Real-Coded Genetic Algorithm Benchmarked on Noiseless Black-Box Optimization Testbed

Thanh-Do Tran
Dept. of Control and Instrumentation Eng.
Korea Maritime University
Busan 606-791, Korea
tdtran@hhu.ac.kr

Gang-Gyoo Jin
Div. of Comp., Control, and Elec. Comm. Eng.
Korea Maritime University
Busan 606-791, Korea
ggjin@hhu.ac.kr

ABSTRACT

Genetic algorithms—a class of stochastic population-based optimization techniques—have been widely realized as the effective tools to solve complicated optimization problems arising from the diverse application domains. Originally developed version was a genetic algorithm with the binary representation of candidate solutions (i.e. chromosomes), the real-coded versions are, however, basically superior and frequently utilized in tackling complex real-valued optimization tasks. In this paper, a real-coded genetic algorithm (RCGA), which employs an adaptive-range variant of the well-known non-uniform mutation, is furnished with a multiple independent restarts mechanism to benchmark the noise-free black-box optimization testbed. The maximum number of function evaluations for each run is set to 50000 times the search space dimension. For low search space dimensions, the algorithm shows encouraging results on several functions. Although the algorithm is unable to solve all the functions to the highest required accuracy, for each type of functions, some of them can be solved, especially to lower precision, with the dimension up to 40.

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

General Terms

Algorithms, Experimentation

Keywords

Benchmarking, Black-box optimization, Evolutionary computation, Real-coded genetic algorithm

1. INTRODUCTION

Genetic algorithms (GAs) are general-purpose stochastic search algorithms belonging to the earliest and the most fundamental branch of evolutionary computation. Soon after John Holland introduced the simple genetic algorithm (SGA) in 1975 [5], GAs have gained widespread and increasing acceptance due to their simplicity and ease of implementation. As a result, a multitude of variations to the SGA have been developed based on various aspects of the basic framework, i.e. representation schemes, genetic operators (selection, crossover, mutation), and other strategies (e.g. elitism).

Among representation schemes available in GAs, floating-point technique, in which each candidate solution (or chromosome vector) is encoded as a vector of floating-point valued numbers of the same length as the dimension of the search space, is the most widely used. Among several selection schemes, tournament selection is one of the commonly used operators owing to its ease of implementation.

The role of selection phase is to create an intermediate population for the subsequent application of other operators (e.g. crossover, mutation). Its main principle is "the better is an individual, the higher is its chance of being parent" [9]. Once an intermediate population is established, the crossover operator will be applied to produce offspring. The offspring may be created through various manners. Notably, arithmetical crossover [7] has become a favorite operator to recombine information of parent solutions so as to yield an offspring solution that benefits from the advantageous information of both parents. Besides, non-uniform mutation [6] has also gained in popularity to introduce new exploratory information into the intermediate population while ensuring exploitation at later stages of the evolution process in order to produce solutions with high precision.

In this work, a genetic algorithm using float-point representation together with the tournament selection, arithmetical crossover and adaptive-range variant of the non-uniform mutation—which is referred to as the real-coded genetic algorithm (RCGA)—is implemented and benchmarked on the noiseless functions of the BBOB-2010 testbed. More details about the RCGA being in use is presented in the following sections.

2. REAL-CODED GENETIC ALGORITHM

Over the years, several variations have been developed within the framework of GAs, however they have still conformed to the general structure as illustrated in Algorithm 1.

2.1 Selection Operator

Genetic algorithms, and evolutionary algorithms in general, search for near-optimal solutions by evolving a population of candidate solutions gradually over generations,

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

Algorithm 1 General structure of a Genetic Algorithm

Input: Fitness function f; parameters; stopping_condition
Output: Best solution, $\mathbf{x}_b$, $f(\mathbf{x}_b)$

Initialize $\mathcal{P}(0), \mathcal{P}(0) = \{\mathbf{x}_i(0), 1 \le i \le N\}$
$\{f(\mathbf{x}_i(0)), 1 \le i \le N\} \leftarrow$ Evaluate($\mathcal{P}(0)$)
$t \leftarrow 1$ // generation counter
while not stopping_condition **do**
  $\mathcal{P}'(t) \leftarrow$ Selection($\mathcal{P}(t)$) // $\mathcal{P}'(t) = \{\mathbf{x}'_i(t)\}$
  $\mathcal{P}'(t) \leftarrow$ Crossover($\mathcal{P}'(t)$)
  $\mathcal{P}'(t) \leftarrow$ Mutation($\mathcal{P}'(t)$)
  $\{f(\mathbf{x}'_i(t))\} \leftarrow$ Evaluate($\mathcal{P}'(t)$)
  $\mathcal{P}(t+1) \leftarrow$ Replace($\mathcal{P}(t), \mathcal{P}'(t)$)
  $t \leftarrow t + 1$ // advance to the next generation
end while
return $\mathbf{x}_b$, $f(\mathbf{x}_b)$

in consequence, which individuals of the current population selected for crossover may have significant influence on the next population.

The underlying idea for selection methods is that better individuals, i.e., higher fitness, have higher chance of being parents in recombination process (crossover and mutation). The rationale for this principle is that, in a population, highly fit members possess good properties that, if recombined in a right way, could lead to even better solutions. The impact of a selection phase can be regulated by the selection pressure which may drive the population to better members. On the contrary, worst members should not be discarded and should have some chance to be selected since they may lead to useful genetic material. In generational genetic algorithms, however, not every individual makes it into the intermediate population. This issue is concerned in all selection schemes and referred to as the *loss of diversity*.

A commonly used selection scheme is the fitness proportional selection, also known as "roulette-wheel selection". As the name suggests, individuals get drawn with probabilities directly proportional to their fitness values. This is in contrast to using the *rank* of an individual to assign these probabilities. This selection scheme, however, may introduce a bias towards an exceptionally fit individual at the beginning of the search which will probably dominate the intermediate population, leading to a loss of diversity and premature convergence. To overcome such drawbacks of the roulette-wheel selection, tournament selection scheme is used in this work.

Tournament selection selects a group of n_{ts} individuals uniformly at random from the population, where $n_{ts} < N$, N is the population size and n_{ts} is the tournament size. The performance of n_{ts} selected individuals is then compared in terms of fitness and the best individual from this group is selected. To select N individuals, the tournament procedure is repeated N times. The pseudo-code of this operator is presented in Algorithm 2.

