Title	Continuous non-revisiting genetic algorithm with random search space re-partitioning and one-gene-flip mutation
Author(s)	Chow, CK; Yuen, SY
Citation	The 6th IEEE World Congress on Computational Intelligence cum IEEE Congress on Evolutionary Computation (WCCI-CEC 2010), Barcelona, Spain, 18-23 July 2010. In IEEE Transactions on Evolutionary Computation, 2010, p. 1-8
Issued Date	2010
URL	http://hdl.handle.net/10722/196678
Rights	Creative Commons: Attribution 3.0 Hong Kong License

Continuous Non-revisiting Genetic Algorithm with Random Search Space Re-partitioning and One-Gene-Flip Mutation

Chi Kin Chow and Shiu Yin Yuen

***Abstract*—In continuous non-revisiting genetic algorithm (cNrGA), the solution set with different order leads to different density estimation and hence different mutation step size. As a result, the performance of cNrGA depends on the order of the evaluated solutions. In this paper, we propose to remove this dependence by a search space re-partitioning strategy. At each iteration, the strategy re-shuffles the solutions into random order. The re-ordered sequence is then used to construct a new density tree, which leads to a new space partition sets. Afterwards, instead of randomly picking a mutant within a partition, a new adaptive one-gene-flip mutation is applied. Motivated from the fact that the proposed adaptive mutation concerns only small amount of partitions, we propose a new density tree construction algorithm. This algorithm refuses to partition the sub-regions which do not contain any individual to be mutated, which simplifies the tree topology as well as speeds up the construction time. The new cNrGA integrated with the proposed re-partitioning strategy (cNrGA/RP/OGF) is examined on 19 benchmark functions at dimensions ranging from 2 to 40. The simulation results show that cNrGA/RP/OGF is significantly superior to the original cNrGA at most of the test functions. Its average performance is also better than those of six benchmark EAs.**

I. Introduction

Continuous Non-Revisiting GA (cNrGA) [1] is an extension of (discrete) NrGA [2] to continuous variables. It uses a binary space partitioning (BSP) tree archive, namely *density tree*, to record the positions of evaluated solutions as well as to represent the density distribution of the solutions. Each node of the tree represents a sub-region in a search space. Suppose a parent node has two child nodes **l** and **r**. The sub-regions represented by **l** and **r** are disjoint and their union is the partition of the parent (i.e., the child nodes binary divide the parent sub-region).

Definition 1: The partition of **x**

Suppose **x** is a solution in the search space S, i.e. $\mathbf{x} \in S$, and S is partitioned into the partition set $H = \cup_i h_i$ by a density tree T, we define the partition $h \subseteq H$ as the '*partition of* **x**' if $\mathbf{x} \in h$ and h is represented by a leaf node of T.

Since the search space is partitioned according to the distribution of the evaluated solutions, the partition size is small (large) if the corresponding evaluated solution is close to (far from) its neighbor. Thus, the density of the evaluated solution at **x** can be estimated from the partition size of **x** (i.e. from the tree). cNrGA uses this density information to decide the mutation step size, which implements an adaptive and parameter-less mutation.

As new solutions are constantly generated and evaluated during the evolution, the density of the evaluated solutions changes from iteration to iteration. cNrGA responds to this change by inserting a new partition to the original space partitioning scheme. Obviously, different solution orders generate different space partitioning schemes. Note that the mutation step size in cNrGA is computed from the partition size; thus the sequential estimation approach infers that the step size does not only depend on the distribution of the evaluated solutions; but also depends on the order of the solutions. This introduces a subtle bias to the algorithm.

In this paper, we propose a new solution-density estimation approach to remove the dependence on the order of the solutions. At each iteration, the proposed approach re-shuffles the evaluated solutions into a random order. Then a new density tree is built from the re-ordered solution sequence. After this, we apply a new one-gene-flip mutation operator to obtain the mutation step size and to mutate an individual. Comparing to the adaptive mutation in the original cNrGA which randomly mutates within the partition, the proposed mutation is less disruptive to the schemata, which helps speed up the convergence. To maintain a fast solution-density estimation, the density tree of the re-ordered sequence is built by a modified tree construction method. This modified method does not only speed up the construction time but also simplifies the topology of the tree.

The rest of this paper is organized as follows: Section II presents a new adaptive mutation scheme that uses the space partitioning information provided by a randomly re-shuffled search history. Section III reports the improved cNrGA by this mutation. Section IV reports experimental results and section V gives the conclusion.

II. Adaptive Mutation with Search Space Re-partitioning

In this section, we present an adaptive mutation scheme for which the dependence on the order of the evaluated solutions is removed. The scheme starts by re-shuffling the order of the evaluated solutions. Afterwards, the corresponding density tree is re-built and is used to compute the mutation regions.

Chi Kin Chow is with the department of Electronic Engineering, City University of Hong Kong, Hong Kong SAR (E-mail: chowchi@cityu.edu.hk).

Shiu Yin Yuen is with the department of Electronic Engineering, City University of Hong Kong, Hong Kong SAR (E-mail: kelviny.ee@cityu.edu.hk).

The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 124409].

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

Different from the original cNrGA, a mutant is generated by a newly proposed algorithm called *One-Gene-Flip* mutation (OGF). In the following sub-section, we first present a simplified density tree construction which speeds up the tree construction process. Afterwards, we present the details of OGF.

