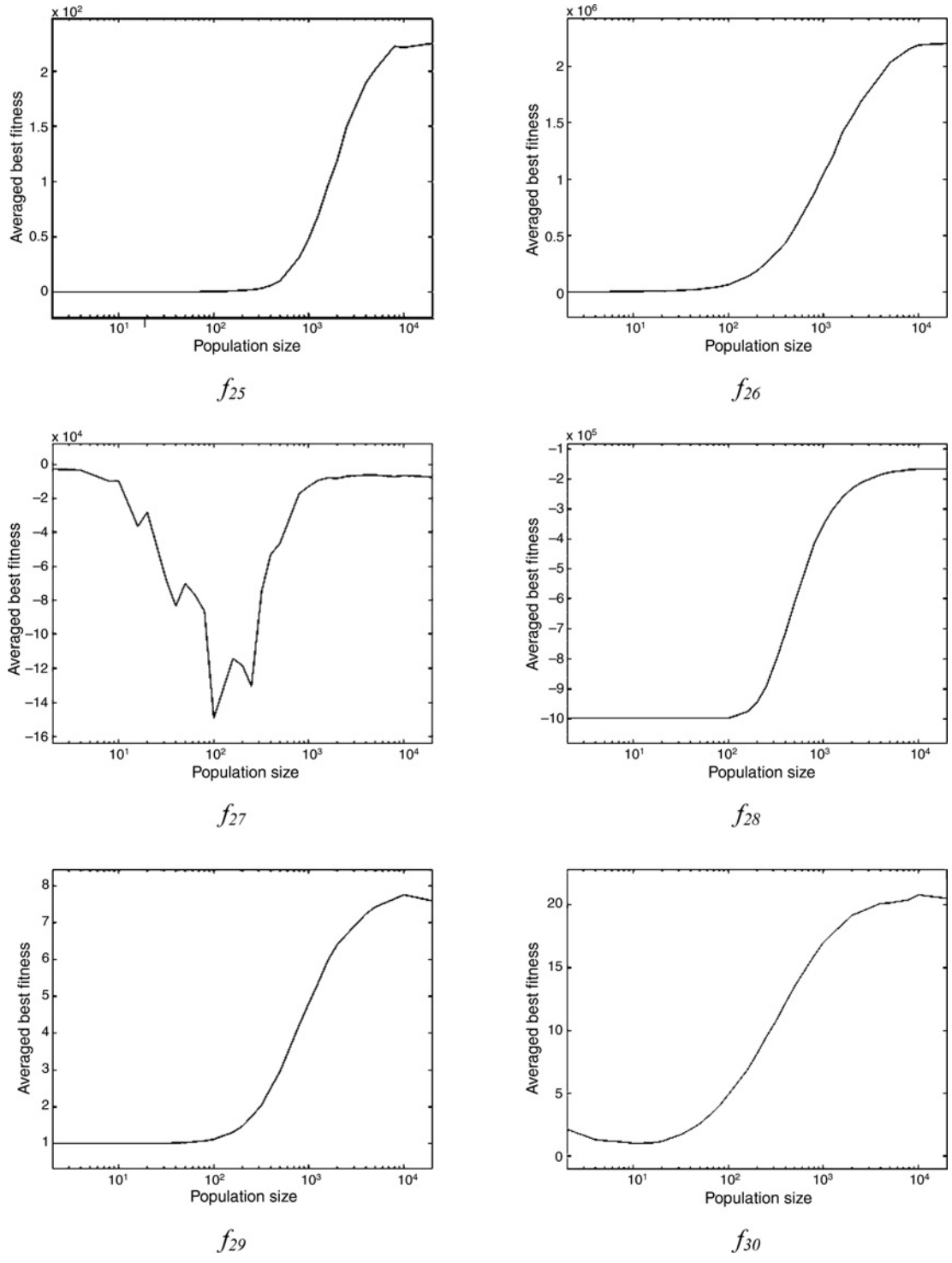


Fig. 10 (*Continued*)

