Hindawi Publishing Corporation
The Scientific World Journal
Volume 2015, Article ID 734957, 11 pages
http://dx.doi.org/10.1155/2015/734957

Research Article

Benchmarking RCGAu on the Noiseless BBOB Testbed

Babatunde A. Sawyerr,[1,2] Aderemi O. Adewumi,[1] and M. Montaz Ali[3]

[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

Academic Editor: Albert Victoire

RCGAu is a hybrid real-coded genetic algorithm with "*uniform random direction*" search mechanism. The *uniform random direction* search mechanism enhances the local search capability of RCGA. In this paper, RCGAu was tested on the BBOB-2013 noiseless testbed using restarts till a maximum number of function evaluations (#FEs) of $10^5 \times D$ are reached, where D is the dimension of the function search space. RCGAu was able to solve several test functions in the low search dimensions of 2 and 3 to the desired accuracy of 10^8. 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 most of the test functions with dimensions up to 40 with lower precisions.

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

(1) *Initialize* $P_{t=0}, P_t = \{x_{1,t}, x_{2,t}, \ldots, x_{N,t}\}$ *from* X
(2) $f(x_{i,t}) = evaluate(P_t), \{1 \leq i \leq N\}$
(3) *While not stopping condition, do Steps 4–12*
(4) *Calculate* $\sigma(f(P_t))$, *if* $\sigma(f(P_t)) \leq \varepsilon$ *do Step 5 else do Step 6*
(5) $\widehat{P}_t = perturb(P_t)$
(6) $\widehat{P}_t = tournamentSelection(P_t)$
(7) $C_t = blend\text{-}\alpha Crossover(\widehat{P}_t, p_c)$
(8) $M_t = non\text{-}uniformMutation(C_t, p_m)$
(9) $\Upsilon_t = ulsearch(M_t)$
(10) $f(x_{i,t}) = evaluate(\Upsilon_t)$
(11) $P_{t+1} = replace(P_t, \Upsilon_t)$
(12) $t = t + 1$
(13) *end while*

Algorithm 1: The RCGAu Algorithm.

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 $x_{i,t}$ 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 $x_{i,t}, \in P_t$, $\widehat{P}_t$ is the mating pool containing the parent solutions, C_t is the population of offspring solutions obtained after applying crossover on the parents in $\widehat{P}_t$, 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, and Υ_t is the population of solutions obtained after ulsearch has been applied to M_t, where ulsearch denotes 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 $P_{t=0}$ from the search space $X \subset \mathfrak{R}^n$. The domain of X is defined by specifying upper (u^j) and lower (l^j) limits of each jth component of x; that is, $l^j \leq x^j \leq u^j$ and $l^j, u^j \in \mathfrak{R}$, $j = 1, 2, \ldots, n$. Next, the fitness value $f(x_{i,t})$, $\forall x_{i,t} \in P_0$, is calculated and the population diversity of P_t is measured by calculating the standard deviation $\sigma(f(P_t))$ of $f(x_{i,t})$.

