Black-Box Optimization Benchmarking of PRCGA on Noisy Testbed

Babatunde A. Sawyerr
Department of Computer Sciences
University of Lagos
Lagos, Nigeria
bsawyerr@unilag.edu.ng

Ebun-Oluwa P. Fasina
Department of Computer Sciences
University of Lagos
Lagos, Nigeria
efasina@unilag.edu.ng

Omowunmi O. Adeyemo
Department of Computer Sciences
University of Ibadan
Ibadan, Nigeria
wumiglory@yahoo.com

ABSTRACT
Real-coded genetic algorithms (RCGAs) are a class of probabilistic optimization algorithms developed specifically for the continuous parameter optimization domain. Projection-based RCGA (PRCGA) is a hybrid RCGA that is benchmarked on a set of 30 noisy testbed functions. The recently introduced COMparison of Continuous Optimizers (COCO) methodology was used in carrying out the experiment reported in this paper. In the experiment, PRCGA was implemented as a generational RCGA with a multiple independent restart mechanism, tournament selection, α-blend crossover, nonuniform mutation and a mechanism to prevent stagnation and premature convergence. The maximum number of function evaluations (#FEs) for each test run is \(10^8\) times the problem dimension. The results of this experiment show that PRCGA is able to solve more than half of the test functions with the dimension up to 40 with lower precision and can only solve about four test functions to the desired level of accuracy of \(10^{-8}\).

Keywords
Benchmarking, Black-box optimization, Real-coded genetic algorithm, Projection

1. INTRODUCTION
Real-coded genetic algorithms (RCGAs) are a class of evolutionary algorithms (EAs) designed specifically for the continuous parameter optimization domain. RCGAs use real-valued vectors to represent individual solutions instead of the classical binary representation used in binary-coded genetic algorithms (BCGAs). RCGAs were introduced to address the drawbacks of BCGAs. A major drawback of BCGAs is the expensive computational cost of handling problems with large search space. Despite the advantages of RCGAs over BCGAs, RCGAs have the problem of falling into premature convergence. This is partly due to lack of population diversity and high selection pressure. Another drawback of RCGAs is that they do not exploit the local basins of solutions in the population [17].

To address this problem, hybridizing RCGAs with local optimization algorithms was proposed and has been used successfully by several researchers who have shown that the performance of RCGAs has been greatly improved by hybridization. In this section, some recent examples of hybrid RCGAs are highlighted.


Over the last three decades, several test functions have been used to benchmark different algorithms by researchers in the optimization and Evolutionary Computation community. In 2005, two groups of researchers independently proposed two sets of benchmark testbeds. Ali et al [1], proposed a set of test functions with varying degrees of difficulty and Suganathan et al [18] also proposed a set of testbed functions called CEC 2005. These two testbeds have served as the basic testbeds for benchmarking optimization algorithms.

In 2009, Hansen et al [10, 11] proposed two sets of testbed functions namely noiseless and noisy testbeds for the BBOB-2009 workshop. These testbeds were designed to determine the potentials and weaknesses of an algorithm. The framework can determine the differences between algorithms at every stage of the search. The COMparison of Continuous Optimizers (COCO) methodology was chosen because it consist of two sets of carefully chosen scalable noiseless and noisy benchmark functions that represent real world problems with varying levels of difficulties.

PRCGA was benchmarked on the noiseless testbed and its performance in the BBOB 2013 workshop was not satisfactory overall but interesting because it was able outperform all its competitors in solving one of the most difficult problems in the workshop. This observation serves as a motivation to extend the benchmark of PRCGA to the BBOB noisy testbeds. The noisy testbed functions were designed to mimic real-world problems. PRCGA is a hybrid RCGA that is equipped with a novel search operator called projection. Projection is an idea borrowed from the projection of vectors in linear algebra. The use of projection as a search operator in evolutionary computation was reported for the first time in [15, 17].

