An Evaluation of Sequential Model-Based Optimization for Expensive Blackbox Functions

Frank Hutter
Freiburg University
Department of Computer Science
fh@informatik.uni-freiburg.de

Holger Hoos
University of British Columbia
Department of Computer Science
hoos@cs.ubc.ca

Kevin Leyton-Brown
University of British Columbia
Department of Computer Science
kevinlb@cs.ubc.ca

ABSTRACT

We benchmark a sequential model-based optimization procedure, SMAC-BBOB, on the BBOB set of blackbox functions. We demonstrate that with a small budget of $10 \times D$ evaluations of D-dimensional functions, SMAC-BBOB in most cases outperforms the state-of-the-art blackbox optimizer CMA-ES. However, CMA-ES benefits more from growing the budget to $100 \times D$, and for larger number of function evaluations SMAC-BBOB also requires increasingly large computational resources for building and using its models.

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

1. INTRODUCTION

This paper discusses results obtained by applying the sequential model-based algorithm configuration procedure SMAC [7] to standard blackbox function optimization problems. We first describe the general SMAC procedure and the particular variant, SMAC-BBOB, used here. Then, we evaluate SMAC-BBOB empirically.

2. SMAC AND SMAC-BBOB

SMAC has been designed to target blackbox functions that arise in the optimization of algorithm parameters. Formally, the *algorithm configuration* (AC) problem solved by SMAC can be stated as follows. We are given a parameterized algorithm A with configuration space Θ, a distribution D of problem instances, and a performance metric $m(\boldsymbol{\theta}, \pi)$ capturing A's performance with parameter settings $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ on instances $\pi \sim D$. Let $f(\boldsymbol{\theta}) = \mathbb{E}_{\pi \sim D}[m(\boldsymbol{\theta}, \pi)]$ denote the expected performance of A given parameter setting $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ (where the expectation is over instances π drawn from D; in the case of randomized algorithms, it would also be over random seeds). The problem is then to find a parameter setting $\boldsymbol{\theta}$ of A that minimizes $f(\boldsymbol{\theta})$.

SMAC is rooted in the statistics literature on sequential model-based optimization (SMBO) of expensive functions [13, 12]. To minimize functions $f : \boldsymbol{\Theta} \mapsto \mathbb{R}$ that do not have closed-form representations, are costly to evaluate, and do not allow the computation of gradients, SMBO first gathers initial data and then iterates over the following steps:

1. based on the data collected thus far, construct a model that predicts a probability distribution for f's value at arbitrary points $\boldsymbol{\theta} \in \boldsymbol{\Theta}$;
2. use the model to quantify the desirability $d(\boldsymbol{\theta})$ of learning $f(\boldsymbol{\theta})$ for each $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ and to select $\boldsymbol{\theta} \in \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} d(\boldsymbol{\theta})$; and
3. evaluate $f(\boldsymbol{\theta})$, resulting in a new data point $\langle \boldsymbol{\theta}, f(\boldsymbol{\theta}) \rangle$.

The desirability function d serves to address the exploration/exploitation tradeoff between learning about new, unknown parts of the parameter space and intensifying the search locally in the best known region. SMAC uses a classic desirability function that evaluates each configuration $\boldsymbol{\theta}$ by its *expected positive improvement* $\mathbb{E}[I(\boldsymbol{\theta})] = \mathbb{E}[\max\{0, f_{\min} - f(\boldsymbol{\theta})\}]$ over the lowest function value $f_{\min}$ seen so far, where the expectation is taken with respect to model predictions and can be computed in closed form given a model with Gaussian predictive distribution $\mathcal{N}(\mu_{\boldsymbol{\theta}}, \sigma^2_{\boldsymbol{\theta}})$ (see, *e.g.*, [12] for the exact formula).

In developing SMAC, we have extended existing SMBO approaches in various ways in order to obtain a procedure that is applicable to the practical problem of algorithm configuration. Specifically, we developed mechanisms for handling

- discrete and conditional parameter spaces [7] (allowing for the optimization of categorical algorithm parameters and parameters among which dependencies exist);
- substantial, non-Gaussian noise [8] (due to variance in the distribution of algorithm runtime over problem instances and multiple independent runs on the same instance);
- partially censored function evaluations [6] (due to prematurely terminated algorithm runs);
- a budget on the total time available for algorithm configuration, rather than on the number of function evaluations [9]; and

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

- distributed optimization on computer clusters [5].

While crucial for effectively tackling algorithm configuration problems, none of these features is relevant to the task of optimizing deterministic, continuous, sequential blackbox functions that is the subject of this study. We therefore modified SMAC in two ways, resulting in a version we call SMAC-BBOB. Firstly, by default, SMAC uses random forest (RF) models instead of the more commonly-used Gaussian process (GP) models, because RF models improve performance in discrete optimization and can be relatively easily adapted to accommodate censored data and conditional parameters. SMAC-BBOB uses GP models instead, since these have been found to perform better for continuous optimization problems [9]. Secondly, in its default variant, SMAC uses a relatively inexpensive multi-start local search procedure in order to optimize $\mathbb{E}[I(\boldsymbol{\theta})]$ over $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. The blackbox functions tackled here are assumed to be expensive compared to the time required by the subsidiary optimization procedure. Therefore, in addition to SMAC's local search, for each EI optimization, SMAC-BBOB also executes one run of DIRECT [11] (with a budget of $10 \times D$ evaluations of $\mathbb{E}[I(\boldsymbol{\theta})]$) and 10 runs of CMA-ES [4] (each using $100 \times D$ evaluations of $\mathbb{E}[I(\boldsymbol{\theta})]$); we note that these subsidiary EI optimization runs only use (cheap) model evaluations rather than evaluations of the (expensive) blackbox function to be optimized. After performing these subsidiary EI optimization runs, SMAC-BBOB queries the expensive function at the resulting configuration $\boldsymbol{\theta}$ with highest $\mathbb{E}[I(\boldsymbol{\theta})]$.[1]

We note that the resulting SMAC-BBOB variant resembles the classic SMBO method EGO ("efficient global optimization" [12]). This is not surprising, since EGO was the original starting point for SMAC's development, and for the SMAC-BBOB variant we removed several of SMAC's new components. Nevertheless, some differences remain; we discuss and evaluate these in Section 3.2.