Algorithm 2 Tournament selection

Input: $\mathcal{P}(t) = \{\mathbf{x}_i(t)\}$, $\mathcal{F}(t) = \{f(\mathbf{x}_i(t))\}$, N, n_{ts}
Output: $\mathcal{P}'(t) = \{\mathbf{x}'_i(t)\}$, $\mathcal{F}'(t) = \{f(\mathbf{x}'_i(t))\}$

for $i = 1$ **to** N **do**
  Tournament_Group $\leftarrow$ Pickup ($\mathcal{P}(t), \mathcal{F}(t), n_{ts}$)
  $(\mathbf{x}_{win}, f(\mathbf{x}_{win})) \leftarrow$ ChooseBest(*Tournament_Group*)
  $(\mathbf{x}'_i, f(\mathbf{x}'_i)) \leftarrow (\mathbf{x}_{win}, f(\mathbf{x}_{win}))$
end for
return $\mathcal{P}'(t) = \{\mathbf{x}'_i(t)\}$, $\mathcal{F}'(t) = \{f(\mathbf{x}'_i(t))\}$

2.2 Crossover Operator

The role of crossover operators is to inherit some characteristics of two parents to generate offspring. For RCGAs, several crossover operators have been suggested. In this version of RCGA, we employ the arithmetical crossover. The implementation of this operator is described as follows. Suppose $\mathbf{x}_1(t) = [x^t_{1,1}, x^t_{1,2}, \cdots, x^t_{1,D}]$ and $\mathbf{x}_2(t) = [x^t_{2,1}, x^t_{2,2}, \cdots, x^t_{2,D}]$ are two parent individuals, chosen uniformly at random from the population, subject to be crossed at generation t, $\mathbf{x}'_1(t) = [x'^t_{1,1}, x'^t_{1,2}, \cdots, x'^t_{1,D}]$ and $\mathbf{x}'_2(t) = [x'^t_{2,1}, x'^t_{2,2}, \cdots, x'^t_{2,D}]$ are the corresponding offspring pair generated by the crossover with probability $p_c \in (0, 1]$. Then each element of the produced offspring is a combination of two corresponding elements from the two parents:

$$x'^t_{1,j} = \lambda_j x^t_{1,j} + (1 - \lambda_j) x^t_{2,j}$$
$$x'^t_{2,j} = (1 - \lambda_j) x^t_{1,j} + \lambda_j x^t_{2,j},$$

where λ_j is a random number in $(0, 1)$ which is uniformly drawn anew for each element index j.

2.3 Mutation Operator

Several mutation schemes have been suggested to enhance the performance of RCGAs. Among those, non-uniform mutation is a remarkable scheme which was introduced by Janikow and Michalewicz [6]. It is designed for assuring uniform exploration of the search space initially while giving a fine-tuning capability aimed at achieving high precision at the final stages in the course of evolution. For a given parent individual $\mathbf{x}_i(t) = [x^t_{i,1},\ x^t_{i,2}, \cdots, x^t_{i,D}]$ at generation t, each element $x^t_{i,j}$ $(1 \le j \le D)$ of it has exactly equal chance of undergoing the mutative process. Assuming that the element $x^t_{i,k}$ is, with a mutation probability p_m, selected for mutation, then the resulting offspring is $\mathbf{x}'_i(t) = [x^t_{i,1}, \cdots, x'^t_{i,k}, \cdots, x^t_{i,D}]$. The mutation-generated element $x'^t_{i,k}$ is produced by adding to or subtracting from the original element $x^t_{i,k}$ a perturbation amount as follows:

$$x'^t_{i,k} = \begin{cases} x^t_{i,k} \ + \ \Delta(t, ub_i - x^t_{i,k}) & \text{with prob. } q \\ x^t_{i,k} \ - \ \Delta(t, x^t_{i,k} - lb_i) & \text{with prob. } 1{-}q, \end{cases} \tag{1}$$

where $\Delta(t, y)$ is the perturbation function, dependent on generation t and position y of the original value relative to the search boundaries. This function returns a value in the range $[0, y]$ in such a way that the returned value approaches 0 as t increases. This property causes the operator to search the space uniformly at early stages and very locally at later stages. The function $\Delta(t, y)$ is given as follows:

$$\Delta(t, y) \ = \ y \left[1 - r^{\gamma(t)}\right]. \tag{2}$$

Here r is a random number uniformly distributed in the interval [0, 1] and $\gamma(t)$ is the strategy function of generation t which provides the fine-tuning capability according to the following relation:

$$\gamma(t) = \left(1 - \frac{t}{T}\right)^{\beta}, \tag{3}$$

where T is the maximum number of generation and β the exogenous strategy parameter which determines the degree of non-uniformity across generations.

Originally, the value of q is proposed to be 0.5 [7] which means that mutation to the left and right of the original value is equally likely. Neubauer [8] demonstrated that non-uniform mutation is not a zero-mean deviation operator. By forming an expression for the expected value of mutation and using the above perturbation, the mutation was shown to concentrate the search between the parent value and the center of the search range. As a modification to overcome such limitations, an adaptive range variant was introduced by Austin [1]. The modified operator establishes a mutation range $x \pm \Delta(t, y)$ based on the generation number t and a fixed preset value y, then randomly selects a point within this range. The mutation range is redefined as follows:

$$\begin{cases} \sigma_L = \mathbf{max}\{lb_i,\ x_{i,k}^t - \Delta(t,y)\} \\ \sigma_U = \mathbf{min}\{ub_i,\ x_{i,k}^t + \Delta(t,y)\}, \end{cases} \tag{4}$$

while $y = ub_i - lb_i$. The mutation returns a random value within the range $[\sigma_L,\ \sigma_U]$ with the assurance of symmetry about the parent value $x_{i,k}^t$, that is:

$$x_{i,k}'^t = \begin{cases} x_{i,k}^t\ -\ (1-2p)(x_{i,k}^t - \sigma_L) & \text{if } q \le 0.5 \\ x_{i,k}^t\ +\ (2p-1)(\sigma_U - x_{i,k}^t) & \text{otherwise.} \end{cases} \tag{5}$$

2.4 Replacement Strategy

The replacement phase concerns the survivor selection of both parent and offspring population. Since the size of the population is constant, it allows to withdraw individuals according to a given strategy. There are two main strategies in GAs, namely generational and steady-state. Generational replacement will concern the whole population of size N. The offspring population will replace systematically the parent population. On the contrary, in steady-state replacement, at each generation, only one offspring is generated. For instance, it replaces the worst individual of the parent population. In this implementation of RCGA, the generational replacement strategy is utilized. Besides, an elitist strategy is also used to ensure the survival of the best individual of the current population to the next generation.

