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 '*survivial 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-α crossover and non-uniform mutation operators to drive the genetic search.

2. PRCGA ALGORITHM

PRCGA consists of five operators namely; tournament selection, blend-α crossover, non-uniform mutation, projection and a stagnation alleviation mechanism. The outline of PRCGA is shown in Algorithm 1.

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.

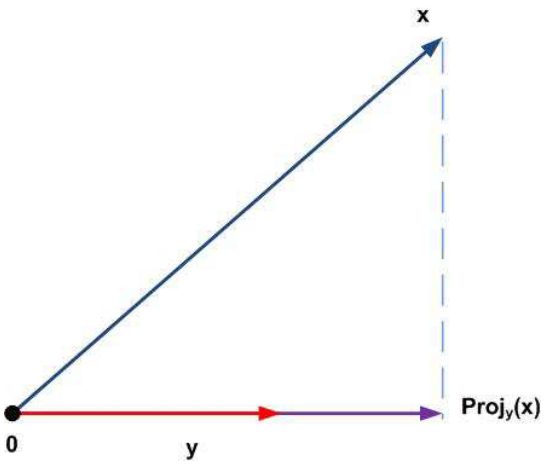


Figure 1: Projection of vector x on vector y

The notations used are defined as:

Let x^* be the global minimizer of the objective function, $f : S \subset \Re^n \rightarrow \Re$ if $f(x^*) \leq f(x), \forall x \in S$. The point $x = \left(x^1, x^2, \ldots, x^n\right)^T \in \Re^n$. The domain of S, is defined by specifying upper $\left(u^j\right)$ and lower $\left(l^j\right)$ limits of each j^{th} component of x, i.e., $l^j \leq x^j \leq u^j$ and $l^j, u^j \in \Re, j = 1, 2, \ldots, n$. 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, ζ_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, Φ_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 $f(x_{i,t}), \forall x_{i,t} \in P_0$ is calculated. At each time step t, the population diversity of P_t is measured by calculating the standard deviation ζ_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 η_{tour} number of solutions uniformly at random from P_t, where η_{tour} is the tournament size, $\eta_{tour} < N$. The fitness values of the selected individuals in η_{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 been 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_{best}; $f(x_{best})$*

1. *Initialize $P_{t=0}, P_t = \{x_{1,t}, x_{2,t}, \ldots, x_{N,t}\}$ from S*
2. *$f(x_{i,t}) =$ 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. *$\hat{P}_t =$ perturb(P_t)*
6. *$\hat{P}_t =$ tournamentSelection(P_t)*
7. *$C_t =$ blend-αCrossover($\hat{P}_t, p_c$)*
8. *$M_t =$ non-uniformMutation(C_t, p_m)*
9. *$\Phi_t =$ projection(M_t)*
10. *$f(x_{i,t}) =$ evaluate(Φ_t)*
11. *$P_{t+1} =$ replace(P_t,Φ_t)*
12. *$t = t + 1$*
13. *end while*

Blend-α crossover is carried out on a pair $(x_{i,t}, x_{k,t})$ when 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-α crossover uniformly draws the new 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

$$\begin{aligned} c^j_{1,t} &= (\min(x^j_{i,t}, x^j_{k,t}) - \alpha * d^j, \max(x^j_{i,t}, x^j_{k,t}) + \alpha * d^j) \\ c^j_{2,t} &= (\min(x^j_{i,t}, x^j_{k,t}) - \alpha * d^j, \max(x^j_{i,t}, x^j_{k,t}) + \alpha * d^j), \end{aligned} \tag{2}$$

where $(1 \leq k \leq N)$, $\alpha = 0.3 + 0.2 \times z$, z is a uniform random number drawn from the interval $[0, 1]$, $d^j = \left|x^j_{i,t} - x^j_{k,t}\right|$. 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

$$m^j_{i,t} = \begin{cases} c^j_{i,t} + \Delta(t, u^j - c^j_{i,t}) & \text{if} \quad \tau \leq 0.5, \\ c^j_{i,t} - \Delta(t, c^j_{i,t} - l^j) & \text{otherwise.} \end{cases} \tag{3}$$

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

$$\Delta(t, y) = y(1 - r^{(1 - \frac{t}{T})})^{\beta}, \tag{4}$$

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 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 Φ_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,

$$\omega_{i,t} = \frac{m_{k,t}^T m_{i,t}}{m_{i,t}^T m_{i,t}} m_{i,t} = \frac{m_{k,t}^T m_{i,t}}{\|m_{i,t}\|^2} m_{i,t} = \left(\frac{\|m_{k,t}\| \cos(\theta)}{\|m_{i,t}\|} m_{i,t} \right) \quad (5)$$

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 θ 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 $\Phi_t = \{\omega_{1,t}, \omega_{2,t}, \ldots, \omega_{N,t}\}$.