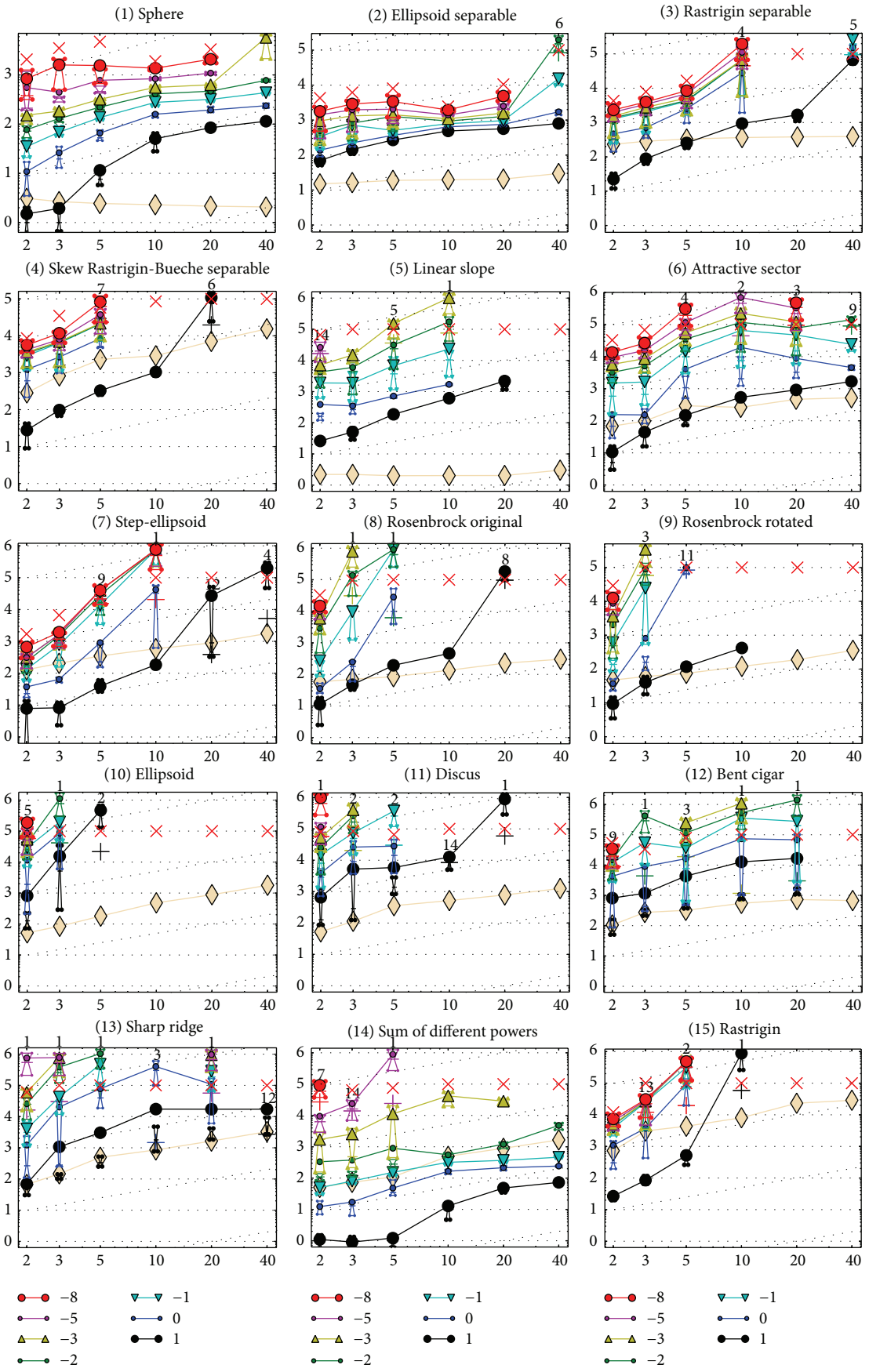
If $\sigma(f(P_t)) \leq \epsilon$ and the global optimum has not been found, then 90% of P_t is refreshed with newly generated solutions using the function perturb (P_t). P_t is refreshed by sorting the solutions according to their fitness values and preserving the top 10% of P_t. The remaining 90% of P_t are replaced with uniformly generated random values from the interval $[-4, 4]^D$ and the resultant population; $\widehat{P}_t = \{x_{1,t}, x_{2,t}, \ldots, x_{m,t}\}$ is created. m is the size of the mating pool $\widehat{P}_t$ 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 $\widehat{P}_t$.

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

After the mating pool has been created, blend-α crossover is applied to a pair of parent solutions $(x_{i,t}, x_{k,t})$ if a randomly generated number τ 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 $(c_{1,t}, c_{2,t})$ from the interval $[\min(x^j_{i,t}, x^j_{k,t}) - \alpha * d^j, \max(x^j_{i,t}, x^j_{k,t}) + \alpha * d^j]$ as follows:

