A Genetic Algorithm That Adaptively Mutates and Never Revisits

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

***Abstract*—A novel genetic algorithm is reported that is non-revisiting: It remembers every position that it has searched before. An archive is used to store all the solutions that have been explored before. Different from other memory schemes in the literature, a novel binary space partitioning tree archive design is advocated. Not only is the design an efficient method to check for revisits, if any, it in itself constitutes a novel adaptive mutation operator that has no parameter. To demonstrate the power of the method, the algorithm is evaluated using 19 famous benchmark functions. The results are as follows. 1) Though it only uses finite resolution grids, when compared with a canonical genetic algorithm, a generic real-coded genetic algorithm, a canonical genetic algorithm with simple diversity mechanism, and three particle swarm optimization algorithms, it shows a significant improvement. 2) The new algorithm also shows superior performance compared to Covariance Matrix Adaptation Evolution Strategy (CMA-ES), a state-of-the-art method for adaptive mutation. 3) It can work with problems that have large search spaces with dimensions as high as 40. 4) The corresponding CPU overhead of the binary space partitioning tree design is insignificant for applications with expensive or time-consuming fitness evaluations, and for such applications, the memory usage due to the archive is acceptable. 5) Though the adaptive mutation is parameter-less, it shows and maintains a stable good performance. However, for other algorithms we compare, the performance is highly dependent on suitable parameter settings.**

***Index Terms*—Adaptive mutation, binary space partitioning, diversity maintenance, genetic algorithm, no revisits, premature convergence.**

I. Introduction

MANY stochastic algorithms (e.g., genetic algorithm, simulated annealing, etc) do not *memorize* places that they have visited. A notable exception is Tabu search [1], which incorporates the memory of recent search results to guide the next search step. However, even with Tabu search, not the entire list of visited positions is recorded. Thus revisits are inevitable for existing methods. A *revisit* to a search position s is defined as a reevaluation of the function (or fitness) of s which has been evaluated before. Since in many real world applications, function evaluation is the most computational intensive process within the stochastic algorithm, such revisits are clearly a waste of computational resources. In the terminology of no-free-lunch (NFL) theorems [2], all non-revisiting (stochastic or deterministic) algorithms have the same average performance when the problem distribution is uniform. A revisiting algorithm A searches the same sequence of distinct points as a non-revisiting algorithm A' when revisited points in the sequence are taken out. This implies that A' is superior to A as A' has the same performance but with fewer function evaluations. Thus it is always beneficial to eliminate all the revisits.

Genetic algorithm (GA) is a famous stochastic optimization algorithm pioneered by Holland [3]. It mimics the evolutionary process of a population of individuals over time. The hallmark of GA is the provision of a population (multiple parallel search capability), selection (survival of the fittest), crossover (sexual reproduction), and mutation (random incremental changes). GA is closely related to evolutionary strategies (ES) and evolutionary programming (EP). In GA, crossover is a key operator while mutation is usually a background operator. Traditional GA uses binary coded strings called chromosomes, but other codings, such as gray code, integer code, and real code are also in practice. Though mutation is usually a background operator, many authors advocate that an adaptive mutation schedule would enhance GA performance significantly, see, for example, survey [4] and a recent book [5]. In ES and EP, self adaptive mutation is the key operator, and traditional ES and EP are real coded [6]. However, the binary coded forms of GA allow it to be applied to both discrete/combinatorial and continuous/real problems. The schools of GA, ES, and EP are best viewed as different dialects of the same fundamental evolutionary algorithm (EA) [6, p. 2].

It is confirmed by numerous empirical studies that *duplicate removal* can enhance the performance of GA significantly. Mauldin's [7] uniqueness operator only allows a new child to be inserted into the population if its Hamming distance to all members of the population is greater than a threshold. Davis [8] reports that, for a comparable number of child evaluations, a binary coded GA that removes duplicates in the population has superior performance. Recently, Friedrich *et al.* [9] analyze an EA with a population greater than 1 using uniform bit mutation but no crossover. He shows that the above duplicate removal method changes the time complexity of optimizing a plateau function from exponential to polynomial. Crossover is omitted in the above work because it is a difficult operator to analyze. This result provides some theoretical support to the power of duplicate removal. The above papers compare each child with each solution in the current population. For a population with δ individuals, δ comparisons need to be made. To further enhance the efficiency, Ronald [10] reports the use of Hash table to reduce the number of comparisons to $O(1)$. However, these efforts only compare a child with the current population. It does not guarantee non-revists, i.e., no duplicates in the entire

Manuscript received December 24, 2007; revised April 03, 2008 and June 13, 2008. First published December 09, 2008; current version published April 01, 2009. This work was supported by City University of Hong Kong under Grant 7001859.

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

Digital Object Identifier 10.1109/TEVC.2008.2003008

search. Povinelli and Feng [11] use a small hash table to store all visited individuals. When the table is full, it is thrown away and a larger table is used. Kratica [12] uses a small fixed size cache to store all visited individuals. When it is full, an old entry is thrown away to make place for a new entry using the least-recently-used strategy. He investigates experimentally the best cache size for a plant location problem. These two papers extend the use of memory to beyond one generation. They report a substantial improvement to GA by adding the hash table or cache, but they do not store all the individuals and thus do not guarantee non-revists, nor do they try to organize the knowledge of the visited positions to guide the search to promising regions.

From another point of view, *premature convergence* is an undesirable phenomenon often reported in the literature as to cause poor performance of GA. It refers to the state when all of the population consists of a single suboptimal individual. It is generally agreed by the GA community that to prevent premature convergence, an appropriate diversity in the population has to be maintained. The reason is that once the entire population converges to a single kind of individual, crossover will be useless and GA reduces to parallel mutation climbing. Numerous operators that modify the selection, crossover, or mutation to diversify the population have been proposed. Two well-known examples are fitness sharing [13] and island model [14].

From yet another viewpoint, it is known from studies [4], [5] that an adaptive mutation rate is beneficial for GA/ES/EP to find the global optimum more efficiently. Three types of parameter controls are distinguished in [4]. A time evolution equation is used in deterministic control; feedback from the search process is used in adaptive control; the parameters itself are coded as part of the solutions and evolve in self adaptive control. However, all inevitably involve introducing more control parameters, for which the determination of suitable values requires parameter tuning. Such tuning is notoriously difficult. A state-of-the-art method for adaptive mutation is Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [15]. It performs well in the 2005 CEC algorithm contest, and is recommended by experts in the field [5].

In this paper, we propose one solution to the revisit problem in the context of GA. We show that an archive design using *binary space partitioning* (BSP) can be naturally integrated with GA so that revisits are completely eliminated. A modification of GA by incorporating an adaptive mutation naturally arises from the integration. The new algorithm will be abbreviated as NrGA, which stands for *Non-Revisiting Genetic Algorithm with Parameter-less Adaptive Mutation*. Fig. 1 shows the communication between GA and BSP tree.

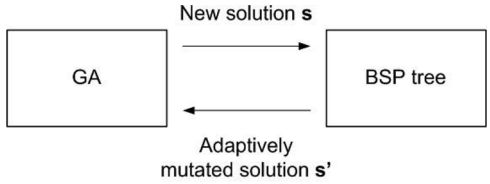

Fig. 1. Communication between GA and BSP tree.

NrGA has four key strengths.

1) It takes advantage of the "free lunch" result that a non-revisiting algorithm is superior to a revisiting algorithm.
2) It automatically assures diversity maintenance through duplicate removal. Every individual in a GA population is guaranteed to be different. Moreover, the *entire* previous search history is used to guide the search to find the next unvisited position.
3) It has *no* premature convergence problem. By nature, it is impossible for a population to consist of one kind of individual only. Moreover, even in the special case that the population consists of very similar individuals, the population would eventually diversify because of the non-revisiting property.
4) It naturally leads to an adaptive mutation mechanism and requires no tuning parameter. It is not required to set any mutation parameter or decide upon the mutation scheme. It incorporates a natural mechanism that adapts its mutation rate to the frequency of visiting a search region. Moreover, it would automatically increase its mutation rate if it is too small. Furthermore, there is a physical meaning to the adjective "small"—the mutation must be big enough to find the nearest unvisited neighbor.

Though the new algorithm only uses finite resolution grids, when compared with a canonical genetic algorithm, a generic real-coded genetic algorithm, a canonical genetic algorithm with simple diversity mechanism, and three particle swarm optimization algorithms, it shows a significant improvement in the quality of solutions found under fixed number of evaluations. Moreover, the new algorithm shows superior performance compared to CMA-ES. Furthermore, though the adaptive mutation is parameter-less, it shows and maintains a stable good performance. However, for other algorithms we compare, the performance is highly dependent on suitable parameter settings. The new algorithm can work with large search spaces with dimensions as high as 40.

The complexity that the algorithm can work with is bounded by the memory resources available, which in today's computer technology, is large but not unlimited. This moot point, however, is not an issue for applications involving expensive or time consuming fitness evaluations, or both, since the number of fitness evaluations cannot be too large in such applications. As a point of fact, many GA designs tacitly assume that the number of fitness evaluations is not extremely large. There are many real world applications involving expensive or time consuming fitness evaluations. Two illustrative examples are 3-D object registration in computer vision [16] and heating, ventilating, and air conditioning (HVAC) engineering [17]. There are many other such applications, especially for day to day engineering problems. For example, the IEEE CEC conference 2007 devotes two sessions on "Evolutionary Computation for Expensive Optimization Problems." For such applications, a fitness evaluation may take 1 second to 1 day. This gives impetus to some theoretical computer scientists to abstract away the overhead of solution generation in EA and only count the number of fitness evaluations. They refer to such a scenario as "black box optimization" [18]. The CPU overhead of NrGA is negligible and the memory requirement is acceptable for such applications.

There does exist some problems for which the fitness evaluation cost is cheap and comparable to the solution generation cost. A well-known example is the traveling salesman problem (TSP); the cost of evaluating the distance of a tour is trivial but the problem itself is NP hard. For such problems, NrGA is bounded by the memory available. Even so, we are positive for the future because of Moore's law.

To the best of our knowledge, this is the first paper to advocate a non-revisiting design for evolutionary algorithm. Use of memory to enhance search and optimization performance is unquestionably a vibrant paradigm in computer science. Constraint satisfaction problem [19] and game tree evaluation [20] are just two examples in many. In search problems, one already has the powerful Tabu search, which stores recently searched results in a Tabu list. However, our algorithm is based on fundamentally different design considerations compared to Tabu search: 1) Our algorithm is non-revisiting whereas Tabu search is not. 2) It uses the BSP tree, a more complicated construct than the Tabu list. 3) It does not just passively check whether there is revisit; it uses the clue to actively adjust the search strategy and turn it into a parameter-less adaptive mutation operator. 4) Our algorithm is not a simple hybridized version of GA with Tabu search. It is fundamentally a different conceptualization of search: the *non-revisiting stochastic search* (in this case GA). Though it is out of scope of this paper, the non-revisiting (Nr) search idea can also be used in other stochastic search algorithms [21], [22]. 5) Our algorithm does not introduce additional control parameter but Tabu search does.

This paper is organized as follows: Section II reports the GA with dynamic BSP tree archive method. Section III reports the experimental results. Section IV gives the conclusion. A preliminary version of this paper has appeared in [23].

II. Genetic Algorithm With Dynamic Tree Archive

NrGA may be conceptualized as a GA that interacts with a BSP tree archive *Ar*. BSP has been used in computer graphics and computational geometry [24], [25]. In particular, the Oct tree is a special case of the BSP tree. Here, the BSP tree is used as an efficient data structure to enable fast query of whether a solution has been visited. A GA can be visualized as generating a sequence of solutions $\mathbf{sq} = (s(1), s(2), \ldots)$. For a generational GA with population size δ, δ solutions are generated in the same cycle. For a steady-state GA, solutions are generated one by one until δ solutions are generated. Both can be considered as generating a sequence of solutions. The function of an archive *Ar* is to return an amended sequence $\mathbf{sq}' = (s(1)', s(2)', \ldots)$ such that no two $s(i)'$ and $s(j)'$ satisfy $s(i)' = s(j)'$ unless $i = j$. For ease of explanation, let us ignore revisits for the moment. That is, $s(i) \neq s(j)$ unless $i = j$. Then each solution $x = (x_1, \ldots, x_D)$ (D is function dimension and $\mathbf{x}$ can be $s(1)$, $s(2)$, etc.) will insert a leaf node $\mathbf{x}$ to the tree. x_j are variables with cardinality d (axis resolution). The following metric partitions the search space.