3. INDEPENDENT RESTARTS SCHEME

A multiple independent restarts version of the RCGA is implemented in this work. For each restart, the initial solutions $\{\mathbf{x}_i(0)\}$ are uniformly sampled at random within the search bound, i.e., $[-5,\ 5]^D$. Whenever the stopping conditions are met, the algorithm will be reinitialized and restarted without inheriting any information about the last run. This process is iterated until the mission is accomplished, i.e., the objective function value is less than the target function value, or the total number of function evaluations surpasses 5×10^4.

There are two stopping conditions for the multiple independent restarts scheme. The first one is the maximum number of iterations in each run of the algorithm. This value is computed according to the search space dimensionality, where the function $\mathsf{floor}(100 + 3800D\sqrt{D})$ is chosen empirically in this study. The second condition is that the best objective function values obtained so far and during the last $(50+25D)$ generations do not vary more than 10^{-12}. It means that there is no significant improvement in the population after evolving over several consecutive generations. Whenever each of these two conditions is met, the algorithm will be terminated and start again from scratch.

4. PARAMETER SETTINGS

In order to implement the proposed version of RCGA, we used the population size $N = 100$; crossover rate $p_c = 0.95$, tournament size $n_{ts} = 2$; mutation rate $p_m = \{0.05, 0.1, 0.2\}$ which was in turn assigned to each restart, and the degree of non-uniformity for the mutation was $\beta = 5$.

No parameter tuning has been conducted. The final parameter settings were identical for all functions and therefore the crafting effort [3] computes to CrE $= 0$.

5. CPU TIMING EXPERIMENTS

CPU timing experiments were conducted using the same multiple independent restarts RCGA which was particularly run on f_8 with a maximum of $10^5 \times D$ function evaluations and restarted until at least 30 seconds had passed (according to Figure 2 in [3]). These experiments have been implemented on an Intel Core 2 Duo CPU E8400 running at 3.00 GHz under Windows XP SP2 with Matlab 7.8.0 (R2009a). The results are given in Table 3. These results show that the multiple independent restarts implementation of the RCGA takes on approximately 30, 20, 12, 6.0, 3.0, 3.5, and 4.7 times 10^{-5} seconds per function evaluation for 2-, 3-, 5-, 10-, 20-, 40-, and 80-dimensional search space, respectively. The implemented RCGA is a CPU-inexpensive algorithm, i.e., an increase in dimension, up to 40-D, has negligible influence on the CPU time. Moreover, up to 80-D, the dependency of CPU time on the search space dimensionality is trivial.

6. EXPERIMENTAL RESULTS

The results from the experiments according to [3] on the benchmark functions given in [2, 4] are shown in Figures 1, 2, 3 and in Tables 1, 2.

The obtained results from Figure 1 show that RCGA has encouraging performance on all types of test functions, there is at least one representative function within each type that can be solved in the dimension of 5 or even 10. Besides, it can be observed that the expected number of function evaluations to reach a given target function value often scales quadratically with the dimension on almost all functions (Figure 1). To low precision, such as 10^{-3}, the scaling is maintained or even better than quadratical on functions f_5, f_6, f_{14}, f_{17}, f_{21}.

Some functions that turn out to be laborious for this RCGA implementation are Skew Rastrigin-Bueche (f_4), Rosenbrock (f_8, f_9), Ellipsoid (f_{10}), Discuss (f_{11}), Lunacek bi-Rastrigin (f_{24}). Apart from these functions, which fall into different types, performance of the algorithm with a not-so-high required resolution is acceptable within the given computational budget, fruitful results are accomplished on functions Sphere (f_1), Linear slope (f_5), Attractive sector (f_6), Sum of different powers (f_{14}), Schaffer F7 with condition 10 (f_{17}), Gallagher 101 peaks (f_{21}). No matter whether these reported results are competitive or not, the overall outcome from all experiments demonstrates that the implemented version of RCGA generally exploits the separability of the problems.

7. CONCLUSIONS

The real-coded genetic algorithm, which is composed of the tournament selection, the arithmetical crossover, and the adaptive-range non-uniform mutation, incorporated with a multiple independent restarts mechanism has presented encouraging results when benchmarked on noiseless black-box

Table 2: ERT loss ratio (see Figure 3) compared to the respective best result from BBOB-2009 for budgets given in the first column. The last row $\mathrm{RL_{US}}$/D gives the number of function evaluations in unsuccessful runs divided by dimension. Shown are the smallest, 10%-tile, 25%-tile, 50%-tile, 75%-tile and 90%-tile values (smaller values are better).

	f_1–f_{24} in 5-D, maxFE/D=50000					
#FEs/D	best	10%	25%	**med**	75%	90%
2	1.2	1.6	2.0	2.6	3.7	8.3
10	1.6	3.1	3.5	5.1	8.3	50
100	4.2	6.7	8.7	17	24	96
1e3	5.0	11	29	61	89	4.2e2
1e4	22	31	81	2.9e2	4.7e2	2.3e3
1e5	23	61	1.3e2	6.7e2	2.3e3	3.6e3
$\mathrm{RL_{US}}$/D	5e4	5e4	5e4	5e4	5e4	5e4

	f_1–f_{24} in 20-D, maxFE/D=50000					
#FEs/D	best	10%	25%	**med**	75%	90%
2	1.0	5.1	11	31	40	40
10	4.8	6.9	11	51	2.0e2	2.0e2
100	6.4	6.8	14	26	37	2.9e2
1e3	14	18	25	60	1.8e2	4.7e2
1e4	6.6	32	62	2.9e2	5.9e2	4.7e3
1e5	3.4	24	79	6.8e2	2.8e3	6.9e3
1e6	5.8	85	3.0e2	3.3e3	1.8e4	3.1e4
$\mathrm{RL_{US}}$/D	5e4	5e4	5e4	5e4	5e4	5e4

Table 3: Results for CPU timing experiments. Shown are dimension, D, number of times the timing procedure is repeated within 30 seconds, n, and CPU-time per function evaluation, t ($\times 10^{-5}$ second).

D	2	3	5	10	20	40	80
n	125574	119864	108579	78598	6378	1	1
t	30	20	12	6.0	3.0	3.5	4.7

optimization testbed. Without producing impressive results, the algorithm is unable to be considered as a competitive algorithm against other state-of-the-art evolutionary computation approaches, such as some modifications of the covariance matrix adaptation evolution strategy (CMA-ES); nevertheless, the implemented RCGA not only displays potential scaling property but also exhibits the robustness on various types of problems with relatively small number of variables (less than 20). On the other hand, the multiple independent restarts strategy induces comparatively effective search on unstructured multi-modal landscapes. Further modifications so as to adaptively control the parameters, such as crossover rate, mutation rate, degree of non-uniform, and/or restart strategy, during the course of evolution may improve the performance and the efficiency of the algorithm.