A. Simplified Density Tree Construction

Note that for each newly generated solution, cNrGA considers only a portion of leaf nodes (i.e. the leaf nodes which contain the being mutated individuals) in density tree T to compute the mutation step size; whilst the remaining leaf nodes are unneeded. Thus, to simplify the tree topology, the sub-tree(s) of which all leaf nodes are unneeded should be pruned. Instead of first building the whole tree and then pruning those unneeded sub-tree(s), we propose an algorithm that avoids creating them during the tree construction, which simplifies the tree topology as well as shortens the construction time. Suppose $\mathbf{z}$ is an evaluated solution, the decision on either recording it into the tree or ignoring it from the tree construction depends on whether the partition of $\mathbf{z}$ contains the being mutated individual(s). If the partition does not contain any being mutated individual, the partitioning induced from $\mathbf{z}$ will not adjust the mutation regions of the being mutated individuals. It is worthwhile to ignore $\mathbf{z}$ to simplify the tree topology. In other words, each leaf node has to record its being mutated individual(s) for this tree simplification. We name this set of begin mutated individuals as the *mutation se*t, which is formally defined below:

Definition 2: The mutation set of **x, L(x)**

Suppose $\mathbf{x}$ is a leaf node of density tree T and P is a set of individuals to be mutated, we define the individual set $L \subseteq P$ as the '*mutation set of* $\mathbf{x}$' if every individual in L is inside the partition represented by $\mathbf{x}$.

Given a search space S; a set of individuals to be mutated $P = \{\mathbf{p_i}\}_{i=1,2,\ldots,\mu}$ and the evaluated solution set $Z = \{\mathbf{z_i}\}_{i=1,2,\ldots,N}$, the proposed adaptive mutation starts from randomly re-shuffling Z as $\{\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_N}\}$. Meanwhile, the mutation region of $\mathbf{p_i}$ is defined as follows:

Definition 3: The mutation region of $\boldsymbol{p}$, $\mathbf{q}$

The mutation region of $\mathbf{p}$ contains all possible mutants of $\mathbf{p}$. Suppose $\mathbf{p}$ is an individual to be mutated, the mutation region q of $\mathbf{p}$ is defined as $q = \prod_{k=1}^{D} [V(k), U(k)]$ where $V(k)$ and $D(k)$ are the lower and upper bounds at the k^{th} dimension respectively. These bounds are dynamically determined by the binary partitioning process.

All mutation regions $\{q_i\}_{i=1,2,\ldots,\mu}$ are initialized as the whole search space, i.e. $q_i := S$ for all i. Also, the density tree T is initialized to consist of the root node only. Since the root node represents the entire search space, its mutation set is P. After the initialization, the re-shuffled solutions $\{\mathbf{s_i}\}$ are recorded one by one. Suppose $\mathbf{s_i}$ is currently recorded, we first search $h(\mathbf{s_i})$: the partition of $\mathbf{s_i}$. We denote by *Curr_node* as the leaf node in T which represents $h(\mathbf{s_i})$. If the mutation set of *Curr_node*, i.e. L(*Curr_node*), is empty, the partitioning caused by $\mathbf{s_i}$ will not affect the mutation regions of $\{\mathbf{p_i}\}$ and hence $\mathbf{s_i}$ needs not be inserted to T. On the other hand, if L(*Curr_node*) is not empty, T records $\mathbf{s_i}$ by inserting a leaf node under *Curr_node*. This node insertion is equivalent to sub-dividing $h(\mathbf{s_i})$ into two partitions, and they are represented by the left child node $\mathbf{l}$ and the right child node $\mathbf{r}$ of *Curr_node*. Meanwhile, L(*Curr_node*) are also divided into two groups: the first group G_1 contains the individuals which are inside $h(\mathbf{l})$ whilst the second group G_2 contains the individuals inside $h(\mathbf{r})$, i.e. $\mathbf{a} \in h(\mathbf{l})$ for all $\mathbf{a} \in G_1$; $\mathbf{b} \in h(\mathbf{r})$ for all $\mathbf{b} \in G_2$ and L(*Curr_node*) $= G_1 \cup G_2$. Obviously, the mutation set of $\mathbf{l}$ is assigned as G_1 and the mutation set of $\mathbf{r}$ is assigned as G_2. Also, the mutation regions of all individuals in $L(\mathbf{l})$ are updated as $h(\mathbf{l})$; and the mutation regions of all individuals in $L(\mathbf{r})$ are updated as $h(\mathbf{r})$. After considering every evaluated solution in Z according to their orders (either being recorded into the tree or being ignored during the tree construction), we obtain the mutation regions of the individual $\{\mathbf{p_i}\}$. *Algorithm A1* shows the pseudo code for the simplified density tree construction algorithm.

