Benchmarking Projection-Based Real Coded Genetic Algorithm on BBOB-2013 Noiseless Function Testbed

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

Aderemi O. Adewumi  
School of Mathematics, Statistics and Computer Science  
University of KwaZulu-Natal  
Westville, South Africa  
Adewumia@ukzn.ac.za

Montaz M. Ali  
School of Computational and Applied Mathematics  
University of The Witwatersrand  
Johannesburg, South Africa  
Montaz.Ali@wits.ac.za

ABSTRACT

In this paper, a real-coded genetic algorithm (RCGA) which incorporates an exploratory search mechanism based on vector projection termed projection-based RCGA (PRCGA) is benchmarked on the noisefree BBOB 2013 testbed. It is an enhanced version of RCGA-P in [22, 23]. The projection operator incorporated in PRCGA shows promising exploratory search capability in some problem landscape. PRCGA is equipped with a multiple independent restart mechanism and a stagnation alleviation mechanism. The maximum number of function evaluations (#FEs) for each test run is set to $10^5$ times the problem dimension. PRCGA shows encouraging results on several problems in the low and moderate search dimensions. It is able to solve each type of problem with the dimension up to 40 with lower precision but not all the functions to the desired level of accuracy of $10^{-8}$ especially for high conditioning and multi-modal functions within the specified maximum #FEs.

Categories and Subject Descriptors

G.1.6 [Numerical Analysis]: Optimization—global optimization, unconstrained optimization; F.2.1 [Analysis of Algorithms and Problem Complexity]: Numerical Algorithms and Problems

Keywords

Benchmarking, Black-box optimization, Real-coded Genetic Algorithm, Projection

1. INTRODUCTION

Genetic algorithms (GAs) are a class of stochastic algorithms based on the notion of natural selection and natural genetics by Charles Darwin. The simple genetic plan was developed in 1975 by John Holland [18]. GAs later became popular largely due to the outstanding work of the students of Holland especially Kenneth De Jong [10] and David Goldberg [14]. Subsequently, several variants of GAs have been used to solve many real-world optimization problems arising from diverse fields [2, 11, 14, 20].

The simple GA consists of a set of binary strings of potential solutions called chromosomes, a selection operator, a crossover operator and a mutation operator. The selection operator selects solutions for mating based on the principle of ‘survival of the fittest.’ The crossover operator generates new solution pairs by mixing the genetic materials of the selected parent chromosomes. The mutation operator is applied, with low probability, to the population of chromosomes to prevent premature convergence of the solutions to local optimum [14].

Binary string GAs, also known as Binary-coded genetic algorithms (BCGAs) are robust search algorithms that have been successfully used to solve several challenging problems but are computationally expensive in solving continuous and large scale optimization problems [13].

Real-coded genetic algorithms (RCGAs) are RCGAs with real-valued chromosomes. They are designed to tackle the problems encountered by BCGAs in the continuous parameter optimization domain. Recent works on RCGAs can be found in [1, 2, 7, 8, 9, 19, 20]. Despite the advantages of RCGAs in the continuous parameter domain they are still susceptible to the problem of premature convergence, therefore hybridization has been employed by researchers to prevent RCGAs from falling into premature convergence [3, 5, 6].

The projection-based exploration search method designed for RCGA in [22, 23] is based on the concept of orthogonal projection of a vector $x$ on a vector $y$. Figure 1 provides an illustration of projecting a vector $x$ on another vector $y$, a well known concept in linear algebra but relatively new to the field of evolutionary computation. For a detailed description of this concept see [23].

In this paper, an enhanced projection-based RCGA was developed using the well-known tournament selection, blend-$\alpha$ crossover and non-uniform mutation operators to drive the genetic search.