The remaining part of the paper is organized as follows:
Section 2 presents the PRCGA algorithm. Section 3 presents the Experimental Procedure used in this work while Section 4 provides the CPU Timing Experiments, Section 5 presents the results and discussion. Finally in Section 6, conclusion is given.

2. THE PRCGA ALGORITHM

The PRCGA algorithm in Algorithm 1, starts by creating an initial population $P_{t=0}$ from the solution search space $S$ randomly at generation $t = 0$. $P_{t=0}$ is evaluated and the genetic operators are applied to $P_{t=0}$ to create a new population of offspring solutions $O_{t=0}$. Using a replacement strategy, $O_{t=0}$ is used to replace $P_{t=0}$ to give $P_{t=1}$. The genetic operators are applied to the new population of solutions $P_{t+1}$ and the evolutionary process continues until an optimal solution is found or the maximum number of generation is satisfied. The PRCGA algorithm is a generational RCGA with four operators:

- Selection
- Crossover
- Mutation
- Projection

Selection, crossover and mutation are the ‘genetic operators’ while the projection operator used in this algorithm was introduced for the first time in evolutionary computation in [15, 17].

Algorithm 1. The PRCGA Algorithm

\textbf{Input:} Fitness function $f$; Parameters
\textbf{Output:} Best solution $x_{best}; f(x_{best})$

1. Initialize $P_{t=0}, P_{t} = \{x_{1,t}, x_{2,t}, \ldots, x_{N,t}\}$ from $S$
2. $f(x_{i,t}) = \text{evaluate}(P_{t}), \{1 \leq i \leq N\}$
3. While not stopping condition, do steps 4 - 12
4. If $\sigma(f(P_{t})) \leq \epsilon$ do step 5 else step 6
5. $P_{t} = \text{perturb}(P_{t})$
6. $P_{t} = \text{tournamentSelection}(P_{t})$
7. $O_{t} = \text{blend-}\alpha\text{-Crossover}(P_{t}, p_{c})$
8. $M_{t} = \text{non-uniformMutation}(O_{t}, p_{m})$
9. $\Phi_{t} = \text{projection}(M_{t})$
10. $f(x_{i,t}) = \text{evaluate}(\Phi_{t})$
11. $P_{t+1} = \text{replace}(P_{t}, \Phi_{t})$
12. $t = t + 1$
13. end while

2.1 Selection

Selection operators in GAs are used to guide the evolutionary search by selecting solutions with very good fitness values in the current population for mating, i.e., solutions with higher fitness in relation to other members of the same set are selected for mating. The reason is that the solutions with higher fitness value in a population possess good properties that, if paired and mated are passed on to the offspring produced. Examples of selection operators include fitness proportional selection (roulette wheel selection), ranked based selection, tournament selection, etc.

In this work, the tournament selection was used because it maintain a good selection pressure. A GA with good selection pressure will not easily fall into premature convergence. Tournament selection was used to select $\tau$ number of solutions uniformly at random from $P_{t}$ with replacement, where $\tau$ is the tournament size and $\tau$ is less than $N$, the size of the population $P_{t}$. Secondly, the selected solutions in $\tau$ are compared using their fitness values and the best solution is selected and assigned to $P_{t}$, the mating pool. This procedure is repeated $m$ times to populate $P_{t} = \{x_{1,t}, x_{2,t}, \ldots, x_{m,t}\}$, where $m$ is the size of the mating pool and $m \leq N$.

2.2 Crossover

The crossover operator is the primary genetic operator that is used to pass on some of the parents’ genes to their offspring through mating. The offspring inherit traits from the participating parents. Several crossover operators have been proposed for RCGAs. Examples include, Arithmetic crossover, Blend-\alpha crossover, Heuristic crossover, etc. In this work, the Blend-\alpha crossover was used.