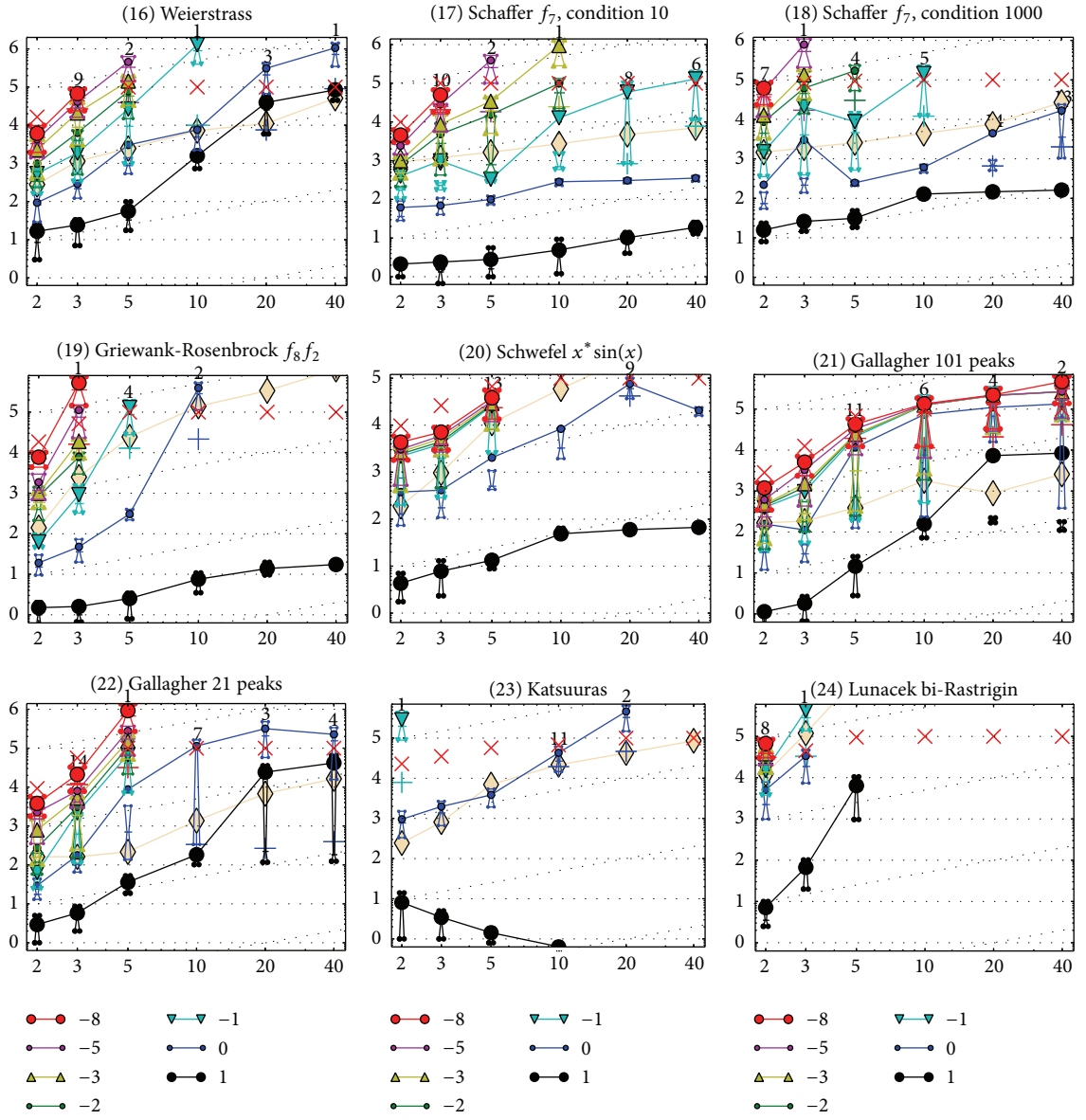
$$c^j_{1,t} = \left(\min\left(x^j_{i,t}, x^j_{k,t}\right) - \alpha * d^j, \max\left(x^j_{i,t}, x^j_{k,t}\right) + \alpha * d^j\right)$$
$$c^j_{2,t} = \left(\min\left(x^j_{i,t}, x^j_{k,t}\right) - \alpha * d^j, \max\left(x^j_{i,t}, x^j_{k,t}\right) + \alpha * d^j\right), \quad (1)$$

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 $d^j = |x^j_{i,t} - x^j_{k,t}|$. 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.



(a)

Figure 1: Continued.



(b)

FIGURE 1: 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 run lengths to reach $f_{\text{opt}} + \Delta f$; all values are divided by dimension and plotted as $\log_{10}$ 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.

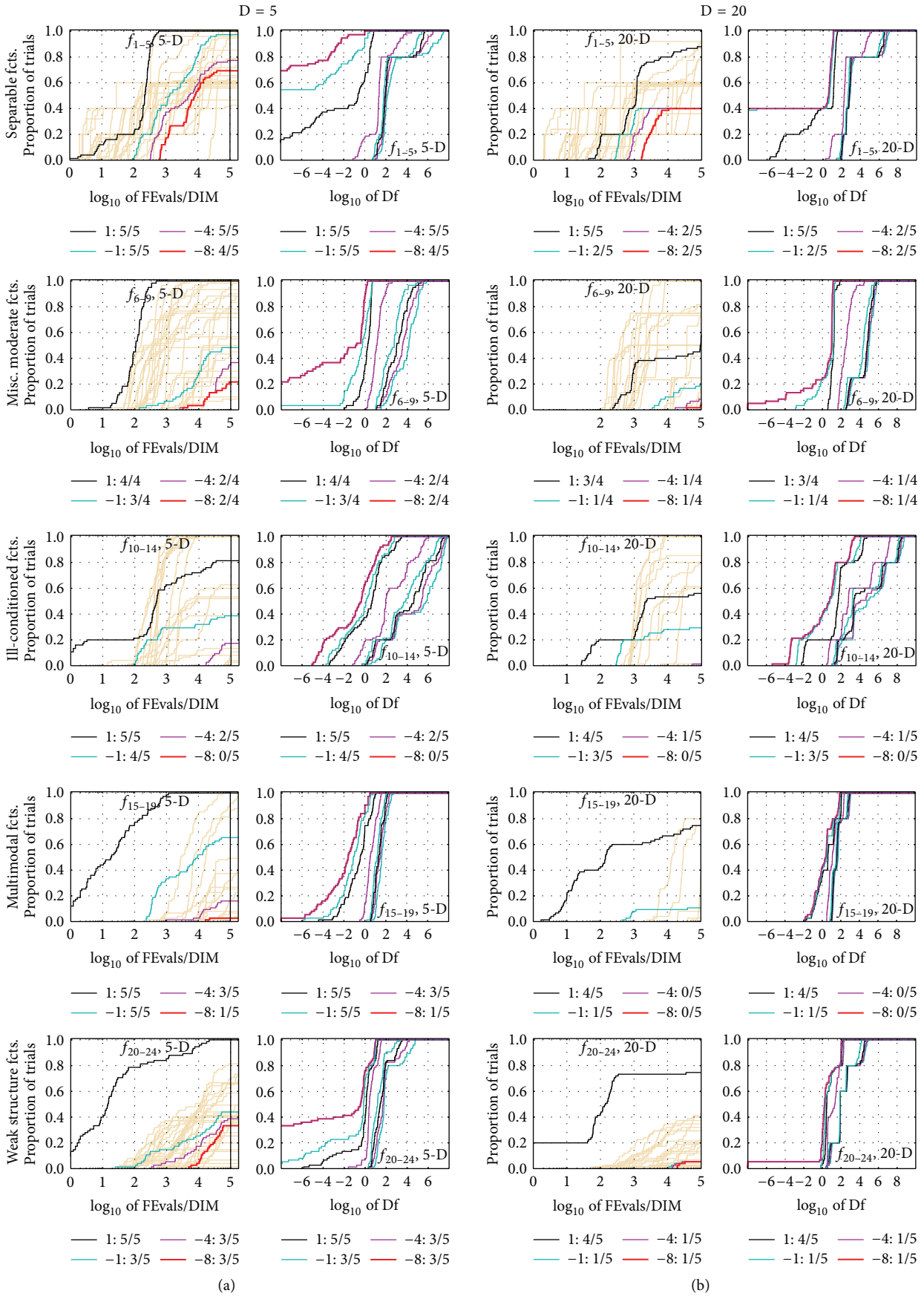
Then the nonuniform mutation [4] is applied to the components of each member of C_t with probability, p_m, as follows :

