Black-Box Optimization Benchmarking Template for the Comparison of Algorithms on the bbob Test Suite

Draft version

Firstname Lastname

ABSTRACT

to be written

CCS CONCEPTS

• **Computing methodologies → Continuous space search**.

KEYWORDS

Benchmarking, Black-box optimization

ACM Reference Format:

Firstname Lastname. 2023. Black-Box Optimization Benchmarking Template for the Comparison of Algorithms on the bbob Test Suite: Draft version. In *Proceedings of The Genetic and Evolutionary Computation Conference 2023 (GECCO '23).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 ALGORITHM PRESENTATION

The algorithm uses two strategies together to find global minima of functions. One is to focus on regions called trust regions. These evolve depending on the performance of the algorithm. The second is to approximate the objective function through a Gaussian process. In this way, the algorithm consists of an iteration of these two strategies that we explain in more detail below.

1.1 Bayesian Optimization

TuRBO uses a statistical model called a Gaussian process to approximate the objective function. This is a collection of random variables (possibly not enumerable) such that each finite subset follows a multivariate normal distribution. As in the finite case, this $\mathcal{GP}$ process is completely determined by a function μ and function Σ that fulfill the role of the mean and the variance respectively.

To make predictions using this approximation, suppose we have some noisy observations of a function f

$$y(x) = f(x) + \epsilon\sigma_y.$$

where $\epsilon \sim \mathcal{N}(0,1)$ ant the parameter σ_y accounts for the amplitude of the noise. We can suppose that the probability of the observations are given by a Gaussian process where the mean is usually taken to be 0. This is,

$$p(f(x)|\theta) = \mathcal{GP}(0, K(x,x') + I\sigma_y).$$

In this way we took Σ to be some function K called the kernel and the identity times σ_y that accounts for the noise in the observations. Several choices of kernel are possible. In the case of turbo the a Matérn-5/2 kernel with ARD covariance function is used. This function gives the covariance of two points as a function of its distance and its given by

$$K(x,x';\zeta,\nu,\lambda_1\ldots\lambda_d) = \frac{s^2}{2^{\nu-1}\Gamma(\nu)}\left(\frac{d(x,x';\lambda_1\ldots\lambda_d)}{\zeta}\right)^{\nu} K_\nu\left(\frac{d(x,x';\lambda_1\ldots\lambda_d)}{\zeta}\right)$$

In the formula the parameter ν is a parameter that accounts for the differentiability of the learned function. In our case ν is taken to be $5/2$ meaning that the learned function will be two times differentiable. d is the distance between x and x' by doing a rescaling according to λ_i. Additionally the parameters s^2 and ζ are the variance and another scaling factor. Finally K_ν is the modified Bessel function.

We can maximize the log likelihood function to estimate the parameters of the $\mathcal{GP}$. In the article the authors seated bounds for the hyper-parameters as follows : $\lambda_i \in [0.005, 2.0]$, $s^2 \in [0.05, 20.0]$ and $\sigma_y^2 \in [0.0005, 0.1]$

1.2 Trust region methods

A trust region method is an optimization method that is limited to a subset of Omega (TR) in which the objective function is approximated with another function with better characteristics. The TR is updated iteratively in such a way that if our prediction is good, the TR grows and if it is bad it shrinks. In this way, we ensure that if the approximation of the objective function is good, the search area can be expanded and if it is bad, it should be decreased in order to obtain a better approximation of the objective function.

In TuRBO methodology, a subset of q points from our function in the trust regions is sampled. These points are then evaluated against the current best approximation. Their performance is categorized as either successes or failures based on whether they improve upon the previous minimum. This categorization drives the adjustment of parameters τ_{fail} and τ_{succ}: if failures outnumber τ_{fail}, the sampling area decreases; conversely, if successes surpass τ_{succ}, the sampling area expands.