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,

$$x_{i,t} = \begin{cases} \omega_{i,t} & \text{if } f(\omega_{i,t}) < f(m_{i,t}), m_{i,t} \in M_t \\ m_{i,t} & \text{otherwise.} \end{cases} \quad (6)$$

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_{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 $\eta_{tour} = 3$, 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 $7.10(R2010a)$.

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^{-8} beyond dimension 3 in Rastrigin Function f_{15} 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 Lunacek 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 tesbed 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^{-8} 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 evolutonary algorithms such as the variants of CMA-ES in [16]. Research is under way to improve PRCGA so that it can comfortable solve most if not all the BBOB-2013 test functions.

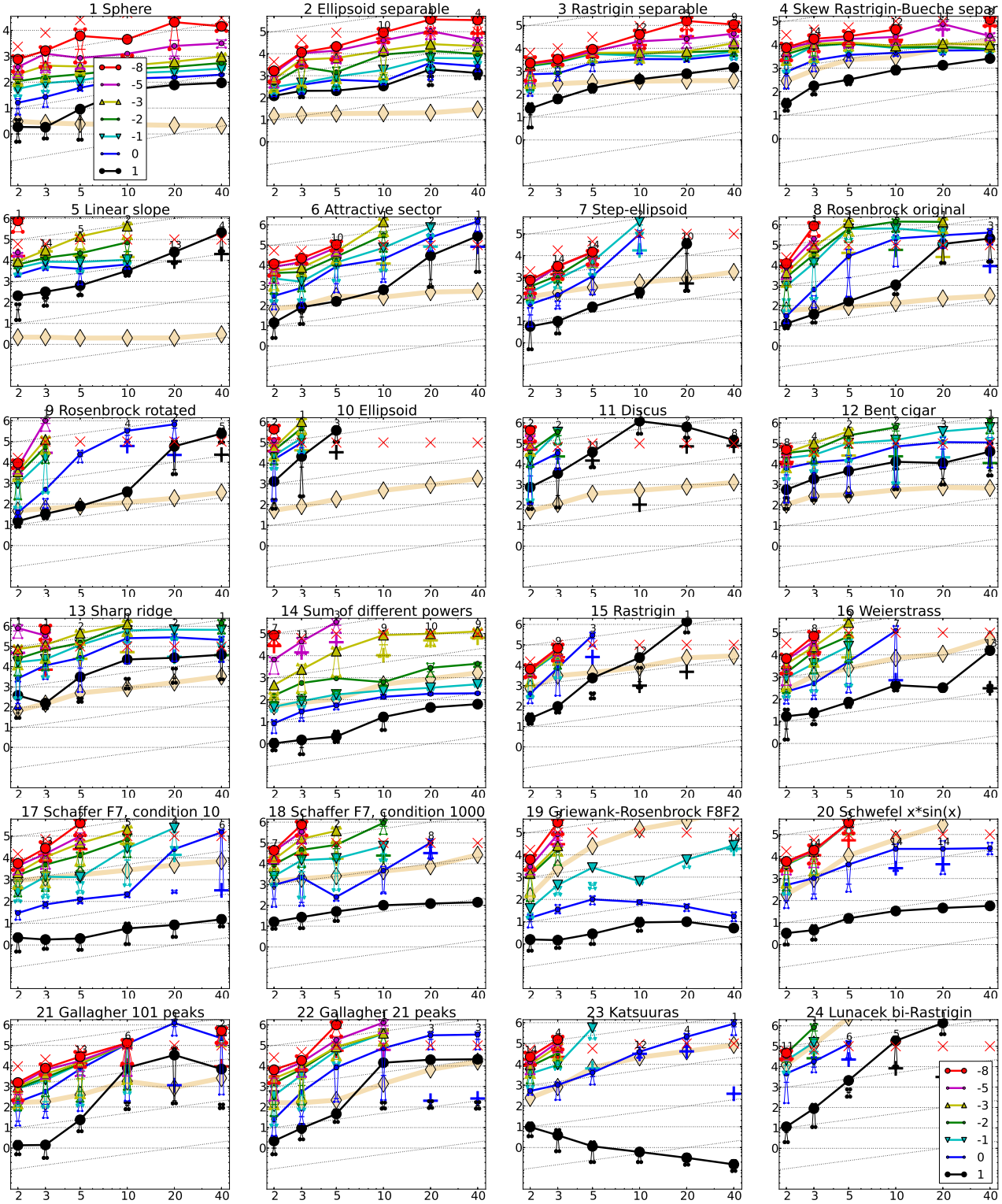


Figure 2: Expected number of f-evaluations (ERT, lines) to reach $f_{\mathrm{opt}} + \Delta f$; median number of f-evaluations (+) to reach the most difficult target that was reached not always but at least once; maximum number of f-evaluations in any trial (×); interquartile range with median (notched boxes) of simulated runlengths to reach $f_{\mathrm{opt}} + \Delta f$; 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.

