Research Article

Benchmarking RCGAu on the Noiseless BBOB Testbed

Babatunde A. Sawyerr, Aderemi O. Adewumi, and M. Montaz Ali

1 School of Mathematics, Statistics and Computer Science, College of Agriculture, Engineering and Science, University of KwaZulu-Natal, Westville, South Africa
2 Department of Computer Sciences, Faculty of Science, University of Lagos, Lagos, Nigeria
3 School of Computational and Applied Mathematics, Faculty of Science and TCSE, Faculty of Engineering and Built Environment, University of the Witwatersrand, Johannesburg, South Africa

Correspondence should be addressed to Aderemi O. Adewumi; adewumia@ukzn.ac.za

Received 19 July 2014; Accepted 9 November 2014

1. Introduction

The simple genetic algorithm (GA) introduced by Holland is a probabilistic algorithm based on the theory of natural selection by Charles Darwin. GA mimics the evolutionary process through the creation of variations in each generation and the survival of the fittest individuals through the blending of genetic traits. Individuals with genetic traits that increase their probability of survival will be given more opportunities to reproduce and their offspring will also profit from the heritable traits. Over the period of time these individuals will eventually dominate the population [1, 2].

GA consists of a set of potential solutions called chromosomes, a selection operator, a crossover operator, and a mutation operator. A chromosome is a string of zeros (0s) and ones (1s). It is a metaphor of the biological chromosome in living organisms. The zeros (0s) and ones (1s) are called genes. A gene is the transfer unit of heredity. It contains genetic traits or information that is passed on from a parent solution to its offspring. The selection operator selects solutions for mating based on the principle of “survival of the fittest.” The crossover operator generates new solution pairs called children by combining the genetic materials of the selected parents. The mutation operator is an exploratory operator that is applied, with low probability, to the population of chromosomes to sustain diversity. Without the mutation operator, GAs can easily fall into premature convergence [1, 3].

The simple GA was designed to work on binary strings and it is directly applicable to pseudoboolean objective functions. However, most real life problems are represented as continuous parameter optimization problems. A decoding function was designed to map the solutions from binary space to the real-valued space. This decoding process can become prohibitively expensive for binary string GAs especially when the problem dimension increases [1, 3]. To tackle this problem real-coded genetic algorithms were introduced [4].

Real-coded genetic algorithms (RCGAs) use real-valued vectors to represent individual solutions. Surveys show that several variants of RCGAs have been proposed and used to solve a wide range of real life optimization problems. Some recent examples can be found in [1, 4–10].

Over the last three decades, researchers have continuously improved the performance of RCGAs through hybridization. RCGAs have been hybridized with other
optimizers such as Nelder-Mead algorithms [11], simplex method [12], quadratic approximation [13], and pattern search [14–16]. In this paper, a set of noiseless testbed from the black-box optimization benchmarking (BBOB) 2013 workshop is used to benchmark RCGAu, a hybrid real-coded genetic algorithm that consists of “uniform random direction” local search technique.

The RCGAu algorithm is presented in Section 2, Section 3 provides the CPU timing for the experiments, Section 4 presents the results and discussion, and finally Section 5 concludes the paper with some recommendations.

2. The RCGAu Algorithm

RCGAu is a hybrid RCGA with a simple derivative-free local search technique called “uniform random direction” local search method. The local search technique operates on all individuals after the mutation operator has been applied to the population of individuals.

The RCGAu used in this work is a modified version of the RCGAu used in [16, 17]. It consists of five major operators, namely, tournament selection, blend-α crossover, nonuniform mutation, uniform random direction local search method, and a stagnation alleviation mechanism. Algorithm 1 shows the RCGAu algorithm.

The notations used in this paper are defined as follows.

- \( P_t \) denotes the population of individual solutions \( x_i \) at time \( t \), \( N \) is the size of \( P_t \), \( \sigma(f(P_t)) \) represents the standard deviation of the fitness values \( f(P_t) \) of all solutions \( x_i \in P_t \), \( \tilde{P}_t \) is the mating pool containing the parent solutions, \( C_0 \) is the population of offspring solutions obtained after applying crossover on the parents in \( \tilde{P}_t \), \( p_c \) is the crossover probability, \( M_t \) is the resultant population of solutions after applying mutation on \( C_0 \), \( p_m \) is the mutation probability, \( Y_t \) is the population of solutions obtained after ulsearch has been applied to \( M_t \), where ulsearch uses the uniform random direction local search. Also, \( \epsilon = 10^{-12} \), a very small positive value [18].