1) Definition 1: Metric $d(\mathbf{x}, \mathbf{y}|j)$: The Euclidean distance between x_j and y_j, where j is the dimension, is designated $d(\mathbf{x}, \mathbf{y}|j)$, i.e., $d(\mathbf{x}, \mathbf{y}|j) = |x_j - y_j|$.

Remark: Other metric, for example Hamming distance, may also be employed.

The following algorithm constructs *Ar* when there are no revisits.

Algorithm A1: (Archive construction when there are no revisits)

1. Initial condition: *Ar* consists of the *root* node
2. A new solution z (generated by GA) is presented to *Ar*
3. *Curr_node* := *root*
4.1 If (*Curr_node* has two child nodes x and y)
4.2 {
4.4 Define the comparing dimension j:
$$j = \arg\max_{k \in [1, D]} d(x, y|k)$$
4.5 If d(x,z | j) ≤ d(y,z | j)
4.6 *Curr_node* := child node x
4.7 Else
4.8 *Curr_node* := child node y
4.9 Repeat (step 4.1)
4.10 }
4.11 Else
4.12 {
4.13 Insert a "virtual" child node to *Curr_node*, create a "real" child node that records z "under" the virtual child node
4.14 Finish
4.15 }

Remark 1: If the objective function is a pseudo Boolean function, the comparing dimension j defined in step 4.4 can be chosen according to some deterministic rule, subject to the condition $x_j \neq y_j$. For example, the smallest dimension j such that $x_j \neq y_j$ may be chosen.

Remark 2: Some explanation of the "virtual" child node in step 4.13 is in order. The virtual node is introduced for explanation of the binary space partitioning concept only. It also records the solution $\mathbf{z}$. When it is used for comparison, it outputs the solution $\mathbf{z}$ for comparing. However, when it is traversed, it acts like an intermediate node and performs no action. In implementation, virtual child nodes are not needed: $\mathbf{x}$ is stored in the parent node and $\mathbf{y}$ is stored in the left leaf node. The search goes to $\mathbf{y}$ or goes down the right branch, depending on the result at the comparing dimension. As a result, the number of BSP tree nodes is strictly equal to the number of generated GA solutions; virtual nodes do not incur any memory usage. In the ensuing discussion, we shall assume a normal BSP tree. The conceptual usage of the virtual node will be assumed to be understood.

Immediately after the full algorithm is described within this section, an illustrative example of how the BSP with virtual nodes is built will be given. Readers interested in the technical details on how to build virtual nodes is directed to the example below.

The archive is a standard BSP tree with the following well established properties.

1) The root represents the entire search space S.
2) Each non-root node represents a "subspace" of S. More precisely, it is a hyper-rectangular box in the search space.
3) Suppose a parent node has two child nodes $\mathbf{x}$ and $\mathbf{y}$, then the subspaces of $\mathbf{x}$ and $\mathbf{y}$ are disjoint and their union is the subspace of the parent, i.e., the child nodes binary partition the parent subspace.

The following properties are specific to the BSP tree proposed for GA.

1) It is constructed dynamically using the data points generated by GA. Thus each run of GA will produce a different random tree.
2) It stores all the search positions $\mathbf{x}$ visited by GA. (Recall that we make the simplifying assumption that there are no revisits for the moment.)
3) For a balanced tree, the mean number of steps to decide whether a search position $\mathbf{x}$ has been visited is at most $O(\log(d^D))$.

The nature of a GA is that good solutions will be more likely to be selected; in the same vein, good recombination of solutions by crossover will be more likely to be selected. In general, the higher the selection pressure, the more likely will revisits occur. In the extreme case, the whole population may consist of only one single individual, which is selected over and over again. Thus revisits will necessarily occur in the course of running the GA. The phenomenon also occurs in other stochastic search algorithms.

Thus let us consider the general case when revisits could occur:

2) Definition 2: Revisits: The ith solution $s(i)$ in the sequence $\mathbf{sq} = (s(1), s(2), \ldots)$ is a revisit if $\exists j < i$, $s(i) = s(j)$. It occurs if and only if $Curr\_node = s(i)$ is found when traversing the tree.

Algorithm A1 is now also used to check for revisit. The modified algorithm is *Algorithm A2*. The added code is in bold.

Algorithm A2: (Archive construction and parsing when there may be revisits)

1. Initial condition: *Ar* consists of the *root* node
   revisit flag RF = 0
2. A new solution z (generated by GA) is presented to *Ar*
3. *Curr_node* := *root*
4. If (*Curr_node* has two child nodes)
   {
   Compare z with child node x and y;

   If (z = x) or (z = y)
   RF = 1

   Define the comparing dimension *j*:
   $j = \arg\max_{k\in[1,D]} d(x, y|k)$
   If d(x,z | j) ≤ d(y,z | j)
   Curr_node := child node x
   Else
   Curr_node := child node y
   Repeat (Step 4)
   }
   Else
   {
   If (RF = 0)
   {
   Insert a child node to *Curr_node* that records z
   Finish
   }
   Else
   Finish
   }

A revisit flag $\mathbf{RF}$ is used. Initially, $RF = 0$. When $\mathbf{z}$ visits a node $\mathbf{x}$, $\mathbf{x}$, and $\mathbf{z}$ are checked. If $\mathbf{x} = \mathbf{z}$, $\mathbf{RF}$ is flagged 1. Recall that each node in *Ar* records a solution in the set $\{s(1), \ldots, s(i-1)\}$. Thus a revisit may occur when Algorithm *A2* is visiting a non-leaf node or a leaf node. If $\mathbf{RF} = 1$, no new node is inserted.

No matter whether the revisit flag $\mathbf{RF}$ is turned on or not, the algorithm will finish at a leaf node at the bottom of the tree. This somewhat artificial feature is part of the adaptive mutation mechanism to be described below.

Before introducing the full algorithm, it is desirable to enlarge the utility of the concept $Curr\_node = \mathbf{x}$ in the following way: $Curr\_node = \mathbf{x}$ may also be interpreted as *Curr_node* recording the *entire subspace* under $\mathbf{x}$. Denote the subspace spanned by $\mathbf{x}$ as X. In the tree, it is represented by introducing a status flag at each node.

3) Definition 3: Open and Closed Nodes: A node $\mathbf{x}$ is flagged *Open* if the subspace represented by $\mathbf{x}$ has not all been visited. Otherwise, it is flagged *Closed*.

Remark: By definition, a node $\mathbf{x}$ for which the subspace contains a single point is always flagged *Closed*, since the single point, which simultaneously is the whole subspace, has been visited at the time when the node is created.

Thus in addition to finding an intermediate node that is a revisit, when the tree traversal visits a *Closed* node, then a revisit must have occurred.

The following algorithm deals with revisits. The newly added codes are in bold.

Algorithm A3 (NrGA)

1. Initial condition: *Ar* consists of the *root* node.
   revisit flag RF = 0
   Flag(*root*) := *Open*
2. A new solution z (generated by the GA) is presented to *Ar*
3. *Curr_node* := *root*
4. **If Flag(*Curr_node*) = *Open***
   {
   If (*Curr_node* has two child nodes)
   {
   Compare z with child node x and y;

   If (z = x) or (z = y)
   RF = 1

   Define the comparing dimension *j*:
   $j = \arg\max_{k\in[1,D]} d(x, y|k)$
   If d(x,z | j) ≤ d(y,z | j)
   Curr_node := child node x
   Else
   Curr_node := child node y
   Repeat (Step 4)
   }
   Else
   {
   If (RF = 0)
   {

```
                Insert a child node (that records) z to Curr_node
                If (child node's subspace is not a singleton)
                  Flag(child node) := Open
                Else
                  Flag(child node) := Closed
                Finish
            }
            Else
            {                                                  // Case 1
                Evaluate the unvisited subspace represented by
                Curr_node
                Create a child node by randomly mutating
                z within the unvisited subspace
                If (child node's subspace is not a singleton)
                    Flag(child node) := Open
                Else
                    Flag(child node) := Closed
                Finish
            }
        }
    }
    ELSE   /* Flag(Curr_node) = Closed */
    {
        Curr_node := Parent
        If (there are two Closed child nodes)                  // Case 2
        {
            Flag(Curr_node) := Closed
            Prune the subtree under Curr_node
            Repeat (Step 4)
        }
        Else                                                   // Case 3
        {
            Curr_node := Open child node
            Repeat (Step 4)
        }
    }
```

The full algorithm is the same as Algorithm *A2* except when revisits occur.

Consider a node $\mathbf{x}$ that is *open*.

Case 1: Node $\mathbf{x}$'s subspace X is the *nearest neighbor unvisited subspace* to $\mathbf{z}$. Now randomly mutate $\mathbf{z}$ to some point $z' \neq z$ within X.

Consider a node $\mathbf{x}$ that is *closed*. This implies that all the points in X have been visited. The algorithm backtracks to its parent node $\mathbf{p}$. Two cases may occur.

Case 2: Node $\mathbf{y}$, the other child node of $\mathbf{p}$, exists and is also *Closed*. Then the entire subspace of $\mathbf{p}$, $P = X \cup Y$, has been visited. P is flagged *Closed* as well. The algorithm backtracks to the parent of $\mathbf{p}$. Meanwhile, the entire subtree under $\mathbf{p}$ is pruned.

Case 3: Node $\mathbf{y}$ exists and is *Open*. Then the entire subspace X has been visited, but there are still some unvisited points within $Y = P - X$. Then the search is directed to node $\mathbf{y}$.

The main idea for handling revisits in NrGA is as follows:

When a node that stores the solution identical to the solution $\mathbf{z}$ generated by GA is found, a revisit has occurred. The tree needs to generate a solution that is not visited before. By nature of the binary space partitioning, it would always find the *nearest neighbor unvisited subspace* to it. A random mutation in this subspace is done to find an unvisited solution (*Case 1* above).

If the current nearest neighbor unvisited subspace is all visited, it backtracks to the parent and check whether the subspace of the parent has all been visited (*Case* 2 above). When this occurs, backtracking is again initiated and the process goes on until the parent's subspace has not been visited entirely (*Case* 3 above). Meanwhile, since the whole subtree under the parent node has been visited, there is no need to keep the subtree. Hence the entire subtree is pruned. For *Case 3*, there is an *Open* child node which means that some solutions under the child node's subspace have been visited, but not all. A depth first-order search commences to go down the subtree of this child node. By nature of the binary space partitioning, it would eventually find the nearest neighbor unvisited subspace that has not been visited entirely (*Case 1* above).

Observe that the position $\mathbf{z}'$ that $\mathbf{z}$ will mutate to is a function of $\mathbf{z}$ and X. It is not a specific mutation with a fixed mutation rate. However, it will mutate to a point in the neighborhood of $\mathbf{z}$. Also, though the mutation *per se* is deterministic, $\mathbf{z}$ and X are both random and their behaviour depends on the problem. Thus the mutation process when taken as a whole is also random. This agrees with the intuitive meaning of the term mutation. It is also problem specific.

Note that *no* other mutation operator needs to be defined within the GA. After selection and crossover, the solution can be passed immediately to the tree. If the solution has not been visited, no mutation will be performed. Otherwise, it will find a nearest unvisited neighbor subspace and randomly finds an unvisited solution in it. Note that we have introduced a novel adaptive mutation operator that has the desirable property of having no parameter that requires user fine tuning, i.e., a parameter-less mutation operator.

Intuitively, the adaptive mutation operator will adjust its mutation step size automatically according to its position in the tree. Interestingly, the more visited is a branch, the more refined is the nearest neighbor unvisited subspace, and so the smaller will be the step size and vice versa. This is reasonable because the more visited is the branch, the more the GA—itself a complex heuristic stochastic process—reasons as promising to exploit the underlying subspace; hence the smaller mutation and vice versa. Moreover, the operator will automatically increase its mutation rate if it is too small; with a physical meaning to the adjective "small"—The mutation must be big enough to find at least the nearest unvisited neighbor.

The above parameter-less adaptive mutation operator is slightly different from that we introduced earlier in [23], but keeps the basic spirit.

