The Age-Layered Population Structure (ALPS) Evolutionary Algorithm

Gregory S. Hornby
University Affiliated Research Center, U.C. Santa Cruz
NASA Ames Research Center, Mail Stop 269-3
Moffett Field, CA USA
Gregory.S.Hornby@nasa.gov

ABSTRACT

To reduce the problem of premature convergence we define a new method for measuring an individual's age and propose the Age-Layered Population Structure (ALPS). This measure of age measures how long the genetic material has been evolving in the population: offspring start with an age of 1 plus the age of their oldest parent instead of starting with an age of 0 as with traditional measures of age. ALPS differs from a typical Evolutionary Algorithm (EA) by segregating individuals into different age-layers by their age and by regularly introducing new, randomly generated individuals in the youngest layer. The introduction of randomly generated individuals at regular intervals results in an EA that is never completely converged and is always exploring new parts of the fitness landscape and by using age to restrict competition and breeding, younger individuals are able to develop without being dominated by older ones. In effect, ALPS is a novel way to run multiple EAs simultaneously.

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

Keywords

Benchmarking, Black-box optimization, Evolutionary computation

1. INTRODUCTION

A common problem experienced in running an evolutionary algorithm (EA) is that after some number of evaluations the population converges to a local optima and no further improvements are made no matter how much longer the EA is run. What has happened is called *premature convergence*, which is that the existing genetic material in the population has converged such that the variation operators cannot produce new individuals which will move the population into a better part of the fitness landscape [2, 5, 8, 17]. Many attempts at creating a more robust EA have been tried and this is still an ongoing area of research in the Evolutionary Computation community.

In the Genetic Programming (GP) community, the Age-Layered Population Structure (ALPS) Evolutionary Algorithm (EA) was introduced as a way of addressing, and significantly reducing, premature convergence [11]. ALPS differs from other EAs in that it segregates the population into multiple layers using a novel measure of age, and it reduces premature convergence by introducing a new group of randomly generated individuals into the bottom layer at regular intervals. This approach can be thought of as combining multiple, independent search runs that are done sequentially into a single, multi-layered meta-run.

One advantage of ALPS is that it does not use any details of the representation and, thus, can be used with any sort of encoding scheme: GP programs, bit-strings, real-valued vectors of parameters, sequences of integers (such as for solving a scheduling or TSP problem), and anything else. Having been demonstrated to be effective on GP problems with GP representations [15, 19, 20], the main objective of this paper is to introduce the ALPS paradigm to the GA community by combining ALPS with a basic GA and running it on the problems in the Black-Box Optimization Benchmarking (BBOB) 2009 workshop at GECCO-09.

2. ALGORITHM PRESENTATION

In the field of optimization algorithms, one type of algorithmic improvement is that of increasing the *speed* at which problems of solvable difficulty can be solved. This is shown in papers in which the comparison is on which algorithm can find the global optima of a benchmark problem in the fewest number of evaluations. Another type of algorithmic improvement is increasing the *robustness* of the algorithm. That is, either increasing the reliability of finding the global optima or being able to find better results then other algorithms. This interest in improving robustness on hard problems, at the cost of slightly slower speed on easy problems, is becoming a more useful tradeoff with the continual increase in computing power available.

One approach for improving the robustness of a particular search algorithm is to regularly restart it, using a different

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

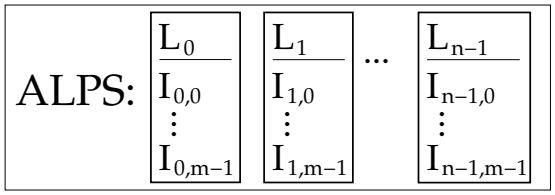

Figure 1: The layout of an ALPS system with n layers (L_0 to L_{n-1}) and m individuals in each layer ($I_{i,0}$ to $I_{i,m-1}$).

random number seed for each restart. Restarting can be done at fixed intervals with a multi-run EA, which divides a total of n generations into m runs of n/m generations. While regularly restarting the EA can improve search performance [3, 18], the challenge becomes developing a good method for deciding when to restart. An alternative to restarting the entire EA is to run multiple EAs simultaneously and only restart one of them, and this is what is done with ALPS.

ALPS was developed to be a more robust EA – especially on hard problems – although not necessarily the fastest one on easy problems [11]. This ability is most easily noticed when using a small population or when performing extremely long evolutionary runs. With ALPS, several instances of the search algorithm are run in parallel, each in its own age-layer (see Figure 1) and the age of solutions is kept track of using a novel age measure. The key properties of ALPS are:

- Multiple instances of a search algorithm are run in parallel, with each instance in its own age layer and having its own population of one or more candidate solutions (*individuals*).
- Each age-layer has a maximum age and it may not contain individuals older than that maximum age.
- The age of individuals is based on when the original genetic material was created from random.
- The search algorithm in a given age-layer can look at individuals in its own population and at the populations in younger age layers but it can only replace individuals in its own population.
- At regular intervals, the search algorithm in the first age-layer is restarted.

We first review the measure of age and then how ALPS manages individuals in its age layers.

2.1 Measuring Age

Over the years *age* has been used in various EAs to try to improve performance [12, 13, 14, 16]. In these systems all individuals, whether they are created randomly or through mutation or recombination, start with an age value of 1. After each generation in which an individual is kept in the population (eg. it is not changed through mutation or recombination) its *age* is increased by one. Thus in these systems *age* is a measure of how long a particular individual with the same set of alleles has been in the population.

With ALPS, *age* is a measure of how long an individual's family of genotypic material has been in the population. Randomly generated individuals, such as those that are created in the initial generation of a canonical EA run, start with an age of 1. Each generation that an individual stays in the population (such as through elitism) its age is increased by one. Individuals that are created through mutation or recombination take the age of their oldest parent and add one to it. For example, if individual $\texttt{Ind}_A$, age 23, and individual $\texttt{Ind}_B$, age 28, are selected as parents for recombination then their offspring, $\texttt{Ind}_C$, will be assigned an age of 29. At the end of the reproduction phase $\texttt{Ind}_A$ will have its age increased to 24 and $\texttt{Ind}_B$ will have its age increased to 29 and $\texttt{Ind}_C$ will keep its age of 29. In contrast, other age-based EA systems would assign an age of 1 to $\texttt{Ind}_C$.