The evolutionary process in Algorithm 1 starts by initializing \( P_{i=0} \) from the search space \( X \subset \mathbb{R}^n \). The domain of \( X \) is defined by specifying upper (\( u_l \)) and lower (\( l_1 \)) limits of each \( j \)th component of \( x \); that is, \( l_j \leq x_j \leq u_j \) and \( l_j, u_j \in \mathbb{R} \), \( j = 1, 2, \ldots, n \). Next, the fitness value \( f(x_{i,j}) \), \( \forall x_{i,j} \in P_0 \), is calculated and the population diversity of \( P_t \) is measured by calculating the standard deviation \( \sigma(f(P_t)) \) of \( f(x_{i,j}) \).

If \( \sigma(f(P_t)) \leq \epsilon \) 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 \([-4, 4]^D\) and the resultant population.

\( \tilde{P}_t = \{x_{1,j}, x_{2,j}, \ldots, x_{m,j}\} \) is created. \( m \) is the size of the mating pool \( \tilde{P}_t \) and \( m \leq N \). If, on the other hand, \( \sigma(f(P_t)) > \epsilon \) then tournament selection is applied on \( P_t \) to create an equivalent mating pool \( \tilde{P}_t \).

The tournament selection scheme works by selecting \( r \) number of solutions uniformly at random from \( P_t \), where \( r < N \). The selected \( r \) individuals are compared using their fitness values and the best individual is selected and assigned to \( \tilde{P}_t \). This procedure is repeated \( m \) times to populate \( \tilde{P}_t \).

After the mating pool has been created, blend-α crossover is applied to a pair of parent solutions \( (x_{i,j}, x_{k,j}) \) if a randomly generated number \( r \) drawn uniformly from the interval \([0, 1]\) is greater than the specified crossover probability threshold \( p_c \). Blend-α crossover creates a pair of offspring \( (c_{1,j}, c_{2,j}) \) from the interval \([\min(x_{1,j}, x_{k,j}) - \alpha \cdot d^1], \max(x_{1,j}, x_{k,j}) + \alpha \cdot d^1] \) as follows:

\[
\begin{align*}
\alpha_{1,j} &= \min(\alpha \cdot d^1, \max(\alpha \cdot d^1) + \alpha \cdot d^1) \\
\alpha_{2,j} &= \min(\alpha \cdot d^1, \max(\alpha \cdot d^1) + \alpha \cdot d^1)
\end{align*}
\]

where \( (1 \leq k \leq N), \alpha = 0.3 + 0.2 \times z, z \) is a uniform random number drawn from the interval \([0, 1]\), and \( d^1 = |x_{1,j} - x_{k,j}| \). The new pair \( (c_{1,j}, c_{2,j}) \) is then copied to the set \( C_t \); otherwise the pair \( (x_{1,j}, x_{k,j}) \) is copied to \( C_t \).
Figure 1: Continued.
Then the nonuniform mutation [4] is applied to the components of each member of $C_t$ with probability, $p_m$, as follows:

$$m^j_{i,t} = \begin{cases} c^j_{i,t} + \Delta \left( t, u^j - c^j_{i,t} \right) & \text{if } u \leq 0.5, \\ c^j_{i,t} - \Delta \left( t, c^j_{i,t} - l^j \right) & \text{otherwise}, \end{cases}$$

(2)

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

$$\Delta \left( t, y \right) = y \left( 1 - r^{\left( 1 - (t/T) \right)} \right)^\beta,$$

(3)

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 nonuniform strength of the mutation operator. The mutated individual $m^j_{i,t}$ is then copied to the set $M_j$; otherwise $c^j_{i,t}$ is copied to $M_j$. 