1.3 TuRBO algorithm

The algorithm consists of using m trust regions with m independent GP models. In each of the trust regions we set $L \longleftarrow L_{\text{init}}$. The trust region will be then an hyper rectangle of volume L^d. Each side of the hyper-rectangle is calculated using the parameters λ_i of the respective $\mathcal{GP}$ of the trust region. in this way we have $L_i = \lambda_i L / \left(\prod_{j=1}^{d} \lambda_j\right)^{1/d}$. After this we do the sampling of the q point within the trust region and we evaluate the performance of the

[*]Submission deadline: to be announced (likely late March/early April.

GECCO '23, July 15–19, 2023, Lisbon, Portugal
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

algorithm. The update of the trust region is done by setting $L \leftarrow \min\{L_{\max}, 2L\}$ if we decide to increase the trust region, or $L \leftarrow L/2$ if we shrink it. In the case $L < L_{\min}$ we discard the trust region and create another one.

Finally to solve the problem of how to sample our badge of q pints from the trust regions in each iteration TuRBO uses Thomson sampling to allocate the points in an efficient way

1.4 parameters of the algorithm

- $L_{\max}$: Maximum size of the side of the hyperrectangle
- $L_{\min}$: Minimum size of the side of the hyperrectangle
- L_{init}: This magnitude is used to calculate the sides of the TR hyperrectangle. The true lengths are calculated such that the initial volume is L_{init}^d and are scaled using the $\mathcal{GP}$ λ_i parameters
- τ_{succ}: If this number of successes is exceeded, the trust region is expanded.
- τ_{fail}: If this number of failures is exceeded, the trust region is shrunk.

2 CPU TIMING

In order to evaluate the CPU timing of the algorithm, we have run the TuRBO-optimize-fmin-001 with restarts on the entire bbob test suite [3] for $100n$ function evaluations according to [4]. The C/Java/Matlab/Octave/Python code was run on a Mac Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz with 1 processor and 4 cores and (compile) options xxx. The time per function evaluation for dimensions 2, 3, 5, 10, 20, 40 equals $x.x$, $x.x$, $x.x$, xx, xxx, and xxx seconds respectively.

3 RESULTS

Results from experiments according to [4] and [?] on the benchmark functions given in [1] are presented in Figures 1 and 3 and Tables 1 and **??**.

The experiments were performed with COCO [2], version 2.6, the plots were produced with version 2.6.

The **expected runtime (ERT)**, used in the tables, depends on a given target precision, $I_{\text{target}} = f_{\text{opt}} + \Delta f$, and is computed over all relevant trials as the number of function evaluations executed during each trial while the best function value did not reach f_t, summed over all trials and divided by the number of trials that actually reached f_t [5?]. **Statistical significance** is tested with the rank-sum test for a given target Δf_t using, for each trial, either the number of needed function evaluations to reach f_t (inverted and multiplied by -1), or, if the target was not reached, the best Δf-value achieved, measured only up to the smallest number of overall function evaluations for any unsuccessful trial under consideration.

REFERENCES

[1] S. Finck, N. Hansen, R. Ros, and A. Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Presentation of the Noiseless Functions.* Technical Report 2009/20. Research Center PPE. http://coco.lri.fr/downloads/download15.03/bbobdocfunctions.pdf Updated February 2010.

[2] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. 2021. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *Optimization Methods and Software* 36 (2021), 114–144. Issue 1. https://doi.org/10.1080/10556788.2020.1808977

[3] N. Hansen, S. Finck, R. Ros, and A. Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions.* Technical Report RR-6829. INRIA. http://hal.inria.fr/inria-00362633/en/

[4] N. Hansen, T. Tušar, O. Mersmann, A. Auger, and D. Brockhoff. 2016. COCO: The Experimental Procedure. *ArXiv e-prints* arXiv:1603.08776 (2016).