3. EXPERIMENTAL ANALYSIS

We now turn to our empirical assessment of SMAC-BBOB. Our software and the data underlying the experiments reported here is available online at http://www.cs.ubc.ca/labs/beta/Projects/SMAC/.

3.1 Experimental Setup

We carried out experiments following [2] on the BBOB benchmark functions given in [1, 3]. Our experiments use a budget between 10×D and 100×D function evaluations, where D is the dimensionality of the given function. The functions are assumed to be expensive enough to impose such a limit and to completely dominate the running time of the procedures studied here. The **expected running time (ERT)**, used in the figures and table, depends on a given target function value, $f_{\mathrm{t}} = f_{\mathrm{opt}} + \Delta f$. It is computed across all runs of an optimizer on a function, as the total number of function evaluations before reaching f_{t} divided by the number of trials that actually reached f_{t} [2, 14]. We use a rank-sum test to assess **statistical significance** of differences in performance for a given target Δf_{t}; here, the performance for each trial is computed as either the number of function evaluations needed to reach Δf_{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.

[1]We regularly observed cases in which the best EI result was found by any of the subsidiary optimizers: SMAC's local search, DIRECT or CMA-ES. We used only $10 \times D$ evaluations for DIRECT, since its internal computations were a bottleneck for high dimensions.

3.2 Assessing Different SMAC Versions

We first performed preliminary experiments with several variants of SMAC, using a budget of $10 \times D$ function evaluations to assess the impact of an initial design and the type of response surface model used. We assessed the impact of four different design decisions:

- **Choice of model: random forest (RF) vs. Gaussian process (GP).** We found that for the all-continuous inputs to be optimized here, RF models were suboptimal, due to their poor extrapolation and uncertainty estimates. They performed particularly poorly for the (simple) case of optimizing the linear function f5: SMAC based on RFs only reached an error of 10^{-8} in 1/15 runs for $D = 5$ (as compared to 15/15 runs when based on GPs).

- **Noise-free vs. noisy kernel.** In our experiments, it was quite detrimental to allow additive observation noise in the GP kernel; using this noise, the GP often produced response surfaces much smoother than the real function. This could potentially be remedied by using a kernel that better captures the typical characteristics of the BBOB functions. As one step in this direction, we used a Matern kernel, which is less smooth than the standard squared exponential kernel and yielded somewhat better results.

- **Isotropic vs. Automatic Relevance Detection (ARD) kernel.** Kernels with different length scales for each dimension (so-called *automatic relevance detection (ARD)* kernels [15]) consistently scaled worse with increasing dimensionality D than isotropic kernels. We attribute this to over-fitting in the hyperparameter optimization of the $O(D)$ kernel parameters based on very sparse data.

- **Initial design.** In our experiments, using an initial design did not improve SMAC's final result but (of course) decreased initial performance.

Based on these results, we defined SMAC-BBOB as using a GP with a noise-free isotropic Matern kernel and no initial design. In contrast, the classic EGO algorithm [12] uses different length scales for each dimension and optimizes them via maximum likelihood; our experiments indicate that SMAC-BBOB's isotropic Matern kernel performs substantially better in high dimensions. The second difference to EGO lies in the initial design, for which EGO uses $10 \times D$ function evaluations. As our experiments in the following sections will show, compared to other methods, SMAC-BBOB performed particularly well for small function evaluation budgets up to $10 \times D$, precisely the range in which EGO still evaluates its initial design.

3.3 Comparison against CMA-ES

We now compare SMAC-BBOB to the state-of-the-art blackbox optimization method CMA-ES [4]. Following the advice of Nikolaus Hansen, we used the publicly available Matlab implementation 3.61 and enabled the option *active CMA* [10].

At a high level, Figures 1–3 demonstrate that, for a small budget of 10×D function evaluations, SMAC-BBOB performed better than CMA-ES in many cases, but that CMA-ES caught up for a larger budget of 100×D function evaluations. Note that Figures 1 and 2 show data for a budget of only 10×D function evaluations, since SMAC-BBOB targets the case of very expensive function evaluations.[2]

[2]Equivalent plots for 100×D function evaluations can be generated

Figure 1 shows the ERTs of SMAC-BBOB and CMA-ES for a given target function value in six different dimensions. With the small budget of $10 \times D$ function evaluations, for 13/24 functions SMAC performed significantly better than CMA-ES for at least one of the six dimensions tested; improvements were particularly significant for functions 5 (linear slope), 6 (attractive sector), 7 (step-ellipsoid), 11 (discuss), and 13 (sharp ridge). The figure also shows that SMAC-BBOB's and CMA-ES's ERTs tended to scale quite similarly with dimensionality: often roughly constant with the best GECCO 2009 data point. The same experiment based on $100 \times D$ function evaluations (results not shown here, for lack of space) showed CMA-ES to catch up: it performed better on 8/24 functions, SMAC performed better on 7/24.