$$m_{i,t}^j = \begin{cases} c_{i,t}^j + \Delta\left(t, u^j - c_{i,t}^j\right) & \text{if } u \le 0.5, \\ c_{i,t}^j - \Delta\left(t, c_{i,t}^j - l^j\right) & \text{otherwise,} \end{cases} \tag{2}$$

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 $\Delta(t, u^j - c_{i,t}^j)$ given below takes a value in the interval $[0, y]$:

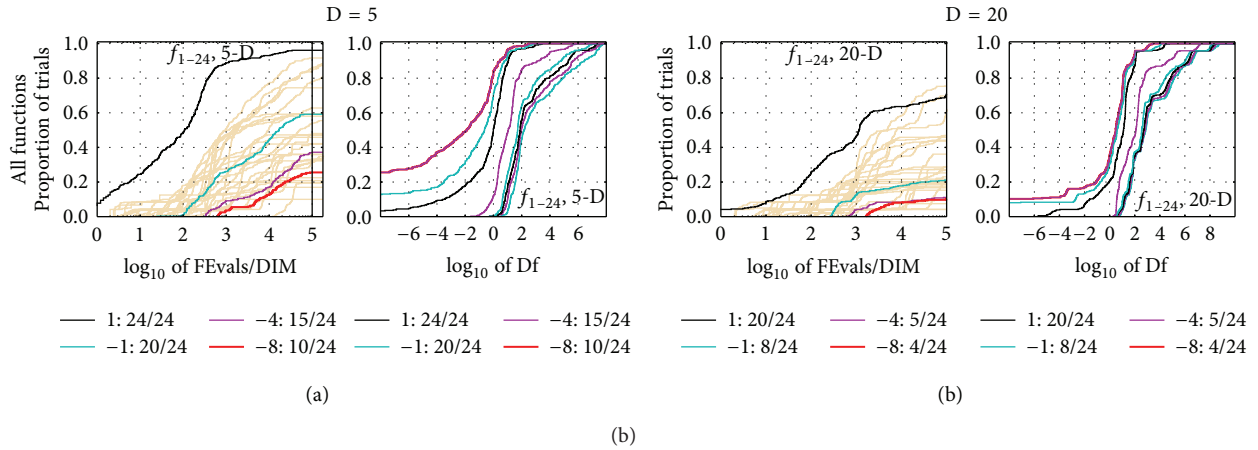
$$\Delta(t, y) = y\left(1 - r^{(1-(t/T))}\right)^{\beta}, \tag{3}$$

where r is a uniformly distributed random number in the interval $[0, 1]$, T is the maximum number of generations, and β is a parameter that determines the nonuniform 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.



(a)

Figure 2: Continued.



(b)

Figure 2: Empirical cumulative distribution functions (ECDF), plotting the fraction of trials with an outcome not larger than the respective values 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 Δf 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 Δf-value (red), where Δf and Df denote the difference to the optimal function value. Light brown lines in the background show ECDF for the most difficult target of all algorithms benchmarked during BBOB-2009.