---

**Figure 1**: Expected number of $f$-evaluations (ERT, lines) to reach $f_{opt} + \Delta f$; 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 run lengths to reach $f_{opt} + \Delta f$; all values are divided by dimension and plotted as log values versus dimension. Also, $\Delta f = 10^{\{1,0,-1,-2,-3,-5,-8\}}$ are shown. Numbers above ERT-symbols (if appearing) indicate the number of trials reaching the respective targets. The light thick line with diamonds indicates the respective best results from BBOB-2009 for $\Delta f = 10^{-8}$. Horizontal lines mean linear scaling and slanted grid lines depict quadratic scaling.
Figure 2: Continued.
Then ulsearch is applied on each solution \( m_{ij} \in M_i \) with the aim of performing local searches around the neighborhood of each solution. ulsearch works by randomly selecting a solution \( m_{ij} \in M_i \) and creating a trial point \( y_{ij} \) using
\[
y_{ij} = m_{ij} + \Delta_i U,
\]
where \( \Delta_i \) is a step size parameter and \( U = (U_1, U_2, \ldots, U_n)^T \) is a directional cosine with random components
\[
U_j = \frac{R_j}{(R_1^2 + \cdots + R_n^2)^{1/2}}, \quad j = 1, 2, \ldots, n,
\]
where \( R_j \sim \text{Unif}([-1, 1]) \). There are cases when the components of the trial point \( y_{ij} = (y_{i1}, y_{i2}, \ldots, y_{in}) \) generated by (4) fall outside the search space \( X \) during the search. In these cases, the components of \( y_{ij} \) are regenrated using
\[
y_{ij} = \begin{cases} m_{ij} + \lambda (u - m_{ij}), & \text{if } y_{ij} > u_i \\ m_{ij} + \lambda (m_{ij} - l_i), & \text{if } y_{ij} < l_i, \end{cases}
\]
where \( \lambda \sim \text{Unif}(0, 1) \) and \( m_{ij} \) is the corresponding component of the randomly selected solution \( m_{ij} \in M_i \).

The step size parameter, \( \Delta_i \), is initialized at time \( t = 0 \) according to [15, 16] by
\[
\Delta_0 = \tau \times \max \{u_j - l_j \mid j = 1, 2, \ldots, n\},
\]
where \( \tau \in [0, 1] \). The idea of using (7) to generate the initial step length is to accelerate the search by starting with a suitably large step size to quickly traverse the search space and as the search progresses the step size is adaptively adjusted at the end of each generation, \( t \), by
\[
\Delta_{t+1} = \frac{1}{K} \sum_{j=1}^{K} y^j_i,
\]
where \( K \) is the number of Euclidean distances \( \{y^1_i, y^2_i, \ldots, y^K_i\} \) between \( K \) nearest points to the mean \( \overline{X} \) and \( \overline{X} \) of a set of randomly selected distinct points \( \Omega = \{x_1, x_2, \ldots, x_n\} \subset P_i \).

After the trial point \( y_{ij} \in Y \) has been created, it is evaluated and compared with \( m_{ij} \). If \( y_{ij} < m_{ij} \), then \( y_{ij} \) is used to replace \( m_{ij} \); otherwise the search direction is changed by changing the sign of the step length. The new step length is used to recalculate a new trial point. After a new trial point has been recalculated and evaluated, it is used to replace \( m_{ij} \in M_i \) with \( y_{ij} \); if \( y_{ij} < m_{ij} \); otherwise \( m_{ij} \in M_i \) is retained.

At the end of ulsearch, \( P_i \) is updated with \( M_i \) to form \( P_{i+1} \) and elitism is used to replace the worst point in \( P_i \) with the best point in \( P_t \) because the generational model is the replacement strategy adopted in this work [19].

### 3. Experimental Procedure and Parameter Settings

The experimental setup was carried out according to [20] on the benchmark functions provided in [21, 22]. Two independent restart strategies were used to restart RClogAu whenever \( P_i \) stagnates or when the maximum number of generations is exceeded and \( f_{\text{target}} \) is not found. For each restart strategy, the experiment is reinitializd with an initial population \( P_0 \) which is uniformly and randomly sampled from the search space \( [-4, 4]^D \) [6, 18].

Two stopping conditions used for the restart strategies are as follows.