4) Example: The search space X is $[3,9] \times [3,6]$ and the axis resolution d is chosen as 3 (i.e., the possible values of the first gene are 3, 6, and 9, and the possible values of the second gene are 3 and 6). Suppose GA randomly generates $sq = (s(1), s(2), s(3), s(4)) = (s_1 = (9,6), s_2 = (6,6), s_3 = (9,3)$ and $s_4 = (9,3))$. Note that s_4 is a revisit. In Figs. 2(a), 3(a), 4(a), and 6(a), the grey-filled boxes represent the positions of the evaluated solutions while the circle sign represent the positions of the solutions either for revisit check or to be included into the archive. Figs. 2(b), 3(b), 4(b), 5(a), and 6(b) show the topology of the normal BSP trees. Figs. 2(c), 3(c), 4(c), 5(b),

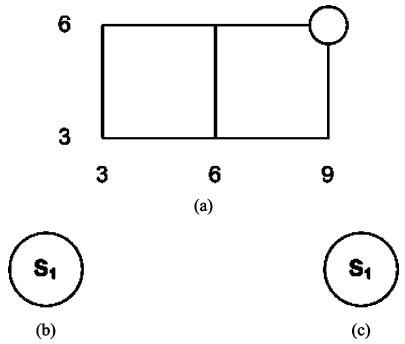

Fig. 2. (a) Content of X when s_1 is presented to the archive: the circle indicates the position of s_1. (b) Topology of the normal BSP tree after inserting s_1. (c) Topology of the optimized BSP tree after inserting s_1.

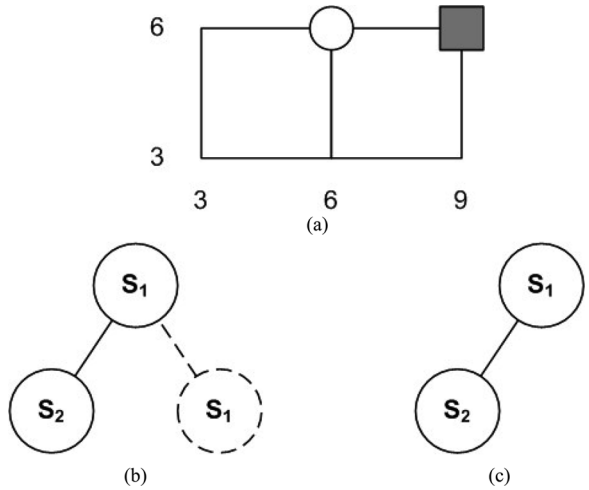

Fig. 3. (a) Content of X when s_2 is presented to the archive: the grey-filled box represents the position of the evaluated solutions and the circle indicates the position of s_2. (b) Topology of the normal BSP tree after inserting s_2. (c) Topology of the optimized BSP tree after inserting s_2.

and 6(c) show the topology of the optimized BSP trees in actual programming implementation. Open and Closed nodes of the tree are denoted by white and grey filled circles, respectively.

Initially, the archive is empty. It consists only of the root node. s_1 is inserted as an open child node of the root (Fig. 2(b)—the normal BSP tree and Fig. 2(c)—the optimized BSP tree).

s_2 is next generated by GA [Fig. 3(a)]. Since there is only one node, the search visits node s_1. A revisit check of s_2 with s_1 is conducted. Since $|s_2 - s_1| > 0$, s_2 is not a revisit. Hence it is inserted as another open child of the *root* (Fig. 3(b)—the normal BSP tree and Fig. 3(c)—the optimized BSP tree). The comparing dimension is $cd = 1$ because $|6 - 9| > |6 - 6|$. As a result, this insertion is equivalent to partitioning X into two subspaces: $X_1 = [7.5, 9] \times [3, 6]$ and $X_2 = [3, 7.5] \times [3, 6]$ where $s_1 \in X_1$ and $s_2 \in X_2$.

s_3 is next generated by GA (Fig. 4(a)). Since $|s_{3,1} - s_{1,1}| \leq |s_{3,1} - s_{2,1}|$, it goes down the right branch. A revisit check of s_3 with s_1 is conducted. Since $|s_3 - s_1| > 0$, s_3 is not a revisit and it is inserted as an open node under s_1 (Fig. 4(b)—normal BSP tree and Fig. 4(c)—the optimized BSP tree). After this insertion, X_1 is further divided into two nonoverlapping subspaces:

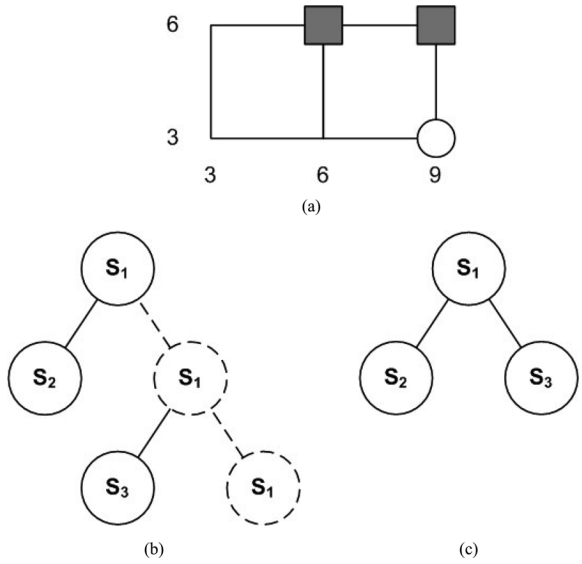

Fig. 4. (a) Content of X when s_3 is presented to the archive: the grey-filled boxes represent the positions of the evaluated solutions and the circle indicates the position of s_3. (b) Topology of the normal BSP tree after inserting s_3. (c) Topology of the optimized BSP tree after inserting s_3.

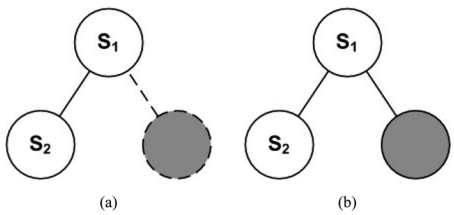

Fig. 5. (a) Topology of the normal BSP tree after pruning and (b) the topology of the optimized BSP tree after pruning.

$[7.5, 9] \times [3, 4.5]$ and $[7.5, 9] \times [4.5, 6]$. Since all possible solutions in X_1 are visited, s_3 is pruned and the right branch of s_1 is marked a *Closed* node (Fig. 5(a)—the normal BSP tree and Fig. 5(b)—the optimized BSP tree).

s_4 is next generated by GA [Fig. 6(a)]. By similar reasoning, it would select the right branch to s_1. Since the right child of s_1 is *Closed*, s_4 is a revisit and an adaptive mutation should be performed. The mutation begins by backtracking. According to *Case* 2 of step 4 of the algorithm $A3$, the mutated s_4 should be within X_2 and it is randomly selected from the set X_2. Say for example, s_4 is mutated to (3, 3) [Fig. 6(b)]. It is added under s_2 (Fig. 6(c)—the normal BSP tree and Fig. 6(d)—the optimized BSP tree).

III. Experimental Results

A. Test Function Set

A real-valued function set $F = \{f_1(x), f_2(x), \ldots, f_{19}(x)\}$ consisting of 19 well-known benchmark functions are employed to illustrate the performance of the proposed non-revisiting scheme. These 19 test functions are as follows.

1. Sphere function [26].
2. Schwefel's problem 2.22 [26].
3. Schwefel's problem 1.2 [26].
4. Schwefel's problem 2.21 [26].
5. Generalized Rosenbrock function [26].

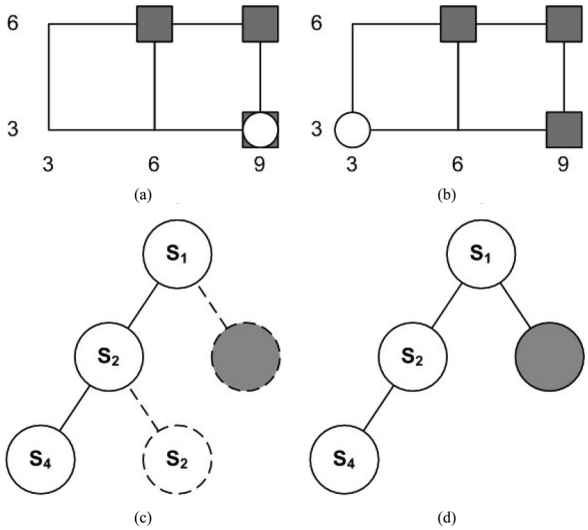

Fig. 6. (a) Content of X when s_4 is presented to the archive: the grey-filled boxes represent the positions of the revisits and the circle indicates the position of s_4. (b) Content of X after mutating s_4. (c) Topology of the normal BSP tree after inserting s_4. (d) Topology of the optimized BSP tree after inserting s_4.

6. Quaric function [26].
7. Generalized Rastrigin function [26].
8. Generalized Griewank function [26].
9. Generalized Schwefel's problem 2.26 [26].
10. Ackley function [26].
11. Shekel's Foxholes function [26].
12. Six-Hump Camel-Back function [26].
13. Branin function [26].
14. Goldstein–Price function [26].
15. Rotated high conditioned elliptic function [27].
16. Rotated Griewank's function [27].
17. Rotated Rastrigin's function [27].
18. Rotated Weierstrass's function [27].
19. Hybrid Composition function [27].

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

The first six functions are unimodal functions; the next eight are multimodal functions designed with a considerable amount of local minima. Meanwhile, the dimensions of the first ten and the last five functions are adjustable while the dimensions of $f_{11} - f_{14}$ are fixed at two. $f_{15} - f_{18}$ are rotated multimodal functions. The last function f_{19} is a hybrid composition function. The value of the global minimum is shown along with each function in Appendix I. All functions with the exceptions of f_9, f_{11}, f_{12}, and f_{13} have the global minimum at the origin or very close to the origin. Simulations are carried out to find the global minimum of each function.

B. Algorithms for Comparison

To evaluate the impact of the proposed non-revisiting algorithm, we compare the optimal fitness found by NrGA with seven benchmark evolutionary algorithms. The design and settings of NrGA and the algorithms for comparison are summarized below. Simulation settings are given in section C.

Test algorithm 1—Canonical GA (CGA): An individual of CGA is represented by a bit string. Meanwhile, crossover and mutation are operated at bit level. The offspring are generated by uniform crossover together with uniform mutation.

Test algorithm 2—Real coded GA (RC-GA): RC-GA is tailored for optimization in real-valued search spaces. In contrast to CGA, the gene consists of real-valued object parameters; evolution operates on the *natural* representation, i.e., each gene is represented by a 64-bit floating point number. The mechanisms of RC-GA genetic operators are similar to those of CGA. Two RC-GA genetic operators employed in this paper are summarized as follows.

1) Definition 4: Uniform Crossover of RC-GA: Suppose $X_1 = [x_{1,1}, x_{1,2}, \ldots, x_{1,D}]$ and $X_2 = [x_{2,1}, x_{2,2}, \ldots, x_{2,D}]$ are chosen from the population uniformly at random. The corresponding crossover-generated offspring pair $O_1 = [o_{1,1}, o_{1,2}, \ldots, o_{1,D}]$ and $O_2 = [o_{2,1}, o_{2,2}, \ldots, o_{2,D}]$ are reproduced for which $o_{1,i} = x_{1,i}$ and $o_{2,i} = x_{2,i} (i = 1, \ldots, D)$ with probability r_x and $o_{1,i} = x_{2,i}$ and $o_{2,i} = x_{1,i}$ with probability $1 - r_x$, where r_x is the crossover rate.

2) Definition 5: Mutation of RC-GA: Suppose $\mathbf{X}$ is a randomly selected individual from the current population, the corresponding mutation-generated offspring $\mathbf{O}$ is defined as $\mathbf{O} = \mathbf{X} + \mathbf{N}(\sigma_m)$ where $\mathbf{N}(\sigma_m)$ is a 1 by D vector consisting of D Gaussian random variables with zero mean and standard deviation σ_m (denoted as mutation standard deviation).

Test algorithm 3—Non-revisiting GA (NrGA): NrGA is a real-coded GA adapted with the proposed non-revisiting scheme. The number of possible real numbers for each gene is controlled by the axis resolution d. Since ultimately a real number is being represented, the gene representation and the mechanisms of the genetic operators follow those in RC-GA. To illustrate the adaptive mutation effect of the non-revisiting scheme, no mutation operator is used in the offspring reproduction. The resolution d is chosen as 80 for $f_1 - f_{10}$ and $f_{15} - f_{18}$. For $f_{11} - f_{14}$, d is chosen as 4096 (the influence of axis resolution on the best fitness will be studied in the subsection: *Adaptive Mutation Stability of NrGA* in Section III-D5).

Test algorithm 4—CMA-ES[15]: CMA-ES is an evolution strategy that adapts the full covariance matrix of a normal search (mutation) distribution. Compared to many other evolutionary algorithms, 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, and to use that information to modify the covariance matrix of the mutation distribution in a goal directed, derandomized 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 passively decrease. 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 [31] (Aug. 2007 version).