Figure 2 compares the ERTs of SMAC-BBOB and CMA-ES for many different target function values in six different dimensions. As the figure shows, for the small budget of $10 \times D$ function evaluations, SMAC-BBOB often outperformed CMA-ES, and in some cases (f1, f5–f11, f13, f16, f20, and f21) also scaled better to finding excellent target function values. In contrast, when considering a larger budget of $100 \times D$ function evaluations (results not shown here, for lack of space), CMA-ES typically was more competitive for finding excellent target function values.

Figure 3 shows run-length distributions for SMAC-BBOB and CMA-ES for $D = 5$ and $D = 20$, aggregating over various families of functions. It also gives distributions of speedups, which show that, compared to CMA-ES, SMAC-BBOB performed particularly well for the separable, multimodal, and weakly-structured functions. Our method performed better for $10 \times D$ function evaluations, but CMA-ES in many cases caught up and overtook SMAC-BBOB when allowed $100 \times D$ function evaluations. The initial speedups of SMAC-BBOB also tended to be more pronounced for higher dimensions (compare $D = 20$ to $D = 5$).

Overall, from this comparison we conclude that for small budgets of $10 \times D$ function evaluations, SMAC-BBOB in many cases performs better than CMA-ES (particularly for separable, multimodal, and weakly-structured functions) and that CMA-ES typically catches up somewhere between $10 \times D$ and $100 \times D$ function evaluations.

3.4 Comparison Against Best BBOB-09 Results

Table 1 compares SMAC-BBOB's performance to the best results from the Blackbox Optimization Benchmarking Workshop 2009 (BBOB-09). The table demonstrates that for small function evaluation budgets, SMAC-BBOB was competitive with, and in some cases significantly better than, the best results from BBOB-09 (for 2 functions and a combined 3 target values in $D = 5$; and for 7 functions and a combined 11 target values in $D = 20$). In several cases, SMAC-BBOB was very competitive for finding target function values of moderate quality, but not for finding the very best ones. One interesting exception is for the multimodal function f22 in $D = 20$ dimensions, for which SMAC-BBOB's performance relative to the best results from BBOB-09 improves as better target function values are required.

Finally, Figure 4 provides some additional standard plots for SMAC-BBOB to facilitate comparisons with other blackbox optimizers. As the figure shows, SMAC-BBOB's performance, measured as ERT/D to achieve comparable target function values, is quite similar for $D = 5$ and $D = 20$ dimensions.

based on the data we submitted to the BBOB Workshop, but those plots lack data points for $D = 40$ for SMAC-BBOB due to limited computational resources.

3.5 Computational Requirements

While the standard version of SMAC strives to limit the complexity of learning models and using them to select the next query point, SMAC-BBOB's design has not been led by such considerations. It simply targets expensive blackbox functions, whose evaluation cost dominates the time spent inside SMAC-BBOB. For example, it does *not* use approximations of GPs (as we did use in [9]).[3] The complexity of fitting a GP is cubic in the number of data points, and thus SMAC-BBOB's per-step complexity grows over time.

Empirically, for small budgets of $10 \times D$ function evaluations, SMAC-BBOB required between tens of seconds for $d = 2$ and about 20 minutes for $D = 20$. For larger budgets of $100 \times D$ function evaluations, SMAC-BBOB required between five minutes for $d = 2$ and 15–120 hours for $D = 20$. While, in principle, SMAC-BBOB can be made more effective using the techniques discussed in [9], we see its prime use case in the optimization of expensive functions with very tight budgets on the number of function evaluations.

4. CONCLUSIONS

We introduced and evaluated SMAC-BBOB, a variant of the sequential model-based algorithm configuration procedure SMAC for the standard BBOB set of continuous blackbox optimization benchmarks. Despite the fact that these deterministic continuous optimization problems do not benefit from many of the features developed for SMAC's primary use in algorithm configuration, SMAC-BBOB showed very competitive performance for the optimization of expensive blackbox functions when using up to $10 \times D$ function evaluations. In particular, in this low-budget setting, it often outperformed the state-of-the-art blackbox optimization procedure CMA-ES, and in several cases performed competitively with the best results from the Blackbox Optimization Benchmarking Workshop 2009 (BBOB-09). It performed particularly well for multi-modal and weakly structured functions. As the number of function evaluations was increased to $100 \times D$, SMAC-BBOB became more computationally expensive and also less effective, such that CMA-ES caught up in performance and overtook SMAC-BBOB in several cases. We thus see SMAC-BBOB as a competitive method (only) for the optimization of rather expensive functions.

Finally, we emphasize that SMAC-BBOB's performance hinges on the effective and computationally cheap subsidiary optimization procedures it uses in each search step to decide which function value to query next. As such, part of its success is due to its subsidiary optimizers, CMA-ES [4] and DIRECT [11].

Acknowledgements

We thank Nikolaus Hansen for providing the implementation of CMA-ES we used and for technical support with the postprocessing software.

5. REFERENCES

[1] 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.

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

[3] This is because we wished to utilize a noise-free kernel, but the GPML implementation of GPs [15] we use requires additive noise in the covariance function in order to apply an approximation.