[5] Kenneth Price. 1997. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation.* IEEE, Piscataway, NJ, USA, 153–157. https://doi.org/10.1109/ICEC.1997.592287

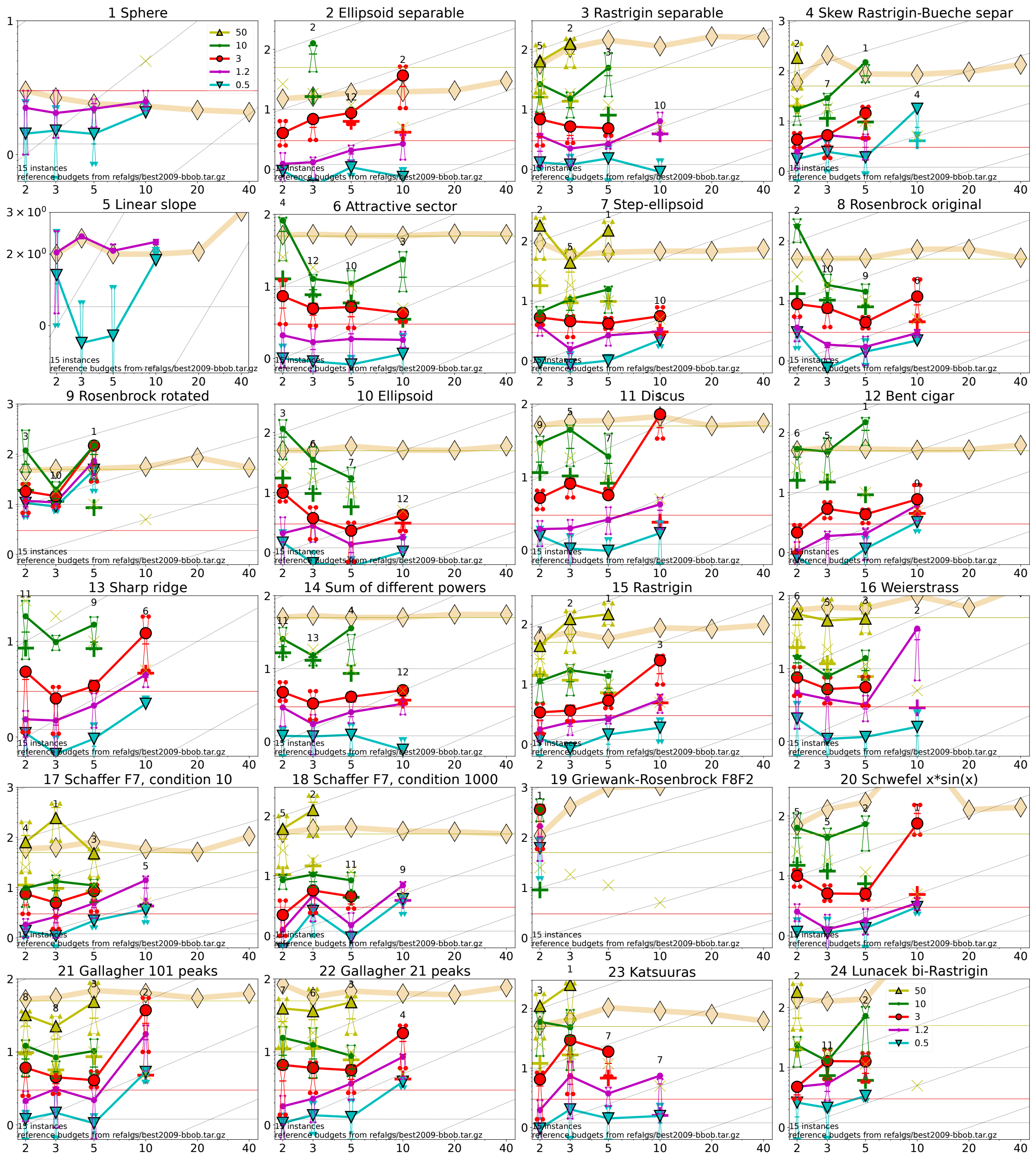

Figure 1: Scaling of runtime with dimension to reach certain target values Δf. Lines: expected runtime (ERT); Cross (+): median runtime of successful runs to reach the most difficult target that was reached at least once (but not always); Cross (×): maximum number of f-evaluations in any trial. Notched boxes: interquartile range with median of simulated runs; All values are divided by dimension and plotted as $\log_{10}$ values versus dimension. Shown is the ERT for targets just not reached by the best algorithm from BBOB 2009 within the given budget $k \times \mathrm{DIM}$, where k is shown in the legend. Numbers above ERT-symbols (if appearing) indicate the number of trials reaching the respective target. The light thick line with diamonds indicates the best algorithm from BBOB 2009 for the most difficult target. Slanted grid lines indicate a scaling with $\mathcal{O}(\mathrm{DIM})$ compared to $\mathcal{O}(1)$ when using the respective reference algorithm.

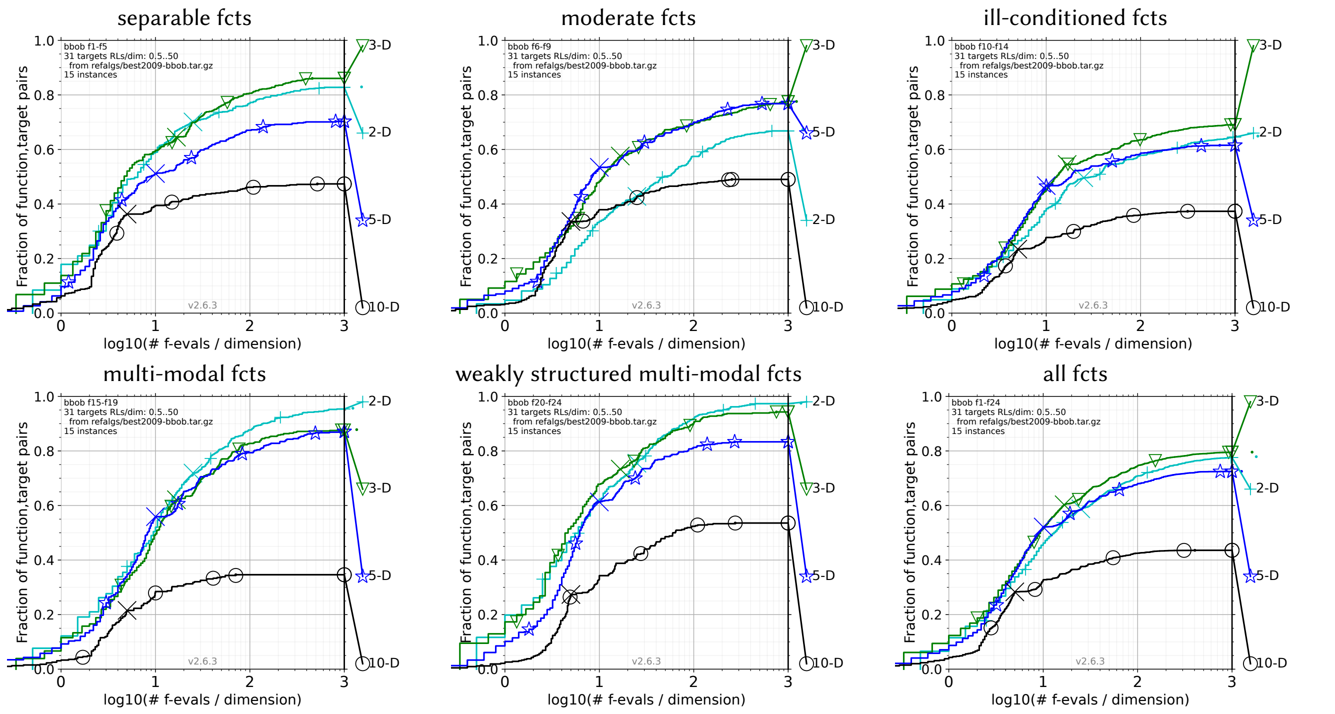

Figure 2: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of f-evaluations divided by dimension (FEvals/DIM) for all function groups and all dimensions and for those targets in $10^{[-8..2]}$ that have just not been reached by the best algorithm from BBOB 2009 in a given budget of $k \times$ DIM, with 31 different values of k chosen equidistant in logscale within the interval $\{0.5, \ldots, 50\}$. The aggregation over all 24 functions is shown in the last plot.