Acknowledgments

The authors would like to acknowledge the constructive comments and suggestions of Dr. Anne Auger at The French National Institute of Research in Computer Science and Control (INRIA).

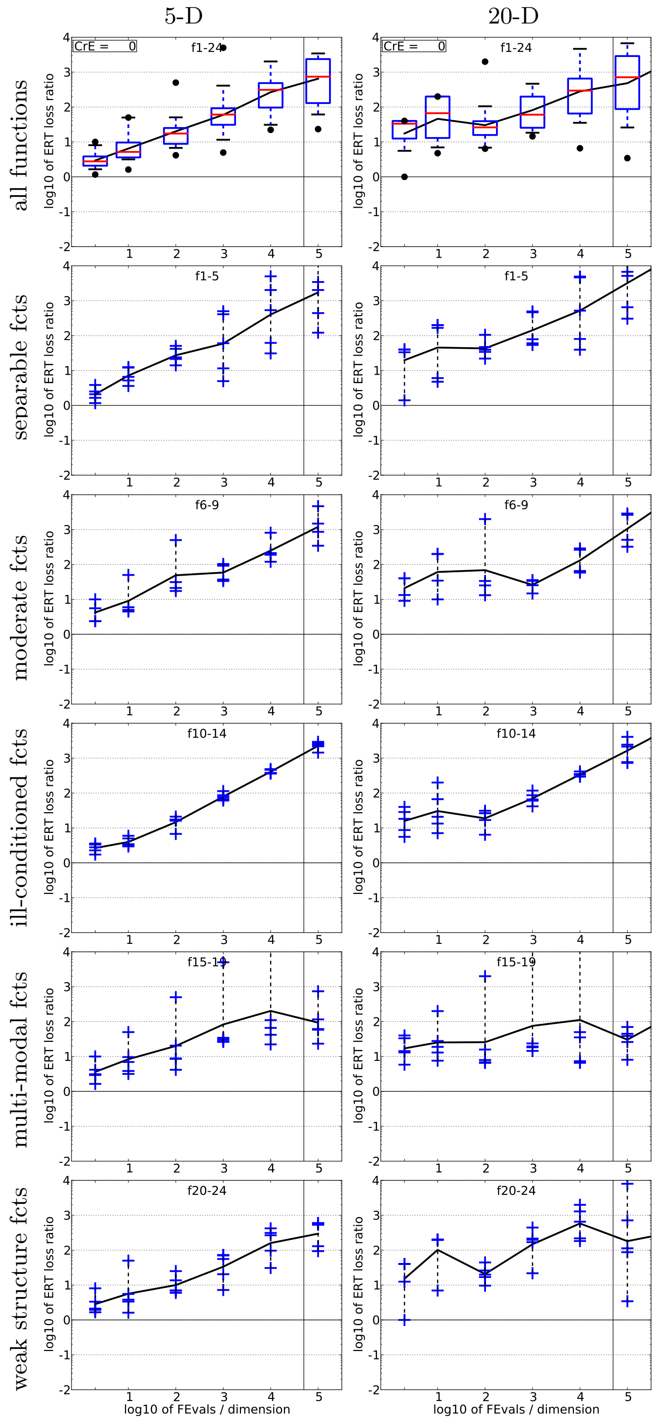

Figure 3: ERT loss ratio versus given budget FEvals. The target value f_t for ERT (see Figure 1) is the smallest (best) recorded function value such that $\mathrm{ERT}(f_t) \leq \mathrm{FEvals}$ for the presented algorithm. Shown is FEvals divided by the respective best $\mathrm{ERT}(f_t)$ from BBOB-2009 for functions f_1–f_{24} in 5-D and 20-D. Each ERT is multiplied by $\exp(\mathrm{CrE})$ correcting for the parameter crafting effort. Line: geometric mean. Box-Whisker error bar: 25-75%-tile with median (box), 10-90%-tile (caps), and minimum and maximum ERT loss ratio (points). The vertical line gives the maximal number of function evaluations in this function subset.