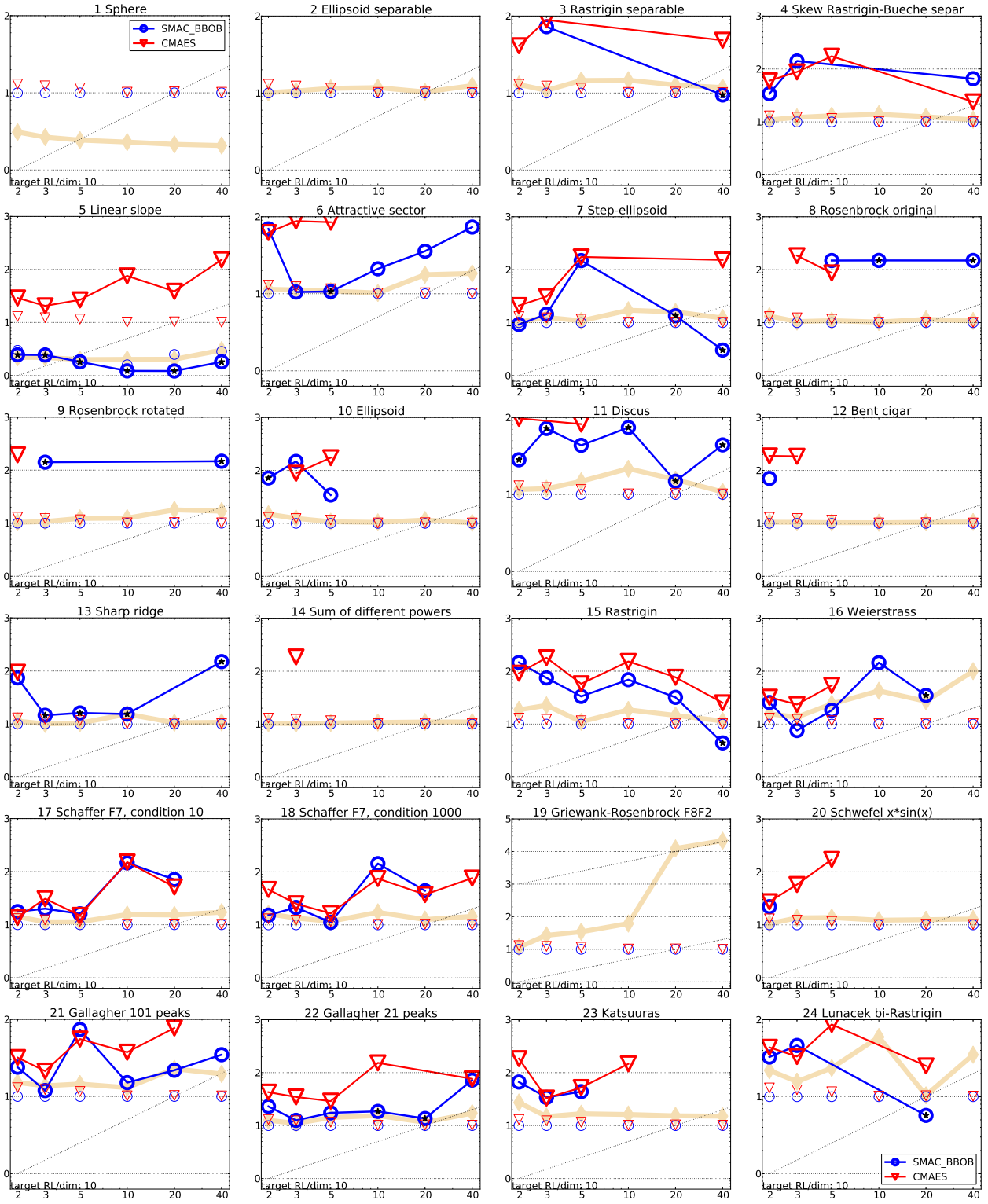

Figure 1: Expected running time (ERT in number of f-evaluations as $\log_{10}$ value) divided by dimension versus dimension. The target function value is chosen such that the bestGECCO2009 artificial algorithm just failed to achieve an ERT of $10 \times$ DIM. Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Light symbols give the maximum number of function evaluations from the longest trial divided by dimension. Black stars indicate statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: ○:SMAC-BBOB, ▽:CMAES.