At every generation $t$, two parent solutions $(x_{i,t}, x_{j,t})$ are selected randomly and blend-\alpha crossover is carried out on the pair, if a randomly generated number $\mu, (0 \leq \mu \leq 1)$ is greater than the specified crossover probability threshold, i.e., $\mu_{i} > p_{c}$. Blend-\alpha crossover is used to uniformly generated the offsprings $(o_{1,t}, o_{2,t})$ from the interval $[\min(x_{i,t}^{k}, x_{j,t}^{k}) - \alpha \times d^{k}, \max(x_{i,t}^{k}, x_{j,t}^{k}) + \alpha \times d^{k}]$ as follows

$$o_{1,t}^{k} = (\min(x_{i,t}^{k}, x_{j,t}^{k}) - \alpha \times d^{k}, \max(x_{i,t}^{k}, x_{j,t}^{k}) + \alpha \times d^{k})$$
$$o_{2,t}^{k} = (\min(x_{i,t}^{k}, x_{j,t}^{k}) - \alpha \times d^{k}, \max(x_{i,t}^{k}, x_{j,t}^{k}) + \alpha \times d^{k}),$$

where $k = 1, 2, \ldots, n$ represents the index of a component of $o_{1,t}, (1 \leq j \leq N), \alpha = 0.3 + 0.2 \times z$, $z$ is a uniform random number drawn from the interval $[0, 1]$, $d^{k} = |x_{i,t}^{k} - x_{j,t}^{k}|$. The new pair $(o_{1,t}, o_{2,t})$ is then copied to the set $O_{t}$, otherwise the pair $(x_{i,t}, x_{j,t})$ is copied to $O_{t}$.

2.3 Mutation

The mutation operator is used for exploring the search space through some random jump within the search region. There are several mutation operators for RCGA but in this work the non-uniform mutation [14] was applied to the components of each member of $O_{t}$ with probability $p_{m}$ as follows

$$m_{i,t}^{k} = \begin{cases} o_{i,t}^{k} + \Delta(t, u^{k} - o_{i,t}^{k}) & \text{if } a \leq 0.5, \\ o_{i,t}^{k} - \Delta(t, o_{i,t}^{k} - l^{k}) & \text{otherwise}. \end{cases}$$

The mutated individual $m_{i,t}^{k}$ is copied to the set $M_{t}$, otherwise $o_{i,t}^{k}$ is copied to $M_{t}$, $a$ is a uniformly distributed random number in the interval $[0, 1]$, $u^{k}$ and $l^{k}$ are the upper and lower boundaries of $x \in S$, respectively. The function
2.4 Projection

The projection-based operator works by projecting a vector, $y$ onto another vector, $x$ to give a displacement of the positions of the two points to create a third point, $\hat{y}$. This operation is similar to the way crossover operators work in genetic algorithms except that the projection operator produces only one offspring per operation. The projection operator is able to find good solutions through fitness guided search in the solution landscape. It performs exploration at an earlier stage of the genetic search and exploitation at the later stage. Figure 1 provides an illustration of projecting a vector $x$ on another vector $y$.

Projection operation is used to generate $\Phi_t$ from $M_t$ by randomly taking a pair of solutions, $(m_{i,t}, m_{j,t})$, for each $m_{i,t} \in M_t$, $(m_{i,t}, m_{j,t})$ are evaluated, if $f(m_{i,t})$ is better than $f(m_{j,t})$ then $m_{j,t}$ is projected on $m_{i,t}$ otherwise $m_{i,t}$ is projected on $m_{j,t}$ and a projected solution $\phi_{i,t}$ is $\Phi_t$ is created. This operation is defined by Eq. (4).

$$\phi_{i,t} = \frac{m_{j,t} \cdot m_{i,t}}{m_{i,t} \cdot m_{i,t}} m_{i,t} = \frac{m_{j,t} \cdot m_{i,t}}{||m_{i,t}||^2} m_{i,t}$$

$$= \left( \frac{||m_{j,t}|| \cos(\theta)}{||m_{i,t}||} \right) m_{i,t}$$

(4)

Note that the projected vector $\phi_{i,t}$ (the offspring) will be in the same direction as $x_{i,t}$ unless $\frac{\pi}{2} < \theta < \frac{3\pi}{2}$ in which case the angle $\theta$ between the two vectors is such that $\cos(\theta) < 0$, thereby projecting the vector in the opposite direction.