Rather than counting generations, age is implemented by taking the number of evaluations in which an individual's genetic material has been around and dividing it by the size of the population. Randomly generated individuals store the number of evaluations that have been performed so far, and individuals created through mutation and recombination store the smallest (which is equivalent to "oldest") value of their parents. The equation for calculating the age of an individual is:

$$age = 1 + (evals_{current} - evals_{created})/popsize \quad (1)$$

Where: $evals_{current}$ is the number of evaluations that have been performed so far; $evals_{created}$ is the number of evaluations that had been performed when the individual's genetic material was first created; and *popsize* is the total number of individuals in all layers. A constant of 1 is added so that the age of randomly generated individuals is 1 at creation time.

2.2 ALPS-EA

With the ALPS paradigm, the population is segregated into multiple age layers, with each layer having an upper age limit. The EA acts on each age layer somewhat independently of the others, with an exception being that parents can be selected from both the current layer and the layers below. When an individual is too old for its current layer, L_i, the algorithm can move it up to the next layer, L_{i+1}. Also, at regular intervals the bottom layer is replaced with a new sub-population of randomly generated individuals, each with an age of 1.

Table 1: Different systems for setting the age-limits for each age-layer and the corresponding maximum age in each layer for an `age-gap` of 1.

	Max age in layer						
Aging-scheme	0	1	2	3	4	5	6
Linear	1	2	3	4	5	6	7
Fibonacci	1	2	3	5	8	13	21
Polynomial (n^2)	1	2	4	9	16	25	49
Exponential (2^n)	1	2	4	8	16	32	64

In setting up an ALPS run, the number of age layers and the age limits for each layer are parameters that are set by the user. Different systems can be used for setting these values, such as by using linearly, polynomially or exponentially increasing limits (Table 1). To keep the size of the population and number of layers manageable, and since there

is generally little need to segregate individuals which are within a few "generations" of each other, these values are then multiplied by an `age-gap` parameter. Also, there is no maximum age for the last layer and a single-layer version of ALPS operates exactly as the standard EA. For example, in a system with five layers, a polynomial aging-scheme and an age gap of seven the maximum ages for the five layers are: 7, 14, 28, 56, and ∞.

With an ALPS-EA, evolution occurs in each layer somewhat independently of the others. When selecting parents to create new individuals for a given layer, parents are selected from individuals in that layer as well as the previous one. Restricting the selection of parents in this way limits selection competition to those individuals of similar ages and prevents the older individuals from dominating the younger ones. Once individuals have been in the "bottom" layer of the population for as many generations as its age limit, all individuals in this layer are replaced with randomly created individuals. For example, with the example in the previous paragraph, the bottom layer is replaced with a new group of randomly created individuals every 7 generations.

2.3 Pseudo-Code

An ALPS-EA works as follows. The algorithm starts by configuring the age layers and then creating, and evaluating, an initial, random population. Once the initial population is created and evaluated, `ALPS-EA` enters its main loop which consists of cycling through the layers, from L_{n-1} to L_0, and then evolving the EA in that layer for one generation:

```
1: procedure ALPSEA( )
2:     gen ← 0
3:     while not done do
4:         for i = n − 1 to 0 do
5:             if (i == 0) && (gen % age_gap == 0) then
6:                 Restart L_0 with a new random population.
7:             end if
8:             Evolve(L_i).                    ▷ For one generation.
9:         end for
10:    end while
11: end procedure
```

ALPS allows for different methods for selecting the parents (eg tournament selection or some form of roulette wheel selection) and different representations and variation operators. One way for genetic material to move up from one layer to the next is by implementing the selection method so to pick individuals from layers L_{i-1} and L_i. In addition, elitism can be added either to just the top layer or to all layers.

2.4 Additional Comments

The above algorithm allows for different methods of selecting the parents (eg tournament selection or some form of roulette wheel selection). In addition, elitism can be added either to just the top layer or to all layers. More generally, ALPS can be thought of as an approach for solving the restart problem when trying to perform multiple, sequential optimization runs. In thinking of ALPS as a method for combining multiple, independent optimization runs into a single run, there are two important criteria.

Individuals in the population have an age, and this age value is based on when the genotypic material was randomly created. That is, randomly created individuals start with an age of 1 and individuals created through mutation and offspring inherit the age of the parent. Age can be measured in generations (Generational ALPS) or in number of evaluations divided by population size (Steady-State ALPS) since the two are equivalent. By increasing an individual's age each generation in which it is used as a parent, it will eventually age its way up the layers and, likely, out of the population. An individual is only guaranteed to stay in the population forever it if is at the global optima, otherwise it will eventually be replaced as better individuals are evolved. Likely any measure of age that is relatively similar to the one proposed will work just as well.

The second important criteria is that selection and replacement are restricted to individuals of similar ages. The version of ALPS described above has explicit age-layers, and in this case the selection of parents is restricted to parents who are young enough that their offspring will not be too old for that layer. In this implementation parents are only selected from the current and previous layer, but it would also be acceptable to select from all previous layers. Explicit age layers may not be necessary as long as the ALPS implementation restricts breeding and competition between similarly aged individuals.

An advantage of using explicit age layers is that any optimization algorithm could be used in each layer. In the experiments presented in this paper a fairly standard EA is used, but it would be perfectly acceptable to combine ALPS with CMA-ES [1], Differential Evolution [6], Particle Swarm Evolution [4], Hill-Climbing, Simulated Annealing, or anything else.