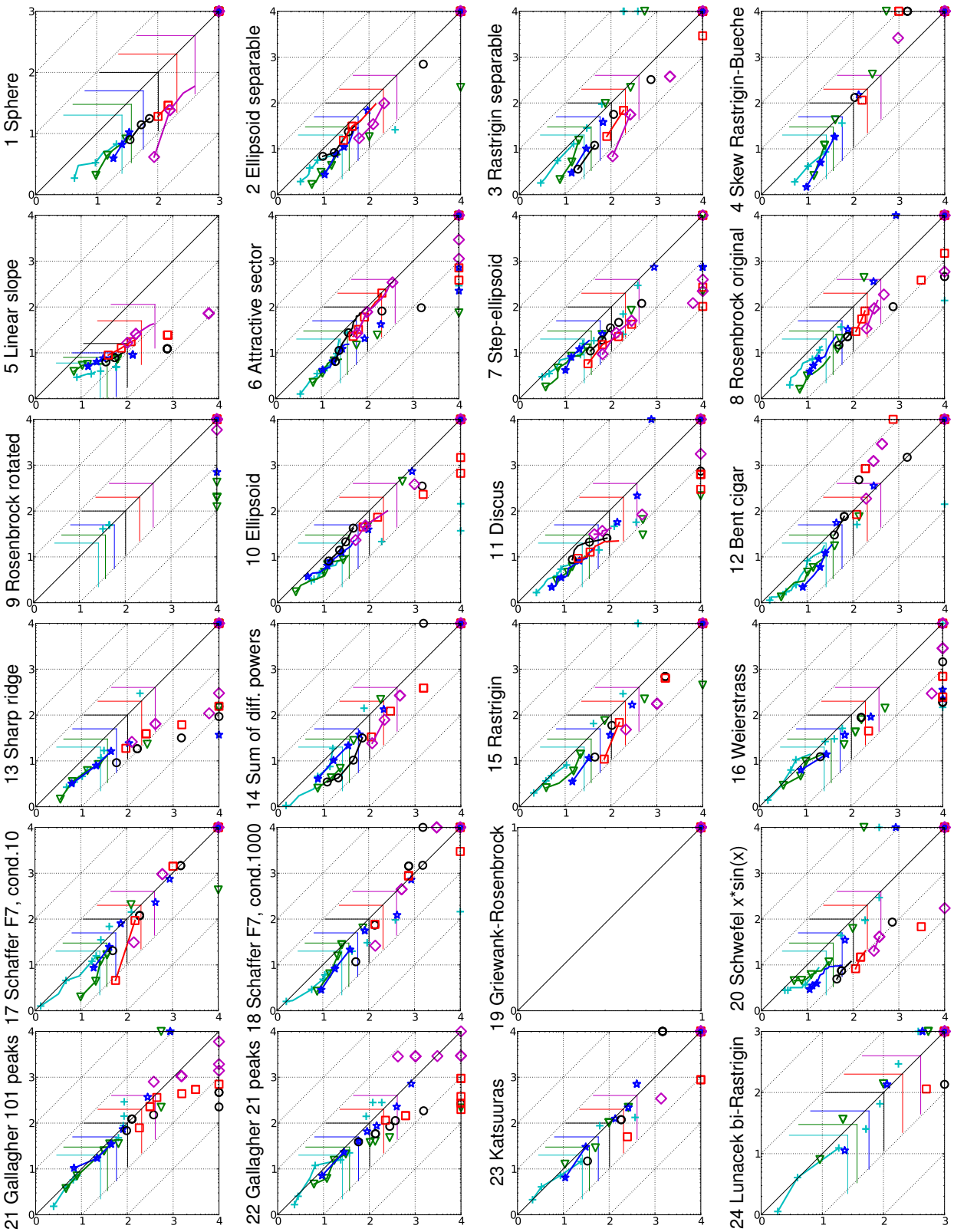

Figure 2: Expected running time (ERT in $\log_{10}$ **of number of function evaluations) of SMAC-BBOB** (y**-axis) versus CMAES** (x**-axis) for** 8 **runlength-based target function values for budgets between** $0.5 \times \mathrm{DIM}$ **and** $50 \times \mathrm{DIM}$ **evaluations. Each runlength-based target** f**-value is chosen such that the** ERT**s of the bestGECCO2009 artificial algorithm for the given and a slightly easier target bracket the reference budget. Markers on the upper or right edge indicate that the respective target value was never reached. Markers represent dimension: 2:+, 3:▽, 5:⋆, 10:○, 20:□, 40:◇.**