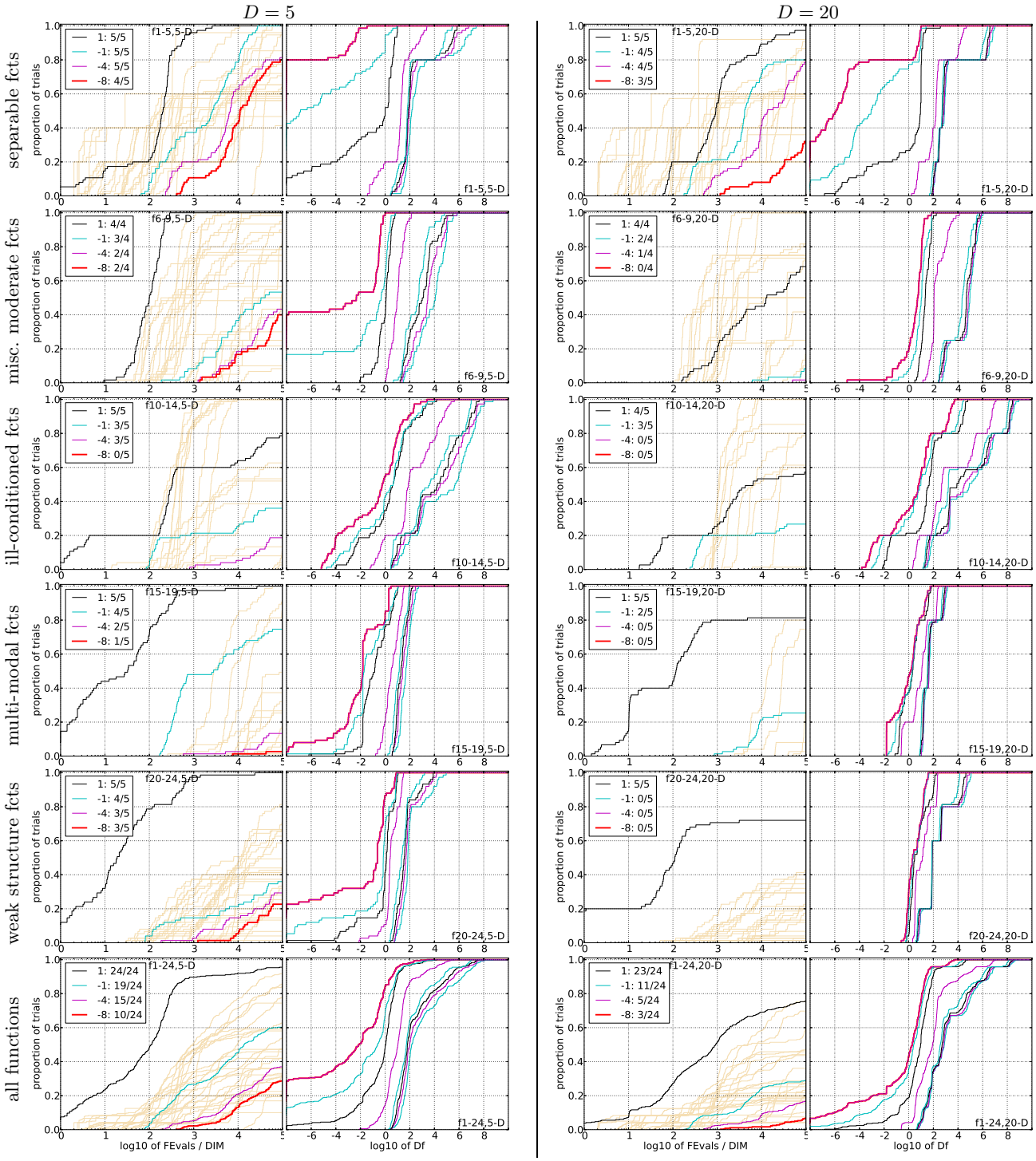


Figure 3: Empirical cumulative distribution functions (ECDF), plotting the fraction of trials with an outcome not larger than the respective value on the x-axis. Left subplots: ECDF of the number of function evaluations (FEvals) divided by search space dimension D, to fall below $f_{\mathrm{opt}} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. The thick red line represents the most difficult target value $f_{\mathrm{opt}} + 10^{-8}$. 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 ECDFs for $\Delta f = 10^{-8}$ of all algorithms benchmarked during BBOB-2009.