3. EXPERIMENTAL PROCEDURE

The objective of these experiments is to compare the performance of an ALPS GA against other mainstream algorithms in the GA community on the BBOB functions. For this comparison we combine ALPS with a standard GA and configure it as follows: [1]

- 12 age layers are used with a variation of the exponential aging scheme and an age-gap of 3. The maximum ages for each age layer are: 3, 5, 9, 17, 33, 65, 129, 257, 513, 1025, 2049 and ∞.
- The GA in each age layer has 30 individuals.
- An elitism of 4 is used by the GA in each age layer and the other individuals are created by using one of two recombination operators (chosen with equal likelihood).
- To create a new individual tournament selection is used with a tournament size of 4. Individuals are chosen from the population in current layer (L_i) or, with 20% likelihood, from the GA's population in the previous layer (L_{i-1}).
- The first recombination operator uses the winner of the tournament as a "center" point and uses another member of the tournament (chosen at random) as the second parent. The values for each gene are selected at random using: $C_i = P_{1,i} + k * (P_{1,i} - P_{2,i})$, where k is a random number with a normal distribution.

[1]Source code for ALPS is available online at http://idesign.ucsc.edu

- The second operator is the similar to first except instead of generating a new random number k for each gene, the same value of k is used for all genes. This results in the offspring, C, falling somewhere on the line through P_1 and P_2.

Because ALPS is running several GAs simultaneously, and the GA in the first layer is being regularly restarted – in this case, every 3 generations – each trial consists of a single ALPS run for 5 million evaluations.

4. RESULTS

Results from experiments according to [9] on the benchmark functions given in [7, 10] are presented in Figures 2 and 3 and in Table 2.

5. REFERENCES

[1] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 1769–1776. IEEE Press, 2005.

[2] J. E. Baker. Adaptive selection methods for genetic algorithms. In J. J. Grefenstette, editor, *Proc. of the First Intl. Conf. on Genetic Algorithms*, pages 101–111, 1985.

[3] E. CantÃž-Paz and D. E. Goldberg. Are multiple runs of genetic algorithms better than one? In E. C.-P. et al., editor, *Proc. of the Genetic and Evolutionary Computation Conference*, LNCS 2724, pages 801–812, Berlin, 2003. Springer-Verlag.

[4] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6:58–73, 2002.

[5] K. A. DeJong. *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Dept. Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975.

[6] V. Feoktistov. *Differential Evolution: In Search of Solutions (Springer Optimization and Its Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[7] 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.

[8] D. E. Goldberg and P. Segrest. Finite markov chain analysis of genetic algorithms. In J. J. Grefenstette, editor, *Proc. of the Second Intl. Conf. on Genetic Algorithms*, pages 1–8. Lawrence Erlbaum Associates, 1987.

[9] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA, 2009.

[10] 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.

[11] G. S. Hornby. ALPS: The age-layered population structure for reducing the problem of premature convergence. In M. K. et al., editor, *Proc. of the Genetic and Evolutionary Computation Conference, GECCO-2006*, pages 815–822, Seattle, WA, 2006. ACM Press.

[12] G. S. Hornby, M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata. Autonomous evolution of gaits with the sony quadruped robot. In Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiel, and Smith, editors, *Proc. of the Genetic and Evolutionary Computation Conference*, pages 1297–1304. Morgan Kaufmann, 1999.

[13] A. Huber and D. A. Mlynski. An age-controlled evolutionary algorithm for optimization problems in physical layout. In *International Symposium on Circuits and Systems*, pages 262–265. IEEE Press, 1998.

[14] J.-H. Kim, J.-Y. Jeon, H.-K. Chae, and K. Koh. A novel evolutionary algorithm with fast convergence. In *IEEE International Conference on Evolutionary Computation*, pages 228–29. IEEE Press, 1995.

[15] M. F. Korns and L. Nunez. Profiling symbolic regression-classification. In R. L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice VI*, Genetic and Evolutionary Computation, chapter 14, pages 215–229. Springer, Ann Arbor, 15-17May 2008.

[16] N. Kubota, T. Fukuda, F. Arai, and K. Shimojima. Genetic algorithm with age structure and its application to self-organizing manufacturing system. In *IEEE Symposium on Emerging Technologies and Factory Automation*, pages 472–477. IEEE Press, 1994.

[17] S. J. Louis and G. J. E. Rawlins. Syntactic analysis of convergence in genetic algorithms. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 141–151. Morgan Kaufmann, 1993.

[18] S. Luke. When short runs beat long runs. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 74–80, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[19] T. McConaghy, P. Palmers, G. Gielen, and M. Steyaert. Genetic programming with reuse of known designs. In R. L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 10, pages 161–186. Springer, Ann Arbor, 17-19May 2007.

[20] A. Willis, S. Patel, and C. D. Clack. GP age-layer and crossover effects in bid-offer spread prediction. In *Proceedings of the 10th annual conference on Genetic and Evolutionary Computation Conference*, Atlanta, GA, July 12-16 2008.