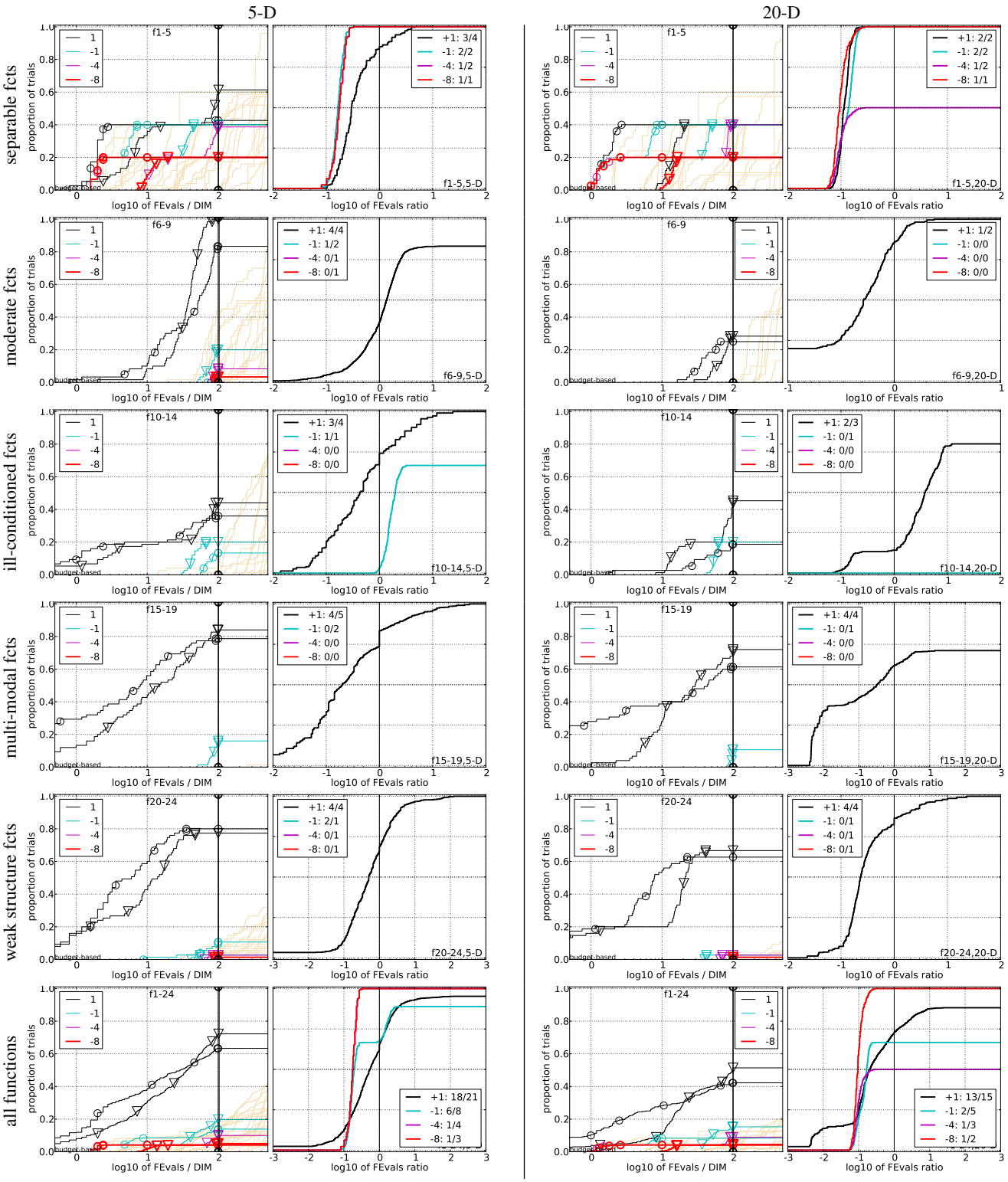

Figure 3: Empirical cumulative distributions (ECDF) of run lengths and speedup ratios in 5-D (left) and 20-D (right). Left sub-columns: ECDF of the number of function evaluations divided by dimension D (FEvals/D) to reach a target value $f_{\text{opt}} + \Delta f$ with $\Delta f = 10^k$, where $k \in \{1, -1, -4, -8\}$ is given by the first value in the legend, for SMAC-BBOB ($\circ$) and CMAES ($\triangledown$). Light beige lines show the ECDF of FEvals for target value $\Delta f = 10^{-8}$ of all algorithms benchmarked during BBOB-2009. Right sub-columns: ECDF of FEval ratios of SMAC-BBOB divided by CMAES, all trial pairs for each function. Pairs where both trials failed are disregarded, pairs where one trial failed are visible in the limits being > 0 or < 1. The legends indicate the number of functions that were solved in at least one trial (SMAC-BBOB first).

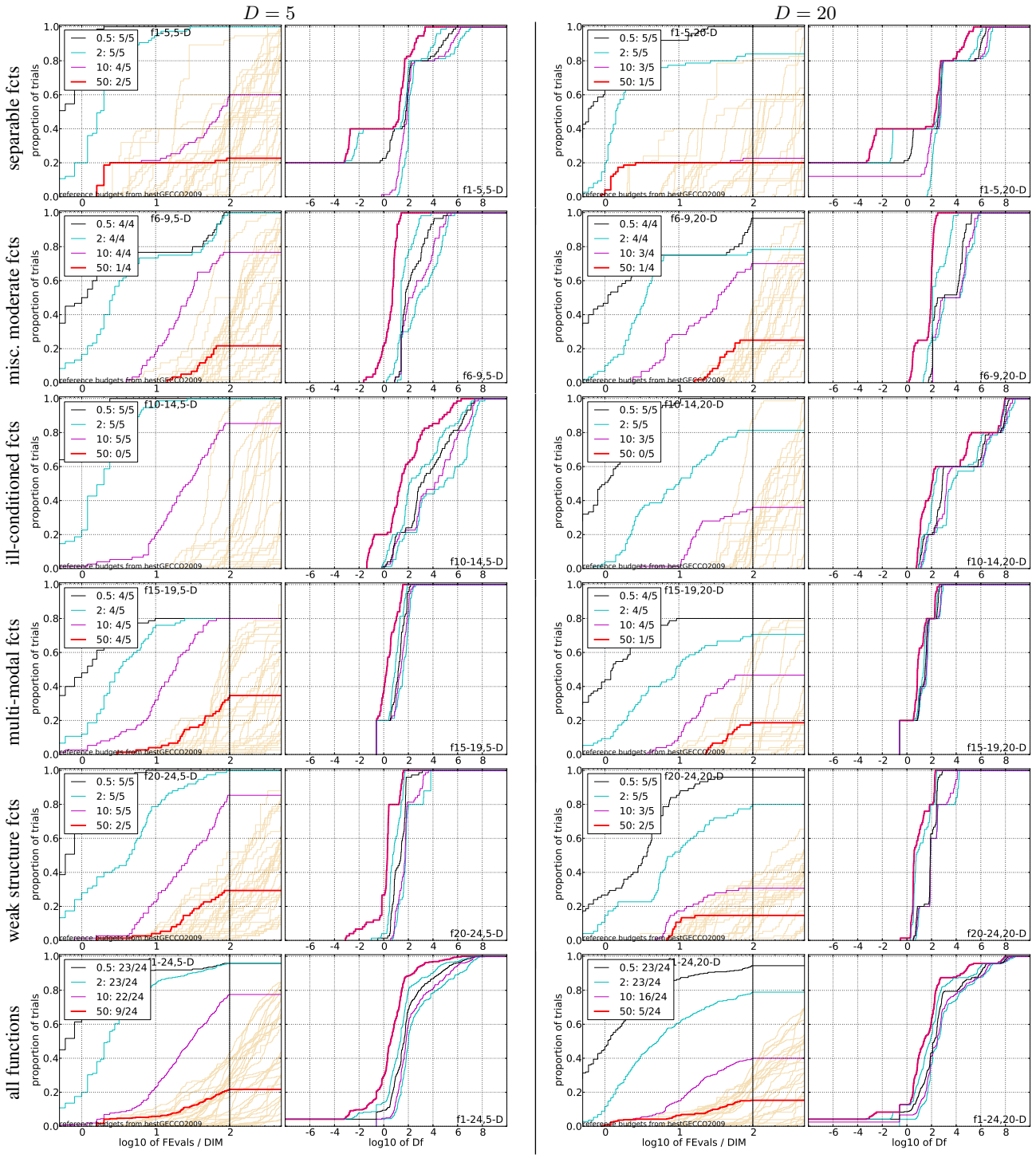

Figure 4: Empirical cumulative distribution functions (ECDF), plotting the fraction of trials with an outcome not larger than the respective value on the x-axis. Left subplots: ECDF of number of function evaluations (FEvals) divided by search space dimension D, to fall below $f_{\mathrm{opt}} + \Delta f$ where Δf is the target just not reached by the GECCO-BBOB-2009 best algorithm within a budget of $k \times \mathrm{DIM}$ evaluations, where k is the first value in the legend. 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 the most difficult target of all algorithms benchmarked during BBOB-2009.