Algorithm A1: Simplified density tree construction

Input: 1) the evaluated solution set $Z = \{\mathbf{z_i}\}_{i=1,2,\ldots,N}$, 2) the set of individuals to be mutated $P = \{\mathbf{p_i}\}_{i=1,2,\ldots,\mu}$, 3) search space S

1. Random re-shuffle the sequence of Z as $\{\mathbf{s_i}\}_{i=1,2,\ldots,N}$
2. $q_i = \prod_{k=1}^{D} [V_i(k), U_i(k)] := S$ for all $i \in [1, \mu]$
3. Initialize density tree T to consist of root node only.
4. $L(root) := P$
5. **For** $i = 1$ **to** N
6. *Curr_node* := *BSPTreeSearch*($\mathbf{s_i}$)
7. **If** $|L(Curr\_node)| > 0$ **then**
8. Suppose $\mathbf{y}$ is the evaluated solution in the partition represented by *Curr_node*.
9. Define the comparing dimension j: $j = \arg\max_{k \in [1,D]} d(\mathbf{y}, \mathbf{s_i} | k)$
10. **If** $\mathbf{s_i}(j) < \mathbf{y}(j)$ **then**
11. Create left child node $\mathbf{l}$ that records $\mathbf{s_i}$
12. Create right child node $\mathbf{r}$ that records $\mathbf{y}$
13. **Else**
14. Create left child node $\mathbf{l}$ that records $\mathbf{y}$
15. Create right child node $\mathbf{r}$ that records $\mathbf{s_i}$
16. **EndIf**
17. $L(\mathbf{l}) := \varnothing$
18. $L(\mathbf{r}) := \varnothing$
19. **For** $m = 1$ **to** $|L(Curr\_node)|$
20. Let $\mathbf{a}$ be the m^{th} element of L(*Curr_node*) and the mutation region of $\mathbf{a}$ is $\prod_{k=1}^{D} [V(k), U(k)]$

```
            If a(j) < (s_i(j) + y(j)) / 2 then
                L(l) := L(l) ∪ a
                U(j) := (s_i(j) + y(j)) / 2
            Else
                L(r) := L(r) ∪ a
                V(j) := (s_i(j) + y(j)) / 2
            EndIf
        Next m
    EndIf
Next i
```

Output: the mutation region set $\{q_i\}$

Example:

Suppose $S = [0,1]^2$ is the search space; ($\mathbf{s_1}$, $\mathbf{s_2}$, $\mathbf{s_3}$, $\mathbf{s_4}$) are the set of all evaluated solutions after being randomly re-shuffled; and $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$ are the individuals to be mutated in the next generation. The distributions of $\{\mathbf{s_i}\}$ and of $\{\mathbf{p_i}\}$ are shown in Fig. 1. The adaptive mutation on $\{\mathbf{p_i}\}$ starts by initializing the density tree T to consist of the root node only. Meanwhile, the parent set of the root node, $L(root)$, is initialized as $\{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}\}$.

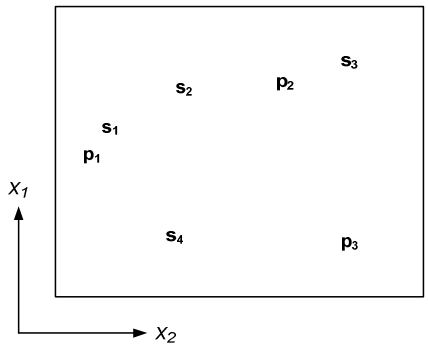

Fig. 1 Distributions of the evaluated solutions $\{\mathbf{s_i}\}$ and of the solutions to be mutated $\{\mathbf{p_i}\}$.

The first two evaluated solutions $\mathbf{s_1}$ and $\mathbf{s_2}$ divide S into two partitions. Fig. 2(a) shows the corresponding space partitioning. The dotted line represents the decision boundary. This space partitioning is equivalent to inserting two child nodes $\mathbf{s_1}$ and $\mathbf{s_2}$. Fig. 2(b) shows the topology of the corresponding T. Seen from Fig. 2(a), the partition of $\mathbf{s_2}$ consists of all individuals to be mutated. Thus, the mutation set of $\mathbf{s_1}$, $L(\mathbf{s_1})$, is assigned as $\{\mathbf{p_1}\}$ whilst the mutation set of $\mathbf{s_2}$, $L(\mathbf{s_2})$, is assigned as $\{\mathbf{p_2}, \mathbf{p_3}\}$. Meanwhile, the mutation regions of $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$ are identified as $h(\mathbf{s_1})$, $h(\mathbf{s_2})$ and $h(\mathbf{s_2})$ respectively.

When $\mathbf{s_3}$ is considered, the space partitioning is updated to that shown in Fig. 2(c), and the topology of the corresponding T is shown in Fig. 2(d). Seen from Fig. 2(c), the updated partition of $\mathbf{s_2}$ does not contain any individual whilst the partition of $\mathbf{s_3}$ contains $\mathbf{p_2}$ and $\mathbf{p_3}$. The mutation set $L(\mathbf{s_2})$ does not change and the mutation set $L(\mathbf{s_3})$ remains empty.

$\mathbf{s_4}$ is the next evaluated solution being considered. It is found that $\mathbf{s_4}$ is inside the partition of $\mathbf{s_2}$. As the mutation set of $\mathbf{s_2}$ is empty, we simply ignore $\mathbf{s_4}$ in the tree construction. Fig. 2(e) and Fig. 2(f) show the space partitioning and the topology of T after considering $\mathbf{s_4}$. In Fig. 2(e), $\mathbf{s_4}$ is in light-gray, which indicates that $\mathbf{s_4}$ is not recorded by the density tree. When all evaluated solutions are considered, the individuals $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$ are mutated in the partitions $h(\mathbf{s_1})$, $h(\mathbf{s_3})$ and $h(\mathbf{s_3})$ respectively. Fig. 3 (a) and (b) show the space partitioning by the evaluated solutions in another two solution orders: $\{\mathbf{s_1}, \mathbf{s_4}, \mathbf{s_2}, \mathbf{s_3}\}$ and $\{\mathbf{s_1}, \mathbf{s_2}, \mathbf{s_4}, \mathbf{s_3}\}$ respectively.