#FEs/D	0.5	1.2	3	10	50	#succ
f_1	*2.5e+1*:5.0	*1.6e+1*:8.0	*1.0e-8*:12	*1.0e-8*:12	*1.0e-8*:12	15/15
	1.4(1)	1.4(0.6)	∞	∞	∞ *50*	0/15
f_2	*1.6e+6*:3.0	*4.0e+5*:11	*2.5e+4*:16	*6.3e+2*:58	*1.0e-8*:58	15/15
	1.8(1)	0.94(0.5)	2.7(2)	∞	∞ *50*	0/15
f_3	*1.6e+2*:4.0	*1.0e+2*:15	*6.3e+1*:23	*2.5e+1*:73	*1.0e+1*:73	15/15
	1.9(1)	0.88(0.5)	1.0(0.4)	3.4(5)	∞ *50*	0/15
f_4	*2.5e+2*:3.0	*1.6e+2*:10	*1.0e+2*:19	*4.0e+1*:65	*1.6e+1*:65	15/15
	3.2(3)	2.2(3)	3.8(4)	12(21)	∞ *50*	0/15
f_5	*6.3e+1*:4.0	*4.0e+1*:10	*1.0e-8*:10	*1.0e-8*:10	*1.0e-8*:10	15/15
	1.1(1)	1.0(0.2)	∞	∞	∞ *50*	0/15
f_6	*1.0e+5*:3.0	*2.5e+4*:8.0	*1.0e+2*:16	*2.5e+1*:54	*2.5e-1*:54	15/15
	1.4(2)	1.2(0.9)	1.6(2)	1.0(0.9)	∞ *50*	0/15
f_7	*1.6e+2*:4.0	*6.3e+1*:11	*2.5e+1*:20	*4.0e+0*:54	*1.0e+0*:54	15/15
	1.3(1)	1.2(0.9)	1.1(0.4)	1.4(1)	2.3(2)	1/15
f_8	*1.0e+4*:5.0	*6.3e+3*:7.0	*1.0e+3*:18	*6.3e+1*:54	*1.6e+0*:54	15/15
	1.4(1)	1.2(0.9)	1.2(0.4)	1.3(1)	∞ *50*	0/15
f_9	*2.5e+1*:20	*1.6e+1*:26	*1.0e+1*:35	*4.0e+0*:62	*1.6e-2*:62	15/15
	12(11)	14(14)	21(25)	12(17)	∞ *50*	0/15
f_{10}	*2.5e+6*:3.0	*6.3e+5*:7.0	*2.5e+5*:17	*6.3e+3*:54	*2.5e+1*:54	15/15
	0.98(1)	0.98(1)	0.68(0.6)	1.6(2)	∞ *50*	0/15
f_{11}	*1.0e+6*:3.0	*4.0e+4*:8.0	*6.3e+2*:16	*6.3e+1*:74	*6.3e-1*:74	15/15
	1.6(2)	1.6(1)	1.8(0.7)	1.3(1)	∞ *50*	0/15
f_{12}	*4.0e+7*:4.0	*1.6e+7*:8.0	*4.0e+6*:19	*1.6e+4*:52	*1.0e+0*:52	15/15
	1.4(1)	1.3(1)	1.1(0.5)	14(21)	∞ *50*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f_{13}	*1.0e+3*:3.0	*6.3e+2*:8.0	*4.0e+2*:17	*6.3e+1*:52	*6.3e-2*:52	15/15