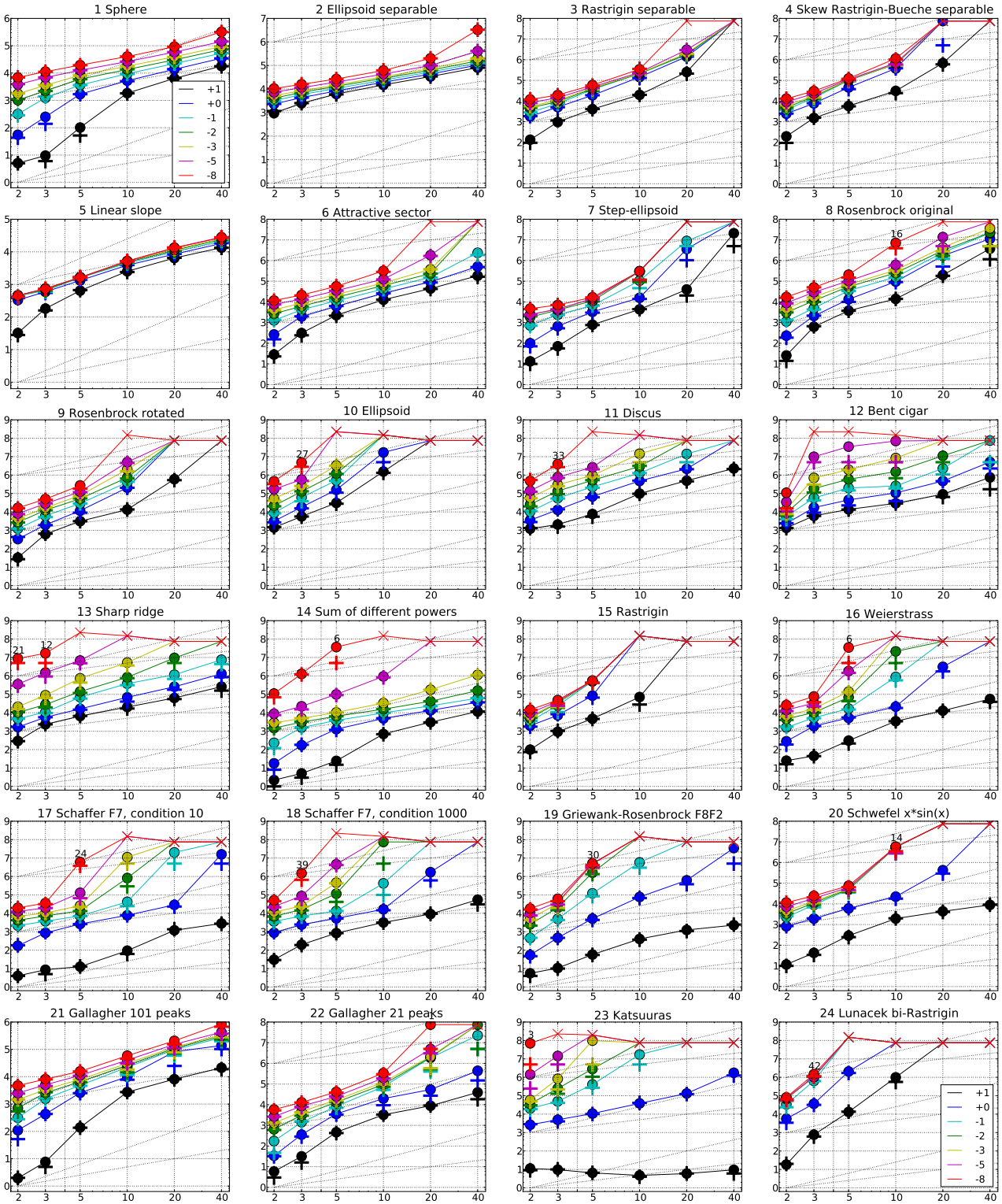

Figure 2: Expected Running Time (ERT, ●) **to reach** $f_{\mathrm{opt}} + \Delta f$ **and median number of function evaluations of successful trials** (+), **shown for** $\Delta f = 10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}$ **(the exponent is given in the legend of** f_1 **and** f_{24}**) versus dimension in log-log presentation. The** $\mathrm{ERT}(\Delta f)$ **equals to** $\#\mathrm{FEs}(\Delta f)$ **divided by the number of successful trials, where a trial is successful if** $f_{\mathrm{opt}} + \Delta f$ **was surpassed during the trial. The** $\#\mathrm{FEs}(\Delta f)$ **are the total number of function evaluations while** $f_{\mathrm{opt}} + \Delta f$ **was not surpassed during the trial from all respective trials (successful and unsuccessful), and** f_{opt} **denotes the optimal function value. Crosses** (×) **indicate the total number of function evaluations** $\#\mathrm{FEs}(-\infty)$**. Numbers above ERT-symbols indicate the number of successful trials. Annotated numbers on the ordinate are decimal logarithms. Additional grid lines show linear and quadratic scaling.**