Test algorithm 5—$(\mu + 1)\ EA_d$ (Div-GA): A canonical GA with a simple diversity mechanism is employed: Div-GA. Div-GA tries to avoid duplicates by diversifying with respect to the search points. The initial population consists of μ different individuals. In each generation, the algorithm chooses two individuals x_1 and x_2 uniformly at random from P. Two offspring are generated by crossover of x_1 and x_2. Afterwards, one of the randomly chosen offspring dies out and another one is mutated to $\mathbf{X}'$. If $\mathbf{X}'$ is equal to any individual in the population P, it is discarded and the process repeats until $\mathbf{X}'$ is not equal to any individual in P. Then, this $\mathbf{X}'$ is included in P. Finally, an individual with the lowest fitness value in P, which may or may not be $\mathbf{X}'$, is deleted. If more than one individual has the same lowest fitness value, one is chosen uniformly at random. Then one generation has passed.

Test algorithm 6—Dissipative particle swarm optimization (DPSO): DPSO [28] is a modified particle swarm optimization (PSO) which introduces random mutation that helps particles to escape from local minima. Its formula is described as follows:

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

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.

Test algorithm 7—PSO with spatial particle extension (SEPSO): SEPSO [29] is another modified PSO which introduces the spatial particle extension model to increase the diversity when particles start to cluster. Two particles collide when their distance is smaller than a given radius r. If collision occurs, the corresponding particles bounce off by adjusting their velocities.

Test algorithm 8—PSO with mutation (PSOMS): PSOMS [30] prevents premature convergence according to the averaged similarity between each particle and the historical best particle explored by the swarm. The clustering degree of particle swarm is computed to measure the swarm diversity and the position of a particle is re-initialized if

$$\eta_4 < \alpha \times c(t) \times s(i, g)$$

where α is a predefined constant, $c(t)$ is the collectivity at tth generation, and $s(i, g)$ denotes the similarity of the ith particle to the current best particle.

For CGA and Div-GA, each gene of an individual is represented by 12 bits, and hence the corresponding individual length γ is $12D$. Suppose the bit string segment $b = [b_1, b_2, \ldots, b_{12}] \in \{0, 1\}^{12}$ represents the ith variable of an objective function, $S = \prod_{i=1}^{D} [L_i, U_i]$ is the search space of an objective function, the actual value represented by b, namely r, is computed as $r = (U_i - L_i) \times \sum_{j=1}^{12} b_j 2^{j-13} + L_i$. For GA-class test algorithms, the crossover rate r_x is chosen as 0.5. This is the recommended setting in [6, p. 48]. Plus $(+)$ selection (elitism selection) is employed. The mutation rate of CGA and Div-GA is $1/\gamma$. The mutation standard deviation σ_m in RC-GA is set to $0.05 \max_i (U_i - L_i)$.

For PSO-class test algorithms, the values of c_1 and c_2 are set to 0.8. The inertia w is linearly decreasing from 1 to 0. Suppose $X = \prod_{i=1}^{D} [L_i, U_i]$ is the search space of a D-dimensional objective function, the maximum velocity $V_{\max}$ is set to $0.1R$ where $R = \max(U_i - L_i)$. The parameters used in DPSO, SEPSO, and PSOMS are assigned to be the same as suggested in the original works: the parameters C_v and C_m of DPSO are chosen to be 0.001 and 0.002, respectively. For SEPSO, the radius s is assigned as $0.005RD^{0.5}$ and simple velocity line bouncing with bouncing factor -1 is used. For PSOMS, the parameters $d_{\min}$, $d_{\max}$, β, and α are set as $0.001RD^{0.5}$, $0.01RD^{0.5}$, 1 and 3, respectively.

C. Simulation Settings

For NrGA, RC-GA, and CGA, the population sizes are set to 100 and 200 offspring are generated at each generation. For Div-GA, the population size μ is also set to 100 and a nonduplicated offspring is generated at each generation. For CMA-ES, the population size λ is chosen by the suggested setting in [31] (i.e., $\lambda = 4 + \lfloor 3 \ln D \rfloor$). For DPSO, SEPSO, and PSOMS, the swarm sizes are set to 100 and 100 offspring are reproduced at each generation. To provide a fair comparison among the test algorithms, the total number of function evaluations of all algorithms is kept a constant: For functions $f_1 - f_{10}$ and $f_{15} - f_{19}$, NrGA, RC-GA, and CGA, are terminated after 200 generations, DPSO, SEPSO, and PSOMS are terminated after 400 generations, Div-GA is terminated after 40 000 generations, CMA-ES is terminated after 40 100 function evaluations: The total number of fitness evaluations of all the algorithms is fixed at 40 100. Similarly, for functions $f_{11} - f_{14}$, the total number of fitness evaluations is fixed at 4100.

All test functions with the exception of $f_{11} - f_{14}$, which are two-dimensional, are tested with dimensions 10, 20, 30, and 40. Since the algorithms are stochastic, their performance on each test function is evaluated based on statistics obtained from 100 independent runs. All simulations are performed on a PC with 3.2-GHz CPU and 1 GB of memory. All GA-class test algorithms (NrGA, RC-GA, CGA, Div-GA, DPSO, SEPSO, and PSOMS) are implemented in C language. CMA-ES uses source code in [31] and MATLAB version 6.1.

D. Simulation Results

1) Performance Measures Studied: First, we observe the performance in terms of accuracy (quality of the averaged optimal fitness) of NrGA in comparison with RC-GA, CMA-ES, CGA, Div-GA, DPSO, SEPSO, and PSOMS. To be a practical solution to real world problems, the processing time and archive size of the BSP tree of NrGA should be within a reasonable range. Thus the overhead of NrGA related to other test algorithms and the archive size statistics are also observed. Finally, the stability of the test algorithms is studied by observing the standard deviation on the averaged best fitness under different mutation parameters.

Detailed comparison results for 100 trials are presented in Tables I– VIII. In Table IX, the averaged overheads of NrGA compared to RC-GA, CGA, Div-GA, DPSO, SEPSO, and PSOMS are presented. Table X lists the average, minimum and maximum of the archive sizes for 100 trials. Table XI

	Unimodal																				*Noisy Unimodal*			
Function	f_1				f_2				f_3				f_4				f_5				f_6			
D	10	20	30	40	10	20	30	40	10	20	30	40	10	20	30	40	10	20	30	40	10	20	30	40
NrGA										2	5	5	6	7	7	7	3	2	2		2	2	2	2
RC-GA																								
CMA-ES																								
CGA																								
Div-GA																								
DPSO																								
SEPSO																								
PSOMS																								

	Multimodal																			
Function	f_7				f_8				f_9				f_{10}				f_{11}	f_{12}	f_{13}	f_{14}
D	10	20	30	40	10	20	30	40	10	20	30	40	10	20	30	40	2	2	2	2
NrGA									2							3	7			
RC-GA																				
CMA-ES																				
CGA																				
Div-GA																				
DPSO																				
SEPSO																				
PSOMS																				

	Rotated Multimodal																*Hybrid Composition*			
Function	f_{15}				f_{16}				f_{17}				f_{18}				f_{19}			
D	10	20	30	40	10	20	30	40	10	20	30	40	10	20	30	40	10	20	30	40
NrGA	3	2	4	3					2	2	2	2								
RC-GA																				
CMA-ES																				
CGA																				
Div-GA																				
DPSO																				
SEPSO																				
PSOMS																				

Fig. 7. Indicators of the best test algorithm in our experiments: The cell with grey color represents that the corresponding test algorithm outperforms the others for a particular function and a particular function dimension.

TABLE I
AVERAGE, STANDARD DEVIATION, AND CONFIDENCE INTERVAL OF THE BEST FITNESS VALUES FOUND BY NrGA, RC-GA, CMA-ES, CGA, Div-GA, DPSO, SEPSO, AND PSOMS: f_1 AND f_2

fitness function		f_1				f_2			
D		10	20	30	40	10	20	30	40
NrGA	average	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
	std. dev.	(0.0)	(0.0)	(0.0)	(0.0)	(0.000)	(0.000)	(0.000)	(0.000)
RC-GA	average	12.553	95.795	254.790	493.757	2.204	8.593	17.052	27.464
	std. dev.	(3.12)	(12.12)	(32.15)	(41.13)	(0.305)	(0.585)	(0.831)	(1.307)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.950%	99.950%	99.950%	99.950%
CMA-ES	average	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
	std. dev.	(0.0)	(0.0)	(0.0)	(0.0)	(0.000)	(0.000)	(0.000)	(0.000)
	C (t-test)	-	-	-	-	-	-	-	-
CGA	average	**0.000**	**0.000**	**0.000**	**0.000**	0.024	0.049	0.073	0.098
	std. dev.	(0.0)	(0.0)	(0.0)	(0.0)	(0.000)	(0.000)	(0.000)	(0.000)
	C (t-test)	-	-	-	-	99.950%	99.950%	99.950%	99.950%
Div-GA	average	**0.000**	**0.000**	**0.000**	**0.000**	0.024	0.049	0.073	0.098
	std. dev.	(0.0)	(0.0)	(0.0)	(0.0)	(0.000)	(0.000)	(0.000)	(0.000)
	C (t-test)	-	-	-	-	99.950%	99.950%	99.950%	99.950%
DPSO	average	3.428	26.635	72.963	130.115	0.954	3.992	7.779	35.763
	std. dev.	(1.30)	(2.88)	(4.50)	(5.21)	(0.489)	(0.810)	(1.061)	(7.803)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.950%	99.950%	99.950%	99.950%
SEPSO	mean	0.428	7.953	37.960	112.980	0.636	3.443	11.031	54.117
	std. dev.	(0.55)	(1.85)	(3.55)	(5.74)	(0.722)	(1.042)	(4.469)	(8.827)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.950%	99.950%	99.950%	99.950%
PSOMS	mean	0.528	9.816	45.470	122.106	0.575	3.670	11.949	53.433
	std. dev.	(0.58)	(2.15)	(3.96)	(5.95)	(0.527)	(1.075)	(4.571)	(8.765)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.950%	99.950%	99.950%	99.950%

TABLE II
AVERAGE, STANDARD DEVIATION, AND CONFIDENCE INTERVAL OF THE BEST FITNESS VALUES FOUND BY NrGA, RC-GA, CMA-ES, CGA, Div-GA, DPSO, SEPSO, AND PSOMS: f_3 AND f_4

Fitness function		f_3				f_4			
D		10	20	30	40	10	20	30	40
NrGA	average	**0.028**	133.1	2734.9	11401.7	4.375	20.350	33.525	41.750
	std. dev.	(0.0)	(105.8)	(1417.4)	(4319.3)	(2.509)	(4.704)	(5.614)	(5.900)
RC-GA	average	116.8	1621.000	7954.0	23477.7	5.085	11.535	17.519	23.043
	std. dev.	(30.1)	(262.4)	(1466.7)	(4088.3)	(0.713)	(0.943)	(1.102)	(1.123)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.500%	99.950%	99.950%	99.950%
CMA-ES	average	296.200	**0.004**	**1346.700**	**2615.400**	98.701	100.0	100.0	100.0
	std. dev.	(1811.9)	(0.0)	(3500.0)	(39090.9)	(0.0)	(0.0)	(0.0)	(0.0)
	C (t-test)	90.0%	99.950%	99.950%	97.500%	99.950%	99.950%	99.950%	99.950%
CGA	average	78.6	3158.2	11098.7	24112.3	0.052	**1.363**	12.064	23.567
	std. dev.	(198.6)	(1948.1)	(3697.6)	(6946.8)	(0.082)	(3.194)	(6.824)	(4.549)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%
Div-GA	average	209.100	3751.500	12181.500	23026.3	**0.050**	1.817	11.506	21.955
	std. dev.	(416.9)	(2117.5)	(3933.7)	(6239.9)	(0.039)	(4.485)	(9.362)	(7.550)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%
DPSO	average	61.212	529.139	1638.092	3156.773	2.675	7.125	**9.696**	**11.411**
	std. dev.	(5.634)	(13.599)	(21.660)	(28.798)	(0.854)	(1.190)	(1.265)	(1.276)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%
SEPSO	mean	31.867	580.701	1896.046	3851.822	1.606	8.241	12.824	16.383
	std. dev.	(4.384)	(14.226)	(23.382)	(30.534)	(0.875)	(1.560)	(1.538)	(1.750)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%
PSOMS	mean	39.140	642.645	2045.424	4077.512	1.800	8.837	13.844	16.412
	std. dev.	(4.856)	(14.800)	(24.491)	(32.826)	(0.922)	(1.534)	(1.793)	(1.664)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%	99.950%

TABLE III
AVERAGE, STANDARD DEVIATION, AND CONFIDENCE INTERVAL OF THE BEST FITNESS VALUES FOUND BY NrGA, RC-GA, CMA-ES, CGA, Div-GA, DPSO, SEPSO, AND PSOMS: f_5 AND f_6