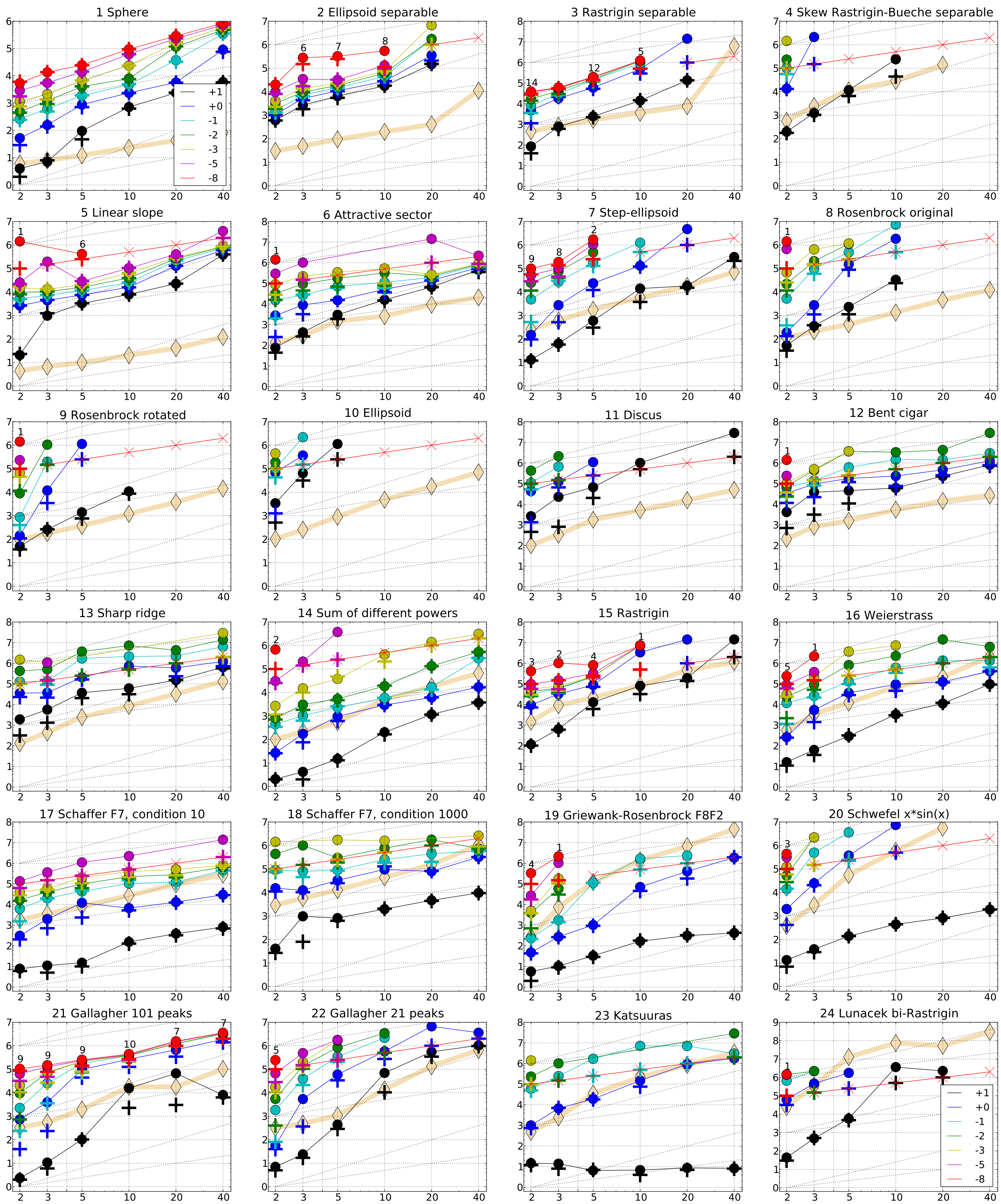

Figure 1: Expected Running Time (ERT, ●) to reach $f_{\rm opt}+\Delta f$ and median number of f-evaluations from successful trials (+), for $\Delta f = 10^{\{+1,0,-1,-2,-3,-5,-8\}}$ (the exponent is given in the legend of f_1 and f_{24}) versus dimension in log-log presentation. For each function and dimension, $\mathrm{ERT}(\Delta f)$ equals to $\#\mathrm{FEs}(\Delta f)$ divided by the number of successful trials, where a trial is successful if $f_{\rm opt}+\Delta f$ was surpassed. The $\#\mathrm{FEs}(\Delta f)$ are the total number (sum) of f-evaluations while $f_{\rm opt}+\Delta f$ was not surpassed in the trial, from all (successful and unsuccessful) trials, and $f_{\rm opt}$ is the optimal function value. Crosses (×) indicate the total number of f-evaluations, $\#\mathrm{FEs}(-\infty)$, divided by the number of trials. Numbers above ERT-symbols indicate the number of successful trials. Y-axis annotations are decimal logarithms. The thick light line with diamonds shows the single best results from BBOB-2009 for $\Delta f = 10^{-8}$. Additional grid lines show linear and quadratic scaling.

Δf	**f_1 in 5-D**, N=15, mFE=29500					**f_1 in 20-D**, N=15, mFE=292100				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	9.5e1	1.0e1	2.7e2	9.5e1	15	2.4e3	1.8e3	3.1e3	2.4e3
1	15	8.8e2	5.2e2	1.6e3	8.8e2	15	5.7e3	4.7e3	6.8e3	5.7e3
1e−1	15	1.9e3	1.1e3	2.8e3	1.9e3	15	3.7e4	1.2e4	6.8e4	3.7e4
1e−3	15	6.8e3	5.9e3	8.8e3	6.8e3	15	1.6e5	1.5e5	1.8e5	1.6e5
1e−5	15	1.4e4	1.2e4	1.6e4	1.4e4	15	2.3e5	2.3e5	2.4e5	2.3e5
1e−8	15	2.5e4	2.3e4	2.9e4	2.5e4	15	2.9e5	2.9e5	2.9e5	2.9e5

Δf	**f_2 in 5-D**, N=15, mFE=250000					**f_2 in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	6.5e3	3.4e3	1.0e4	6.5e3	15	1.5e5	1.3e5	1.6e5	1.5e5
1	15	1.1e4	6.3e3	1.5e4	1.1e4	13	3.6e5	1.7e5	1.2e6	2.0e5
1e−1	15	1.5e4	1.2e4	1.9e4	1.5e4	6	1.7e6	2.1e5	4.7e6	2.2e5
1e−3	15	2.3e4	1.7e4	2.6e4	2.3e4	2	6.8e6	7.8e5	1.3e7	2.8e5
1e−5	15	3.2e4	2.9e4	3.5e4	3.2e4	0	*51e−2*	*47e−5*	*16e−1*	3.4e5
1e−8	7	3.3e5	4.0e4	7.9e5	4.0e4	.	.	.	.	.

Δf	**f_3 in 5-D**, N=15, mFE=250000					**f_3 in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	2.5e3	1.2e3	3.3e3	2.5e3	15	1.4e5	1.3e5	1.6e5	1.4e5
1	15	6.0e4	4.0e4	1.0e5	6.0e4	1	1.4e7	2.2e6	3.2e7	2.4e5
1e−1	13	1.5e5	4.7e4	3.1e5	1.1e5	0	*41e−1*	*10e−1*	*60e−1*	3.3e5
1e−3	12	1.9e5	5.8e4	4.4e5	1.3e5	.	.	.	.	.
1e−5	12	1.9e5	6.5e4	3.9e5	1.3e5	.	.	.	.	.
1e−8	12	2.0e5	7.2e4	4.0e5	1.4e5	.	.	.	.	.

Δf	**f_4 in 5-D**, N=15, mFE=250000					**f_4 in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	1.2e4	4.6e3	3.1e4	1.2e4	0	*22e+0*	*18e+0*	*26e+0*	2.0e5
1	0	*30e−1*	*20e−1*	*40e−1*	1.1e5	.	.	.	.	.
1e−1	.	.	.	.	.	.	.	.	.	.
1e−3	.	.	.	.	.	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_5 in 5-D**, N=15, mFE=250000					**f_5 in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	3.3e3	2.9e3	3.8e3	3.3e3	15	2.3e4	2.0e4	2.5e4	2.3e4
1	15	7.5e3	6.5e3	8.4e3	7.5e3	15	1.3e5	1.2e5	1.4e5	1.3e5
1e−1	15	1.1e4	1.0e4	1.3e4	1.1e4	15	2.1e5	2.0e5	2.2e5	2.1e5
1e−3	15	2.0e4	1.8e4	2.2e4	2.0e4	15	2.9e5	2.9e5	3.0e5	2.9e5
1e−5	15	2.9e4	2.6e4	3.1e4	2.9e4	14	4.0e5	3.3e5	1.3e6	3.3e5
1e−8	6	4.1e5	3.8e4	6.6e5	3.9e4	0	*64e−7*	*33e−7*	*97e−7*	3.4e5