	1.6(1)	1.3(0.8)	1.0(0.3)	1.4(1)	∞ *50*	0/15
f_{14}	*1.6e+1*:3.0	*1.0e+1*:10	*4.0e+0*:22	*2.5e-1*:53	*1.0e-5*:53	15/15
	2.1(2)	1.3(0.7)	0.94(0.3)	3.4(4)	∞ *50*	0/15
f_{15}	*1.6e+2*:3.0	*1.0e+2*:13	*6.3e+1*:24	*4.0e+1*:55	*1.6e+1*:55	5/5
	2.4(2)	1.0(0.6)	1.1(0.6)	1.3(1)	2.5(4)	1/15
f_{16}	*4.0e+1*:5.0	*2.5e+1*:16	*1.6e+1*:46	*1.0e+1*:120	*4.0e+0*:120	15/15
	1.2(1)	1(1.0)	0.61(0.4)	0.58(0.6)	0.73(0.7)	3/15
f_{17}	*1.0e+1*:5.0	*6.3e+0*:26	*4.0e+0*:57	*2.5e+0*:110	*6.3e-1*:110	15/15
	2.2(1)	0.94(0.7)	0.75(0.8)	0.50(0.3)	0.59(0.5)	3/15
f_{18}	*6.3e+1*:3.0	*4.0e+1*:7.0	*2.5e+1*:20	*1.6e+1*:58	*1.6e+0*:58	15/15
	1.6(2)	1.1(0.9)	1.1(0.4)	0.73(1)	∞ *50*	0/15
f_{19}	*1.6e-1*:172	*1.0e-1*:242	*6.3e-2*:675	*4.0e-2*:3078	*2.5e-2*:3078	15/15
	∞	∞	∞	∞	∞ *50*	0/15
f_{20}	*6.3e+3*:5.0	*4.0e+3*:8.0	*2.5e+1*:16	*2.5e+0*:69	*1.0e+0*:69	15/15
	1.3(1)	1.1(0.8)	1.6(0.5)	5.3(7)	∞ *50*	0/15
f_{21}	*4.0e+1*:4.0	*2.5e+1*:11	*1.6e+1*:31	*6.3e+0*:73	*1.6e+0*:73	5/5
	1.3(1)	0.99(0.9)	0.66(0.4)	0.71(0.4)	0.70(0.9)	3/15
f_{22}	*6.3e+1*:4.0	*4.0e+1*:15	*2.5e+1*:32	*1.0e+1*:71	*1.6e+0*:71	5/5
	1.6(1)	1.2(0.8)	0.88(0.8)	0.62(0.4)	0.70(0.5)	3/15
f_{23}	*1.0e+1*:3.0	*6.3e+0*:9.0	*4.0e+0*:33	*2.5e+0*:84	*1.0e+0*:84	15/15
	2.4(2)	2.1(1)	2.9(4)	∞	∞ *55*	0/15
f_{24}	*6.3e+1*:15	*4.0e+1*:37	*4.0e+1*:37	*2.5e+1*:118	*1.6e+1*:118	15/15
	1.1(0.6)	1.7(2)	1.7(1)	3.1(2)	∞ *50*	0/15