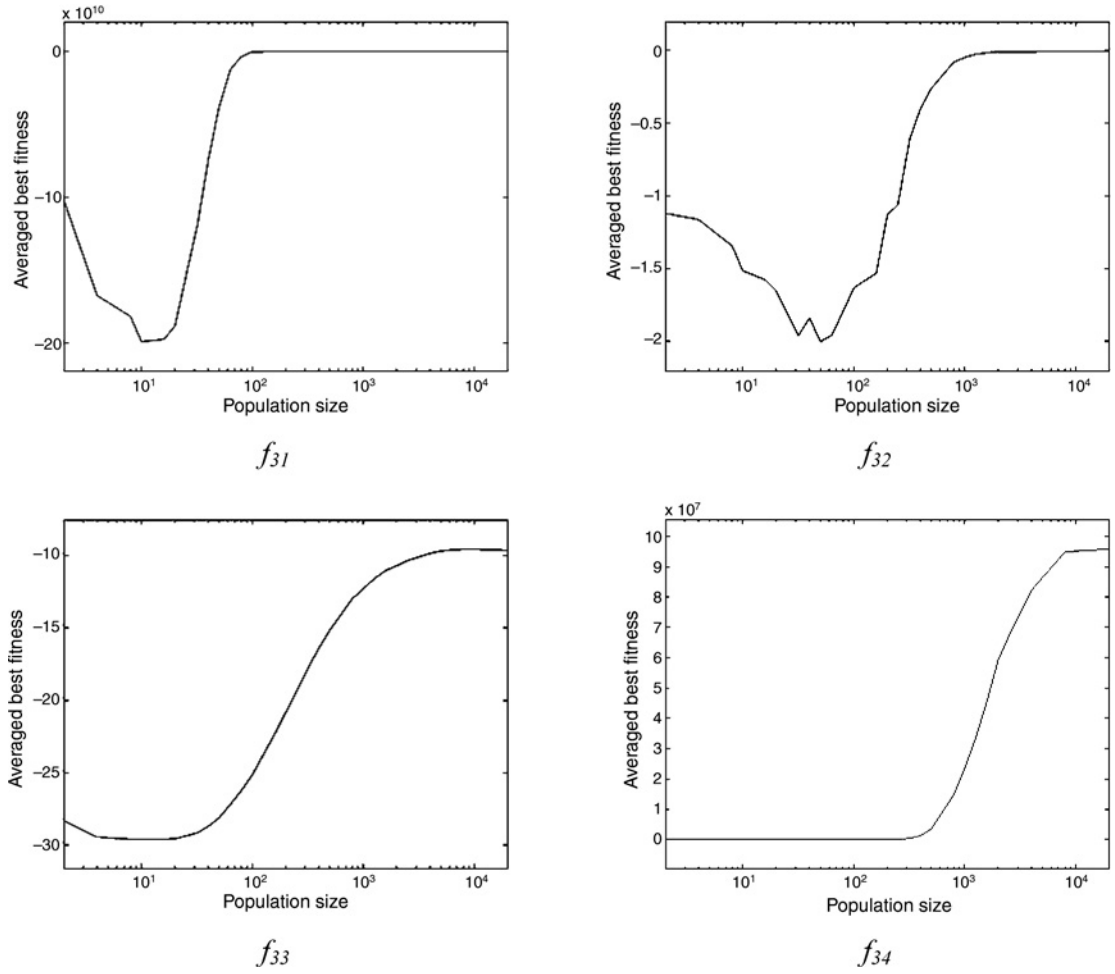


Fig. 10 (*Continued*)

ACKNOWLEDGMENT

The authors would like to thank S. W. Leung for proofreading the manuscript.

REFERENCES

[1] V. K. Koumousis and C. P. Katsaras, "A sawtooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance," *IEEE Trans. Evol. Comput.*, vol. 10, no. 1, pp. 19–28, Feb. 2006.

[2] X. F. Xie, W. J. Zhang, and Z. L. Yang, "A dissipative particle swarm optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2002, pp. 1666–1670.

[3] L. Pedro and J.-A. Lozano, Eds. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Boston, MA: Kluwer, 2002.

[4] R. Storn and K. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," Berkeley, CA, Tech. Rep. TR-95-012, 1995.

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

[6] S. Y. Yuen and C. K. Chow, "A non-revisiting genetic algorithm," in *Proc. IEEE Congr. Evol. Comput.*, 2007, pp. 4583–4590.

[7] S. Y. Yuen and C. K. Chow, "A genetic algorithm that adaptively mutates and never revisits," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 454–472, Apr. 2009.