Fitness function		f_5				f_6			
D		10	20	30	40	10	20	30	40
NrGA	average	83.4	225.3	490.4	**607.200**	1.411	4.736	8.614	12.821
	std. dev.	(159.5)	(296.5)	(615.3)	(691.1)	(0.19)	(0.32)	(0.46)	(0.52)
RC-GA	average	1401.1	37711.4	194518.3	591738.5	1.556	5.293	9.707	14.906
	std. dev.	(528.8)	(991.1)	(3840.6)	(9608.2)	(0.20)	(0.38)	(0.50)	(0.69)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.95%	99.95%	99.95%	99.95%
CMA-ES	average	3291.9	41791.8	10117.1	2725.2	**0.141**	**0.181**	**0.2**	**0.240**
	std. dev.	(13780.3)	(15784.5)	(11872.6)	(90419.3)	(0.10)	(0.09)	(0.09)	(0.08)
	C (t-test)	97.5%	99.950%	99.950%	<50%	99.95%	99.95%	99.95%	99.95%
CGA	average	**34.9**	**200.7**	**314.1**	649.0	1.479	4.826	8.966	13.169
	std. dev.	(159.3)	(471.9)	(608.7)	(962.9)	(0.21)	(0.34)	(0.49)	(0.75)
	C (t-test)	97.5%	<50%	97.5%	<50%	99.%	95.%	99.95%	99.95%
Div-GA	average	73.7	426.8	994.3	1000.5	1.485	4.804	8.803	13.229
	std. dev.	(233.7)	(177.8)	(311.3)	(303.5)	(0.17)	(0.34)	(0.48)	(0.74)
	C (t-test)	<50%	99.950%	99.950%	99.950%	99.50%	90.0%	99.50%	99.95%
DPSO	average	225.584	2444.185	8407.942	18691.251	1.789	5.676	9.846	14.539
	std. dev.	(12.738)	(33.656)	(65.043)	(83.420)	(0.46)	(0.63)	(0.82)	(0.87)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.95%	99.95%	99.95%	99.95%
SEPSO	mean	190.453	1005.395	5404.892	16847.908	1.787	5.537	10.044	14.582
	std. dev.	(17.579)	(28.798)	(54.806)	(93.673)	(0.49)	(0.67)	(0.85)	(0.87)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.95%	99.95%	99.95%	99.95%
PSOMS	mean	261.671	1296.899	6674.390	21132.674	1.764	5.562	9.993	14.502
	std. dev.	(21.102)	(31.968)	(67.598)	(110.465)	(0.45)	(0.65)	(0.82)	(0.95)
	C (t-test)	99.950%	99.950%	99.950%	99.950%	99.95%	99.95%	99.95%	99.95%

lists the stability factors (see the subsection: *Adaptive Mutation Stability of NrGA* in section D for definition) of the test algorithms.

2) Accuracy: Fig. 7 presents a summary of the results. The shaded cells in Fig. 7 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 NrGA indicate the ranking of NrGA on a particular test function at a particular function dimension when it is not the best algorithm. It can be observed that NrGA outperforms the other test

TABLE IV
AVERAGE, STANDARD DEVIATION, AND CONFIDENCE INTERVAL OF THE BEST FITNESS VALUES FOUND BY NrGA, RC-GA, CMA-ES, CGA, Div-GA, DPSO, SEPSO, AND PSOMS: f_7 AND f_8

Fitness function		f_7				f_8			
D		10	20	30	40	10	20	30	40
NrGA	average	**0.244**	**4.520**	**12.538**	**24.428**	**0.0**	**0.0**	**0.0**	**0.0**
	std. dev.	(0.53)	(2.42)	(4.27)	(6.65)	(0.0)	(0.0)	(0.0)	(0.0)
RC-GA	average	26.499	101.299	190.625	287.740	1.724	6.398	15.615	28.857
	std. dev.	(4.38)	(9.42)	(11.44)	(10.77)	(0.17)	(0.74)	(1.51)	(2.18)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	average	13.581	32.485	51.091	74.224	0.011	0.003	0.001	**0.0**
	std. dev.	(6.73)	(9.94)	(13.70)	(13.30)	(0.01)	(0.01)	(0.0)	(0.0)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.0%	-
CGA	average	4.154	18.845	37.096	55.673	0.070	0.063	0.079	0.113
	std. dev.	(1.94)	(5.96)	(9.15)	(10.95)	(0.04)	(0.06)	(0.09)	(0.11)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
Div-GA	average	4.469	18.128	36.696	54.342	0.059	0.067	0.070	0.085
	std. dev.	(2.05)	(5.33)	(7.97)	(9.95)	(0.04)	(0.06)	(0.07)	(0.09)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
DPSO	average	13.752	59.687	122.504	191.583	1.119	1.947	3.533	5.836
	std. dev.	(1.95)	(3.71)	(4.12)	(4.87)	(0.33)	(0.56)	(0.77)	(1.04)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	mean	14.406	45.218	95.184	154.353	0.845	1.298	2.371	4.858
	std. dev.	(2.49)	(3.63)	(4.50)	(4.85)	(0.36)	(0.37)	(0.68)	(1.04)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
PSOMS	mean	15.304	48.514	99.830	162.287	0.877	1.340	2.738	5.368
	std. dev.	(2.47)	(3.31)	(4.33)	(5.11)	(0.39)	(0.39)	(0.76)	(1.12)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

TABLE V
AVERAGE, STANDARD DEVIATION, AND CONFIDENCE INTERVAL OF THE BEST FITNESS VALUES FOUND BY NrGA, RC-GA, CMA-ES, CGA, Div-GA, DPSO, SEPSO, AND PSOMS: f_9 AND f_{10}

Fitness function		f_9				f_{10}			
D		10	20	30	40	10	20	30	40
NrGA	average	-4158.7	**-8273.1**	**-12343.4**	**-16312.2**	**1.323**	**0.771**	**0.0**	1.857
	std. dev.	(37.40)	(74.30)	(138.80)	(201.90)	(0.0)	(0.0)	(0.0)	(4.37)
RC-GA	average	-3985.9	-7119.7	-9506.4	-11369.6	3.719	6.621	9.146	11.279
	std. dev.	(47.50)	(171.10)	(283.30)	(434.70)	(0.22)	(0.30)	(0.30)	(0.28)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	average	-1797.3	-3590.1	-5406.8	-7187.4	16.772	18.968	18.768	19.368
	std. dev.	(116.20)	(108.80)	(92.60)	(184.10)	(7.36)	(4.37)	(4.77)	(3.42)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CGA	average	**-4165.9**	-8108.1	-11987.5	-15780.2	1.341	0.798	1.144	**1.753**
	std. dev.	(42.30)	(124.60)	(200.60)	(277.20)	(0.0)	(0.0)	(0.97)	(0.24)
	C (t-test)	80.00%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	<50%
Div-GA	average	-4139.7	-7968.7	-11716.8	-15410.6	1.341	0.798	1.585	1.798
	std. dev.	(58.40)	(173.80)	(245.60)	(288.10)	(0.0)	(0.0)	(0.83)	(0.25)
	C (t-test)	99.50%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	<50%
DPSO	average	-2346.140	-3386.548	-4256.034	-5079.228	2.511	4.094	5.660	6.861
	std. dev.	(14.80)	(19.56)	(21.35)	(24.51)	(0.51)	(0.61)	(0.74)	(0.81)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	mean	-2961.512	-5009.208	-6742.595	-7948.183	1.848	3.589	5.810	8.007
	std. dev.	(17.13)	(22.64)	(27.37)	(29.20)	(0.46)	(0.68)	(0.87)	(0.93)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
PSOMS	mean	-2868.515	-5006.783	-6562.549	-7718.631	1.962	3.665	6.040	8.184
	std. dev.	(17.60)	(22.53)	(25.87)	(29.02)	(0.47)	(0.73)	(0.95)	(0.96)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

algorithms: NrGA ranks first (or joint first) in 39 and is second in 13 out of a total of 64 cases.

The detailed simulation results (mean and standard deviation) are listed in Tables I–VIII. It lists the average and the standard deviation (inside brackets) of the optimal fitness for 100 trials. A value in bold indicates that the corresponding algorithm is the best among the algorithms on a particular test function at a particular function dimension. To illustrate the significance of the indicator in Fig. 7, the confidence intervals C (in term of %) for t-tests comparing the averaged optimal fitness values of NrGA

TABLE VI
AVERAGE, STANDARD DEVIATION, AND CONFIDENCE INTERVAL OF THE BEST FITNESS VALUES FOUND BY NrGA, RC-GA, CMA-ES, CGA, Div-GA, DPSO, SEPSO, AND PSOMS: f_{11}, f_{12}, f_{13}, f_{14}, AND f_{15}

Fitness function		f_{11}	f_{12}	f_{13}	f_{14}	f_{15}			
D		2	2	2	2	10	20	30	40
NrGA	average	1.679	**-1.0319**	**0.3980**	**3.001**	256×10^4	1786×10^4	11520×10^4	14299×10^4
	std. dev.	(0.03)	(0.01)	(0.01)	(0.01)	(150×10^4)	(773×10^4)	(5509×10^4)	(5212×10^4)
RC-GA	average	1.091	-1.0307	0.3985	3.0099	223×10^4	2395×10^4	16671×10^4	14959×10^4
	std. dev.	(0.07)	(0.04)	(0.01)	(0.02)	(75×10^4)	(594×10^4)	(2680×10^4)	(2332×10^4)
	C (t-test)	99.95%	<50%	<50%	99.95%	95.00%	99.95%	99.95%	80.00%
CMA-ES	average	111.886	-1.0153	12.2453	112.8640	**0.0**	**0.0**	**0.271**	**67792.052**
	std. dev.	(45.87)	(0.01)	(5.88)	(67.23)	(1117×10^4)	(3253×10^5)	(4017×10^5)	(4599×10^5)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	<50%	<50%	99.50%	99.75%
CGA	average	1.017	-1.0316	0.3987	3.0002	713×10^4	2371×10^4	6904×10^4	10722×10^4
	std. dev.	(0.00)	(0.00)	(0.01)	(0.00)	(553×10^4)	(1248×10^4)	(2583×10^4)	(3750×10^4)
	C (t-test)	99.95%	<50%	<50%	<50%	99.95%	99.95%	99.95%	99.95%
Div-GA	average	1.122	-1.0319	0.3985	3.0669	1167×10^4	3125×10^4	9424×10^4	15035×10^4
	std. dev.	(0.01)	(0.01)	(0.00)	(0.01)	(946×10^4)	(1558×10^4)	(3915×10^4)	(5220×10^4)
	C (t-test)	99.95%	<50%	<50%	99.95%	99.95%	99.95%	99.75%	80.00%
DPSO	average	**0.998**	-1.0313	0.3982	3.0020	719×10^5	10516×10^5	32889×10^5	32216×10^5
	std. dev.	(0.01)	(0.01)	(0.01)	(0.02)	(4797.86)	(21537.33)	(22232.14)	(25094.24)
	C (t-test)	99.95%	75.00%	<50%	75.00%	99.95%	99.95%	99.95%	99.95%
SEPSO	mean	1.018	-1.0309	0.3981	3.0013	4137×10^4	16529×10^4	12151×10^5	10683×10^5
	std. dev.	(0.37)	(0.01)	(0.01)	(0.01)	(4415.33)	(9117.72)	(20462.34)	(20110.60)
	C (t-test)	99.95%	90.00%	<50%	99.95%	99.95%	99.95%	99.95%	99.95%
PSOMS	mean	1.008	-1.0310	0.3982	3.0230	3419×10^4	16285×10^4	11502×10^5	10008×10^5
	std. dev.	(0.31)	(0.01)	(0.01)	(0.01)	(4098.86)	(9839.79)	(18681.51)	(18036.60)
	C (t-test)	99.95%	90.00%	<50%	99.95%	99.95%	99.95%	99.95%	99.95%

TABLE VII
AVERAGE, STANDARD DEVIATION, AND CONFIDENCE INTERVAL OF THE BEST FITNESS VALUES FOUND BY NrGA, RC-GA, CMA-ES, CGA, Div-GA, DPSO, SEPSO, AND PSOMS: f_{16} AND f_{17}