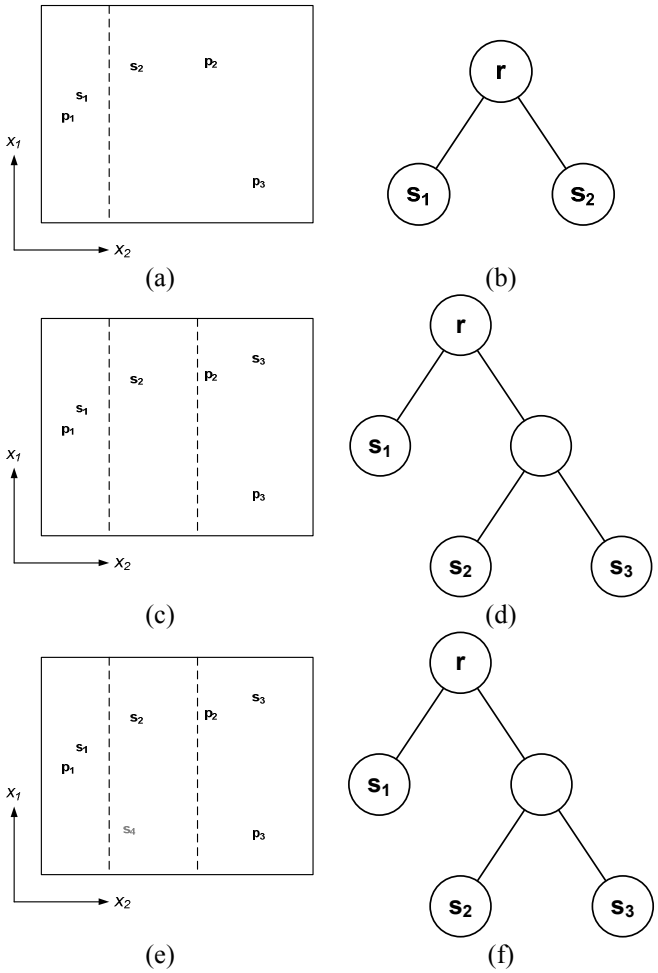

Fig. 2 Illustration of the construction of randomly re-partitioned search space.

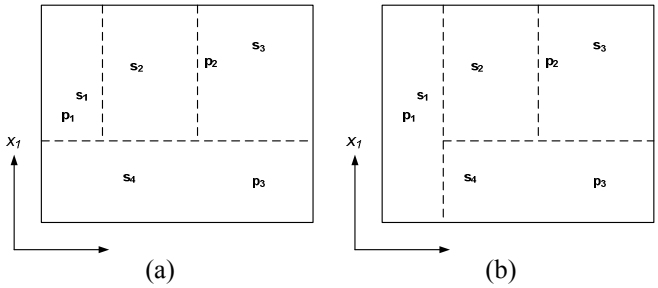

Fig. 3 The space partitioning by another two individual sequences.

B. One-Gene-Flip mutation for cNrGA

One-Gene-Flip mutation (OGF), as its name suggests, extends the one-bit-flip mutation in the conventional genetic

algorithm to handle continuous search space. Similar to one-bit-flip mutation, OGF mutates only one gene in the individual *within the partition* and this gene is randomly selected. OGF is a parameter-less adaptive mutation of cNrGA, in the sense that the mutation is done randomly within the bounds of the gene, *which are in turn defined by the partition*. Comparing to the adaptive mutation in the original cNrGA which *randomly mutates within the partition*, OGF is less disruptive to the schemata structure of the crossover. Suppose **p** is a *D*-dimensional individual to be mutated and $\prod_{k=1}^{D}[V(k), U(k)]$ is the mutation region of **p**, OGF starts from randomly selecting a dimension $j \in \{1,2,\ldots,D\}$. Then **p** is mutated as **p'** by replacing the j^{th} element of **p** with a random number in the range $[V(j), U(j)]$. The values of the genes in the rest of the dimensions are unchanged.

Algorithm A2: One-Gene-Flip mutation for cNrGA

Input: 1) the individuals to be mutated **p**, 2) the mutation region of $\mathbf{p} := \prod_{k=1}^{D}[V(k), U(k)]$

1. $\mathbf{p_i'} := \mathbf{p_i}$
2. Randomly pick a dimension $j \in \{1,2,\ldots,D\}$.
3. $\mathbf{p_i'}(j) := Rand([V(j). U(j)])$

Output: $\mathbf{p_i'}$

III. cNrGA with Adaptive Re-partitioning

cNrGA with randomly re-partitioned density tree (cNrGA/RP/OGF) is a real-coded genetic algorithm. It improves the original cNrGA in the sense that the adaptive mutation in cNrGA/RP/OGF is less sensitive to the ordering sequence of the evaluated solutions and is less disruptive to schemata structure induced by crossover. Fig. 5 shows the structure of cNrGA/RP/OGF. cNrGA/RP/OGF consists of a long-term memory and a short-term memory. The long-memory, namely *Evaluated Solution List* (ESL), is a list structure archive which records the set of all evaluated solutions. On the other hand, the density tree in cNrGA/RP/OGF is a short-term memory which represents the space partitioning scheme. Different from the original cNrGA, the density tree of cNrGA/RP/OGF will be re-built from scratch in every iteration.