2. PRCGA ALGORITHM

PRCGA consists of five operators namely: tournament selection, blend-$\alpha$ crossover, non-uniform mutation, projection and a stagnation alleviation mechanism. The outline of PRCGA is shown in Algorithm 1.
The notations used are defined as:

Let \( x^* \) be the global minimizer of the objective function, \( f : S \subset \mathbb{R}^n \rightarrow \mathbb{R} \) if \( f(x^*) \leq f(x), \forall x \in S \). The point \( x = (x^1, x^2, \ldots, x^n)^T \in \mathbb{R}^n \). The domain of \( S \), is defined by specifying upper \( (u^i) \) and lower \( (l^i) \) limits of each \( j^{th} \) component of \( x \), i.e., \( l^j \leq x^j \leq u^j \) and \( l^j, u^j \in \mathbb{R}, j = 1, 2, \ldots, n \). Without loss of generality, only minimization problems are considered since maximizing \( f \) is equivalent to minimizing \(-f\).

\( P_t \) denotes the population of solutions at time \( t \), \( N \) is the number of solutions in \( P_t \) i.e. the population size, \( \sigma(f(P_t)) \) represents the standard deviation function, \( \zeta_t \) is the standard deviation of the fitness values \( f(P_t) \) of all solutions \( x_{i,t} \in P_t \), \( \hat{P}_t \) is the mating pool containing the parent solutions, \( C_t \) is the population of offspring solutions obtained after applying crossover on the parents in \( \hat{P}_t, p_c \) is the crossover probability, \( M_t \) is the resultant population of solutions after applying mutation on \( C_t, p_m \) is the mutation probability, \( \Phi_t \) is the population of solutions obtained after projection has been applied to \( M_t \) and \( \epsilon = 10^{-12} \), a very small positive value.

For each genetic run in Algorithm 1, \( P_0 \) is initialized from the search space \( S \) and the fitness value \( f(x_{i,0}), \forall x_{i,0} \in P_0 \) is calculated. At each time step \( t \), the population diversity of \( P_t \) is measured by calculating the standard deviation \( \zeta_t \) as follows

\[
\zeta_t = \sigma(f(P_t)). \tag{1}
\]

If \( \zeta_t \leq \epsilon \) and the global optimum has not been found, then 90% of \( P_t \) is refreshed with newly generated solutions using the function perturb(\( P_t \)). \( P_t \) is refreshed by sorting the solutions according to their fitness values and preserving the top 10% of \( P_t \). The remaining 90% of \( P_t \) are replaced with uniformly generated random values from the interval \([-4, 4]^D\). The resultant population is the mating pool, \( \hat{P}_t = (x_{1,t}, x_{2,t}, \ldots, x_{m,t}) \), where \( m \) is the size of the mating pool and \( m \leq N \). On the other hand, if \( \zeta_t > \epsilon \) then tournament selection is applied on \( P_t \) to create the mating pool \( \hat{P}_t \).

Tournament selection was used to select \( \eta_{\text{tour}} \) number of solutions uniformly at random from \( P_t \), where \( \eta_{\text{tour}} \) is the tournament size, \( \eta_{\text{tour}} < N \). The fitness values of the selected individuals in \( \eta_{\text{tour}} \) are compared and the best individual is selected and assigned to \( \hat{P}_t \), the mating pool. This procedure is repeated \( m \) times to populate \( \hat{P}_t \). All individual in \( P_t \) have the probability of being selected several times, i.e., tournament selection with replacement was used.

**Algorithm 1. The PRCGA Algorithm**

**Input:** Fitness function \( f \); Parameters

**Output:** Best solution \( x_{\text{best}}; f(x_{\text{best}}) \)

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. \( \zeta_t = \sigma(f(P_t)), \) if \( \zeta_t \leq \epsilon \) do step 5 else step 6
5. \( \hat{P}_t = \text{perturb}(P_t) \)
6. \( \hat{P}_t = \text{tournamentSelection}(P_t) \)
7. \( C_t = \text{blend-}\alpha\text{Crossover}(\hat{P}_t, P_c) \)
8. \( M_t = \text{non-uniformMutation}(C_t, p_m) \)
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