Δf	#	ERT	10%	90%	RT_{succ}
f_1 in 5-D, N=45, mFE=20996					
10	45	1.0e2	8.2e1	1.2e2	1.0e2
1	45	1.7e3	1.6e3	1.8e3	1.7e3
1e−1	45	3.6e3	3.6e3	3.7e3	3.6e3
1e−3	45	8.3e3	8.2e3	8.4e3	8.3e3
1e−5	45	1.3e4	1.2e4	1.3e4	1.3e4
1e−8	45	2.0e4	1.9e4	2.0e4	2.0e4
f_1 in 20-D, N=15, mFE=100032					
10	15	6.4e3	6.1e3	6.6e3	6.4e3
1	15	1.4e4	1.3e4	1.4e4	1.4e4
1e−1	15	2.2e4	2.2e4	2.3e4	2.2e4
1e−3	15	3.9e4	3.8e4	4.0e4	3.9e4
1e−5	15	5.8e4	5.7e4	5.9e4	5.8e4
1e−8	15	9.2e4	9.1e4	9.3e4	9.2e4
f_2 in 5-D, N=45, mFE=29358					
10	45	6.1e3	6.0e3	6.2e3	6.1e3
1	45	8.3e3	8.1e3	8.4e3	8.3e3
1e−1	45	1.1e4	1.0e4	1.1e4	1.1e4
1e−3	45	1.5e4	1.5e4	1.5e4	1.5e4
1e−5	45	2.0e4	1.9e4	2.0e4	2.0e4
1e−8	45	2.7e4	2.6e4	2.7e4	2.7e4
f_2 in 20-D, N=15, mFE=225513					
10	15	3.4e4	3.3e4	3.4e4	3.4e4
1	15	4.3e4	4.3e4	4.4e4	4.3e4
1e−1	15	5.3e4	5.2e4	5.4e4	5.3e4
1e−3	15	7.4e4	7.3e4	7.5e4	7.4e4
1e−5	15	1.0e5	9.8e4	1.0e5	1.0e5
1e−8	15	2.0e5	2.0e5	2.1e5	2.0e5
f_3 in 5-D, N=45, mFE=120523					
10	45	4.1e3	3.8e3	4.3e3	4.1e3
1	45	1.9e4	1.8e4	2.0e4	1.9e4
1e−1	45	3.7e4	3.2e4	4.1e4	3.7e4
1e−3	45	4.2e4	3.8e4	4.7e4	4.2e4
1e−5	45	4.8e4	4.4e4	5.3e4	4.8e4
1e−8	45	5.9e4	5.5e4	6.4e4	5.9e4
f_3 in 20-D, N=15, mFE=5.00e6					
10	15	2.5e5	2.1e5	3.0e5	2.5e5
1	15	1.4e6	1.3e6	1.5e6	1.4e6
1e−1	15	1.8e6	1.5e6	2.0e6	1.8e6
1e−3	15	2.1e6	1.8e6	2.3e6	2.1e6
1e−5	15	3.0e6	2.8e6	3.3e6	3.0e6
1e−8	0	*39e−8*	*81e−9*	*31e−7*	4.5e6
f_4 in 5-D, N=45, mFE=222315					
10	45	5.8e3	5.6e3	6.0e3	5.8e3
1	45	4.6e4	4.0e4	5.1e4	4.6e4
1e−1	45	1.0e5	9.0e4	1.1e5	1.0e5
1e−3	45	1.0e5	9.4e4	1.2e5	1.0e5
1e−5	45	1.1e5	1.0e5	1.2e5	1.1e5
1e−8	45	1.2e5	1.1e5	1.4e5	1.2e5
f_4 in 20-D, N=15, mFE=5.00e6					
10	15	6.9e5	6.0e5	7.8e5	6.9e5
1	1	7.5e7	3.7e7	>7e7	5.0e6
1e−1	0	*30e−1*	*20e−1*	*40e−1*	4.5e6
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_5 in 5-D, N=45, mFE=2786					
10	45	6.4e2	6.0e2	6.9e2	6.4e2
1	45	1.3e3	1.3e3	1.4e3	1.3e3
1e−1	45	1.6e3	1.5e3	1.7e3	1.6e3
1e−3	45	1.6e3	1.5e3	1.7e3	1.6e3
1e−5	45	1.6e3	1.5e3	1.7e3	1.6e3
1e−8	45	1.6e3	1.5e3	1.7e3	1.6e3
f_5 in 20-D, N=15, mFE=18392					
10	15	6.4e3	6.1e3	6.8e3	6.4e3
1	15	8.5e3	8.2e3	8.8e3	8.5e3
1e−1	15	1.0e4	9.6e3	1.0e4	1.0e4
1e−3	15	1.2e4	1.1e4	1.3e4	1.2e4
1e−5	15	1.3e4	1.3e4	1.4e4	1.3e4
1e−8	15	1.3e4	1.3e4	1.4e4	1.3e4
f_6 in 5-D, N=45, mFE=70473					
10	45	2.3e3	2.1e3	2.4e3	2.3e3
1	45	6.1e3	5.8e3	6.3e3	6.1e3
1e−1	45	1.1e4	1.1e4	1.2e4	1.1e4
1e−3	45	2.3e4	2.2e4	2.3e4	2.3e4
1e−5	45	3.5e4	3.4e4	3.6e4	3.5e4
1e−8	45	5.7e4	5.5e4	5.8e4	5.7e4
f_6 in 20-D, N=15, mFE=5.00e6					
10	15	4.4e4	4.1e4	4.7e4	4.4e4
1	15	9.4e4	8.7e4	1.0e5	9.4e4
1e−1	15	1.6e5	1.5e5	1.7e5	1.6e5
1e−3	15	3.8e5	3.4e5	4.1e5	3.8e5
1e−5	15	1.8e6	1.6e6	2.1e6	1.8e6
1e−8	0	*14e−7*	*46e−8*	*56e−7*	4.5e6
f_7 in 5-D, N=45, mFE=23854					
10	45	7.9e2	7.3e2	8.5e2	7.9e2
1	45	3.3e3	3.1e3	3.5e3	3.3e3
1e−1	45	6.7e3	6.4e3	6.9e3	6.7e3
1e−3	45	1.4e4	1.3e4	1.5e4	1.4e4
1e−5	45	1.4e4	1.3e4	1.5e4	1.4e4
1e−8	45	1.7e4	1.6e4	1.8e4	1.7e4
f_7 in 20-D, N=15, mFE=5.00e6					
10	15	4.0e4	2.2e4	6.0e4	4.0e4
1	10	3.6e6	2.5e6	5.3e6	2.4e6
1e−1	6	8.9e6	6.0e6	1.5e7	3.6e6
1e−3	0	*48e−2*	*31e−3*	*19e−1*	4.5e6
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_8 in 5-D, N=45, mFE=570079					
10	45	3.8e3	3.6e3	3.9e3	3.8e3
1	45	1.3e4	1.2e4	1.5e4	1.3e4
1e−1	45	3.5e4	3.2e4	3.8e4	3.5e4
1e−3	45	6.9e4	6.6e4	7.3e4	6.9e4
1e−5	45	1.0e5	9.8e4	1.1e5	1.0e5
1e−8	45	2.1e5	1.8e5	2.3e5	2.1e5
f_8 in 20-D, N=15, mFE=5.00e6					
10	15	2.0e5	1.6e5	2.4e5	2.0e5