Similar to a simple GA, cNrGA/RP/OGF starts by initializing the population **X** consisting of μ individuals. This population then generates μ offspring individuals $\{\mathbf{p_i}\}$ by a crossover operator. Afterwards, the adaptive mutation module uses the solutions stored in ESL to simultaneously reconstruct the density tree and obtain the mutation regions of $\{\mathbf{p_i}\}$ (i.e. *Algorithm A1*). After performing OGF mutation on $\{\mathbf{p_i}\}$, according to *Algorithm A2*, the corresponding mutant set $\{\mathbf{p_i'}\}$ are evaluated and are recorded by ESL. A $(\mu+\mu)$ elitism selection is performed to choose the best μ individuals from $\{\mathbf{X}, \{\mathbf{p_i'}\}\}$. The reproduction and the selection processes are repeated until the stopping criteria is satisfied. The circled numbers in Fig. 5 represent the order of the steps in a cNrGA/RP/OGF iteration.

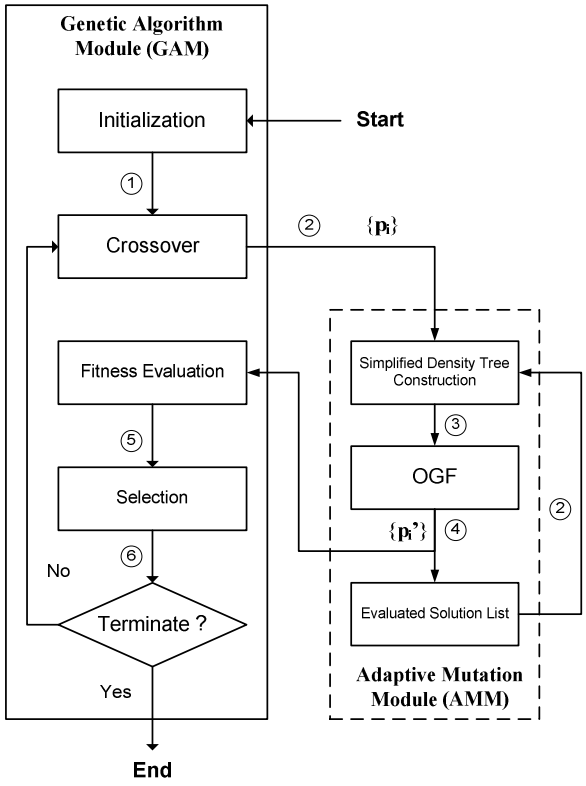

Fig. 4 Flow diagram of cNrGA/RP/OGF

IV. Experimental Results

A. Test function set

A real valued function set $\mathbf{F} = \{f_1(\mathbf{x}), f_2(\mathbf{x}),\ldots, f_{19}(\mathbf{x})\}$ consisting of 19 functions are employed to illustrate the performance of cNrGA/RP/OGF. The 19 test functions are as follows:

1. Sphere function
2. Schwefel's problem 2.22
3. Schwefel's problem 1.2
4. Schwefel's problem 2.21
5. Generalized Rosenbrock function
6. Quartic function
7. Generalized Rastrigin function
8. Generalized Griewank function
9. Generalized Schwefel's problem 2.26
10. Ackley function
11. Shekel's Foxholes function
12. Six-Hump Camel-Back function
13. Branin function
14. Goldstein-Price function
15. Rotated high conditioned elliptic function
16. Rotated Griewank's function

17. Rotated Rastrigin's function
18. Rotated Weierstrass's function
19. Hybrid Composition function

They are well known benchmark test functions. The first 14 functions are taken from [3] whilst the remaining 5 are taken from [4]. For details of the test functions (i.e. the mathematical forms, the search space and the optima), please refers to [2].

The first six functions are uni-modal functions; the remaining thirteen are multi-modal 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. Simulations are carried out to find the global minimum of each function.

B. Test algorithms

In this section, we compare the performance of cNrGA/RP/OGF with those of cNrGA and six benchmark evolutionary algorithms. The search spaces of all test algorithms are continuous. The designs and settings of cNrGA/RP/OGF and the algorithms for comparison are summarized below.

Test algorithm 1 – Continuous non-revisiting genetic algorithm [1] (cNrGA)
Test algorithm 2 – Continuous non-revisiting genetic algorithm with randomly re-partitioned BSP tree (cNrGA/RP/OGF).
Test algorithm 3 – CMA-ES [5].
Test algorithm 4 – Differential Evolution [6] (DE).
Test algorithm 5 – Opposition-Based Differential Evolution [7] (ODE)
Test algorithm 6 – Dissipative particle swarm optimization [8] (DPSO)
Test algorithm 7 – PSO with Spatial Particle Extension [9] (SEPSO)

For cNrGA/RP/OGF and cNrGA, the crossover operator is chosen to be uniform crossover where the crossover rate r_x is chosen as 0.5. This is the recommended setting in [10, pg. 48].