[8] S. Y. Yuen and C. K. Chow, "A non-revisiting simulated annealing algorithm," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 1886–1892.

[9] C. K. Chow and S. Y. Yuen, "A non-revisiting particle swarm optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 1879–1885.

[10] F. Glover and M. Laguna, *Tabu Search*. Norwell, MA: Kluwer, 1997.

[11] J.-F. M. Barthelemy and R. T. Haftka, "Approximation concepts for optimum structural design: A review," *Struct. Optimiz.*, vol. 5, no. 3, pp. 129–144, Sep. 1993.

[12] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Comput. J.*, vol. 9, no. 1, pp. 3–12, 2005.

[13] D. Büche, N. N. Schraudolph, and P. Koumoutsakos, "Accelerating evolutionary algorithms with Gaussian process fitness function models," *IEEE Trans. Evol. Comput.*, vol. 35, no. 2, pp. 183–194, May 2005.

[14] R. G. Regis and C. A. Shoemaker, "Local function approximation in evolutionary algorithms for the optimization of costly functions," *IEEE Trans. Evol. Comput.*, vol. 8, no. 5, pp. 490–505, Oct. 2004.

[15] F. V. Jensen, *Bayesian Networks and Decision Graphs*. Berlin, Germany: Springer, 2001.

[16] J.-J. Liou and Y.-P. Chen, "Adaptive discretization on multidimensional continuous search spaces," in *Proc. Genetic Evol. Comput. Conf.*, 2008, pp. 977–984.

[17] G. B. Huang, Q. Y. Zhu, and C. K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 2. 2004, pp. 985–990.

[18] Y. S. Ong, P. B. Nair, and A. J. Keane, "Evolutionary optimization of computationally expensive problems via surrogate modeling," *Am. Instit. Aeronautics Astronautics J.*, vol. 41, no. 4, pp. 687–696, 2003.

[19] D. Lim, Y. S. Ong, Y. Jin, and B. Sendhoff, "A study on metamodeling techniques, ensembles, and multisurrogates in evolutionary computation," in *Proc. Genetic Evol. Comput. Conf.*, London, U.K., 2007, pp. 1288–1295.

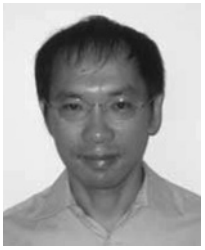
[20] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[21] S. Y. Yuen and C. K. Chow, "Applying non-revisiting genetic algorithm to traveling salesman problem," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 2217–2224.