Sometimes the components $\phi_{i,t}^k$ of the trial point $\phi_{i,t}$ may fall outside the search space. In such cases, the corresponding component $\phi_{i,t}^k$ is regenerated. After the projected vector is generated, its fitness value $f(\phi_{i,t})$ is determined.

After the projected vector is generated, its fitness value $f(\phi_{i,t})$ is determined and a new population, $P_{t+1}$, is created with $x_{i,t}$, where,

$$x_{i,t} = \begin{cases} \phi_{i,t} & \text{if } f(\phi_{i,t}) < f(m_{i,t}), m_{i,t} \in M_t \\ m_{i,t} & \text{otherwise.} \end{cases}$$

(5)

2.5 Replacement Strategy

There are two replacement strategies in genetic algorithms, namely generational and steady state models. The replacement strategy adopted in this work is the generational model [7]. The generational replacement strategy systematically replaces the whole parent population $P_t$ of size $N$ with the offspring population $P_{t+1}$.

Finally, elitism is used to ensure that the best solution in the current population $P_t$ survives and is carried over to the next population $P_{t+1}$.

2.6 Stagnation prevention mechanism

A stagnation prevention mechanism was introduced to prevent premature convergence of the algorithm. The mechanism works by measuring the population diversity of $P_t$. This was done by calculating the standard deviation, $\sigma (f(P_t))$ of the fitness values $f(P_t)$ of all solutions $x_{i,t}, \in P_t$. If $\sigma (f(P_t)) \leq \epsilon = 10^{-12}$, a very small positive value and the global minimum $f(x_{\text{min}})$ is not yet found, then $x_{i,t}, \in P_t$ are sorted according to their fitness values and the top 10% preserved. The remaining 90% of $x_{i,t}, \in P_t$ are replaced with uniformly generated random values from the interval $[-4, 4]^D$. The resultant population is the mating pool, $P_t$. On the other hand, if $\sigma (f(P_t)) > \epsilon$ then tournament selection is applied on $P_t$ to create the mating pool $P_t$ [16].

3. EXPERIMENTAL PROCEDURE

The experimental setup was carried out using the COMparison of Continuous Optimizers (COCO) methodology [9] on the benchmark functions provided in [8, 12]. Two independent restart strategies were used for PRCGA. A restart strategy is used to restart the experiment whenever the population of solutions stagnates or when the maximum number of generations have been met and $f(x_{\text{min}})$ is not found. For each restart strategy, the algorithm is restarted with an initial population $P_0$ uniformly sampled from the search space $[-4, 4]^D$ according to [16].

There are two stopping conditions for the restart strategies. The first restart strategy checks if the best solution obtained so far and during the last $50 + 25 \times D$ generations do not vary by more than $10^{-12}$ as in [4] while the second restart condition is when the maximum number of generations is satisfied and $f(x_{\text{min}})$ is not found. Whenever the stopping conditions are met, the algorithm is reinitialized and restarted from the beginning without using any information from the previous run.

3.1 The COCO Framework

Numerical experiment of PRCGA was carried out on a testbed consisting of 30 noisy test functions [8, 12]. These functions are carefully designed to reflect the difficulties inherent in real-world application problems. The functions are categorized as functions with moderate noise, function with severe noise and highly multimodal functions with severe noise. All the functions are scalable with the dimension $D$ ranging from 2 to 40 and their search domain is $[-5, 5]^D$.

A successful run for an algorithm is achieved if the best solution found satisfy $f_t = f_{\text{opt}} + \Delta f_t$, where $f_{\text{opt}}$ is the optimal function value, $\Delta f_t$ is the precision to reach and $\Delta f_t \in [10^{-8}, 10^2]$. If an algorithm solves a function to the desired precision value of $10^{-8}$, then it has found the optimum and it has also solved the different optimization prob-
lems on the way to the optimum. If on the other hand it cannot reach the optimum, it would have solved the function partially.