Then ulsearch is applied on each solution $m_{i,t} \in M_t$ with the aim of performing local searches around the neighborhood of each solution. ulsearch works by randomly selecting a solution $m_{i,t} \in M_t$ and creating a trial point $y_{i,t}$ using

$$y_{i,t} = m_{i,t} + \Delta_t U, \tag{4}$$

where Δ_t is a step size parameter and $U = (U_1, U_2, \ldots, U_n)^T$ is a directional cosine with random components

$$U_j = \frac{R_j}{\left(R_1^2 + \cdots + R_n^2\right)^{1/2}}, \quad j = 1, 2, \ldots, n, \tag{5}$$

where $R_j \sim \text{Unif}([-1, 1])$. There are cases when the components of the trial point $y_{i,t} = (y_{i,t}^1, y_{i,t}^2, \ldots, y_{i,t}^n)$ generated by (4) fall outside the search space X during the search. In these cases, the components of $y_{i,t}$ are regenerated using

$$y_{i,t}^j = \begin{cases} m_{i,t}^j + \lambda\left(u^j - m_{i,t}^j\right), & \text{if } y_{i,t}^j > u^j \\ m_{i,t}^j + \lambda\left(m_{i,t}^j - l^j\right), & \text{if } y_{i,t}^j < l^j, \end{cases} \tag{6}$$

where $\lambda \sim \text{Unif}([0, 1])$ and $m_{i,t}^j$ is the corresponding component of the randomly selected solution $m_{i,t} \in M_t$.

The step size parameter, Δ_t, is initialized at time $t = 0$ according to [15, 16] by

$$\Delta_0 = \tau \times \max\left\{u^j - l^j \mid j = 1, 2, \ldots, n\right\}, \tag{7}$$

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

$$\Delta_{t+1} = \frac{1}{K}\sum_{i=1}^{K} \gamma^i, \tag{8}$$

where K is the number of Euclidean distances $\{\gamma^1, \gamma^2, \ldots, \gamma^K\}$ between K nearest points to the mean $\overline{x}$ and $\overline{x}$ of a set of randomly selected distinct points $\Omega = \{x_1, x_2, \ldots, x_q\} \subset P_t$.

After the trial point $y_{i,t} \in \Upsilon$ has been created, it is evaluated and compared with $m_{i,t}$. If $y_{i,t} < m_{i,t}$, then $y_{i,t} \in \Upsilon$ is used to replace $m_{i,t} \in M_t$; 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 $m_{i,t} \in M_t$ with $y_{i,t}$, if $y_{i,t} < m_{i,t}$; otherwise $m_{i,t} \in M_t$ is retained.

At the end of ulsearch, P_t is updated with M_t to form P_{t+1} and elitism is used to replace the worst point in P_{t+1} 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 RCGAu whenever P_t stagnates or when the maximum number of generations is exceeded and f_{target} is not found. For each restart strategy, the experiment is reinitialized 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].