Blend-\( \alpha \) crossover is carried out on a pair \((x_{i,t}, x_{k,t})\) when a randomly generated number \( \mu_t \) \((0 \leq \mu_t \leq 1\) is greater than the specified crossover probability threshold, i.e., \( \mu_t > p_c \). Blend-\( \alpha \) crossover uniformly draws the new pair of offspring \((c_{1,t}, c_{2,t})\) from the interval \([\min(x_{i,t}, x_{k,t}) - \alpha \ast d_t, \max(x_{i,t}, x_{k,t}) + \alpha \ast d_t]\),

\[
c_{1,t} = (\min(x_{i,t}, x_{k,t}) - \alpha \ast d_t, \max(x_{i,t}, x_{k,t}) + \alpha \ast d_t) \tag{2}
\]

where \((1 \leq k \leq N)\), \( \alpha = 0.3 + 0.2 \times z \), \( z \) is a uniform random number drawn from the interval \([0, 1]\), \( d_t = |x_{i,t} - x_{k,t}| \). The new pair \((c_{1,t}, c_{2,t})\) is then copied to the set \( C_t \), otherwise the pair \((x_{i,t}, x_{k,t})\) is copied to \( C_t \).

Then the non-uniform mutation [20] is applied to the components of each member of \( C_t \) with probability, \( p_m \) as follows

\[
m_{i,t} = \begin{cases} 
    c_{j,t}^i + \Delta(t, w^i - c_{j,t}^i) & \text{if } \tau \leq 0.5, \\
    c_{j,t}^i - \Delta(t, c_{j,t}^i - l^i) & \text{otherwise}
\end{cases} \tag{3}
\]

where \( \tau \) is a uniformly distributed random number in the interval \([0, 1]\), \( w^i \) and \( l^i \) are the upper and lower boundaries of \( x \in S \), respectively. The function \( \Delta(t, w^i - c_{j,t}^i) \) given below takes a value in the interval \([0, y]\)

\[
\Delta(t, y) = y(1 - r^T(1 - \frac{1}{r}))^\beta,
\]

where \( r \) is a uniformly distributed random number in the interval \([0, 1]\), \( T \) is the maximum number of generations and \( \beta \) is a parameter that determines the non-uniform strength.
of the mutation operator. The mutated individual $m_{i,t}$ is then copied to the set $M_t$, otherwise $c_{i,t}$ is copied to $M_t$.

Projection operation is used to generate $\Phi_t$ from $M_t$ by randomly taking a pair of solutions, $(m_{i,t}, m_{k,t})$, for each $m_{i,t} \in M_t$ and a projected solution $\omega_{i,t} \in \Phi_t$ is created. This operation works by comparing the fitness of the two selected parents and the weaker parent is projected onto the better parent so that the resultant offspring will be derived along the path of the better parent as follows:

If $f(m_{i,t})$ is better than $f(m_{k,t})$ then,

$$\omega_{i,t} = \frac{m_{i,t}^T m_{k,t}}{m_{i,t}^T m_{i,t}} m_{i,t} = \frac{m_{i,t}^T m_{k,t}}{\|m_{i,t}\|^2} m_{i,t}$$

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

As a result, the projected vector is in the opposite direction (the reflection of $m_{i,t}$ about the origin). Hence $\Phi_t = \{\omega_{i,t}, \omega_{j,t}, ..., \omega_{N,t}\}$.

Sometimes the components $\omega_{i,t}^j$ of the trial point $\omega_{i,t}$ may fall outside the search space $S$. In such cases, the corresponding component $\omega_{i,t}^j$ is regenerated. After the projected vector is generated, its fitness value $f(\omega_{i,t})$ is determined and a new population, $P_{t+1}$, is created with $x_{i,t}$, where,

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

Finally, elitism is used to replace the worst point(s) in $P_{t+1}$ with the best solution(s) in $P_t$ because the replacement strategy used is the generational model [10].