1	12	1.7e6	9.3e5	2.7e6	1.2e6
1e−1	12	2.4e6	1.7e6	3.4e6	1.8e6
1e−3	11	4.3e6	3.3e6	5.8e6	3.0e6
1e−5	5	1.4e7	9.2e6	2.4e7	4.4e6
1e−8	0	*26e−6*	*33e−7*	*23e−1*	4.5e6
f_9 in 5-D, N=45, mFE=1.02e6					
10	45	3.1e3	3.0e3	3.2e3	3.1e3
1	45	1.2e4	1.1e4	1.3e4	1.2e4
1e−1	45	2.9e4	2.6e4	3.1e4	2.9e4
1e−3	45	7.8e4	7.3e4	8.3e4	7.8e4
1e−5	45	1.3e5	1.2e5	1.3e5	1.3e5
1e−8	45	2.7e5	2.3e5	3.2e5	2.7e5
f_9 in 20-D, N=15, mFE=5.00e6					
10	15	6.0e5	4.9e5	7.1e5	6.0e5
1	0	*48e−1*	*25e−1*	*60e−1*	4.5e6
1e−1	.	.	.	.	.
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{10} in 5-D, N=45, mFE=5.00e6					
10	45	3.1e4	2.8e4	3.4e4	3.1e4
1	45	1.6e5	1.4e5	1.8e5	1.6e5
1e−1	45	5.1e5	4.6e5	5.6e5	5.1e5
1e−3	39	3.3e6	2.9e6	3.7e6	2.9e6
1e−5	0	*21e−5*	*29e−6*	*11e−4*	4.5e6
1e−8	.	.	.	.	.
f_{10} in 20-D, N=15, mFE=5.00e6					
10	0	*16e+1*	*36e+0*	*29e+1*	4.5e6
1	.	.	.	.	.
1e−1	.	.	.	.	.
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{11} in 5-D, N=45, mFE=5.00e6					
10	45	7.7e3	6.6e3	8.7e3	7.7e3
1	45	6.9e4	6.2e4	7.5e4	6.9e4
1e−1	45	2.3e5	2.1e5	2.5e5	2.3e5
1e−3	45	9.5e5	8.8e5	1.0e6	9.5e5
1e−5	42	2.7e6	2.4e6	2.9e6	2.5e6
1e−8	0	*72e−8*	*93e−9*	*79e−7*	4.5e6
f_{11} in 20-D, N=15, mFE=5.00e6					
10	15	5.0e5	4.5e5	5.4e5	5.0e5
1	15	2.2e6	2.0e6	2.4e6	2.2e6
1e−1	5	1.4e7	9.5e6	2.4e7	4.6e6
1e−3	0	*12e−2*	*47e−3*	*30e−2*	4.5e6
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{12} in 5-D, N=45, mFE=5.00e6					
10	45	1.4e4	1.3e4	1.4e4	1.4e4
1	45	4.5e4	3.8e4	5.3e4	4.5e4
1e−1	45	2.0e5	1.7e5	2.3e5	2.0e5
1e−3	42	1.9e6	1.7e6	2.2e6	1.8e6
1e−5	6	3.5e7	2.3e7	6.9e7	5.0e6
1e−8	0	*24e−5*	*55e−7*	*90e−5*	4.0e6
f_{12} in 20-D, N=15, mFE=5.00e6					
10	15	9.1e4	6.5e4	1.2e5	9.1e4
1	15	4.7e5	3.6e5	5.9e5	4.7e5
1e−1	13	2.3e6	1.6e6	3.2e6	1.9e6
1e−3	0	*33e−3*	*11e−4*	*29e−2*	4.5e6
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{13} in 5-D, N=45, mFE=5.00e6					
10	45	6.6e3	6.4e3	6.8e3	6.6e3
1	45	1.6e4	1.5e4	1.7e4	1.6e4
1e−1	45	7.0e4	6.2e4	7.8e4	7.0e4
1e−3	45	7.4e5	5.5e5	9.5e5	7.4e5
1e−5	24	6.8e6	5.7e6	8.4e6	3.7e6
1e−8	0	*76e−7*	*38e−8*	*16e−5*	4.5e6
f_{13} in 20-D, N=15, mFE=5.00e6					
10	15	6.3e4	5.5e4	7.3e4	6.3e4
1	15	2.6e5	2.0e5	3.2e5	2.6e5
1e−1	15	1.2e6	8.3e5	1.5e6	1.2e6
1e−3	0	*14e−3*	*31e−4*	*47e−3*	4.5e6
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{14} in 5-D, N=45, mFE=5.00e6					
10	45	2.4e1	1.9e1	2.9e1	2.4e1
1	45	1.3e3	1.2e3	1.4e3	1.3e3
1e−1	45	3.8e3	3.7e3	3.9e3	3.8e3
1e−3	45	1.0e4	1.0e4	1.1e4	1.0e4
1e−5	45	9.9e4	9.1e4	1.1e5	9.9e4
1e−8	6	3.7e7	2.5e7	7.3e7	5.0e6
f_{14} in 20-D, N=15, mFE=5.00e6					
10	15	3.1e3	2.8e3	3.3e3	3.1e3
1	15	1.3e4	1.2e4	1.4e4	1.3e4
1e−1	15	2.4e4	2.3e4	2.5e4	2.4e4
1e−3	15	1.8e5	1.7e5	1.9e5	1.8e5
1e−5	0	*34e−6*	*19e−6*	*57e−6*	4.5e6
1e−8	.	.	.	.	.
f_{15} in 5-D, N=45, mFE=1.40e6					
10	45	4.7e3	4.5e3	4.9e3	4.7e3
1	45	8.4e4	7.2e4	9.6e4	8.4e4
1e−1	45	5.1e5	4.4e5	5.8e5	5.1e5
1e−3	45	5.1e5	4.4e5	5.9e5	5.1e5
1e−5	45	5.2e5	4.5e5	6.0e5	5.2e5
1e−8	45	5.5e5	4.8e5	6.3e5	5.5e5
f_{15} in 20-D, N=15, mFE=5.00e6					
10	0	*20e+0*	*16e+0*	*24e+0*	4.5e6
1	.	.	.	.	.
1e−1	.	.	.	.	.
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{16} in 5-D, N=45, mFE=5.00e6					
10	45	3.1e2	2.5e2	3.6e2	3.1e2
1	45	5.2e3	4.9e3	5.6e3	5.2e3
1e−1	45	1.7e4	1.5e4	1.8e4	1.7e4
1e−3	45	1.4e5	1.2e5	1.6e5	1.4e5
1e−5	42	1.8e6	1.5e6	2.2e6	1.7e6
1e−8	6	3.5e7	2.3e7	7.0e7	4.2e6
f_{16} in 20-D, N=15, mFE=5.00e6					
10	15	1.3e4	1.2e4	1.4e4	1.3e4
1	11	3.1e6	2.3e6	4.0e6	2.6e6