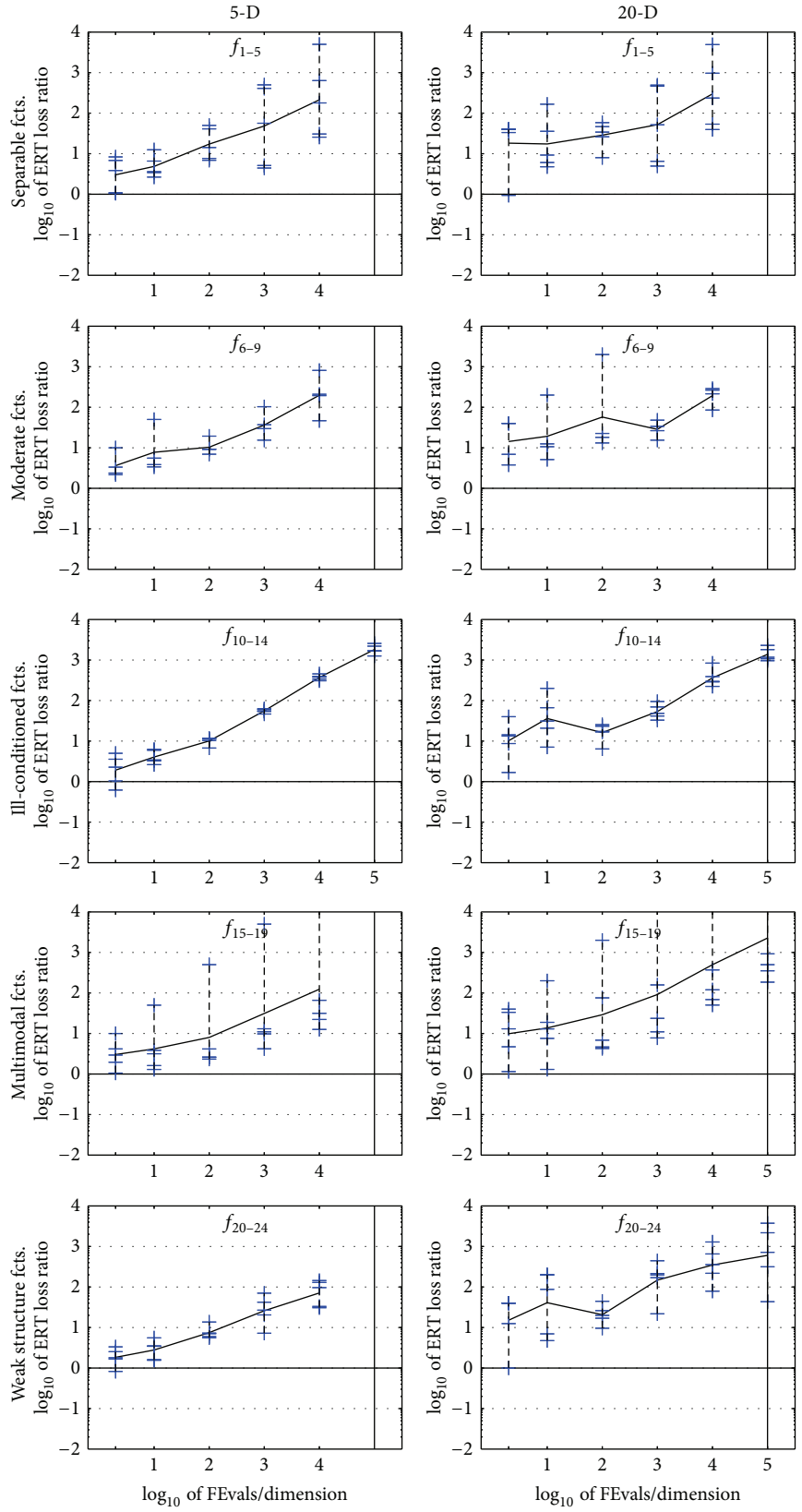


Figure 3: ERT loss ratios (see Table 2 for details). Each cross (+) represents a single function and the line is the geometric mean.

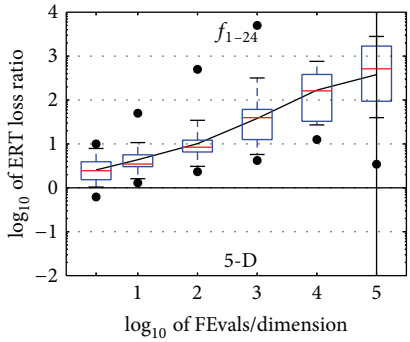
Table 1: Expected running time (ERT in number of function evaluations) divided by the best ERT measured during BBOB-2009. The ERT and, in braces, as dispersion measure, the half difference between 90 and 10 percentile of bootstrapped run lengths appear in the second row of each cell, the best ERT in the first. The different target Δf-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\text{opt}} + 10^{-8}$.