3. EXPERIMENTAL PROCEDURE

The experimental setup was carried out according to [15] on the benchmark functions provided in [12, 17]. Two independent restart strategies were used for PRCGA in this work. For each restart strategy, the genetic run is initiated with an initial population $P_0$ which is uniformly and randomly sampled from the search space $[-4, 4]^D$.

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

4. PARAMETER SETTINGS

The parameters used for the proposed version of PRCGA are: Population size is dimension dependent with population size of min$(100, 10 \times D)$, maximum number of evaluation $\#FEs = 10^5 \times D$, tournament size $\mu_{\text{tour}} = 3$, crossover rate $p_c = 0.8$, mutation rate $p_m = 0.15$, the non-uniformity factor for the mutation $\beta = 15$. The parameter setting mentioned above was used for all functions. The crafting effort is $CrE = 0$ [15].

5. CPU TIMING EXPERIMENT

The CPU timing experiment was conducted for PRCGA using the same independent restart strategies on the function $f_8$ for a duration of 30 seconds on an AMD Turion(tm) II Ultra Dual-Core mobile M620 CPU processor, running at 2.50GHz under a 32-bit Microsoft Windows 7 Professional service pack 1 with 2.75GB RAM usable and Matlab 7.10(f2010a).

The time per function evaluation was 7.1, 7.5, 6.9, 6.9, 7.1 and 8.0 times $10^{-5}$ seconds for PRCGA in dimensions 2, 3, 5, 10, 20 and 40 respectively.

6. RESULTS

The results of PRCGA from experiments carried out according to [15] on the benchmark functions given in [12, 17] are presented in Figures 2, 3, 4 and 5 and in Table 1.

Figure 2 shows the performance of PRCGA on all the benchmark functions with dimensions 2, 3, 5, 10, 20 and 40. PRCGA performed quite well on separable functions $f_1$ – $f_4$ and Gallagher’s Gaussian 101-me Peaks Function $f_{21}$, which is a multi-modal function with weak global structure. PRCGA showed some encouraging performance in solving problems $f_6$ – $f_7$ in dimensions 2 – 10, while it could not achieve the desired precision of $10^{-6}$ beyond dimension 3 in Rastrigin Function $f_{13}$ and Weierstrass Function $f_{16}$ neither could it achieve the desired precision beyond dimension 5 in Schaffers F7 $f_{17}$ and Schwefel Function $f_{20}$.

Some functions that prove to be hard and laborious for PRCGA are Ellipsoidal Function $f_{10}$, Discus $f_{11}$ and Lu-nacek bi-Rastrigin Function $f_{24}$. Apart from these functions, PRCGA performed reasonably well with low levels of precision on majority of the test functions.

By comparing the performance of PRCGA with previous GAs benchmarked on these test functions, DBRCGA [4] outperformed PRCGA probably due to the direction-based crossover used in DBRCGA while PRCGA performed better than the variant of RCGA in [24] and simpleGA [21].

From all the results presented above, it can be seen that GAs are able to exploit the separability properties of the test functions but are not able to navigate through deceptive, highly multimodal, highly rugged functions and non-separable quadratic functions with local irregularities.

7. CONCLUSION

The benchmarking of PRCGA, a real-coded genetic algorithm based on vector projection on noiseless black-box optimization tested has shown the strength and weaknesses of the algorithm. PRCGA has produced some impressive results and shown to be better than some other variants of RCGA [21, 24]. The performance of PRCGA on BBOB-2013 shows that PRCGA is an improvement over RCGA-P which could not achieve the desired precision of $10^{-6}$ in most of the test functions. Further modifications to RCGA are needed and are currently being carried out based on the observed weaknesses of PRCGA.