3.2 Parameter Settings

The parameters used for PRCGA are:

4. CPU TIMING EXPERIMENTS

The CPU timing experiment conducted for PRCGA used the same independent restart strategies on the function $f_6$ for a duration of 30 seconds on an Intel(R) Core(TM) i7 – 2600 CPU processor, running at 3.40GHz under a 64-bit Microsoft Windows 7 Enterprise N with 4.00GB RAM and Matlab 7.10(R2010a).

The time per function evaluation was $5.2, 5.4, 5.3, 6.0, 8.3$ times $10^{-5}$ and $1.3$ times $10^{-4}$ seconds for PRCGA in dimensions 2, 3, 5, 10, 20 and 40 respectively.

5. RESULTS

The results of PRCGA from experiments carried out according to [9] on the benchmark functions given in [8, 13] are presented in Figures 2, 3 and 4 in Tables 2 and 3. Figures 2 shows the performance of PRCGA on all the noisy benchmark functions with dimensions 2, 3, 5, 10, 20 and 40. The performance of PRCGA on the noisy benchmarks is not too impressive.

PRCGA was able to solve problems $f_{101}, f_{103}, f_{105}$ and $f_{110}$ up to dimension 40 with the desired accuracy of $10^{-8}$. In dimension 20 it was able to solve problems $f_{107}, f_{109}$ and $f_{128}$ while it solved problems $f_{113}, f_{114}$ and $f_{129}$ in dimension 5.

PRCGA succeeded in solving more than half of the test problems with the precision of $10^{-2}$. It also performed better than the RCAGA in [19] in test problems $f_{106}, f_{114}, f_{122}$ and $f_{124}$ but generally the RCAGA in [19] solved more test problems at the required level of accuracy than PRCGA.

6. CONCLUSION

The performances of RCGAs on noisy black-box optimization testbed have been average, despite the fact that RCGAs are robust solvers and have been used to solve a wide variety of problems. The benchmarking of RCGAs on noisy BBOB testbed shows that RCGAs can still be improved through hybridization.

Currently, the authors are examining ways of improving RCGAs using different hybridization schemes.

7. REFERENCES


Figure 2: Expected number of $f$-evaluations (ERT, lines) to reach $f_{\text{opt}} + \Delta f$; median number of $f$-evaluations (+) to reach the most difficult target that was reached not always but at least once; maximum number of $f$-evaluations in any trial (×); interquartile range with median (notched boxes) of simulated runlengths to reach $f_{\text{opt}} + \Delta f$; all values are divided by dimension and plotted as $\log_{10}$ values versus dimension. Shown are $\Delta f = 10^{(1, 0, -1, -2, -3, -5, -8)}$. Numbers above ERT-symbols (if appearing) indicate the number of trials reaching the respective target. The light thick line with diamonds indicates the respective best result from BBOB-2009 for $\Delta f = 10^{-8}$. Horizontal lines mean linear scaling, slanted grid lines depict quadratic scaling.
Figure 3: Empirical cumulative distribution functions (ECDFs), plotting the fraction of trials versus running time (left subplots) or versus $\Delta f$ (right subplots). The thick red line represents the best achieved results. Left subplots: ECDF of the running time (number of function evaluations), divided by search space dimension $D$, to fall below $f_{opt} + \Delta f$ with $\Delta f = 10^k$, where $k$ is the first value in the legend. Right subplots: ECDF of the best achieved $\Delta f$ divided by $10^k$ (upper left lines in continuation of the left subplot), and best achieved $\Delta f$ divided by $10^{-8}$ for running times of $D, 10D, 100D, \ldots$ function evaluations (from right to left cycling black-cyan-magenta). The legends indicate the number of functions that were solved in at least one trial. FEvals denotes number of function evaluations, $D$ and DIM denote search space dimension, and $\Delta f$ and $Df$ denote the difference to the optimal function value. Light brown lines in the background show ECDFs for target value $10^{-8}$ of all algorithms benchmarked during BBOB-2009.
Table 1: Parameter Settings for the Experiment