5-D

#FEs/D	0.5	1.2	3	10	50	#succ
$\mathbf{f_1}$	*2.5e+1*:4.8	*1.6e+1*:7.6	*1.0e-8*:12	*1.0e-8*:12	*1.0e-8*:12	15/15
	0.79(0.7)	0.84(0.5)	∞	∞	∞*500*	0/15
$\mathbf{f_2}$	*1.6e+6*:2.9	*4.0e+5*:11	*4.0e+4*:15	*6.3e+2*:58	*1.0e-8*:95	15/15
	1.0(0.9)	0.74(0.7)	1.7(2)	8.0(7)	∞*500*	0/15
$\mathbf{f_3}$	*1.6e+2*:4.1	*1.0e+2*:15	*6.3e+1*:23	*2.5e+1*:73	*1.0e+1*:716	15/15
	0.73(0.6)	0.74(1.0)	2.6(4)	4.4(5)	5.1(5)	2/15
$\mathbf{f_4}$	*2.5e+2*:2.6	*1.6e+2*:10	*1.0e+2*:19	*4.0e+1*:65	*1.6e+1*:434	15/15
	0.56(0.4)	0.54(0.6)	1.8(3)	14(13)	∞*500*	0/15
$\mathbf{f_5}$	*6.3e+1*:4.0	*4.0e+1*:10	*1.0e-8*:10	*1.0e-8*:10	*1.0e-8*:10	15/15
	1.3(0.2)	0.63(0.2)	0.95(0.1)	0.95(0.1)	0.95(0.1)	15/15
$\mathbf{f_6}$	*1.0e+5*:3.0	*2.5e+4*:8.4	*1.0e+2*:16	*2.5e+1*:54	*2.5e-1*:254	15/15
	1.4(1)	1.1(1)	1.5(2)	1.9(2)	∞*500*	0/15
$\mathbf{f_7}$	*1.6e+2*:4.2	*1.0e+2*:6.2	*2.5e+1*:20	*4.0e+0*:54	*1.0e+0*:324	15/15
	1.3(1)	1.1(0.9)	1.5(1)	1.6(0.8)	0.88(0.9)	13/15
$\mathbf{f_8}$	*1.0e+4*:4.6	*6.3e+3*:6.8	*1.0e+3*:18	*6.3e+1*:54	*1.6e+0*:258	15/15
	0.99(1.0)	0.91(1)	1.2(1)	3.3(2)	∞*500*	0/15
$\mathbf{f_9}$	*2.5e+1*:20	*1.6e+1*:26	*1.0e+1*:35	*4.0e+0*:62	*1.6e-2*:256	15/15
	14(6)	12(4)	12(8)	120(122)	∞*500*	0/15
$\mathbf{f_{10}}$	*2.5e+6*:2.9	*6.3e+5*:7.0	*2.5e+5*:17	*6.3e+3*:54	*2.5e+1*:297	15/15
	1.3(0.9)	0.80(0.6)	0.58(0.6)	2.5(2)	∞*500*	0/15
$\mathbf{f_{11}}$	*1.0e+6*:3.0	*6.3e+4*:6.2	*6.3e+2*:16	*6.3e+1*:74	*6.3e-1*:298	15/15
	0.73(0.5)	0.94(0.9)	1.9(2)	0.94(0.8)	∞*500*	0/15
$\mathbf{f_{12}}$	*4.0e+7*:3.6	*1.6e+7*:7.6	*4.0e+6*:19	*1.6e+4*:52	*1.0e+0*:268	15/15
	0.57(0.4)	1.3(2)	3.6(5)	34(34)	∞*500*	0/15
$\mathbf{f_{13}}$	*1.0e+3*:2.8	*6.3e+2*:8.4	*4.0e+2*:17	*6.3e+1*:52	*6.3e-2*:264	15/15
	1(1)	1.1(1)	0.96(0.5)	1.1(0.5)	∞*500*	0/15
$\mathbf{f_{14}}$	*1.6e+1*:3.0	*1.0e+1*:10	*6.3e+0*:15	*2.5e-1*:53	*1.0e-5*:251	15/15
	1.2(2)	0.62(0.6)	0.76(1)	4.9(2)	∞*500*	0/15
$\mathbf{f_{15}}$	*1.6e+2*:3.0	*1.0e+2*:13	*6.3e+1*:24	*4.0e+1*:55	*1.6e+1*:289	5/5
	1.1(2)	0.83(0.6)	1.6(1)	2.2(2)	8.1(9)	3/15
$\mathbf{f_{16}}$	*4.0e+1*:4.8	*2.5e+1*:16	*1.6e+1*:46	*1.0e+1*:120	*4.0e+0*:334	15/15
	1.7(2)	0.77(0.5)	**0.53**(0.5)$^{\downarrow 2}$	**0.42**(0.3)$^{\downarrow 2}$	0.45(0.6)	15/15
$\mathbf{f_{17}}$	*1.0e+1*:5.2	*6.3e+0*:26	*4.0e+0*:57	*2.5e+0*:110	*6.3e-1*:412	15/15
	2.5(4)	1.6(2)	1.9(2)	2.1(2)	2.5(3)	6/15
$\mathbf{f_{18}}$	*6.3e+1*:3.4	*4.0e+1*:7.2	*2.5e+1*:20	*1.6e+1*:58	*1.6e+0*:318	15/15
	1.1(1)	0.85(0.6)	0.97(1.0)	1.1(1)	11(12)	2/15
$\mathbf{f_{19}}$	*1.6e-1*:172	*1.0e-1*:242	*6.3e-2*:675	*4.0e-2*:3078	*2.5e-2*:4946	15/15
	∞	∞	∞	∞	∞*500*	0/15
$\mathbf{f_{20}}$	*6.3e+3*:5.1	*4.0e+3*:8.4	*4.0e+1*:15	*2.5e+0*:69	*1.0e+0*:851	15/15
	0.57(0.2)	**0.44**(0.2)$^{\downarrow}$	0.73(0.3)	4.3(3)	∞*500*	0/15
$\mathbf{f_{21}}$	*4.0e+1*:3.9	*2.5e+1*:11	*1.6e+1*:31	*6.3e+0*:73	*1.6e+0*:347	5/5
	1.8(2)	1.8(2)	1.0(0.9)	1.0(0.9)	1.0(1)	11/15
$\mathbf{f_{22}}$	*6.3e+1*:3.6	*4.0e+1*:15	*2.5e+1*:32	*1.0e+1*:71	*1.6e+0*:341	5/5
	1.8(3)	1.5(1)	1.0(0.8)	0.90(0.8)	1.0(1)	11/15
$\mathbf{f_{23}}$	*1.0e+1*:3.0	*6.3e+0*:9.0	*4.0e+0*:33	*2.5e+0*:84	*1.0e+0*:518	15/15
	1.6(2)	2.9(3)	2.6(3)	3.3(3)	∞*500*	0/15
$\mathbf{f_{24}}$	*6.3e+1*:15	*4.0e+1*:37	*2.5e+1*:118	*1.6e+1*:692	*1.6e+1*:692	15/15
	0.51(0.5)	2.6(3)	7.4(8)	∞	∞*500*	0/15