Fitness function		f_{16}				f_{17}			
D		10	20	30	40	10	20	30	40
NrGA	average	**0.0**	**0.0**	**0.0**	**0.055**	3.98	12.93	28.85	36.81
	std. dev.	(0.0)	(0.0)	(0.0)	(0.36)	(6.49)	(8.73)	(13.55)	(18.01)
RC-GA	average	1.706	6.625	17.360	33.055	27.55	103.26	194.96	297.61
	std. dev.	(0.18)	(0.77)	(1.50)	(2.90)	(4.26)	(8.29)	(10.39)	(14.36)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	average	**0.0**	**0.0**	**0.0**	0.18	**2.15**	**8.56**	**18.13**	**23.26**
	std. dev.	(0.01)	(0.01)	(0.0)	(0.0)	(6.29)	(4.13)	(21.97)	(30.79)
	C (t-test)	-	-	-	99.95%	97.50%	99.95%	99.95%	99.95%
CGA	average	0.152	0.184	0.169	0.202	15.68	46.23	80.10	138.84
	std. dev.	(0.11)	(0.21)	(0.20)	(0.18)	(7.83)	(20.72)	(34.94)	(65.91)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
Div-GA	average	0.173	0.237	0.238	0.195	15.18	39.40	70.52	108.28
	std. dev.	(0.15)	(0.25)	(0.24)	(0.21)	(7.67)	(16.56)	(22.26)	(44.35)
	C (t-test)	99.95%	99.95%	99.95%	99.90%	99.95%	99.95%	99.95%	99.95%
DPSO	average	1.240	2.780	5.578	9.218	25.98	100.14	179.46	275.72
	std. dev.	(0.34)	(0.73)	(1.19)	(1.38)	(2.75)	(3.81)	(4.48)	(5.10)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	mean	1.025	1.948	5.184	9.761	17.53	66.13	127.40	204.21
	std. dev.	(0.32)	(0.63)	(1.20)	(1.35)	(2.75)	(3.80)	(4.91)	(4.74)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
PSOMS	mean	1.045	2.139	5.052	9.870	19.34	68.47	138.91	205.33
	std. dev.	(0.30)	(0.66)	(1.21)	(1.49)	(2.42)	(3.83)	(5.36)	(4.75)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

with each algorithm is listed. In the tables, a larger confidence interval infers a higher significance of a comparison result. The confidence intervals of NrGA compared to an algorithm is indicated by "-" if the averaged best fitness values found by both algorithms are equal to the global optimum.

The significance of the rank of NrGA is summarized as follows: We are 99.95% confident that NrGA performs the best among all test algorithms in 32 test cases. For the remaining seven test cases that NrGA ranks or jointly ranks first, three out of the seven cases are with 90% confidence interval, and the remaining four cases are with confidence intervals smaller than 50%. Though small confidence intervals ($< 50\%$) are obtained in those cases, the performance of NrGA is most consistent among all test algorithms as the standard deviation of

TABLE VIII
AVERAGE, STANDARD DEVIATION, AND CONFIDENCE INTERVAL OF THE BEST FITNESS VALUES FOUND BY NrGA, RC-GA, CMA-ES, CGA, Div-GA, DPSO, SEPSO, AND PSOMS: f_{18} AND f_{19}

Fitness function		f_{18}				f_{19}			
D		10	20	30	40	10	20	30	40
NrGA	average	**0.09**	**0.18**	**0.54**	**1.08**	**3833**	**4393**	**4409**	**5128**
	std. dev.	(0.08)	(0.12)	(0.41)	(0.03)	(633.88)	(269.24)	(278.90)	(404.06)
RC-GA	average	3.34	10.22	18.72	28.68	162147	244852	297445	347577
	std. dev.	(0.47)	(0.78)	(0.96)	(1.66)	(10798)	(12298)	(12270)	(12359)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	average	0.12	0.43	0.97	1.12	4920	6037	6410	6698
	std. dev.	(0.03)	(0.05)	(0.11)	(0.07)	(412.33)	(177.64)	(362.67)	(324.05)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
CGA	average	1.75	8.36	14.97	23.73	11834	17099	22145	26344
	std. dev.	(1.46)	(2.48)	(3.44)	(5.30)	(2907.5)	(4190.0)	(4432.2)	(5319.3)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
Div-GA	average	2.40	9.14	15.46	24.27	15870	25210	31736	38896
	std. dev.	(1.67)	(2.46)	(3.22)	(4.51)	(4495.3)	(6455.9)	(5621.9)	(6457.2)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
DPSO	average	3.20	8.98	15.01	20.75	3637336	7583812	9554016	10921395
	std. dev.	(0.81)	(1.23)	(1.31)	(1.37)	(1157.1)	(1183.3)	(1107.7)	(1060.2)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	mean	3.58	10.28	17.90	24.72	367559	1300268	2346665	3818739
	std. dev.	(1.12)	(1.54)	(1.75)	(1.75)	(858.1)	(1349.2)	(1560.1)	(1893.6)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
PSOMS	mean	3.62	10.96	17.24	26.07	312463	1175650	3080096	3860594
	std. dev.	(1.08)	(1.50)	(1.85)	(1.79)	(734.2)	(1302.9)	(1782.5)	(1802.8)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

the fitness values obtained by NrGA is the smallest. For all test cases that NrGA ranks second, the corresponding confidence intervals are larger than 99.5%. From the above, we conclude that the comparisons of NrGA with other algorithms are statistically significant.

3) Overhead of NrGA: During the search process, a test algorithm spends its computation time on both *solution generation* and *function evaluation*. Different algorithms use different strategies to guess a solution. As a result, the corresponding processing time is different. For example, the major computation load of RC-GA is the floating point number operation in mutation; CMA-ES requires heavy computation in matrix operation such as eigen-decomposition; the bit-string individual of CGA has to be converted into real-valued vector before function evaluation; the search for duplicated individual is a major computation load for Div-GA; DPSO spends time on checking if any particle has to be randomly mutated; SEPSO and PSOMS compute the mutual distances in the swarm at every iteration. For NrGA, the access to the BSP tree is time consuming. Longer processing time of NrGA compared with RC-GA, CGA, Div-GA, and DPSO is expected. On the other hand, NrGA needs not compute the similarities among chromosomes, thus its processing time is shorter than SEPSO and PSOMS. As a practical solution to real world applications, the processing time of NrGA should be within an acceptable range. In this section, the computation load of NrGA, in term of overhead related to the other test algorithms, is studied. Since CMA-ES is written in MATLAB, its comparison time is absent.

Table IX lists the average difference of the computation time (in seconds) between NrGA and RC-GA, CGA, Div-GA, DPSO, SEPSO, and PSOMS. The maximum overhead is only 1.891 seconds when using 40 100 fitness evaluations to optimize a 40-dimension function. For optimization of expensive or time consuming fitness functions, this overhead is insignificant.

4) Archive Size of NrGA: The variations of the archive size L of the BSP tree in 100 runs are observed and the results are listed in Table X. The archive size is the number of BSP tree nodes. In general, the archive size increases along with function dimension. One reason is that the increase in function dimension leads to an exponential growth on the size of search space, which reduces the chance of tree node pruning. The archive size is also affected by the nature of an objective function. For Schwefel's problem 1.2 (function f_3), which is non-separable, the crossover operator is likely to be a random search, during which the offspring are less likely to be merged by tree node pruning. Due to the absolute sign $|\cdot|$ of Schwefel's problem 2.21 (function f_4), its fitness landscape is composed of piecewise flat regions. These flat regions mislead the selection strategy and NrGA is more likely to act like a random search, during which the chance of tree pruning is reduced. For Quaric function (function f_6), the random element introduces many small oscillations (local optima) into the fitness landscape. In addition, these local optima are sparsely distributed on the landscape; they can hardly be merged to reduce the archive size. The worst case archive size in all simulations is 40 100 (i.e., no pruning). Given a PC that is commonly configured with 1 GB of memory, the size of the BSP tree consisting of 40 100 nodes is just 0.0896% of the memory (assuming that each node occupies 24 bytes). It can be concluded that the memory usage is acceptable, especially in the backdrop of expensive

TABLE IX
AVERAGE DIFFERENCE OF THE COMPUTATION TIME (IN SECONDS) BETWEEN NrGA AND RC-GA, CGA, Div-GA, DPSO, SEPSO, AND PSOMS

Function	*D*	**RC-GA**	**CGA**	**Div-GA**	**DPSO**	**SEPSO**	**PSOMS**
f_1	10	0.415	0.502	0.325	0.100	-1.110	-0.040
	20	0.374	0.527	0.400	0.210	-1.240	-0.070
	30	0.262	0.660	0.357	0.310	-1.370	-0.110
	40	0.108	0.566	0.138	0.420	-1.710	-0.150
f_2	10	0.232	0.334	0.165	0.120	-1.090	-0.040
	20	0.566	0.580	0.373	0.220	-1.130	-0.080
	30	0.667	0.799	0.452	0.350	-1.280	-0.130
	40	0.410	0.644	0.353	0.440	-1.450	-0.150
f_3	10	0.211	0.311	0.119	0.140	-1.180	-0.050
	20	0.588	0.733	0.408	0.260	-1.130	-0.090
	30	0.648	0.979	0.642	0.390	-1.380	-0.140
	40	0.860	1.453	1.249	0.500	-1.530	-0.160
f_4	10	0.215	0.308	0.175	0.120	-1.180	-0.040
	20	0.400	0.419	0.191	0.230	-1.230	-0.080
	30	0.617	0.732	0.487	0.330	-1.300	-0.110
	40	0.818	0.868	0.636	0.450	-1.350	-0.160
f_5	10	0.420	0.642	0.379	0.100	-1.170	-0.040
	20	0.266	0.698	0.389	0.200	-1.100	-0.070
	30	0.589	1.150	0.911	0.280	-1.340	-0.090
	40	0.534	1.314	1.038	0.290	-1.460	-0.010
f_6	10	0.209	0.319	0.181	0.120	-1.190	-0.040
	20	0.617	0.736	0.679	0.240	-1.260	-0.080
	30	0.468	0.723	0.534	0.340	-1.270	-0.100
	40	0.693	0.953	0.770	0.450	-1.280	-0.120
f_7	10	0.200	0.480	0.296	0.090	-1.100	-0.030
	20	0.045	0.354	0.014	0.190	-1.350	-0.070
	30	0.911	1.126	0.756	0.280	-1.420	-0.070
	40	0.807	1.301	0.810	0.360	-1.560	-0.090
f_8	10	0.403	0.507	0.283	0.120	-1.190	-0.050
	20	0.236	0.526	0.384	0.200	-1.260	-0.070
	30	0.281	0.634	0.474	0.310	-1.320	-0.110
	40	0.333	0.737	0.500	0.400	-1.370	-0.150
f_9	10	0.237	0.327	0.135	0.130	-1.110	-0.060
	20	0.455	0.641	0.422	0.230	-1.170	-0.090
	30	0.596	0.730	0.437	0.350	-1.240	-0.140
	40	0.860	1.034	0.595	0.460	-1.290	-0.180
f_{10}	10	0.235	0.514	0.331	0.130	-1.190	-0.050
	20	0.397	0.530	0.375	0.220	-1.250	-0.070
	30	0.413	0.568	0.353	0.340	-1.300	-0.100
	40	0.898	1.186	0.820	0.440	-1.330	-0.130
f_{11}	2	0.092	0.134	0.136	0.042	-0.678	-0.068
f_{12}	2	0.065	0.060	0.038	0.031	-0.699	-0.009
f_{13}	2	0.038	0.079	0.047	0.031	-0.699	-0.019
f_{14}	2	0.066	0.054	0.044	0.031	-0.699	-0.019
f_{15}	10	0.300	0.671	0.844	0.817	-1.050	-0.783
	20	0.615	0.682	0.832	1.126	-1.056	-1.644
	30	0.744	0.743	1.141	1.419	-1.214	-2.319
	40	0.964	0.802	1.236	1.677	-1.374	-3.026
f_{16}	10	0.662	0.424	0.501	1.040	-0.826	-1.040
	20	0.883	0.508	0.647	1.255	-1.115	-1.685
	30	0.962	0.558	0.864	1.718	-1.412	-2.155
	40	1.138	0.717	1.138	1.891	-1.775	-2.758
f_{17}	10	0.324	0.571	0.541	1.145	-1.421	-0.479
	20	0.465	0.580	0.855	1.483	-1.979	-0.755
	30	0.581	0.770	0.183	1.513	-2.188	-1.045
	40	0.721	0.909	0.610	1.855	-2.250	-1.350
f_{18}	10	0.233	0.052	0.043	1.008	-0.360	-1.027
	20	0.291	0.664	0.717	1.127	-0.343	-1.2076
	30	0.352	0.765	0.974	1.168	-0.341	-1.974
	40	0.439	0.993	1.153	1.175	-0.417	-2.917
f_{19}	10	0.327	0.076	0.068	1.067	-0.463	-1.056
	20	0.212	0.632	0.567	1.112	-0.751	-1.132
	30	0.330	0.819	0.889	1.134	-0.827	-1.876
	40	0.425	0.997	1.102	1.367	-0.934	-2.531

or time consuming fitness evaluations which put an upper bound on total number of function evaluations.