For PSO-class test algorithms, the values of c_1, c_2 are set to 2. The inertia w is linearly decreasing from 1 to 0. Suppose $X = \prod_{i=1,\ldots,D}[V_i, U_i]$ is the search space of a D-dimensional objective function, the maximum velocity V_{max} is set to $0.1R$ where $R = \max(U_i - V_i)$. The parameters used in DPSO and SEPSO 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, a simple velocity line bouncing with bouncing factor -1 is used. These parameter settings are recommended in the original works [9].

For DE and ODE, the crossover rate and the differential amplification factor are set to 0.95 and 0.5 respectively. These values have been used in literature [11]. The mutation strategy is DE/rand/1/bin (classic version of DE) [11]. The jumping rate constant of ODE is chosen to be 0.3 [7].

C. Simulation settings

For cNrGA/RP/OGF, cNrGA, DE and ODE, the population sizes are set to 100. (100+100) selection is used. For CMA-ES, the population size λ is chosen by the suggested setting in [5] (i.e. $\lambda = 4 + \lfloor 3\ln D \rfloor$). For DPSO and SEPSO, the swarm sizes are set to 100 and 100 offspring are reproduced at each generation. All test functions with the exception of $f_{11} - f_{14}$, which are two-dimensional, are tested with dimension 40. To provide a fair comparison of the test algorithms, the total number of function evaluations of all algorithms is kept a constant: For functions $f_1 - f_{10}$, cNrGA/RP/OGF, cNrGA, DPSO, SEPSO, DE and ODE are terminated after 400 generations. CMA-ES is terminated after 40,000 function evaluations, i.e., the total number of fitness evaluations of all the algorithms is fixed at 40,000. Similarly, for functions $f_{11} - f_{14}$, the total number of fitness evaluations is fixed at 1,000. The swarm sizes of DPSO and PSOMS are set to 50. The population sizes of cNrGA/RP/OGF, cNrGA, DE and ODE are set to 50 also. CMA-ES is terminated after 1,000 function evaluations.

Since the test algorithms are stochastic, their performances on each test function are evaluated based on statistics obtained from 100 independent runs. All simulations are done on a PC with 3.2GHz CPU and 1GB memory. The test algorithms: cNrGA/RP/OGF, cNrGA, DPSO, SEPSO and ODE are implemented in C language. CMA-ES uses source code in [5] and MATLAB version 6.1. DE uses source code in [12] and MATLAB version 6.1.

D. Simulation results

The detailed simulation results are reported in Table 2 – Table 6. Fig. 5 presents a summary of the results. The shaded cells in the figure 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 cNrGA/RP/OGF indicate the ranks of cNrGA/RP/OGF on a particular test function when it is not the best algorithm.

Seen from the figure, cNrGA/RP/OGF is superior to cNrGA. It performs better than cNrGA in all 19 test cases (using t tests, 17 of them are with 99.95% significance; 2 out of them is with 95% significance and the remaining one is with 75% significance). In addition, the performance improvement of cNrGA/RP/OGF is significant. For some of the test functions, the improvements by cNrGA/RP/OGF are even in the order of 10^2 or higher. For example, the averaged optimal fitness of f_1 found by cNrGA/RP/OGF and cNrGA are 0.0158 and 2.498376; the averaged optimal fitness found of f_{18} by cNrGA/RP/OGF and cNrGA are 0.002 and 1.804

(i.e. the optimal values of f_1 and f_{18} are 0). These results significantly show the contributions of the proposed re-partitioning scheme and One-Gene-Flip mutation.

In the comparison with all 7 test EAs, cNrGA/RP/OGF ranks or jointly ranks 1st in 7 and is 2nd in 3 out of 19 test cases. It is the second best test algorithm in terms of the number of 1st ranked test cases – CMA-ES ranks first with 10. However, to measure the performance of a test algorithm, one should simultaneously consider (1) the test cases in which it *dominates* the others and (2) the test cases in which it *is dominated* by the others. cNrGA/RP/OGF ranks between 1st and 3rd in 16 out of 19 test cases whilst CMA-ES ranks between 1st and 3rd in 11 out of 19 test cases . More importantly, cNrGA/RP/OGF never ranks lower than 5 but CMA-ES ranks 6 in 2 test cases and ranks 7th (the last) in 1 test case. Therefore though the frequency of 1st ranked test cases of CMA-ES is much more (cf. 10 vs. 7), it has the risk of having poor ranks in some cases, i.e. it is less stable than cNrGA/RP/OGF.

Table 1 lists the averaged ranks of the seven test algorithms over 19 test cases. The averaged rank of cNrGA/RP/OGF is 2.236, which is the lowest amongst the 7 test algorithms; cNrGA/RP/OGF performs the best when both accuracy and stability are considered.

The detailed simulation results (mean and standard deviation) are listed in Table 2 - Table 4. It lists the average and the standard deviation (inside brackets) of the optimal fitness for 100 trials. A value in **boldface** indicates that the corresponding algorithm is the best amongst the algorithms on a particular test function.