Δf	**f_6 in 5-D**, N=15, mFE=250000					**f_6 in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	3.0e3	6.5e2	8.5e3	3.0e3	15	6.7e4	5.3e4	8.0e4	6.7e4
1	15	1.5e4	4.0e3	2.1e4	1.5e4	15	1.4e5	1.3e5	1.5e5	1.4e5
1e−1	14	7.2e4	1.2e4	1.6e5	5.4e4	15	1.9e5	1.9e5	2.0e5	1.9e5
1e−3	7	3.5e5	3.5e4	8.3e5	6.1e4	15	2.7e5	2.6e5	2.8e5	2.7e5
1e−5	0	*47e−4*	*27e−5*	*67e−3*	4.2e4	1	1.4e7	1.3e6	3.2e7	3.2e5
1e−8	.	.	.	.	.	0	*63e−6*	*16e−6*	*28e−5*	3.4e5

Δf	**f_7 in 5-D**, N=15, mFE=250000					**f_7 in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	6.0e2	2.0e2	8.8e2	6.0e2	15	1.8e4	1.2e4	3.4e4	1.8e4
1	15	2.4e4	1.1e3	6.6e4	2.4e4	3	4.7e6	8.7e5	1.1e7	7.5e5
1e−1	12	1.7e5	4.9e4	3.5e5	1.1e5	0	*13e−1*	*81e−2*	*19e−1*	4.9e5
1e−3	3	1.1e6	8.3e4	2.7e6	8.5e4	.	.	.	.	.
1e−5	3	1.1e6	7.5e4	2.8e6	8.5e4	.	.	.	.	.
1e−8	2	1.7e6	1.0e5	4.6e6	9.0e4	.	.	.	.	.

Δf	**f_8 in 5-D**, N=15, mFE=250000					**f_8 in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	2.3e3	7.7e2	7.1e3	2.3e3	0	*17e+0*	*17e+0*	*17e+0*	3.4e5
1	11	1.5e5	1.0e4	3.9e5	6.2e4	.	.	.	.	.
1e−1	6	4.9e5	1.8e4	1.1e6	1.1e5	.	.	.	.	.
1e−3	3	1.2e6	1.4e5	2.1e6	1.7e5	.	.	.	.	.
1e−5	0	*54e−2*	*13e−5*	*23e−1*	1.2e5	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_9 in 5-D**, N=15, mFE=250000					**f_9 in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	1.4e3	6.3e2	3.1e3	1.4e3	0	*17e+0*	*16e+0*	*18e+0*	3.4e5
1	3	1.1e6	1.6e5	2.8e6	1.4e5	.	.	.	.	.
1e−1	0	*17e−1*	*20e−2*	*24e−1*	1.5e5	.	.	.	.	.
1e−3	.	.	.	.	.	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{10} in 5-D**, N=15, mFE=250000					**f_{10} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	3	1.1e6	1.4e5	2.2e6	1.3e5	0	*48e+2*	*25e+2*	*82e+2*	3.4e5
1	0	*72e+0*	*40e−1*	*49e+1*	1.3e5	.	.	.	.	.
1e−1	.	.	.	.	.	.	.	.	.	.
1e−3	.	.	.	.	.	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{11} in 5-D**, N=15, mFE=250000					**f_{11} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	13	6.7e4	3.4e2	2.5e5	2.9e4	0	*51e+0*	*13e+0*	*64e+0*	4.4e5
1	3	1.1e6	1.3e5	2.2e6	1.0e5	.	.	.	.	.
1e−1	0	*26e−1*	*44e−2*	*11e+0*	1.5e5	.	.	.	.	.
1e−3	.	.	.	.	.	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{12} in 5-D**, N=15, mFE=250000					**f_{12} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	14	4.6e4	8.7e3	1.5e5	2.8e4	15	2.3e5	2.2e5	2.4e5	2.3e5
1	11	1.8e5	1.3e4	5.1e5	8.7e4	13	4.7e5	2.5e5	1.0e6	3.1e5
1e−1	5	6.2e5	8.3e4	1.3e6	1.2e5	7	1.4e6	2.7e5	3.3e6	2.8e5
1e−3	1	3.7e6	4.3e5	9.9e6	1.8e5	0	*12e−2*	*45e−4*	*13e−1*	3.4e5
1e−5	0	*29e−2*	*10e−3*	*94e−1*	1.7e5	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{13} in 5-D**, N=15, mFE=250000					**f_{13} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	3.7e4	7.3e3	9.5e4	3.7e4	15	1.5e5	1.4e5	1.6e5	1.5e5
1	11	2.1e5	2.9e4	4.4e5	1.2e5	11	6.0e5	2.2e5	1.2e6	2.4e5
1e−1	2	1.7e6	2.9e5	4.3e6	1.0e5	5	2.3e6	2.7e5	6.3e6	2.7e5
1e−3	0	*50e−2*	*35e−3*	*17e−1*	1.3e5	0	*28e−2*	*37e−4*	*22e−1*	3.4e5
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{14} in 5-D**, N=15, mFE=250000					**f_{14} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	1.5e1	4.0e0	2.6e1	1.5e1	15	1.2e3	8.0e2	1.5e3	1.2e3
1	15	9.6e2	4.1e2	2.2e3	9.6e2	15	6.1e3	5.3e3	6.7e3	6.1e3
1e−1	15	2.3e3	7.9e2	3.8e3	2.3e3	15	1.7e4	1.0e4	3.3e4	1.7e4
1e−3	15	3.8e4	7.2e3	7.5e4	3.8e4	7	1.4e6	2.5e5	3.3e6	2.6e5
1e−5	1	3.7e6	4.9e5	9.9e6	2.4e5	0	*11e−4*	*69e−5*	*15e−4*	3.4e5
1e−8	0	*90e−6*	*16e−6*	*18e−5*	1.9e5	.	.	.	.	.