5) Adaptive Mutation Stability of NrGA: The effect on the averaged optimal fitness is investigated by varying the axis resolution d of NrGA, the mutation standard deviation σ_m of RC-GA, the mutation rates of CGA and Div-GA, C_v in DPSO, the radius s in SEPSO and α in PSOMS. d is varied from 60 to 100 for $f_1 - f_{10}$ and $f_{15} - f_{18}$, and from 3800 to 4200 for $f_{11} - f_{14}$. The value of σ_m is varied from 0.01 to 0.05 of $\max_i(U_i - L_i)$.

The mutation rate of CGA and Div-GA is varied from $1/\gamma$ to 1/3. C_v in DPSO is varied from $0.0005R$ to $0.002R$. The radius s in SEPSO is varied from $0.0005R$ to $0.002R$. α in PSOMS is varied from 1 to 10. The remaining parameters of the test algorithms are maintained the same. For each particular value of parameter i (d for NrGA, σ_m for RC-GA, the mutation rates of CGA and Div-GA, C_v in DPSO, s in SEPSO and α in PSOMS), an averaged best fitness f_i is obtained. The stability factor of an algorithm is defined as the standard deviation of $\{f_i\}$. Since CMA-ES recommends a set of parameters that should be used in all applications to become "parameter-less" [31], the effect of varying parameters of CMA-ES is not studied. Table XI lists the result for 100 trials. A value in bold indicates that the corresponding algorithm is the best among the algorithms on a particular test function.

TABLE X
STATISTICS OF THE ARCHIVE SIZES OF NrGA

Function	*D*	L_{mean}	L_{min}	L_{max}	*Function*	*D*	L_{mean}	L_{min}	L_{max}
f_1	10	20470	14098	26504	f_{10}	10	20439	13755	25557
	20	37678	35268	38951		20	37590	35058	39064
	30	39768	39358	39991		30	39843	39130	40100
	40	40050	39879	40100		40	39988	39639	40100
f_2	10	19033	13621	25049	f_{11}	2	2801	1305	3589
	20	36765	33377	39143					
	30	39604	38757	40100	f_{12}	2	2763	1378	3377
	40	39992	39677	40100					
f_3	10	37235	32205	39662	f_{13}	2	2756	2038	3428
	20	40100	40100	40100					
	30	40100	40100	40100	f_{14}	2	2768	2238	3647
	40	40100	40100	40100					
f_4	10	32173	22243	37466	f_{15}	10	39321	36340	39500
	20	39791	39252	40073		20	40099	40086	40100
	30	40070	39964	40100		30	40100	40100	40100
	40	40098	40059	40100		40	40100	40100	40100
f_5	10	30375	22733	36954	f_{16}	10	26009	20161	31115
	20	39751	37566	40078		20	37685	35619	39056
	30	40098	40056	40100		30	39777	39289	40003
	40	40100	40100	40100		40	40051	39926	40100
f_6	10	38994	37866	39549	f_{17}	10	22845	18394	31782
	20	40095	40072	40100		20	36864	35790	37866
	30	40100	40099	40100		30	39480	38912	39739
	40	40100	40100	40100		40	40002	39885	40088
f_7	10	31702	27172	36188	f_{18}	10	31536	23097	38261
	20	39268	38202	39945		20	39895	39203	40096
	30	39917	39478	40080		30	40089	39964	40100
	40	40040	39868	40100		40	40099	40091	40100
f_8	10	25066	17635	31407	f_{19}	10	34141	27697	38131
	20	39074	37605	39811		20	38411	36658	40096
	30	40046	39914	40100		30	39511	38558	40096
	40	40100	40095	40100		40	39820	39444	40100
f_9	10	19287	15610	25107					
	20	36343	33975	38820					
	30	39241	38415	39838					
	40	39784	39417	40003					

TABLE XI
STABILITY FACTORS OF NrGA, RC-GA, CGA, Div-GA, DPSO, SEPSO, AND PSOMS

Function	*D*	**NrGA**	**RC-GA**	**CGA**	**Div-GA**	**DPSO**	**SEPSO**	**PSOMS**
f_1	40	**0.005**	191.5	0.5	0.5	132.55	52.30	87.87
f_2	40	**0.0078**	8.7	110306.5	140961.6	17.85	40.36	13.87
f_3	40	**1375.1**	6160.4	17404.0	17215.8	1617.37	2415.21	1881.23
f_4	40	**0.9**	6.9	15.5	16.5	4.73	5.31	10.54
f_5	40	**92.5**	2.37×10^{5}	4.82×10^{7}	4.93×10^{7}	20942.20	38544.21	34894.23
f_6	40	**0.062**	0.9	29.8	29.2	9.47	5.59	4.90
f_7	40	**0.8**	82.6	127.5	128.3	78.80	50.22	109.60
f_8	40	**0.034**	10.7	172.8	174.9	4.02	3.10	2.66
f_9	40	**82.5**	1910.0	2844.4	2844.5	2704.05	4707.87	4230.39
f_{10}	40	**0.049**	2.9	5.9	5.9	3.41	1.93	3.44
f_{11}	2	**0.0023**	0.0064	0.0067	0.0060	0.28	0.27	0.39
f_{12}	2	0.0086	0.0076	**0.0055**	0.0088	0.60	0.40	0.60
f_{13}	2	0.0064	0.0081	**0.0049**	0.0087	0.13	0.18	0.23
f_{14}	2	0.0048	**0.0013**	0.0073	0.0067	0.94	1.92	1.66
f_{15}	40	58.3×10^{6}	259.4×10^{6}	212.2×10^{6}	317.6×10^{6}	6.47×10^{4}	$\mathbf{5.71\times10^{4}}$	9.54×10^{4}
f_{16}	40	**0.67**	8.9	5.1	2.2	11.3	12.8	13.5
f_{17}	40	41.65	**31.71**	134.64	121.71	60.53	49.85	47.23
f_{18}	40	**5.31**	8.72	12.63	11.87	6.44	9.71	13.05
f_{19}	40	**345.2**	56785	8831.64	10587.91	245892.3	109873.2	118267.1

It is observed that NrGA has small standard deviations compared with the rest. This shows that the averaged optimal fitness of NrGA is only slightly dependent on the axis resolution for all the test functions. Therefore, a proper selection of the axis resolution is not a key factor for NrGA, at least for the 19 benchmark functions. The use of the proposed scheme can be identified as a good strategy to overcome the difficulties of selecting a proper mutation parameter. However, for other EAs, except CMA-ES which we do not compare, parameter setting is a critical factor for good performance.

IV. Conclusion

We propose a non-revisiting genetic algorithm (NrGA) with a novel adaptive mutation mechanism. The new GA guarantees that no revisits ever occur in its fitness evaluations. It achieves this by interacting with a dynamically constructed binary space partitioning archive (BSP). The concept of BSP originates from the fields of computer graphics and computational geometry. The BSP archive is built up as a random tree for which its growth process reflects the evolution history of the GA, and is a quick method to query whether there is a revisit. The entire previous search history is used to guide the search to find the next unvisited position. Through this, a new adaptive mutation naturally arises. The adaptive mutation is parameter-less. It does not introduce any additional tuning parameter to GA.

The mechanism of adaptive mutation is to find the nearest neighbor unvisited subspace of the current GA, and then perform a random mutation within this subspace. The operation is random and has the nearest neighbor property, which are characteristics that a mutation should have. The more a GA visits a particular region of the search space, the smaller will be the size of its corresponding unvisited subspace and vice versa. This means that the more visited is a part of the search space, the smaller the mutation step and vice versa. This is reasonable, as the more visited is a region, the more likely that the GA—itself a complex stochastic process—views as promising to *exploit* the region; whereas the more unvisited is a region, the more *explorative* effort should be put on the region. From the constructional point of view, the mutation step automatically adjusts based on the position random mutation occurs in the BSP tree. Moreover, the mutation guarantees that it is at least big enough to visit the nearest unvisited neighbor. Recently, it is proved that such a strategy is sufficient to change the average time complexity from exponential to polynomial using an example function [32]. Furthermore, the random mutation within the unvisited subspace contributes to the search stability in change of dimensions and resolutions.

Apart from the parameter-less adaptive mutation, NrGA enjoys the "free lunch" result that a non-revisiting algorithm is superior to its revisiting counterparts. It is also a natural diversity maintenance mechanism and by nature does not suffer from premature convergence. Even in the special case that the population consists of very similar individuals, the population would eventually diversify because of the non-revisiting nature. These two properties are very useful to EA, as many empirical studies, and recently the theoretical study [9], have highlighted the improvement that can be achieved by incorporating diversity maintenance in EA, and premature convergence is commonly attributed as the culprit for failure of GA. It is also clear that genetic diversity in any particular gene can never be lost.

NrGA is compared with canonical GA, real-coded GA, canonical GA with simple diversity mechanism, dissipative PSO, PSO with spatial particle extension and PSO with mutation using 19 well known benchmark functions in the literature for dimensions up to 40. NrGA outperforms all of them. In particular, it outperforms canonical GA with simple diversity mechanism, indicating that it has superior ability to offer for diversification. NrGA also outperforms CMA-ES, a state-of-the-art algorithm in adaptive mutation. Moreover, the mutation part of NrGA is truly parameter-less.

Stability of parameter settings is also investigated. The only parameter that may influence NrGA's adaptive mutation property is the resolution parameter. It is found from experiments that NrGA's performance is insensitive to this parameter. In contrast, for canonical GA, real-coded GA, canonical GA with simple diversity mechanism, dissipative PSO, PSO with spatial particle extension and PSO with mutation, parameter settings will greatly affect the performance. This leads us to conclude the NrGA is stable and little parameter tuning is required. This is a desirable feature as parameter tuning is a difficult problem in adaptive EA design [5].

The CPU overhead and the memory requirement of NrGA are studied. It is concluded that the CPU overhead is negligible and the memory overhead is acceptable for such problems. This makes it ideal to apply NrGA to the large class of engineering problems where the fitness evaluation cost is (much) higher than the solution generation cost.

The present GA part of NrGA implementation is a standard $(\mu + \lambda)$ GA with uniform crossover. The resolution parameter has been shown to be insensitive; the crossover rate is set to the standard rate 0.5. The only two other parameters that need to be set are μ and λ. In this implementation, we have set it to fixed values. Setting them as a function of the dimension, as in CMA-ES, may further improve performance.

For future work, NrGA has an interesting pruning feature which deserves further study. Recently, we also obtain positive results applying NrGA to the traveling salesman problem [33], an application in which fitness evaluations are inexpensive. Rigorous theoretical analysis of NrGA is also an interesting topic.