(i) A test for stagnation is carried out to check 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 [6].

\[ \text{Proportion of trials} \]
\[ \log_{10} \text{of FEvals/DIM} \]
\[ 0.0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0 \]

\[ \log_{10} \text{of Df} \]
\[ 0.0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0 \]

\[ \text{Proportion of trials} \]
\[ \log_{10} \text{of FEvals/DIM} \]
\[ 0.0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0 \]

\[ \log_{10} \text{of Df} \]
\[ 0.0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0 \]

\[ \text{Proportion of trials} \]
\[ \log_{10} \text{of FEvals/DIM} \]
\[ 0.0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0 \]

\[ \log_{10} \text{of Df} \]
\[ 0.0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0 \]

\[ \text{Proportion of trials} \]
\[ \log_{10} \text{of FEvals/DIM} \]
\[ 0.0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0 \]

\[ \log_{10} \text{of Df} \]
\[ 0.0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0 \]
Figure 3: ERT loss ratios (see Table 2 for details). Each cross (+) represents a single function and the line is the geometric mean.
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 percentile of bootstrapped runlengths appear in the second row of each cell, the best ERT in the first. The different target $\Delta f$-values are shown in the top row.

| $\Delta f$ | $f_{opt} + 10^{-4}$ | $f_{opt}$ | $f_{opt} - 10^{-4}$ | $f_{opt} - 10^{-5}$ | $f_{opt} - 10^{-6}$ | $f_{opt} - 10^{-7}$ |
|------------|---------------------|-----------|---------------------|---------------------|---------------------|---------------------|
| $f_1$      | 5.2 (6)             | 5.6 (15)  | 5.6 (15)            | 5.1 (22)            | 5.1 (29)            | 5.1 (32)            |
| $f_2$      | 43                  | 43        | 43                  | 43                  | 43                  | 43                  |
| $f_3$      | 385                 | 386       | 387                 | 386                 | 391                 | 393                 |
| $f_4$      | 5.1 (4)             | 5.1 (4)   | 5.1 (4)             | 5.1 (4)             | 5.1 (4)             | 5.1 (4)             |
| $f_5$      | 12.3 (1)            | 13.5      | 13.5                | 13.5                | 13.5                | 13.5                |
| $f_6$      | 1.8 (1)             | 2.1 (6)   | 2.1 (6)             | 2.1 (6)             | 2.1 (6)             | 2.1 (6)             |
| $f_7$      | 11                  | 12        | 12                  | 12                  | 12                  | 12                  |
| $f_8$      | 8.9                 | 9.0       | 9.0                 | 9.0                 | 9.0                 | 9.0                 |
| $f_9$      | 4.2 (3)             | 4.2 (3)   | 4.2 (3)             | 4.2 (3)             | 4.2 (3)             | 4.2 (3)             |
| $f_{10}$   | 31                  | 32        | 32                  | 32                  | 32                  | 32                  |
| $f_{11}$   | 12                  | 13        | 13                  | 13                  | 13                  | 13                  |
| $f_{12}$   | 205 (236)           | 201 (236) | 201 (236)           | 201 (236)           | 201 (236)           | 201 (236)           |
| $f_{13}$   | 3.1                 | 3.2       | 3.2                 | 3.2                 | 3.2                 | 3.2                 |
| $f_{14}$   | 20                  | 21        | 21                  | 21                  | 21                  | 21                  |
| $f_{15}$   | 10                  | 11        | 11                  | 11                  | 11                  | 11                  |
| $f_{16}$   | 2.3 (3)             | 2.4       | 2.4                 | 2.4                 | 2.4                 | 2.4                 |
| $f_{17}$   | 5.2                 | 5.3       | 5.3                 | 5.3                 | 5.3                 | 5.3                 |
| $f_{18}$   | 1.5 (1)             | 1.6 (2)   | 1.6 (2)             | 1.6 (2)             | 1.6 (2)             | 1.6 (2)             |
| $f_{19}$   | 16                  | 17        | 17                  | 17                  | 17                  | 17                  |
| $f_{20}$   | 4.2 (3)             | 4.3       | 4.3                 | 4.3                 | 4.3                 | 4.3                 |
| Δf  | le + 1 | le + 0 | le - 1 | le - 2 | le - 3 | le - 5 | le - 7 | #succ | Δf  | le + 1 | le + 0 | le - 1 | le - 2 | le - 3 | le - 5 | le - 7 | #succ |
|-----|--------|--------|--------|--------|--------|--------|--------|-------|-----|--------|--------|--------|--------|--------|--------|--------|-------|-------|
| f₁₁ | 8.12   | 1157   | 1674   | 1692   | 1705   | 1729   | 1757   | 14/15 | f₁₁ | 561    | 6541   | 14103  | 14318  | 14643  | 15567  | 17589  | 15/15 |
| f₁₂ | 71     | 386    | 938    | 980    | 1008   | 1040   | 1068   | 14/15 | f₁₂ | 467    | 5580   | 2349   | 24863  | 24948  | 26847  | 1.3e5  | 12/15 |
| f₁₃ | 3.0    | 318    | 1429   | 27890  | 30630  | 34256  | 34137  | 15/15 | f₁₃ | 3.2    | 1014   | 67457  | 67457  | 67457  | 67457  | 67457  | 15/15 |
| f₁₄ | 1622   | 2.2e5  | 6.4e6  | 9.6e6  | 9.6e6  | 1.3e7  | 1.3e7  | 3/15  | f₁₄ | 1.3e6  | 7.3e6  | 5.2e7  | 5.2e7  | 5.2e7  | 5.2e7  | 5.2e7  | 3/15  |
| f₁₅ | 20     | 27     | oo     | oo     | oo     | oo     | oo     | 0/15  | f₁₅ | oo     | oo     | oo     | oo     | oo     | oo     | oo     | 0/15  |
Table 2: ERT loss ratio versus the budget (both in number of \( f \)-evaluations divided by dimension). The target value \( f_t \) for a given budget FEvals is the best target \( f \)-value reached within the budget by the given algorithm. Shown is the ERT of the given algorithm divided by best ERT seen in GECCO-BBOB-2009 for the target \( f_t \), or, if the best algorithm reached a better target within the budget, the budget divided by the best ERT. Line: geometric mean. Box-Whisker error bar: 25–75%-ile with median (box), 10–90%-ile (caps), and minimum and maximum ERT loss ratio (points). The vertical line gives the maximal number of function evaluations in a single trial in this function subset. See also Figure 3 for results on each function subgroup.