Δf	**f_{15} in 5-D**, N=15, mFE=250000					**f_{15} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	1.3e4	1.2e3	4.6e4	1.3e4	15	1.9e5	1.2e5	1.6e5	1.9e5
1	15	8.8e4	5.4e4	1.5e5	8.8e4	1	1.4e7	1.2e6	3.0e7	2.4e5
1e−1	11	2.2e5	8.3e4	4.4e5	1.3e5	0	*50e−1*	*30e−1*	*77e−1*	3.4e5
1e−3	11	2.3e5	6.6e4	4.6e5	1.4e5	.	.	.	.	.
1e−5	9	3.2e5	7.6e4	6.2e5	1.5e5	.	.	.	.	.
1e−8	4	8.1e5	8.3e4	2.0e6	1.2e5	.	.	.	.	.

Δf	**f_{16} in 5-D**, N=15, mFE=250000					**f_{16} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	2.9e2	3.4e1	6.2e2	2.9e2	15	1.1e4	6.7e3	1.5e4	1.1e4
1	15	3.7e4	1.1e4	5.4e4	3.7e4	15	1.3e5	8.6e4	1.7e5	1.3e5
1e−1	13	1.3e5	4.9e4	2.5e5	9.2e4	7	1.4e6	2.1e5	3.2e6	2.2e5
1e−3	1	3.7e6	4.2e5	7.7e6	1.7e5	0	*12e−2*	*18e−3*	*43e−2*	3.4e5
1e−5	0	*16e−3*	*15e−4*	*27e−2*	1.1e5	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{17} in 5-D**, N=15, mFE=250000					**f_{17} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	1.5e1	3.0e0	3.9e1	1.5e1	15	3.9e2	1.4e2	8.4e2	3.9e2
1	15	1.2e4	1.1e3	4.4e4	1.2e4	15	1.3e4	8.9e3	2.0e4	1.3e4
1e−1	15	4.4e4	7.5e3	7.1e4	4.4e4	15	1.2e5	9.6e4	1.4e5	1.2e5
1e−3	10	2.1e5	6.1e4	5.7e5	8.1e4	12	5.1e5	2.5e5	1.3e6	2.6e5
1e−5	3	1.1e6	1.2e5	2.3e6	9.7e4	0	*12e−5*	*16e−6*	*78e−4*	3.4e5
1e−8	0	*30e−5*	*32e−7*	*38e−4*	8.5e4	.	.	.	.	.

Δf	**f_{18} in 5-D**, N=15, mFE=250000					**f_{18} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	8.1e2	2.3e2	1.8e3	8.1e2	15	4.7e3	2.9e3	7.2e3	4.7e3
1	15	3.3e4	3.8e3	6.7e4	3.3e4	15	7.9e4	3.4e4	1.2e5	7.9e4
1e−1	15	9.0e4	5.1e4	1.4e5	9.0e4	12	4.5e5	1.7e5	1.2e6	2.0e5
1e−3	2	1.7e6	1.2e5	3.4e6	1.1e5	0	*16e−3*	*34e−4*	*18e−2*	3.4e5
1e−5	0	*53e−4*	*77e−5*	*38e−3*	1.1e5	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{19} in 5-D**, N=15, mFE=250000					**f_{19} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	3.3e1	5.0e0	5.1e1	3.3e1	15	3.1e2	2.1e2	3.5e2	3.1e2
1	15	1.0e3	4.1e2	1.7e3	1.0e3	15	4.1e5	5.5e4	9.9e5	4.1e5
1e−1	15	1.1e5	2.5e4	2.1e5	1.1e5	5	2.4e6	1.9e5	5.1e6	3.7e5
1e−3	0	*63e−3*	*22e−3*	*96e−3*	1.5e5	0	*23e−2*	*65e−3*	*77e−2*	3.4e5
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{20} in 5-D**, N=15, mFE=250000					**f_{20} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	1.4e2	4.0e1	2.4e2	1.4e2	15	8.2e2	5.3e2	1.1e3	8.2e2
1	8	3.7e5	8.5e4	7.3e5	1.5e5	0	*22e−1*	*16e−1*	*28e−1*	3.7e5
1e−1	1	3.6e6	3.6e5	7.9e6	1.1e5	.	.	.	.	.
1e−3	0	*97e−2*	*62e−2*	*15e−1*	1.6e5	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{21} in 5-D**, N=15, mFE=250000					**f_{21} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	1.0e2	2.0e1	1.8e2	1.0e2	15	6.7e4	2.3e3	1.1e4	6.7e4
1	12	9.7e4	2.1e3	3.1e5	3.5e4	10	6.7e5	5.8e3	2.0e6	1.7e5
1e−1	10	1.8e5	2.0e4	4.4e5	5.0e4	8	1.2e6	1.2e4	3.2e6	3.3e5
1e−3	9	2.2e5	1.7e4	5.8e5	5.4e4	8	1.3e6	4.9e4	2.9e6	3.8e5
1e−5	9	2.3e5	2.3e4	5.5e5	6.7e4	7	1.5e6	1.2e5	3.7e6	3.6e5
1e−8	9	2.5e5	6.0e4	5.7e5	8.6e4	7	1.6e6	2.0e5	3.8e6	4.1e5

Δf	**f_{22} in 5-D**, N=15, mFE=250000					**f_{22} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	4.4e2	3.8e1	6.5e2	4.4e2	11	5.5e5	2.9e3	1.3e6	1.9e5
1	14	6.0e4	6.4e2	1.9e5	4.2e4	2	6.7e6	7.0e5	1.5e7	2.0e5
1e−1	8	3.4e5	5.9e4	8.9e5	1.2e5	0	*73e−1*	*69e−2*	*11e+0*	3.2e5
1e−3	2	1.7e6	2.0e4	4.3e6	8.5e4	.	.	.	.	.
1e−5	2	1.8e6	1.7e5	3.7e6	1.5e5	.	.	.	.	.
1e−8	0	*90e−3*	*79e−7*	*73e−2*	1.5e5	.	.	.	.	.