Appendix
Benchmark Functions

1) Sphere function [26]:

$$f_1(\mathbf{x}) = \sum_{i=1}^{D} x_i^2 \text{ where } \mathbf{x} \in [-100,100]^D$$
$$\min_{\mathbf{x}} f_1(\mathbf{x}) = f_1([0,0,\ldots,0]) = 0$$

2) Schwefel's problem 2.22 [26]:

$$f_2(\mathbf{x}) = \sum_{i=1}^{D} |x_i| + \prod_{i=1}^{D} |x_i|$$
$$\text{where } \mathbf{x} \in [-10,10]^D$$
$$\min_{\mathbf{x}} f_2(\mathbf{x}) = f_2([0,0,\ldots,0]) = 0.$$

3) Schwefel's problem 1.2 [26]:

$$f_3(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2 \text{ where } \mathbf{x} \in [-100,100]^D$$
$$\min_{\mathbf{x}} f_3(\mathbf{x}) = f_3([0,0,\ldots,0]) = 0.$$

4) Schwefel's problem 2.21 [26]:

$$f_4(\mathbf{x}) = \max_{i \in [1,D]} |x_i| \text{ where } \mathbf{x} \in [-100,100]^D$$
$$\min_{\mathbf{x}} f_4(\mathbf{x}) = f_4([0,0,\ldots,0]) = 0.$$

5) Generalized Rosenbrock function [26]:

$$f_5(\mathbf{x}) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$
$$\text{where } \mathbf{x} \in [-29, 31]^D$$
$$\min_{\mathbf{x}} f_5(\mathbf{x}) = f_5([1, 1, \ldots, 1]) = 0.$$

6) Quaric function [26]:

$$f_6(\mathbf{x}) = \sum_{i=1}^{D} ix^4 + random[0, 1]$$
$$\text{where } \mathbf{x} \in [-1.28, 1.25]^D$$
$$\min_{\mathbf{x}} f_6(\mathbf{x}) = f_6([0, 0, \ldots, 0]) = 0.$$

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

7) Generalized Rastrigin function [26]:

$$f_7(\mathbf{x}) = \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$$
$$\text{where } \mathbf{x} \in [-5.12, 5.12]^D$$
$$\min_{\mathbf{x}} f_7(\mathbf{x}) = f_7([0, 0, \ldots, 0]) = 0.$$

8) Generalized Griewank function [26]:

$$f_8(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos \frac{x_i}{\sqrt{i}} + 1$$
$$\text{where} \in [-600, 600]^D$$
$$\min_{\mathbf{x}} f_8(\vec{x}) = f_8([0, 0, \ldots, 0]) = 0.$$

9) Generalized Schwefel's problem 2.26 [26]:

$$f_9(\mathbf{x}) = -\sum_{i=1}^{D} x_i \sin \sqrt{|x_i|} \text{ where } \mathbf{x} \in [-500, 500]^D$$
$$\min_{\mathbf{x}} f_9(\mathbf{x}) = f_9([420.9687, \ldots, 420.9687]) = -418.9829D.$$

10) Ackley function [26]: See equation at the bottom of the page.
11) Shekel's Foxholes function [26]: See equation at the bottom of the page.
12) Six-Hump Camel-Back function [26] see equation at the bottom of the page.
13) Branin function [26]: see equation at the bottom of the next page.
14) Goldstein-Price function [26]: see equation at the bottom of the next page.
15) Rotated high conditioned elliptic function [27]: see equation at the bottom of the next page.

$$f_{10}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x_i\right) + 20 + e$$
$$\text{where } \mathbf{x} \in [-32, 32]^D \text{ and } \min_{\mathbf{x}} f_{10}(\mathbf{x}) = f_{10}([0, 0, \ldots, 0]) = 0$$

$$f_{11}(\mathbf{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{i,j})^6}\right]^{-1} \quad \text{where } \mathbf{x} \in [-98, 34]^2 \text{ and}$$
$$\{a_{i,j}\} = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \ldots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \ldots & 32 & 32 & 32 \end{bmatrix}$$
$$\min_{\mathbf{x}} f_{11}(\mathbf{x}) = f_{11}([-32, -32]) \approx 1$$

$$f_{12}(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$
$$\text{where } \mathbf{x} \in [-4.91017, 5.0893] \times [-5.7126, 4.2874] \text{ and}$$
$$\min_{\mathbf{x}} f_{12}(\mathbf{x}) = f_{12}([0.08983, -0.7126]) = f_{12}([-0.08983, 0.7126]) = -1.0316285$$

16) Rotated Griewank's function [27]:

$$f_{16}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{D} z_i^2 - \prod_{i=1}^{D}\cos\frac{z_i}{\sqrt{i}} + 1$$
$$\text{where } \mathbf{z} = \mathbf{xM}, \quad \mathbf{x} \in [-600, 600]^D \text{ and}$$
$$\min_{\mathbf{x}} f_{16}(\mathbf{x}) = f_{16}([0, 0, ..., 0]) = 0.$$

17) Rotated Rastrigin's function [27]:

$$f_{17}(\mathbf{x}) = \sum_{i=1}^{D}\left[z_i^2 - 10\cos(2\pi z_i) + 10\right]$$
$$\text{where } \mathbf{z} = \mathbf{xM}, \quad \mathbf{x} \in [-5.12, 5.12]^D \text{ and}$$
$$\min_{\mathbf{x}} f_{17}(\mathbf{x}) = f_{17}([0, 0, ..., 0]) = 0.$$

18) Rotated Weierstrass's function [27]: see equation at the bottom of the page.
Note: For $f_{15} - f_{18}$, $\mathbf{M}$ is an arbitrary orthogonal matrix. In this paper, the D by D matrix $\mathbf{M}$ is defined as the unity matrix output from the orthogonal-triangular decomposition of the D by D matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & ... & D \\ D+1 & D+2 & ... & 2D \\ \vdots & \vdots & \ddots & \vdots \\ D^2 - D & D^2 - D + 1 & ... & D^2 \end{bmatrix}.$$

19) Hybrid Composition function [27]:

$$f_{19}(\mathbf{x}) = \sum_{i=1}^{10} w_i(g_i(\lambda_i(\mathbf{x} - \mathbf{o_i})) + b_i)$$
where
$$\mathbf{o_i} = \{o_{i,j}\},$$
$$g_i(\lambda_i\mathbf{x}) = C \times \frac{h_i(\lambda_i\mathbf{x})}{h_i(\lambda_i[5, 5, \ldots, 5])}$$
$$h_{1-2}(\mathbf{x}) = f_7(\mathbf{x})h_{3-4}(\mathbf{x}) = f_{17}(\mathbf{x});$$
the corresponding $\mathbf{M}$ is an D by D identity matrix,
$$h_{5-6}(\mathbf{x}) = f_8(\mathbf{x})$$
$$h_{7-8}(\mathbf{x}) = f_{10}(\mathbf{x})$$
$$h_{9-10}(\mathbf{x}) = f_1(\mathbf{x})$$
$$w_i = \exp\left(-\sum_{j=1}^{D}\frac{(x_j - o_{i,j})^2}{2D\sigma_j^2}\right)$$
$$\sigma_j = 1 \text{ for } j = 1, 2, \ldots, D$$
$$\lambda_i = [1, 1, 0.1, 0.1, 12, 12, 6.4, 6.4, 20, 20]$$
$$b_i = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900]$$
$$\mathbf{o_i} = [u_i, u_i, \ldots, u_i]$$
where
$$u_i = 0.5 \times \text{floor}\left(\frac{i}{2}\right)$$
$$C = 2000$$
$$\mathbf{x} \in [-5, 5]^D$$
and
$$\min_{\mathbf{x}} f_{19}(\mathbf{x}) = f_{19}([0, 0, ..., 0]) = 4500.$$

$$f_{13}(\mathbf{x}) = (x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos x_1 + 10$$
$$\text{where } \mathbf{x} \in [-8.142, 6.858] \times [-12.275, 2.725] \text{ and}$$
$$\min_{\mathbf{x}} f_{13}(\mathbf{x}) = f_{13}([-3.142, 12.275]) = f_{13}([3.142, 2.275]) = f_{13}([9.425, 2.425]) = 0.398$$

$$f_{14}(\mathbf{x}) = g(\mathbf{x}) \times h(\mathbf{x}) \text{ where}$$
$$g(\mathbf{x}) = 1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)$$
$$h(\mathbf{x}) = 30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)$$
$$\mathbf{x} \in [-2, 2] \times [-3, 1] \text{ and } \min_{\mathbf{x}} f_{14}.(\mathbf{x}) = f_{14}([0, -1]) = 3$$

$$f_{15}(\mathbf{x}) = \sum_{i=1}^{D} 10^{6(i-1)/D-1} z_i^2 \text{ where } \mathbf{z} = (\mathbf{x} + [100, \ldots, 100])\mathbf{M}, \quad \mathbf{x} \in [-100,100]^D \text{ and}$$
$$\min_{\mathbf{x}} f_{15}(\mathbf{x}) = f_{15}([-100, -100, ..., -100]) = 0$$

$$f_{18}(\mathbf{x}) = \sum_{i=1}^{D}\sum_{k=0}^{20} a^k \cos(2\pi b^k(z_i - 0.5)) - D\sum_{i=0}^{20} a^k \cos(\pi b^k)$$

$$\text{where } \mathbf{z} = \mathbf{xM}, \quad \mathbf{x} \in [-0.5, 0.5]^D \min_{\mathbf{x}} f_{18}(\mathbf{x}) = f_{18}([0, 0, ..., 0]) = 0$$

ACKNOWLEDGMENT

The authors are grateful to Dr. K. F. Fong and Dr. C. W. Sung for stimulating discussions. They would also like to thank the six anonymous Reviewers, the anonymous Associate Editor, and Prof. Xin Yao, the Editor-in-Chief, for constructive and detailed comments

REFERENCES

[1] F. Glover and M. Laguna, *Tabu search*. Norwell, MA: Kluwer, 1997.
[2] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
[3] J. H. Holland, *Adaptation in natural and artificial systems*. Ann Arbor, MI: Univ. of Michigan Press, 1975.
[4] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 124–141, Jul. 1999.
[5] *Parameter setting in evolutionary algorithms*, G. F. Lobo, C. F. Lima, and Z. Michalewicz, Eds. New York: Springer, 2007.
[6] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. New York: Springer, 2003.
[7] M. L. Mauldin, "Maintaining diversity in genetic search," in *Proc. National Conf. Artif. Intell.*, 1984, pp. 247–250.
[8] L. Davis, *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold, 1991.
[9] T. Friedrich, N. Hebbinghaus, and F. Neumann, "Rigorous analyses of simple diversity mechanisms," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2007, pp. 1219–1225.
[10] S. Ronald, "Duplicate genotypes in a genetic algorithm," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1998, pp. 793–798.
[11] R. J. Povinelli and X. Feng, "Improving genetic algorithms performance by hashing fitness values," in *Proc. Artif. Neural Netw. Eng.*, 1999, pp. 399–404.
[12] J. Kratica, "Improving performances of the genetic algorithm by caching," *Comput. Artif. Intell.*, vol. 18, no. 3, pp. 271–283, 1999.
[13] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proc. Int. Conf. Genetic Algorithms, Lawrence Erlbaum*, 1987, pp. 41–49.
[14] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 443–462, 2002.
[15] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolutionary strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, 2001.
[16] C. K. Chow, H. T. Tsui, and T. Lee, "Surface registration using a dynamic genetic algorithm," *Pattern Recognition*, vol. 37, no. 1, pp. 105–117, 2004.
[17] K. F. Fong, V. I. Hanby, and T. T. Chow, "HVAC system optimization for energy management by evolutionary programming," *Energy and Buildings*, vol. 38, no. 3, pp. 220–231, 2006.
[18] I. Wegener, *Complexity theory*. New York: Springer, 2005.
[19] P. V. Beek, "Backtracking search algorithms," in *Handbook of Constraint Programming*, F. Rossi, P. Beek, and T. Walsh, Eds. New York: Elsevier, 2006.
[20] A. Plaat, J. Schaeffer, W. Pijls, and A. Debruin, "Best-first fixed-depth minimax algorithms," *Artif. Intell.*, vol. 87, no. 1–2, pp. 255–293, 1996.
[21] S. Y. Yuen and C. K. Chow, "A non-revisiting simulated annealing," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 1886–1892.
[22] C. K. Chow and S. Y. Yuen, "A non-revisiting particle swarm optimization," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 1879–1885.
[23] S. Y. Yuen and C. K. Chow, "A non-revisiting genetic algorithm," in *Proc. IEEE Congr. Evol. Comput.*, 2007, pp. 4583–4590.
[24] D. Hearn and M. P. Baker, *Computer Graphics with OpenGl*, 3rd Ed ed. Upper Saddle River, NJ: Prentice-Hall, 2004.
[25] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational geometry, algorithms and applications*, 2nd Ed ed. New York: Springer, 2000.
[26] X. Yao, Y. Liu, and G. M. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Apr. 1999.
[27] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Nanyang Technological Univ., Singapore, 2005, Tech. Rep..
[28] 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.
[29] 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.
[30] J. Liu, X. Fan, and Z. Qu, "An improved particle swarm optimization with mutation based on similarity," in *Proc. IEEE Int. Conf. Natural Comput.*, 2007, pp. 824–828.
[31] N. Hansen, "The CMA evolutionary strategy: A tutorial," 2007 [Online]. Available: Link: www.bionik.tu-berlin.de/user/niko/cmatutorial.pdf
[32] C. W. Sung and S. Y. Yuen, "On the analysis of the $(1 + 1)$ evolutionary algorithm with short-term memory," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2008, pp. 235–241.
[33] S. Y. Yuen and C. K. Chow, "Applying non-revisiting genetic algorithm to traveling salesman problem," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2008, pp. 2217–2224.

Shiu Yin Yuen (M'98) received the Associateship and the M.Phil. degrees from the Department of Electronic Engineering, The Hong Kong Polytechnic (now The 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, U.K., in 1992.

He is currently an Assistant Professor in the Department of Electronic Engineering, City University of Hong Kong. His main research interests are in evolutionary computation, computer vision, and machine learning. He has published more than 50 technical papers on these subjects, including more than ten international journal papers.

Dr. Yuen served as program committee member in nine international conferences and is 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*.

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

His current research interests are neural networks, machine learning, evolutionary computation, image processing, and multiagent systems.