1e−1	0	*78e−2*	*48e−2*	*11e−1*	4.0e6
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{17} in 5-D, N=45, mFE=5.00e6					
10	45	1.3e1	1.1e1	1.4e1	1.3e1
1	45	2.6e3	2.5e3	2.8e3	2.6e3
1e−1	45	7.2e3	6.9e3	7.4e3	7.2e3
1e−3	45	2.6e4	2.3e4	2.9e4	2.6e4
1e−5	45	1.3e5	1.1e5	1.5e5	1.3e5
1e−8	24	6.1e6	5.0e6	7.6e6	3.3e6
f_{17} in 20-D, N=15, mFE=5.00e6					
10	15	1.2e3	1.0e3	1.4e3	1.2e3
1	15	2.9e4	2.4e4	3.4e4	2.9e4
1e−1	3	2.0e7	1.2e7	5.7e7	5.0e6
1e−3	0	*16e−2*	*79e−3*	*27e−2*	3.2e6
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{18} in 5-D, N=45, mFE=5.00e6					
10	45	8.4e2	7.8e2	9.1e2	8.4e2
1	45	5.3e3	5.1e3	5.6e3	5.3e3
1e−1	45	1.3e4	1.3e4	1.4e4	1.3e4
1e−3	45	4.6e5	4.0e5	5.2e5	4.6e5
1e−5	30	4.6e6	3.8e6	5.5e6	3.0e6
1e−8	0	*53e−7*	*46e−8*	*15e−5*	4.5e6
f_{18} in 20-D, N=15, mFE=5.00e6					
10	15	9.2e3	8.5e3	9.7e3	9.2e3
1	14	1.7e6	1.1e6	2.4e6	1.6e6
1e−1	0	*69e−2*	*18e−2*	*96e−2*	4.0e6
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{19} in 5-D, N=45, mFE=5.00e6					
10	45	5.8e1	4.8e1	6.7e1	5.8e1
1	45	5.1e3	4.6e3	5.6e3	5.1e3
1e−1	45	1.2e5	1.0e5	1.4e5	1.2e5
1e−3	30	4.7e6	4.0e6	5.7e6	3.2e6
1e−5	30	4.9e6	4.2e6	5.8e6	3.3e6
1e−8	30	5.1e6	4.4e6	6.0e6	3.5e6
f_{19} in 20-D, N=15, mFE=5.00e6					
10	15	1.2e3	1.1e3	1.4e3	1.2e3
1	15	6.2e5	4.1e5	8.4e5	6.2e5
1e−1	0	*31e−2*	*18e−2*	*48e−2*	4.5e6
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{20} in 5-D, N=45, mFE=262762					
10	45	2.9e2	2.6e2	3.2e2	2.9e2
1	45	6.0e3	5.7e3	6.3e3	6.0e3
1e−1	45	4.8e4	3.8e4	5.9e4	4.8e4
1e−3	45	5.5e4	4.5e4	6.6e4	5.5e4
1e−5	45	6.2e4	5.2e4	7.3e4	6.2e4
1e−8	45	7.5e4	6.6e4	8.7e4	7.5e4
f_{20} in 20-D, N=15, mFE=5.00e6					
10	15	4.4e3	4.0e3	4.7e3	4.4e3
1	15	4.3e5	3.2e5	5.4e5	4.3e5
1e−1	0	*47e−2*	*39e−2*	*66e−2*	4.5e6
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{21} in 5-D, N=45, mFE=23469					
10	45	1.4e2	1.2e2	1.6e2	1.4e2
1	45	2.6e3	2.4e3	2.8e3	2.6e3
1e−1	45	4.8e3	4.5e3	5.2e3	4.8e3
1e−3	45	8.0e3	7.6e3	8.4e3	8.0e3
1e−5	45	1.1e4	1.1e4	1.2e4	1.1e4
1e−8	45	1.6e4	1.5e4	1.6e4	1.6e4
f_{21} in 20-D, N=15, mFE=611471					
10	15	8.1e3	7.4e3	8.8e3	8.1e3
1	15	8.2e4	4.2e4	1.2e5	8.2e4
1e−1	15	9.8e4	6.0e4	1.4e5	9.8e4
1e−3	15	1.2e5	8.4e4	1.7e5	1.2e5
1e−5	15	1.5e5	1.1e5	2.0e5	1.5e5
1e−8	15	2.0e5	1.6e5	2.5e5	2.0e5
f_{22} in 5-D, N=45, mFE=69317					
10	45	4.7e2	4.0e2	5.4e2	4.7e2
1	45	3.6e3	3.4e3	3.8e3	3.6e3
1e−1	45	8.3e3	7.6e3	9.0e3	8.3e3
1e−3	45	1.7e4	1.6e4	1.8e4	1.7e4
1e−5	45	2.5e4	2.3e4	2.7e4	2.5e4
1e−8	45	4.2e4	3.9e4	4.5e4	4.2e4
f_{22} in 20-D, N=15, mFE=5.00e6					
10	15	8.9e3	8.1e3	9.6e3	8.9e3
1	15	5.4e4	3.7e4	7.4e4	5.4e4
1e−1	12	1.9e6	1.0e6	2.9e6	1.4e6
1e−3	12	2.1e6	1.3e6	3.2e6	1.6e6
1e−5	10	4.9e6	3.9e6	6.5e6	3.4e6
1e−8	1	7.5e7	3.7e7	>7e7	5.0e6
f_{23} in 5-D, N=40, mFE=5.00e6					
10	40	6.5e0	5.5e0	7.5e0	6.5e0
1	40	1.1e4	9.8e3	1.2e4	1.1e4
1e−1	40	4.1e5	3.3e5	4.8e5	4.1e5
1e−3	2	9.8e7	4.9e7	>2e8	5.0e6
1e−5	0	*54e−4*	*13e−4*	*24e−3*	3.2e6
1e−8	.	.	.	.	.
f_{23} in 20-D, N=15, mFE=5.00e6					
10	15	5.9e0	4.8e0	7.0e0	5.9e0
1	15	1.3e5	1.1e5	1.5e5	1.3e5
1e−1	0	*29e−2*	*17e−2*	*45e−2*	4.5e6
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{24} in 5-D, N=30, mFE=5.00e6					
10	30	1.3e4	1.2e4	1.4e4	1.3e4
1	28	2.1e6	1.7e6	2.4e6	1.9e6
1e−1	0	*75e−2*	*18e−2*	*94e−2*	4.5e6
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{24} in 20-D, N=15, mFE=5.00e6					
10	0	*22e+0*	*16e+0*	*31e+0*	4.5e6
1	.	.	.	.	.
1e−1	.	.	.	.	.
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.