(ii) A test is carried out to check if the maximum number of generations is satisfied and \( f_{\text{target}} \) is not found.

The parameters used for RCGAu on all functions are

(i) population size = \( \min(100, 10 \times D) \), where \( D \) is the problem dimension;
(ii) maximum number of evaluations \( \# \text{FEs} = 10^5 \times D \);
(iii) tournament size \( r = 3 \);
(iv) crossover rate \( p_c = 0.8 \);
(v) mutation rate \( p_m = 0.15 \);
(vi) nonuniformity factor for the mutation \( \beta = 15 \);
(vii) elitism \( E = 1 \);
(viii) crafting effort \( \text{CrE} = 0 \) [20].

4. CPU Timing Experiment

The CPU timing experiment was conducted for RCGAu 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 2.50 GHz under a 32-bit Microsoft Windows 7 Professional service pack 1 with 2.75 GB RAM usable and Matlab 7.10 (R2010a).

The time per function evaluation was 2.5, 2.6, 2.9, 3.0, 3.2, and 3.5 times \( 10^{-4} \) seconds for RCGAu in dimensions 2, 3, 5, 10, 20, and 40, respectively.
5. Results

The results of the empirical experiments conducted on RCGAu according to [20] on the benchmark functions given in [21,22] are presented in Figures 1, 2, and 3 and in Tables 1 and 2.

Figure 1 shows the performance of RCGAu on all the noiseless problems with the dimensions 2, 3, 5, 10, 20, and 40. RCGAu was able to solve many test functions in the low search dimensions of 2 and 3 to the desired accuracy of $10^8$. It is able to solve most test functions with dimensions up to 40 at lowest precision of $10^1$.

Although RCGAu found it difficult in getting a solution with the desired accuracy $10^8$ for high conditioning and multimodal functions within the specified maximum #FES it was able to solve $f_{21}$ with dimensions up to 40, $f_1$ and $f_2$ with dimensions up to 20, $f_3$ and $f_7$ with dimensions up to 10, and $f_4, f_5, f_{15}, f_{20}$, and $f_{22}$ with dimensions up to 5.

In Figure 2, the left subplot graphically illustrates the empirical cumulative distribution function (ECDF) of the number of function evaluations divided by the dimension of the search space, while the right subplot shows the ECDF of the best achieved $\Delta f$. This figure graphically shows the performance of RCGAu in terms of function evaluation.

Table 1 presents the performance of RCGAu in terms of the expected running time (ERT). This measure estimates the run time of RCGAu by using the number of function evaluations divided by the best ERT measured during BBOB 2009 workshop. This benchmark shows that RCGAu needs some improvement in terms of performance.