	5-D								20-D							
Δf	1e+1	1e+0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ	1e+1	1e+0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f_1	11	12	12	12	12	12	12	15/15	43	43	43	43	43	43	43	15/15
	4.1(7)	25(12)	48(18)	83(24)	166(65)	352(453)	1757(3289)	15/15	36(6)	72(7)	120(27)	182(45)	287(152)	1166(1000)	6963(6562)	15/15
f_2	83	87	88	89	90	92	94	15/15	385	386	387	388	390	391	393	15/15
	13(4)	37(61)	52(88)	80(97)	353(339)	747(1082)	1101(1453)	15/15	101(96)	200(92)	332(267)	699(791)	1402(1249)	5402(5111)	17479(18440)	4/15
f_3	716	1622	1637	1642	1646	1650	1654	15/15	5066	7626	7635	7637	7643	7646	7651	15/15
	1.3(0.6)	7.1(5)	17(18)	19(19)	19(19)	23(24)	26(24)	15/15	3.1(0.7)	8.8(4)	11(4)	18(19)	19(19)	68(70)	275(267)	5/15
f_4	809	1633	1688	1758	1817	1886	1903	15/15	4722	7628	7666	7686	7700	7758	1.4e5	9/15
	2.0(1)	11(8)	32(34)	33(32)	34(31)	46(41)	55(55)	15/15	5.6(2)	15(7)	20(12)	21(12)	29(26)	193(169)	50(49)	0/15
f_5	10	10	10	10	10	10	10	15/15	41	41	41	41	41	41	41	15/15
	318(422)	1999(2627)	4233(3688)	11021(9942)	70495(79261)	∞	∞ *2.7e5*	0/15	2433(16425)	∞	∞	∞	∞	∞	∞ *1.1e6*	0/15
f_6	114	214	281	404	580	1038	1332	15/15	1296	2343	3413	4255	5220	6728	8409	15/15
	7.1(3)	202(355)	243(357)	328(386)	410(442)	362(312)	337(233)	10/15	461(775)	2044(2523)	4301(4653)	∞	∞	∞	∞ *2.0e6*	0/15
f_7	24	324	1171	1451	1572	1572	1597	15/15	1351	4274	9503	16523	16524	16524	16969	15/15
	9.4(5)	17(17)	22(22)	40(65)	45(64)	45(68)	46(67)	14/15	513(801)	∞	∞	∞	∞	∞	∞ *1.1e6*	0/15
f_8	73	273	336	372	391	410	422	15/15	2039	3871	4040	4148	4219	4371	4484	15/15
	11(4)	518(811)	9455(11234)	8747(9309)	16854(19258)	∞	∞ *5.0e5*	0/15	1117(1472)	1580(1768)	2138(2475)	6823(7474)	6759(7822)	∞	∞ *2.0e6*	0/15
f_9	35	127	214	263	300	335	369	15/15	1716	3102	3277	3379	3455	3594	3727	15/15
	11(5)	957(744)	∞	∞	∞	∞	∞ *5.0e5*	0/15	688(798)	4338(4514)	∞	∞	∞	∞	∞ *2.0e6*	0/15
f_{10}	349	500	574	607	626	829	880	15/15	7413	8661	10735	13641	14920	17073	17476	15/15
	5553(6452)	∞	∞	∞	∞	∞	∞ *5.0e5*	0/15	∞	∞	∞	∞	∞	∞	∞ *2.0e6*	0/15
f_{11}	143	202	763	977	1177	1467	1673	15/15	1002	2228	6278	8586	9762	12285	14831	15/15
	1331(1728)	∞	∞	∞	∞	∞	∞ *2.8e5*	0/15	2668(13322)	∞	∞	∞	∞	∞	∞ *2.0e6*	0/15
f_{12}	108	268	371	413	461	1303	1494	15/15	1042	1938	2740	3156	4140	12407	13827	15/15
	214(615)	296(609)	1403(1719)	2982(3388)	4280(5081)	∞	∞ *2.6e5*	0/15	214(226)	1202(1551)	2842(3285)	∞	∞	∞	∞ *2.0e6*	0/15