Δf	**f_{23} in 5-D**, N=15, mFE=250000					**f_{23} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	6.5e0	1.0e0	1.3e1	6.5e0	15	8.7e0	3.0e0	2.4e1	8.7e0
1	15	1.9e4	5.6e3	4.6e4	1.9e4	15	9.2e5	9.6e5	9.9e5	9.2e5
1e−1	2	1.7e6	7.5e4	3.9e6	7.3e4	2	7.1e6	9.9e5	1.6e7	6.2e5
1e−3	0	*53e−2*	*82e−3*	*66e−2*	7.5e4	0	*26e−2*	*94e−3*	*46e−2*	1.0e6
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	**f_{24} in 5-D**, N=15, mFE=250000					**f_{24} in 20-D**, N=15, mFE=1.00e6				
	#	ERT	10%	90%	RT_{succ}	#	ERT	10%	90%	RT_{succ}
10	15	5.7e3	2.2e3	9.6e3	5.7e3	6	2.3e6	6.8e5	5.4e6	7.6e6
1	2	1.8e6	1.6e5	4.0e6	1.4e5	0	*25e+0*	*38e−1*	*60e+0*	1.0e6
1e−1	0	*55e−1*	*82e−2*	*68e−1*	6.9e4	.	.	.	.	.
1e−3	.	.	.	.	.	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Table 1: Shown are, for a given target difference to the optimal function value Δf: the number of successful trials (#); the expected running time to surpass $f_{\mathrm{opt}}+\Delta f$ (ERT, see Figure 1); the 10%-tile and 90%-tile of the bootstrap distribution of ERT; the average number of function evaluations in successful trials or, if none was successful, as last entry the median number of function evaluations to reach the best function value (RT_{succ}). If $f_{\mathrm{opt}}+\Delta f$ was never reached, figures in *italics* denote the best achieved Δf-value of the median trial and the 10% and 90%-tile trial. Furthermore, N denotes the number of trials, and mFE denotes the maximum of number of function evaluations executed in one trial. See Figure 1 for the names of functions.

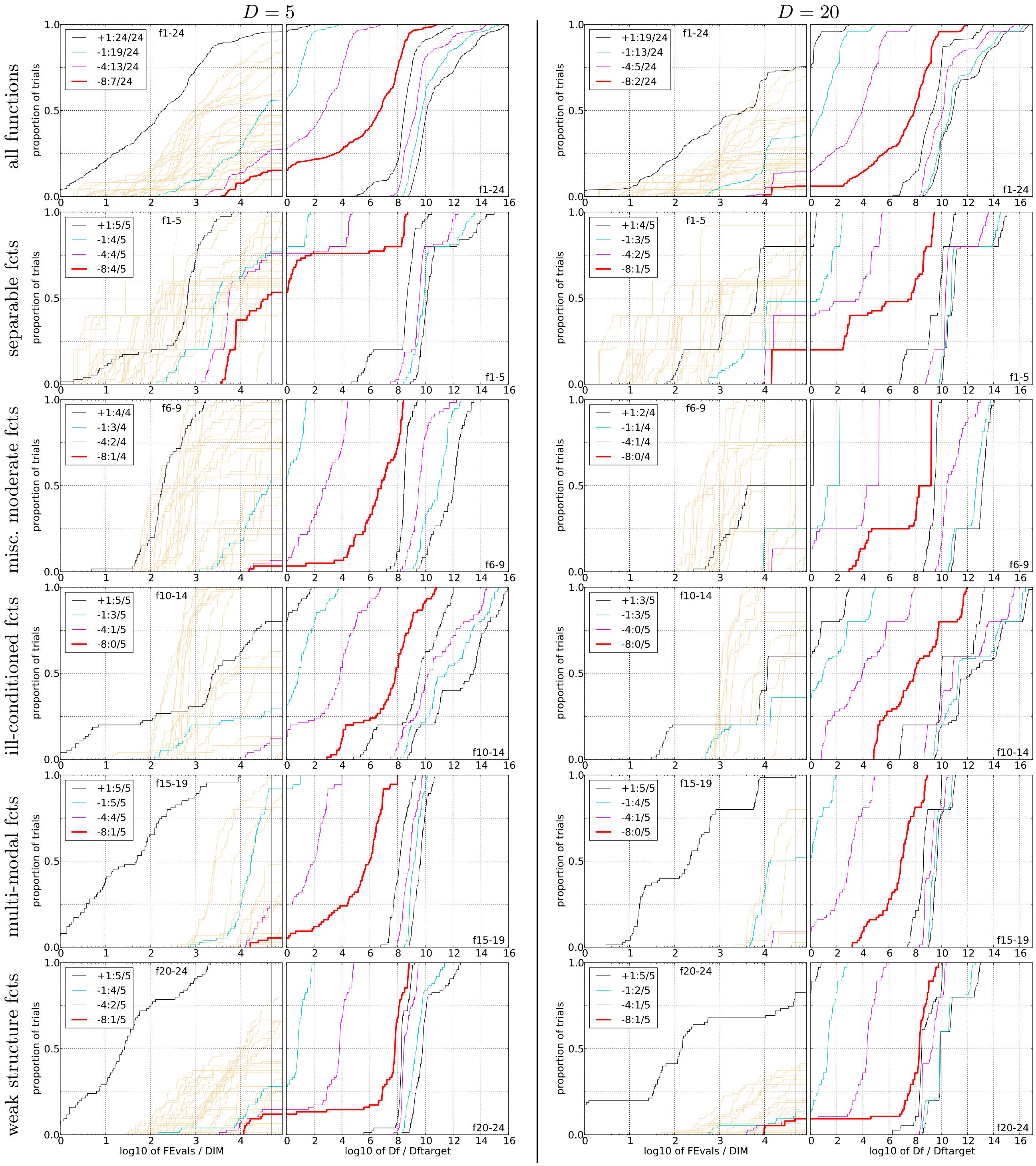

Figure 2: Empirical cumulative distribution functions (ECDFs), plotting the fraction of trials versus running time (left subplots) or versus Δf (right subplots). The thick red line represents the best achieved results. Left subplots: ECDF of the running time (number of function evaluations), divided by search space dimension D, to fall below $f_{\text{opt}} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. Right subplots: ECDF of the best achieved Δf divided by 10^k (upper left lines in continuation of the left subplot), and best achieved Δf divided by 10^{-8} for running times of $D, 10\,D, 100\,D \ldots$ function evaluations (from right to left cycling black-cyan-magenta). The legends indicate the number of functions that were solved in at least one trial. FEvals denotes number of function evaluations, D and DIM denote search space dimension, and Δf and Df denote the difference to the optimal function value. Light brown lines in the background show ECDFs for target value 10^{-8} of all algorithms benchmarked during BBOB-2009.

8. REFERENCES

[1] K. Austin. *Evolutionary Design of Robust Flight Control for a Hypersonic Aircraft*. PhD Dissertation, The Univeristy of Queensland, Department of Mechanical Engineering, 2002.

[2] 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.

[3] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2010: Experimental setup. Technical Report RR-7215, INRIA, 2010.

[4] 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.

[5] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[6] Z. Janikow and Z. Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA 1991)*, pages 31–36, 1991.

[7] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[8] A. Neubauer. A theoretical analysis of the non-uniform mutation operator for the modified genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 93–96, 1997.

[9] E.-G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley, 2009.