Table 2: Shown are, for a given target difference to the optimal function value Δf: the number of successful trials (#); the expected running time to surpass $f_{\text{opt}} + \Delta f$ (ERT, see Figure 2); 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 $f_{\text{opt}} + \Delta f$ 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 2 for the names of functions.

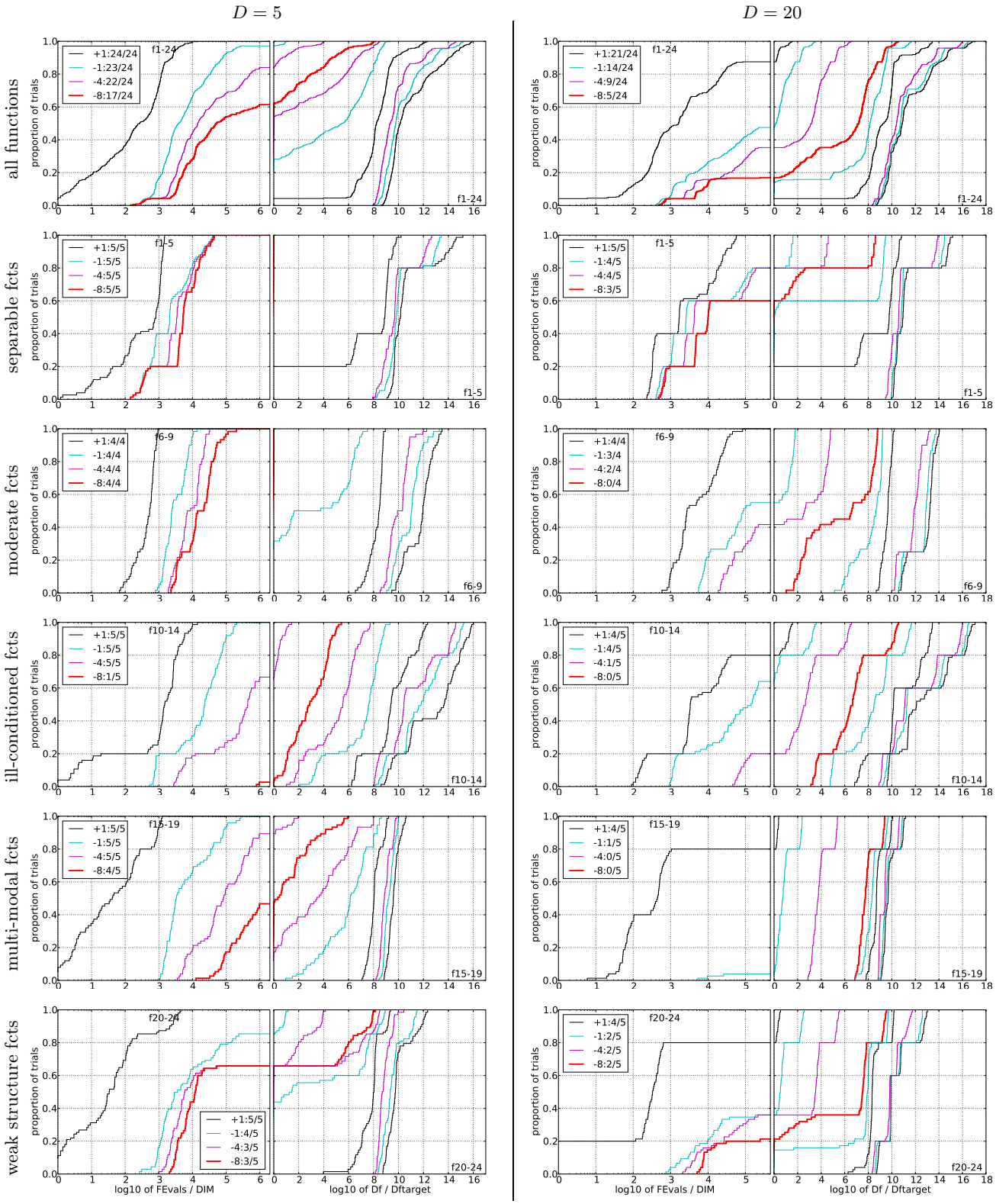

Figure 3: Empirical cumulative distribution functions (ECDFs), plotting the fraction of trials versus running time (left subplots) or versus Δf (right subplots). The thick red line represents the best achieved results. Left subplots: ECDF of the running time (number of function evaluations), divided by search space dimension D, to fall below $f_{\mathrm{opt}} + \Delta f$ 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). Top row: all results from all functions; second row: separable functions; third row: misc. moderate functions; fourth row: ill-conditioned functions; fifth row: multi-modal functions with adequate structure; last row: multi-modal functions with weak structure. 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.