f_{13}	132	195	250	319	1310	1752	2255	15/15	652	2021	2751	3507	18749	24455	30201	15/15
	118(3)	697(1004)	2369(2864)	2342(2568)	1742(1845)	∞	∞ *3.0e5*	0/15	848(1547)	2814(3522)	4952(5411)	∞	∞	∞	∞ *2.0e6*	0/15
f_{14}	10	41	58	90	139	251	476	15/15	75	239	304	451	932	1648	15661	15/15
	1.1(1)	6.8(3)	15(3)	52(4)	580(784)	6773(7095)	∞ *3.4e5*	0/15	12(5)	16(4)	23(6)	126(102)	2131(1910)	∞	∞ *2.0e6*	0/15
f_{15}	511	9310	19369	19743	20073	20769	21359	15/15	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
	23(24)	157(181)	∞	∞	∞	∞	∞ *2.8e5*	0/15	925(1152)	∞	∞	∞	∞	∞	∞ *2.0e6*	0/15
f_{16}	120	612	2662	10163	10449	11644	12095	15/15	1384	27265	77015	1.4e5	1.9e5	2.0e5	2.2e5	15/15
	3.0(2)	40(27)	46(59)	34(44)	139(161)	∞	∞ *3.4e5*	0/15	4.8(2)	∞	∞	∞	∞	∞	∞ *2.0e6*	0/15
f_{17}	5.2	215	899	2861	3669	6351	7934	15/15	63	1030	4005	12242	30677	56288	80472	15/15
	1.9(2)	2.9(2)	7.2(10)	23(30)	64(89)	78(75)	77(76)	2/15	2.6(2)	490(971)	1170(1457)	∞	∞	∞	∞ *1.7e6*	0/15
f_{18}	103	378	3968	8451	9280	10905	12469	15/15	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
	2.5(3)	3.4(2)	23(31)	48(59)	205(245)	∞	∞ *2.7e5*	0/15	3.9(1)	515(566)	∞	∞	∞	∞	∞ *1.6e6*	0/15
f_{19}	1	1	242	1.0e5	1.2e5	1.2e5	1.2e5	15/15	1	1	3.4e5	4.7e6	6.2e6	6.7e6	6.7e6	15/15
	14(12)	502(302)	55(5)	∞	∞	∞	∞ *2.5e5*	0/15	199(20)	931(428)	0.35(0.2)	∞	∞	∞	∞ *2.0e6*	0/15
f_{20}	16	851	38111	51362	54470	54861	55313	15/15	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
	4.9(4)	25(44)	44(49)	33(36)	31(34)	31(32)	31(31)	3/15	11(5)	10(16)	∞	∞	∞	∞	∞ *1.8e6*	0/15
f_{21}	41	1157	1674	1692	1705	1729	1757	15/15	561	6541	14103	14318	14643	15567	17589	15/15
	2.9(4)	31(71)	38(77)	39(76)	42(80)	52(78)	70(79)	13/15	1215(1783)	3803(4344)	∞	∞	∞	∞	∞ *2.0e6*	0/15
f_{22}	71	386	938	980	1008	1040	1068	15/15	467	5580	23491	24163	24948	26847	1.3e5	12/15
	3.3(3)	109(243)	298(347)	416(496)	420(473)	888(901)	2323(2395)	1/15	867(1787)	1121(1382)	∞	∞	∞	∞	∞ *1.6e6*	0/15
f_{23}	3.0	518	14249	27890	31654	33030	34256	15/15	3.2	1614	67457	3.7e5	4.9e5	8.1e5	8.4e5	15/15
	1.9(2)	39(50)	201(209)	∞	∞	∞	∞ *1.9e5*	0/15	2.0(1)	2615(2716)	∞	∞	∞	∞	∞ *1.1e6*	0/15
f_{24}	1622	2.2e5	6.4e6	9.6e6	9.6e6	1.3e7	1.3e7	3/15	1.3e6	7.5e6	5.2e7	5.2e7	5.2e7	5.2e7	5.2e7	3/15
	6.5(2)	3.0(4)	∞	∞	∞	∞	∞ *3.2e5*	0/15	21(23)	∞	∞	∞	∞	∞	∞ *2.0e6*	0/15