From the performance of PRCGA, it is obvious that in its current state it cannot compete with the current state-of-the-art evolutionary algorithms such as the variants of CMA-ES in [16]. Research is under way to improve PRCGA so that it can comfortably solve most if not all the BBOB-2013 test functions.
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 ($\times$); 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 (ECDF), plotting the fraction of trials with an outcome not larger than the respective value on the x-axis. Left subplots: ECDF of the number of function evaluations (FEvals) divided by search space dimension $D$, to fall below $f_{\text{opt}} + \Delta f$ with $\Delta f = 10^k$, where $k$ is the first value in the legend. The thick red line represents the most difficult target value $f_{\text{opt}} + 10^{-8}$. Legends indicate for each target the number of functions that were solved in at least one trial within the displayed budget. Right subplots: ECDF of the best achieved $\Delta f$ for running times of $0.5D, 1.2D, 3D, 10D, 100D, 1000D, \ldots$ function evaluations (from right to left cycling cyan-magenta-black...) and final $\Delta f$-value (red), where $\Delta f$ and $Df$ denote the difference to the optimal function value. Light brown lines in the background show ECDFs for $\Delta f = 10^{-8}$ of all algorithms benchmarked during BBOB-2009.
Table 1: Expected running time (ERT) in number of function evaluations) divided by the best ERT measured during BBOB-2009. The ERT and in braces, as dispersion measure, the half difference between 90 and 10%-tile of bootstrapped run lengths appear in the second row of each cell, the best ERT in the first.

<table>
<thead>
<tr>
<th>Target</th>
<th>5-D</th>
<th>20-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>212</td>
<td>1002</td>
</tr>
<tr>
<td>f2</td>
<td>228</td>
<td>2002</td>
</tr>
<tr>
<td>f3</td>
<td>232</td>
<td>2092</td>
</tr>
<tr>
<td>f4</td>
<td>239</td>
<td>2113</td>
</tr>
<tr>
<td>f5</td>
<td>251</td>
<td>2139</td>
</tr>
<tr>
<td>f6</td>
<td>257</td>
<td>2163</td>
</tr>
<tr>
<td>f7</td>
<td>266</td>
<td>2188</td>
</tr>
<tr>
<td>f8</td>
<td>268</td>
<td>2200</td>
</tr>
<tr>
<td>f9</td>
<td>271</td>
<td>2213</td>
</tr>
<tr>
<td>f10</td>
<td>275</td>
<td>2227</td>
</tr>
<tr>
<td>f11</td>
<td>278</td>
<td>2243</td>
</tr>
<tr>
<td>f12</td>
<td>281</td>
<td>2256</td>
</tr>
<tr>
<td>f13</td>
<td>284</td>
<td>2269</td>
</tr>
<tr>
<td>f14</td>
<td>289</td>
<td>2283</td>
</tr>
<tr>
<td>f15</td>
<td>292</td>
<td>2296</td>
</tr>
<tr>
<td>f16</td>
<td>296</td>
<td>2310</td>
</tr>
<tr>
<td>f17</td>
<td>301</td>
<td>2324</td>
</tr>
<tr>
<td>f18</td>
<td>305</td>
<td>2338</td>
</tr>
<tr>
<td>f19</td>
<td>310</td>
<td>2352</td>
</tr>
<tr>
<td>f20</td>
<td>314</td>
<td>2366</td>
</tr>
<tr>
<td>f21</td>
<td>318</td>
<td>2380</td>
</tr>
<tr>
<td>f22</td>
<td>323</td>
<td>2394</td>
</tr>
<tr>
<td>f23</td>
<td>327</td>
<td>2408</td>
</tr>
<tr>
<td>f24</td>
<td>331</td>
<td>2422</td>
</tr>
</tbody>
</table>

8. ACKNOWLEDGEMENTS

This work was funded by The University of Kwazulu-Natal during the postdoctoral study of the first author at The School of Mathematics, Statistics and Computer Science, University of Kwazulu-Natal, South Africa. The first author would like to thank The University of Lagos for her support during this research work.

9. REFERENCES