V. CONCLUSION

In this paper, we contribute to the continuous non-revisiting genetic algorithm (cNrGA) in the following ways:

1. We point out that, in cNrGA, the solution-density is estimated by a sequential approach; as cNrGA computes mutation step size according to the density, the performance of cNrGA depends on the order of the evaluated solutions. This introduces a subtle bias to the algorithm.
2. To balance between removing the dependence and preserving a fast estimation, we propose to randomly re-shuffle the order of the evaluated solutions at every iteration. Afterwards, we use the re-ordered solution sequence to construct the density tree for the adaptive mutation.
3. We propose an algorithm that constructs density tree whose topology is optimized for the adaptive mutation at the current generation.
4. We propose a new adaptive mutation algorithm called *One-Gene-Flip mutation*. Comparing to the mutation in the original cNrGA, it is less disruptive to the schemata, which helps speed up the convergence.

The cNrGA that employs the idea of solution reordering and one-gene-flip mutation (cNrGA/RP/OGF) is examined on 19 benchmark test functions. Its performance is compared with those of the original cNrGA as well as six bench mark evolutionary algorithms (EAs). The experimental results show that:

1. cNrGA/RP/OGF is significantly superior to the original cNrGA in all 19 test cases, which empirically illustrates that (i) the idea of search space re-partitioing and one-gene-flip mutation can improve the performance of the original cNrGA.
2. The averaged rank of cNrGA/RP/OGF over the 19 test cases is 2.263, which is the lowest amongst the 7 test EAs. Meanwhile, the variation of the ranks of cNrGA/RP/OGF is also the smallest, i.e. its rank is from 1st to 5th. On the other hand, though the frequency of 1st rank test cases obtained by CMA-ES is larger than that of cNrGA/RP/OGF, it ranks 6th in 2 test cases and ranks 7th (the last) in 1 test case. Thus cNrGA/RP/OGF outperforms the six EAs when considering both accuracy and stability.

This paper has introduced two new mechanisms to the original cNrGA, namely, solution re-ordering and one-gene-flip mutation. The new algorithm, cNrGA/RP/OGF is shown to improve the cNrGA significantly. Future work should analyze the relative contributions of each mechanism.

REFERENCES

[1] S. Y. Yuen and C. K. Chow, "Continuous non-revisiting genetic algorithm," in *Proc. IEEE Congr. Evol. Comput., 2009*, pp. 1896 – 1903, 2009.

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

[3] 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.

[4] 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", *Technical Report*, Nanyang Technological University, Singapore, May 2005 and *KanGAL Report* #2005005, IIT Kanpur, India.

[5] N. Hansen, "The CMA evolutionary strategy: A tutorial", Technical Report, code version: 31 Aug. 2007. Link: www.bionik.tu-berlin.de/user/niko/cmatutorial.pdf

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

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

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

[9] T. Krink, J. S. Vesterstrom, J. Riget, "Particle swarm optimization with spatial particle extension," in *Proc. IEEE Congr. Evol. Comput.*, 2002, pp. 1474 – 1497.

[10] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*, Springer 2003.

[11] ——, "Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization 11*, Norwell, MA: Kluwer, 1997, pp. 341–359.

[12] Differential evolution source code link: http://www.icsi.berkeley.edu/~storn/code.html

	Uni-modal						*Multi-modal*												
Function	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}
cNrGA/RP/OGF	3	2	5	3	3	3	1	3	1	2	1	1	1	1	3	1	4	3	2
cNrGA	5	5	6	6	6	4	2	5	2	4	2	2	2	2	5	4	5	5	5
CMA-ES	1	1	1	1	1	1	4	1	5	7	6	4	3	6	4	5	3	1	1
DE	4	4	7	7	4	7	3	4	3	3	7	7	7	7	2	2	2	4	7
ODE	2	3	2	2	2	2	5	2	6	1	5	6	5	5	1	3	1	2	4
DPSO	7	6	3	4	5	5	7	6	7	5	3	5	6	4	7	6	7	6	6
SEPSO	6	7	4	5	7	6	6	7	4	6	4	3	4	3	6	7	6	7	3

Fig. 5 Indicators of the best test algorithm in the 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 1 THE AVERAGED RANK OVER 19 TEST CASES

Test algorithm	**cNrGA/RP/OGF**	**cNrGA**	**CMA-ES**	**DE**	**ODE**	**DPSO**	**SEPSO**
Averaged rank	2.263	4.105	2.842	4.789	3.105	5.526	5.315

TABLE 2 AVERAGE, STANDARD DEVIATION AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY THE EIGHT TEST ALGORITHMS:$F_1 - F_7$