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. The different target Δf-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\text{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Bold entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm in BBOB-2009, with $p = 0.05$ or $p = 10^{-k}$ when the number $k > 1$ is following the $\downarrow$ symbol, with Bonferroni correction by the number of functions.

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

[1] M. M. Ali and A. Törn. A population set-based global optimization algorithms: some modifications and numerical studies. *Computers & Operations Research*, 31(10):1703–1725, 2004.

[2] T. Back. *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. Oxford University press, New York, 1996.

[3] 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*, 148(2):335–348, 2003.

[4] Y.-C. Chuang and C.-T. Chen. Black-box optimization benchmarking for noiseless function testbed using a direction-based rcga. In *GECCO (Companion)*, pages 167–174, 2012.

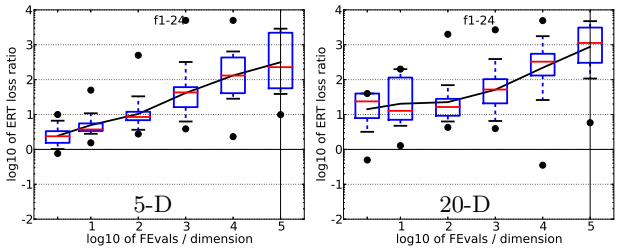
[5] K. Deep and K. N. Das. Quadratic approximation based hybrid genetic algorithm for function optimization. *Applied Mathematics and Computation*, 203:86–98, 2008.

[6] K. Deep and Dipti. A new hybrid self organizing migrating genetic algorithm for function optimization. In D. Srinivasan and L. Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 2796–2803, Singapore, 25-28 September 2007. IEEE Computational Intelligence Society, IEEE Press.

[7] K. Deep and M. Thakur. A new crossover operator for real coded genetic algorithms. *Applied Mathematics and Computation*, 188:895–911, 2007.

[8] K. Deep and M. Thakur. A new mutation operator for real coded genetic algorithms. *Applied Mathematics and Computation*, 193:211–230, 2007.

[9] K. Deep and M. Thakur. A real coded multi parent



f_1–f_{24} in **5-D**, maxFE/D=100004						
#FEs/D	best	10%	25%	**med**	75%	90%
2	0.77	0.91	1.5	2.4	3.3	7.1
10	1.5	2.6	3.4	3.7	5.6	16
100	2.8	3.4	6.9	8.5	12	42
1e3	3.9	5.8	15	43	64	4.2e2
1e4	2.3	27	39	1.3e2	4.4e2	8.6e2
1e5	9.9	37	54	2.3e2	2.2e3	3.0e3
1e6	9.7	26	1.3e2	5.7e2	8.8e3	2.6e4
RL_{US}/D	3e4	5e4	5e4	6e4	9e4	1e5

f_1–f_{24} in **20-D**, maxFE/D=100007						
#FEs/D	best	10%	25%	**med**	75%	90%
2	0.50	2.9	7.1	24	40	40
10	1.3	4.5	7.0	13	1.3e2	2.0e2
100	4.3	6.1	8.8	16	30	2.7e2
1e3	3.9	5.4	20	52	1.1e2	4.7e2
1e4	0.35	24	1.2e2	3.3e2	5.9e2	2.4e3
1e5	5.8	42	2.9e2	1.1e3	3.3e3	5.6e3
1e6	34	1.1e2	7.9e2	5.2e3	9.2e3	1.8e4
RL_{US}/D	4e4	5e4	7e4	1e5	1e5	1e5

Figure 4: 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 5 for results on each function subgroup.**