Table 1: Expected runtime (ERT in number of f-evaluations) divided by the ERT of the best algorithm from BBOB 2009 in dimension 5. This ERT ratio 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 $f_{\mathrm{opt}} + 10^{-8}$. The median number of conducted 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 from BBOB 2009, with $p = 0.05$ or $p = 10^{-k}$ when the number $k > 1$ is following the ↓ symbol, with Bonferroni correction by the number of functions (24).

Data produced with COCO v2.6.3

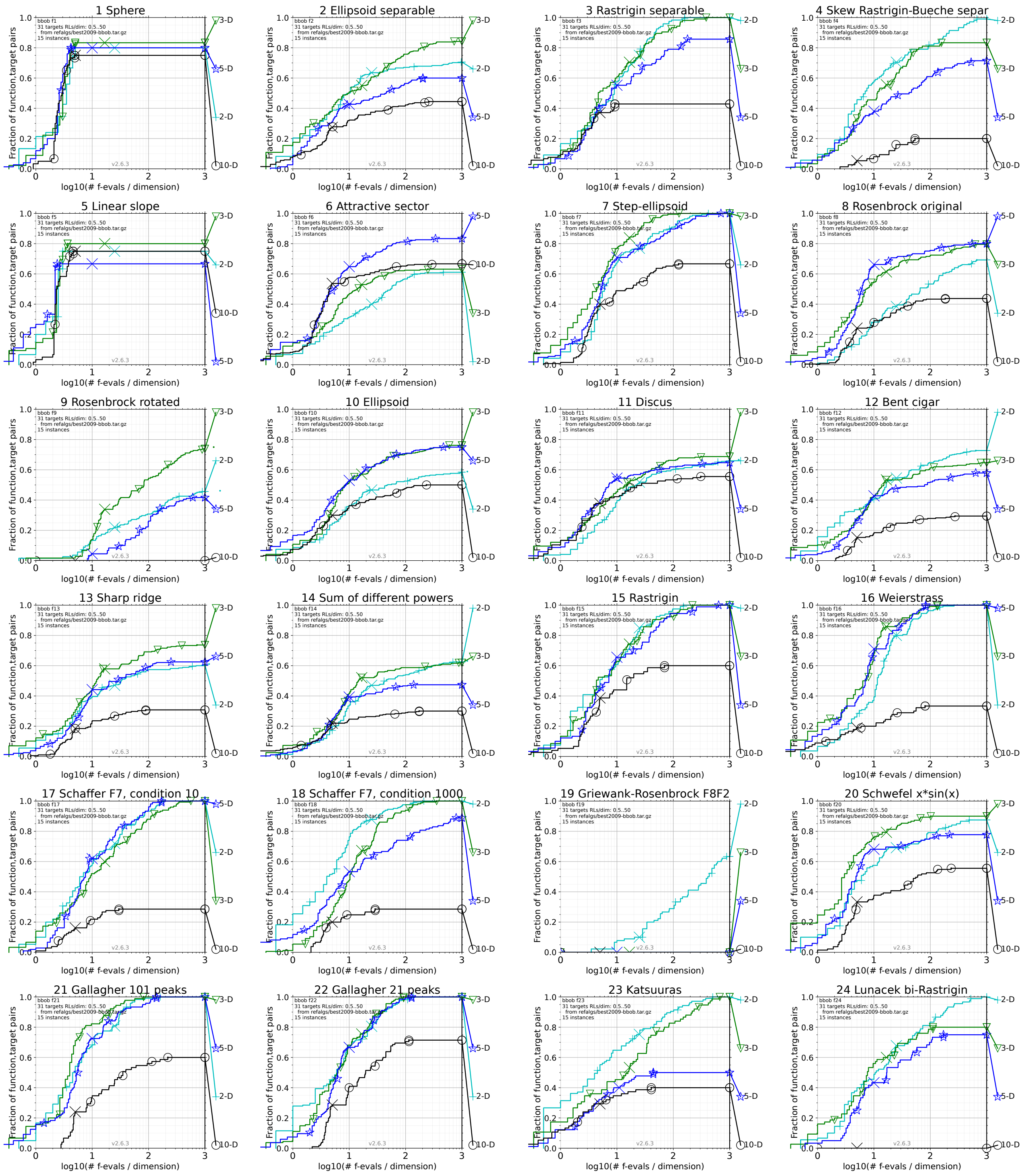

Figure 3: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of f-evaluations divided by dimension (FEvals/DIM) in dimensions 2 to 40 and for those targets in $10^{[-8..2]}$ that have just not been reached by the best algorithm from BBOB 2009 in a given budget of $k \times$ DIM, with 31 different values of k chosen equidistant in logscale within the interval $\{0.5, \ldots, 50\}$.