<table>
<thead>
<tr>
<th>Sno.</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Population size</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>Maximum number of generation</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>Mutation probability (p&lt;sub&gt;m&lt;/sub&gt;)</td>
<td>0.15</td>
</tr>
<tr>
<td>4</td>
<td>Crossover probability (p&lt;sub&gt;cx&lt;/sub&gt;)</td>
<td>0.8</td>
</tr>
<tr>
<td>5</td>
<td>Tournament size η&lt;sub&gt;tour&lt;/sub&gt;</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>Elitism (E)</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Maximum number of evaluation #FEs</td>
<td>10&lt;sup&gt;3&lt;/sup&gt; x D</td>
</tr>
<tr>
<td>8</td>
<td>Non-uniformity factor for the mutation β</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>Crafting effort CE [9]</td>
<td>0</td>
</tr>
</tbody>
</table>

5-D

<table>
<thead>
<tr>
<th>f100</th>
<th>11</th>
<th>37</th>
<th>44</th>
<th>49</th>
<th>62</th>
<th>60</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>f101</td>
<td>7</td>
<td>13</td>
<td>18</td>
<td>23</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31</td>
<td>36</td>
<td>41</td>
<td>45</td>
<td>50</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td></td>
<td>56</td>
<td>61</td>
<td>66</td>
<td>70</td>
<td>75</td>
<td>80</td>
</tr>
</tbody>
</table>

20-D

<table>
<thead>
<tr>
<th>f110</th>
<th>11</th>
<th>37</th>
<th>44</th>
<th>49</th>
<th>62</th>
<th>60</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>f111</td>
<td>7</td>
<td>13</td>
<td>18</td>
<td>23</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31</td>
<td>36</td>
<td>41</td>
<td>45</td>
<td>50</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td></td>
<td>56</td>
<td>61</td>
<td>66</td>
<td>70</td>
<td>75</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 2: Expected running time (ERT in number of function evaluations) divided by the best ERT measured during BBOB-2009 (given in the respective first row) for different Δf values for functions f<sub>1</sub>-f<sub>23</sub>. The median number of conducted function evaluations is additionally given in italics, if ERT(10<sup>-7</sup>) = ∞. #succ is the number of trials that reached the final target f<sub>opt</sub> + 10<sup>-4</sup>.

Table 2 continued...

References:


Table 3: ERT loss ratio compared to the respective best result from BBOB-2009 for budgets given in the first column (see also Figure 4). The last row RLUS/D gives the number of function evaluations in unsuccessful runs divided by dimension. Shown are the smallest, 10%-ile, 25%-ile, 50%-ile, 75%-ile and 90%-ile value (smaller values are better). The ERT Loss ratio equals to one for the respective best algorithm from BBOB-2009. Typical median values are between ten and hundred.

<table>
<thead>
<tr>
<th>#FEs/D</th>
<th>f101–f130 in 5-D, maxFE/D=100012</th>
<th>f101–f130 in 20-D, maxFE/D=100007</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>best</td>
<td>10%</td>
</tr>
<tr>
<td>2</td>
<td>0.55</td>
<td>0.73</td>
</tr>
<tr>
<td>10</td>
<td>0.41</td>
<td>0.73</td>
</tr>
<tr>
<td>100</td>
<td>0.68</td>
<td>0.97</td>
</tr>
<tr>
<td>1e3</td>
<td>0.99</td>
<td>1.6</td>
</tr>
<tr>
<td>1e4</td>
<td>0.10</td>
<td>3.6</td>
</tr>
<tr>
<td>1e5</td>
<td>0.30</td>
<td>1.9</td>
</tr>
<tr>
<td>RLUS/D</td>
<td>5e4</td>
<td>5e4</td>
</tr>
</tbody>
</table>

RR-6829, INRIA, 2009.