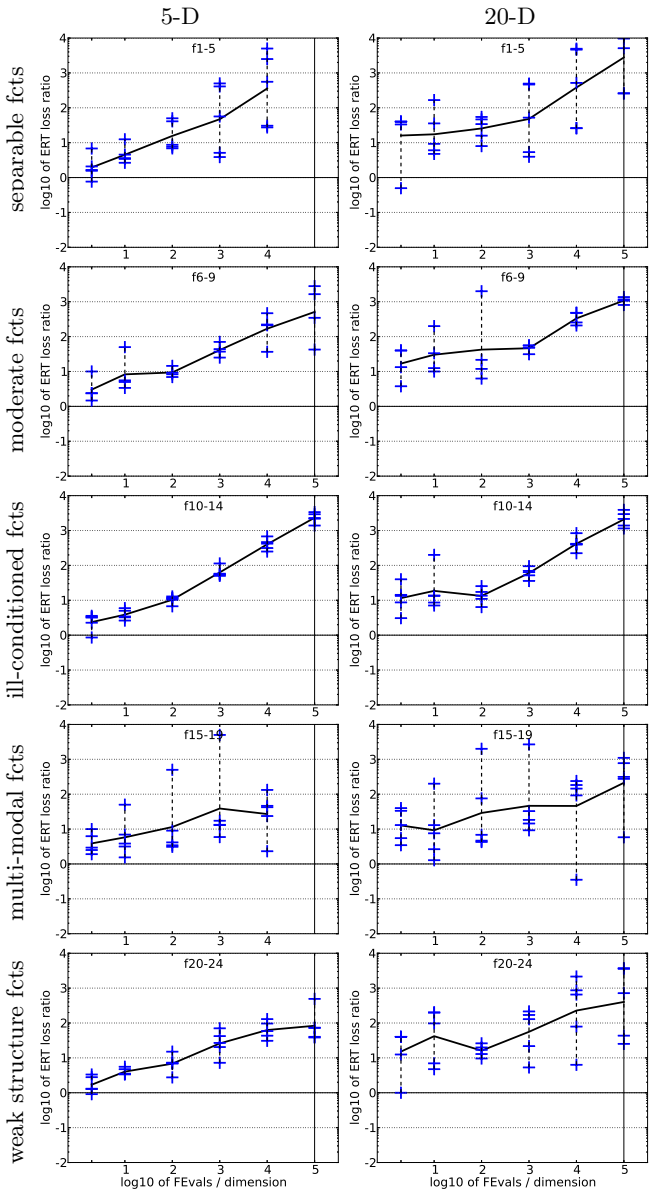


Figure 5: ERT **loss ratios (see Figure 4 for details). Each cross (+) represents a single function, the line is the geometric mean.**

genetic algorithms for function optimization. *Applied Mathematics and Computation*, 1(2):67–83, 2008.

[10] K. A. DeJong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michgan, Ann Arbor, MI, USA, 1975.

[11] A. P. Engelbrecht. *Fundamental of computational swarm intelligence*. John Wiley & Sons, Ltd, West Sussex, England, 2005.

[12] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical Report 2009/20, Research Center PPE, 2009. Updated February 2010.

[13] D. Goldberg. Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems*, 5(2):139–168, 1991.

[14] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[15] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2012: Experimental setup. Technical report, INRIA, 2012.

[16] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, GECCO '10, pages 1689–1696, New York, NY, USA, 2010. ACM.

[17] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking

2009: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2009. Updated February 2010.

[18] J. H. Holland. *Adaptation in natural and artificial systems, An introductory analysis with applications to biology, control and artificial intelligence*. MIT press, Cambridge, Massachusetts, 1975.

[19] P. Kaelo and M. M. Ali. Integrated crossover rules in real coded genetic algorithms. *European Journal of Operational Research*, 176:60–76, 2007.

[20] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, Berlin Heidelberg, N.Y., 1996.

[21] M. Nicolau. Application of a simple binary genetic algorithm to a noiseless testbed benchmark. In *GECCO (Companion)*, pages 2473–2478, 2009.

[22] B. A. Sawyerr. *Hybrid real coded genetic algorithms with pattern search and projection*. PhD thesis, University of Lagos, Lagos, Nigeria, 2010.

[23] 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 and Software*, 26(6):945–970, 2011.

[24] T.-D. Tran and G.-G. Jin. Real-coded genetic algorithm benchmarked on noiseless black-box optimization testbed. In *GECCO (Companion)*, pages 1731–1738, 2010.