[22] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Toward memetic algorithms," Caltech, U.S., Caltech Concurrent Computation Program Rep. 826, 1989.
[23] R. Meuth, M. H. Lim, Y. S. Ong, and D. C. Wunsch, II, "A proposition on memes and meta-memes in computing for higher-order learning," *Memetic Comput.*, vol. 1, no. 2, pp. 85–100, 2009.
[24] X. Yao, Y. Liu, and G. M. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Jul. 1999.
[25] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Nanyang Technol. Univ., Singapore, 2005, Tech. Rep., and IIT Kanpur, Kanpur, India, KanGAL Rep. #2005005, 2005.
[26] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky, "A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems," *J. Global Optimiz.*, vol. 31, no. 4, pp. 635–672, 2005.
[27] I. Ono and S. Kobayashi, "A real-code genetic algorithm for function optimization using unimodal normal distribution crossover," in *Proc. 7th Int. Conf. Genetic Algorithm*, 2007, pp. 246–253.
[28] N. Hansen, "The CMA evolutionary strategy: A tutorial," Tech. Rep., Aug. 31, 2007 [Online]. Available: www.bionik.tu-berlin.de/user/niko/cmatutorial.pdf
[29] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition-based differential evolution," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 64–79, Feb. 2008.
[30] H. R. Tizhoosh, "Opposition-based learning: A new scheme for machine intelligence," in *Proc. Int. Conf. Comput. Intell. Modeling Control Autom.*, vol. 1. Vienna, Austria, 2005, pp. 695–701.
[31] N. Noman and H. Iba, "Accelerating differential evolution using an adaptive local search," *IEEE Trans. Evol. Comput.*, vol. 12, no. 1, pp. 107–125, Feb. 2008.
[32] S. Tsutsui, M. Yamamura, and T. Higuchi, "Multiparent recombination with simplex crossover in real coded genetic algorithms," in *Proc. Genetic Evol. Comput. Conf.*, 1999, pp. 657–664.
[33] T. Krink, J. S. Vesterstrom, and J. Riget, "Particle swarm optimization with spatial particle extension," in *Proc. IEEE Congr. Evol. Comput.*, 2002, pp. 1474–1497.
[34] R. Santana, C. Echegoyen, A. Mendiburu, C. Bielza, J. A. Lozano, P. Larrañaga, R. Armañanzas, and S. Shakya, *MATLAB Toolbox for Estimation of Distribution Algorithms (MATEDA-2.0)*, Feb. 2009 [Online]. Available: http://www.sc.ehu.es/ccwbayes/members/rsantana/software/matlab/IntEDA.tar.gz
[35] R. Santana, C. Echegoyen, A. Mendiburu, C. Bielza, J. A. Lozano, P. Larrañaga, R. Armañanzas, and S. Shakya, "MATEDA: A suite of EDA programs in MATLAB," Univ. Basque Country, Bilbao, Spain, Tech. Rep. EHU-KZAA-IK-2/09, Feb. 2009.
[36] J. Vesterstroem and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proc. IEEE Congr. Evol. Comput.*, vol. 2. 2004, pp. 1980–1987.
[37] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Comput. A Fusion Found. Methodol. Applicat.*, vol. 9, no. 6, pp. 448–462, 2005.
[38] 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 Trans. Evol. Comput.*, vol. 10, no. 6, pp. 646–657, Dec. 2006.
[39] M. M. Ali and A. Törn, "Population set-based global optimization algorithms: Some modifications and numerical studies," *Comput. Oper. Res.*, vol. 31, no. 10, pp. 1703–1725, 2004.
[40] J. Sun, Q. Zhang, and E. P. K. Tsang, "DE/EDA: A new evolutionary algorithm for global optimization," *Inform. Sci.*, vol. 169, nos. 3–4, pp. 249–262, Feb. 2005.
[41] G. C. Onwubolu and B. V. Babu, *New Optimization Techniques in Engineering*. New York: Springer, 2004.
[42] C. K. Chow, H. T. Tsui, and T. Lee, "Surface registration using a dynamic genetic algorithm," *Pattern Recognit.*, vol. 37, no. 1, pp. 105–117, 2004.
[43] K. F. Fong, V. I. Hanby, and T. T. Chow, "HVAC system optimization for energy management by evolutionary programming," *Energy Buildings*, vol. 38, no. 3, pp. 220–231, 2006.
[44] K. F. Fong, V. I. Hanby, and T. T. Chow, "Development of optimal design of solar water heating system by using evolutionary algorithm," *J. Solar Energy Eng.*, vol. 129, no. 4, pp. 499–501, 2007.
[45] K. F. Fong, "Optimized design and energy management of heating, ventilating and air conditioning systems by evolutionary algorithm," Ph.D. thesis, De Montfort University, Leicester, U.K., 2006.

Chi Kin Chow received the B.E., M.Phil., and Ph.D. degrees from the Department of Electronic Engineering, Chinese University of Hong Kong, Hong Kong, China, in 1999, 2001, and 2005, respectively.

He was a Research Assistant and a Senior Research Assistant with the Department of Electronic Engineering, City University of Hong Kong, Kowloon Tong, Hong Kong, China, in 2005 and 2006, respectively. Since 2007, he has been a Research Fellow with the same department. His current research interests include neural networks, machine learning, evolutionary computation, image processing, and multiagent systems.



Shiu Yin Yuen (M'98) received the Associateship and the M.Phil. degrees from the Department of Electronic Engineering, Hong Kong Polytechnic (now Hong Kong Polytechnic University), in 1985 and 1988, respectively, and the D.Phil. degree from the School of Cognitive and Computing Sciences, University of Sussex, Sussex, U.K., in 1992.

He is currently an Associate Professor with the Department of Electronic Engineering, City University of Hong Kong, Kowloon Tong, Hong Kong, China. He has published more than 60 technical papers on these subjects, including more than 15 international journal papers. His main research interests include evolutionary computation, computer vision, and machine learning.

Dr. Yuen served as a Program Committee Member in 11 international conferences and is a Reviewer of many journals and conferences. His name is listed in Who's Who in the World and Who's Who in Science and Engineering.