Fitness function		f_1	f_2	f_3	f_4	f_5	f_6	f_7
D		40	40	40	40	40	40	40
cNrGA/RP/OGF	average	0.0158	0.2799	10301.949	8.292	841.296	13.073	**10.332**
	std. dev.	(0.0044)	(0.0775)	(2634.58)	(0.9004)	(1144.01)	(0.4923)	**(2.9946)**
cNrGA	average	2.498376	4.864231	15137.309	52.476	38713.864	13.220	44.366
	std. dev.	(2.3019)	(2.0009)	(4415.30)	(6.1540)	(59449.6)	(0.6872)	(8.1339)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	95%	99.95%
CMA-ES	average	**0**	**0**	**0.000**	**0.000**	**0.000**	**0.266**	76.174
	std. dev.	**(0)**	**(0)**	**(0)**	**(0)**	**(0)**	**(0.0963)**	(17.1046)
	C (t-test)	**99.95%**	**99.95%**	**99.95%**	**99.95%**	**99.95%**	**99.95%**	99.95%
DE	average	0.167	1.1491	45895.908	60.773	2023.166	16.338	57.394
	std. dev.	(0.0279)	(0.1101)	(4761.74)	(3.8266)	(405.62)	(0.6334)	(5.3769)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	average	0.001	0.3459	409.988	0.153	42.288	12.433	170.719
	std. dev.	(0.0009)	(0.1754)	(230.1221)	(0.4848)	(13.7948)	(0.5279)	(55.5588)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
DPSO	average	6.9563	19.4899	3986.092	13.299	31765.025	14.624	201.557
	std. dev.	(1.2491)	(5.3605)	(35.7403)	(1.4465)	(110.437)	(0.8821)	(4.6532)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	average	6.8806	21.4078	6057.789	16.994	41826.011	15.630	181.948
	std. dev.	(1.5195)	(5.7894)	(41.8452)	(1.663)	(141.3864)	(1.0337)	(4.8482)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%

TABLE 3 AVERAGE, STANDARD DEVIATION AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY THE EIGHT TEST ALGORITHMS: $F_8 - F_{14}$

Fitness function		f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
D		40	40	40	40	40	40	40
cNrGA/RP/OGF	average	0.965	**-18324.08**	0.314	**0.998**	**-1.03162**	**0.3979**	**3.000**
	std. dev.	(0.0323)	**(7.9151)**	(0.06)	**(0.00001)**	**(0)**	**(0)**	**(0)**
cNrGA	average	3.468	-16885.37	5.805	**1.030**	**-1.03160**	0.39792	3.049
	std. dev.	(2.122)	(343.189)	(1.4714)	**(0.24037)**	**(0)**	(0)	(0.2557)
	C (t-test)	99.95%	99.95%	99.95%	**90%**	**99.95%**	99.95%	95%
CMA-ES	average	**0.001**	-7187.400	21.495	13.522	-1.024	**0.398**	7.320
	std. dev.	**(0.0031)**	(184.1)	(0.1658)	(5.3565)	(0.0816)	**(0)**	(16.6041)
	C (t-test)	**99.95%**	99.95%	99.95%	99.95%	80%	< 50%	99%
DE	average	1.154	-15568.75	3.047	13.974	-0.670	1.522	14.839
	std. dev.	(0.0259)	(256.12)	(0.2579)	(21.6527)	(0.3211)	(1.348)	(10.4039)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	average	0.243	-6559.977	**0.118**	2.449	-1.021	0.425	3.521
	std. dev.	(0.1488)	(839.019)	**(0.1698)**	(1.4992)	(0.0109)	(0.0277)	(0.4767)
	C (t-test)	99.95%	99.95%	**99.95%**	99.95%	99.95%	99.95%	99.95%
DPSO	average	7.005	-6196.144	6.942	1.491	-1.023	1.441	3.143
	std. dev.	(1.1996)	(27.7505)	(0.8013)	(0.84)	(0.1171)	(1.6188)	(0.6474)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	75%	99.95%	97.5%
SEPSO	average	7.433	-9108.410	7.749	2.247	-1.025	0.410	3.062
	std. dev.	(1.2926)	(30.4254)	(0.9357)	(1.1537)	(0.104)	(0.1366)	(0.2934)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	< 50%	80%	97.5%

TABLE 4 AVERAGE, STANDARD DEVIATION AND CONFIDENCE LEVEL OF THE BEST FITNESS VALUES FOUND BY THE EIGHT TEST ALGORITHMS: $F_{15} - F_{19}$

Fitness function		f_8	f_9	f_{10}	f_{11}	f_{12}
D		40	40	40	40	40
cNrGA/RP/OGF	average	37739.775	**0.968**	27711.047	0.002	145.617
	std. dev.	(21876.767)	**(0.2116)**	(2001.1961)	(0.0004)	(62.4854)
cNrGA	average	8643721.585	10.009	150157.060	1.804	196.013
	std. dev.	(7718188.14)	(2.185494)	(30256.914)	(1.299734)	(35.577081)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%
CMA-ES	average	116566.807	10.575	6698.980	**0.000**	**0.000**
	std. dev.	(76286.018)	(18.654)	(324.05)	**(0)**	**(0)**
	C (t-test)	99.95%	99.95%	99.95%	**99.95%**	**99.95%**
DE	average	37580.665	3.392	2454.875	0.028	425.763
	std. dev.	(6525.303)	(0.2111)	(77.024)	(0.0059)	(42.4568)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%
ODE	average	**254.327**	4.209	**2003.177**	0.001	164.558
	std. dev.	**(302.6105)**	(0.8853)	**(163.5926)**	(0.0005)	(38.622)
	C (t-test)	**99.95%**	99.95%	**99.95%**	99.95%	99%
DPSO	average	15086214.98	19.525	11335533.32	7.013	278.654
	std. dev.	(2359.020)	(1.3333)	(1009.9892)	(1.8273)	(10.7637)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	99.95%
SEPSO	average	13793165.09	20.951	365663.074	7.460	155.882
	std. dev.	(2260.6129)	(1.5315)	(398.9803)	(1.886)	(6.205)
	C (t-test)	99.95%	99.95%	99.95%	99.95%	90%