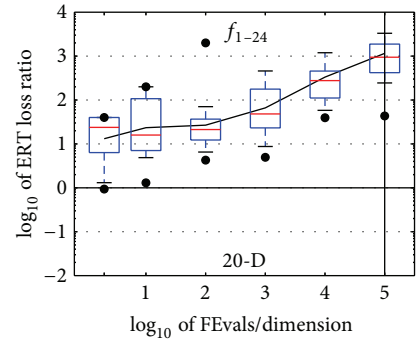
	5D								20D							
Δf	1e + 1	1e + 0	1e − 1	1e − 2	1e − 3	1e − 5	1e − 7	#succ	Δf	1e + 1	1e + 0	1e − 1	1e − 2	1e − 3	1e − 5	1e − 7	#succ
$\mathbf{f}_1$	11	12	12	12	12	12	12	15/15	$\mathbf{f}_1$	43	43	43	43	43	43	43	15/15
	5.2 (6)	27 (15)	56 (15)	91 (22)	131 (29)	319 (193)	580 (800)	15/15		39 (7)	90 (17)	148 (19)	218 (28)	294 (36)	506 (112)	787 (173)	15/15
$\mathbf{f}_2$	83	87	88	89	90	92	94	15/15	$\mathbf{f}_2$	385	386	387	388	390	391	393	15/15
	16 (3)	21 (6)	29 (10)	71 (118)	79 (118)	112 (180)	155 (176)	15/15		29 (5)	39 (8)	50 (8)	66 (17)	81 (30)	128 (41)	189 (78)	15/15
$\mathbf{f}_3$	716	1622	1637	1642	1646	1650	1654	15/15	$\mathbf{f}_3$	5066	7626	7635	7637	7643	7646	7651	15/15
	1.8 (1)	7.8 (9)	13 (15)	13 (15)	16 (16)	21 (20)	24 (19)	15/15		6.6 (6)	∞	∞	∞	∞	∞	∞*1.1e6*	0/15
$\mathbf{f}_4$	809	1633	1688	1758	1817	1886	1903	15/15	$\mathbf{f}_4$	4722	7628	7666	7686	7700	7758	1.4e5	9/15
	2.0 (0.9)	26 (26)	62 (83)	60 (79)	60 (63)	99 (109)	129 (134)	7/15		459 (473)	∞	∞	∞	∞	∞	∞*1.1e6*	0/15
$\mathbf{f}_5$	10	10	10	10	10	10	10	15/15	$\mathbf{f}_5$	41	41	41	41	41	41	41	15/15
	93 (20)	359 (112)	3404 (3910)	15683 (18516)	76807 (77456)	∞	∞*2.7e5*	0/15		1075 (273)	∞	∞	∞	∞	∞	∞*1.2e6*	0/15
$\mathbf{f}_6$	114	214	281	404	580	1038	1332	15/15	$\mathbf{f}_6$	1296	2343	3413	4255	5220	6728	8409	15/15
	6.5 (6)	95 (217)	263 (443)	432 (351)	470 (459)	552 (489)	749 (685)	4/15		14 (3)	75 (28)	277 (325)	365 (470)	465 (474)	971 (1056)	1110 (1193)	3/15
$\mathbf{f}_7$	24	324	1171	1451	1572	1572	1597	15/15	$\mathbf{f}_7$	1351	4274	9503	16523	16524	16524	16969	15/15
	8.4 (6)	14 (22)	58 (89)	98 (114)	126 (139)	126 (133)	125 (134)	9/15		402 (507)	∞	∞	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f}_8$	73	273	336	372	391	410	422	15/15	$\mathbf{f}_8$	2039	3871	4040	4148	4219	4371	4484	15/15
	13 (4)	517 (732)	13503 (15418)	12258 (13199)	∞	∞	∞*2.8e5*	0/15		1796 (1474)	∞	∞	∞	∞	∞	∞*2.0e6*	0/15
$\mathbf{f}_9$	35	127	214	263	300	335	369	15/15	$\mathbf{f}_9$	1716	3102	3277	3379	3455	3594	3727	15/15
	17 (7)	3920 (3354)	∞	∞	∞	∞	∞*5.0e5*	0/15		∞	∞	∞	∞	∞	∞	∞*2.0e6*	0/15
$\mathbf{f}_{10}$	349	500	574	607	626	829	880	15/15	$\mathbf{f}_{10}$	7413	8661	10735	13641	14920	17073	17476	15/15
	6725 (7986)	∞	∞	∞	∞	∞	∞*3.1e5*	0/15		∞	∞	∞	∞	∞	∞	∞*2.0e6*	0/15
$\mathbf{f}_{11}$	143	202	763	977	1177	1467	1673	15/15	$\mathbf{f}_{11}$	1002	2228	6278	8586	9762	12285	14831	15/15
	205 (236)	698 (864)	2442 (2575)	∞	∞	∞	∞*2.6e5*	0/15		17715 (19470)	∞	∞	∞	∞	∞	∞*1.1e6*	0/15
$\mathbf{f}_{12}$	108	268	371	413	461	1303	1494	15/15	$\mathbf{f}_{12}$	1042	1938	2740	3156	4140	12407	13827	15/15
	201 (576)	303 (585)	490 (704)	1392 (1453)	2674 (2913)	∞	∞*2.7e5*	0/15		320 (961)	706 (1035)	2024 (2557)	8893 (9507)	∞	∞	∞*2.0e6*	0/15
$\mathbf{f}_{13}$	132	195	250	319	1310	1752	2255	15/15	$\mathbf{f}_{13}$	652	2021	2751	3507	18749	24455	30201	15/15
	115 (216)	2009 (2341)	9467 (10306)	16356 (17378)	∞	∞	∞*3.3e5*	0/15		529 (907)	1072 (1477)	2052 (2407)	5467 (5911)	1026 (1134)	792 (852)	∞*1.2e6*	0/15
$\mathbf{f}_{14}$	10	41	58	90	139	251	476	15/15	$\mathbf{f}_{14}$	75	239	304	451	932	1648	15661	15/15
	0.62 (0.6)	5.9 (5)	13 (5)	50 (23)	406 (473)	17736 (19499)	∞*3.1e5*	0/15		13 (6)	18 (3)	24 (5)	53 (29)	612 (196)	∞	∞*2.0e6*	0/15
$\mathbf{f}_{15}$	511	9310	19369	19743	20073	20769	21359	14/15	$\mathbf{f}_{15}$	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
	5.1 (4)	50 (60)	81 (91)	120 (151)	118 (142)	115 (142)	114 (130)	2/15		∞	∞	∞	∞	∞	∞	∞*1.2e6*	0/15
$\mathbf{f}_{16}$	120	612	2662	10163	10449	11644	12095	15/15	$\mathbf{f}_{16}$	1384	27265	77015	1.4e5	1.9e5	2.0e5	2.2e5	15/15
	2.3 (2)	25 (7)	50 (79)	38 (49)	69 (80)	196 (217)	∞*2.9e5*	0/15		569 (695)	227 (268)	∞	∞	∞	∞	∞*1.3e6*	0/15
$\mathbf{f}_{17}$	5.2	215	899	2861	3669	6351	7934	15/15	$\mathbf{f}_{17}$	63	1030	4005	12242	30677	56288	80472	15/15
	2.7 (3)	2.3 (0.8)	1.8 (0.6)	27 (33)	47 (57)	308 (353)	∞*2.6e5*	0/15		3.2 (3)	6.0 (2)	294 (397)	∞	∞	∞	∞*1.1e6*	0/15
$\mathbf{f}_{18}$	103	378	3968	8451	9280	10905	12469	15/15	$\mathbf{f}_{18}$	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
	1.5 (1)	3.3 (0.8)	11 (11)	101 (106)	∞	∞	∞*2.6e5*	0/15		4.7 (2)	22 (3)	∞	∞	∞	∞	∞*1.1e6*	0/15
$\mathbf{f}_{19}$	1	1	242	1.0e5	1.2e5	1.2e5	1.2e5	15/15	$\mathbf{f}_{19}$	1	1	3.4e5	4.7e6	6.2e6	6.7e6	6.7e6	15/15
	13 (15)	1532 (999)	2682 (3069)	∞	∞	∞	∞*1.8e5*	0/15		277 (139)	∞	∞	∞	∞	∞	∞*1.1e6*	0/15
$\mathbf{f}_{20}$	16	851	38111	51362	54470	54861	55313	14/15	$\mathbf{f}_{20}$	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
	4.2 (3)	12 (21)	3.4 (3)	2.7 (3)	2.7 (3)	3.0 (3)	3.4 (3)	13/15		15 (3)	33 (33)	∞	∞	∞	∞	∞*1.2e6*	0/15