6. Conclusion

The performance of RCGAu on the suite of noiseless black-box optimization testbed has been average on a number of problems but it has excelled in solving functions $f_1, f_2, f_3, f_7$, and $f_{21}$. Studies have currently been carried out to find out why RCGAs do not efficiently solve highly conditioned problems. Further modifications to RCGAs are needed to exploit the full strength of evolutionary processes.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] T. Back, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University press, New York, NY, USA, 1996.
[2] J. H. Holland, Adaptation in Natural and Artificial Systems, An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, MIT Press, Cambridge, Mass, USA, 1975.
[3] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, Mass, USA, 1989.
[4] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, Berlin, Germany, 1996.
[5] M. M. Ali and A. Törn, “Population set-based global optimization algorithms: some modifications and numerical studies,” Computers & Operations Research, vol. 31, no. 10, pp. 1703–1725, 2004.
[6] Y.-C. Chuang and C.-T. Chen, ”Black-box optimization benchmarking for noiseless function testbed using a direction-based RCGA,” in Proceedings of the 14th International Conference on Genetic and Evolutionary Computation (GECCO ’12), pp. 167–174, Philadelphia, PA, USA, July 2012.
[7] K. Deep and M. Thakur, ”A new crossover operator for real coded genetic algorithms,” Applied Mathematics and Computation, vol. 188, no. 1, pp. 895–911, 2007.
[8] K. Deep and M. Thakur, ”A new mutation operator for real coded genetic algorithms,” Applied Mathematics and Computation, vol. 193, no. 1, pp. 211–230, 2007.
[9] K. Deep and M. Thakur, ”A real coded multi parent genetic algorithms for function optimization,” Applied Mathematics and Computation, vol. 1, no. 2, pp. 67–83, 2008.
[10] P. Kaelo and M. M. Ali, ”Integrated crossover rules in real coded genetic algorithms,” European Journal of Operational Research, vol. 176, no. 1, pp. 60–76, 2007.
[11] R. Chelouah and P. Siarry, ”Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multimimina functions,” European Journal of Operational Research, vol. 148, no. 2, pp. 335–348, 2003.
[12] J. Andre, P. Siarry, and T. Dognon, ”An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization,” Advances in Engineering Software, vol. 32, no. 1, pp. 49–60, 2001.
[13] K. Deep and K. N. Das, ”Quadratic approximation based hybrid genetic algorithm for function optimization,” Applied Mathematics and Computation, vol. 203, no. 1, pp. 86–98, 2008.
[14] W. E. Hart, Adaptive global optimization with local search [Ph.D. thesis], University of California, San Diego, Calif, USA, 1994.
[15] B. A. Sawyerr, M. M. Ali, and A. O. Adewumi, ”A comparative study of some real-coded genetic algorithms for unconstrained global optimization,” Optimization Methods & Software, vol. 26, no. 6, pp. 945–970, 2011.
[16] B. A. Sawyerr, Hybrid real coded genetic algorithms with pattern search and projection [Ph.D. thesis], University of Lagos, Lagos, Nigeria, 2010.
[17] B. A. Sawyerr, A. O. Adewumi, and M. M. Ali, ”Real-coded genetic algorithm with uniform random local search,” Applied Mathematics and Computation, vol. 228, pp. 589–597, 2014.
[18] B. A. Sawyerr, A. O. Adewumi, and M. M. Ali, ”Benchmarking projection-based real coded genetic algorithm on BBOB-2013 noiseless function testbed,” in Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO ’13), pp. 1193–1200, Amsterdam, The Netherlands, July 2013.
[19] K. A. DeJong, An analysis of the behavior of a class of genetic adaptive systems [Ph.D. thesis], University of Michigan, Ann Arbor, Mich, USA, 1975.
[20] N. Hansen, A. Auger, S. Finck, and R. Ros, ”Real-parameter black-box optimization benchmarking 2012: experimental setup,” Tech. Rep., INRIA, 2012.
[21] S. Finck, N. Hansen, R. Ros, and A. Auger, ”Real-parameter black-box optimization benchmarking 2009: presentation of the noiseless functions,” Tech. Rep. 2009/20, Research Center PPE, 2009.
[22] N. Hansen, S. Finck, R. Ros, and A. Auger, ”Real-parameter black-box optimization benchmarking 2009: noiseless functions definitions,” Tech. Rep. RR-6829, INRIA, 2009.