Table 1: Continued.

	5D								20D							
Δf	$1e+1$	$1e+0$	$1e-1$	$1e-2$	$1e-3$	$1e-5$	$1e-7$	#succ	Δf	$1e+1$	$1e+0$	$1e-1$	$1e-2$	$1e-3$	$1e-5$	$1e-7$	#succ
$\mathbf{f_{21}}$	41	1157	1674	1692	1705	1729	1757	14/15	$\mathbf{f_{21}}$	561	6541	14103	14318	14643	15567	17589	15/15
	1.8 (2)	50 (60)	64 (106)	71 (101)	72 (100)	96 (121)	109 (116)	11/15		261 (4)	338 (435)	304 (365)	300 (362)	295 (344)	280 (317)	250 (301)	4/15
$\mathbf{f_{22}}$	71	386	938	980	1008	1040	1068	14/15	$\mathbf{f_{22}}$	467	5580	23491	24163	24948	26847	1.3*e*5	12/15
	2.5 (2)	113 (258)	364 (426)	545 (684)	754 (889)	1337 (1548)	4297 (4718)	1/15		1036 (2143)	1139 (1385)	∞	∞	∞	∞	∞*1.2e6*	0/15
$\mathbf{f_{23}}$	3.0	518	14249	27890	31654	33030	34256	15/15	$\mathbf{f_{23}}$	3.2	1614	67457	3.7*e*5	4.9*e*5	8.1*e*5	8.4*e*5	15/15
	2.4 (1)	37 (42)	∞	∞	∞	∞	∞*1.7e5*	0/15		1.7 (2)	5701 (6238)	∞	∞	∞	∞	∞*1.1e6*	0/15
$\mathbf{f_{24}}$	1622	2.2*e*5	6.4*e*6	9.6*e*6	9.6*e*6	1.3*e*7	1.3*e*7	3/15	$\mathbf{f_{24}}$	1.3*e*6	7.5*e*6	5.2*e*7	5.2*e*7	5.2*e*7	5.2*e*7	5.2*e*7	3/15
	20 (27)	∞	∞	∞	∞	∞	∞*1.8e5*	0/15		∞	∞	∞	∞	∞	∞	∞*1.1e6*	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.





#FEs/D	f_1–f_{24} in 5-D, maxFE/D = 100018					
	best	10%	25%	**med**	75%	90%
2	0.62	1.0	1.4	2.5	4.0	8.5
10	1.3	1.6	2.9	3.5	5.8	16
100	2.3	2.6	6.4	8.4	13	42
1*e*3	4.2	5.0	12	40	62	4.2*e*2
1*e*4	13	25	32	1.6*e*2	3.9*e*2	1.2*e*3
1*e*5	3.4	35	83	5.1*e*2	1.7*e*3	3.0*e*3
1*e*6	3.4	35	1.4*e*2	8.6*e*2	1.2*e*4	2.2*e*4
RL_{US}/D	3*e*4	4*e*4	5*e*4	5*e*4	7*e*4	1*e*5
#FEs/D	f_1–f_{24} in 20-D, maxFE/D = 100004					
	best	10%	25%	**med**	75%	90%
2	0.94	1.1	5.8	24	40	40
10	1.3	4.8	7.0	16	1.3*e*2	2.0*e*2
100	4.3	6.2	11	21	39	2.7*e*2
1*e*3	5.0	7.6	23	48	1.8*e*2	4.7*e*2
1*e*4	39	53	1.0*e*2	2.8*e*2	5.2*e*2	1.7*e*3
1*e*5	43	2.3*e*2	4.0*e*2	9.4*e*2	1.9*e*3	8.3*e*3
1*e*6	2.4*e*2	4.2*e*2	9.9*e*2	3.9*e*3	1.1*e*4	6.8*e*4
RL_{US}/D	4*e*4	5*e*4	5*e*4	6*e*4	1*e*5	1*e*5

(ii) A test is carried out to check if the maximum number of generations is satisfied and f_{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 #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 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 *M*620 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 (*R*2010*a*).

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_6, 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 Δf. 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 multiminima 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 Michgan, 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.