20-D

#FEs/D	0.5	1.2	3	10	50	#succ
$\mathbf{f_1}$	*6.3e+1*:24	*4.0e+1*:42	*1.0e-8*:43	*1.0e-8*:43	*1.0e-8*:43	15/15
	0.80(0.4)	**0.67**(0.2)$^{\downarrow 4}$	∞	∞	∞*2000*	0/15
$\mathbf{f_2}$	*4.0e+6*:29	*2.5e+6*:42	*1.0e+5*:65	*1.0e+4*:207	*1.0e-8*:412	15/15
	0.54(0.4)	0.70(0.6)	23(25)	143(160)	∞*2000*	0/15
$\mathbf{f_3}$	*6.3e+2*:33	*4.0e+2*:44	*1.6e+2*:109	*1.0e+2*:255	*2.5e+1*:3277	15/15
	0.49(0.5)$^{\downarrow 2}$	2.1(3)	124(144)	114(122)	∞*2000*	0/15
$\mathbf{f_4}$	*6.3e+2*:22	*4.0e+2*:91	*2.5e+2*:250	*1.6e+2*:332	*6.3e+1*:1927	15/15
	6.9(10)	102(110)	∞	∞	∞*2000*	0/15
$\mathbf{f_5}$	*2.5e+2*:19	*1.6e+2*:34	*1.0e-8*:41	*1.0e-8*:41	*1.0e-8*:41	15/15
	0.46(0.2)$^{\downarrow 2}$	**0.33**(0.1)$^{\downarrow 4}$	0.66(0.2)	0.66(0.2)	0.66(0.2)	15/15
$\mathbf{f_6}$	*2.5e+5*:16	*6.3e+4*:43	*1.6e+4*:62	*1.6e+2*:353	*1.6e+1*:1078	15/15
	1.6(1)	1.2(0.9)	1.6(1.0)	2.8(3)	∞*2000*	0/15
$\mathbf{f_7}$	*1.0e+3*:11	*4.0e+2*:39	*2.5e+2*:74	*6.3e+1*:319	*1.0e+1*:1351	15/15
	0.58(0.6)	0.61(0.6)	**0.49**(0.3)$^{\downarrow 2}$	**0.39**(0.3)$^{\downarrow 4}$	0.57(0.3)	15/15
$\mathbf{f_8}$	*4.0e+4*:19	*2.5e+4*:35	*4.0e+3*:67	*2.5e+2*:231	*1.6e+1*:1470	15/15
	1.4(2)	1.5(1)	2.5(0.8)	4.1(3)	∞*2000*	0/15
$\mathbf{f_9}$	*1.0e+2*:357	*6.3e+1*:560	*4.0e+1*:684	*2.5e+1*:756	*1.0e+1*:1716	15/15
	5.0(3)	26(27)	∞	∞	∞*2000*	0/15
$\mathbf{f_{10}}$	*1.6e+6*:15	*1.0e+6*:27	*4.0e+5*:70	*6.3e+4*:231	*4.0e+3*:1015	15/15
	3.7(3)	5.5(5)	6.2(5)	18(17)	∞*2000*	0/15
$\mathbf{f_{11}}$	*4.0e+4*:11	*2.5e+3*:27	*1.6e+2*:313	*1.0e+2*:481	*1.0e+1*:1002	15/15
	0.59(0.5)	0.68(0.6)	2.5(3)	7.3(9)	∞*2000*	0/15
$\mathbf{f_{12}}$	*1.0e+8*:23	*6.3e+7*:39	*2.5e+7*:76	*4.0e+6*:209	*1.0e+1*:1042	15/15
	2.8(4)	38(52)	∞	∞	∞*2000*	0/15
$\mathbf{f_{13}}$	*1.6e+3*:28	*1.0e+3*:64	*6.3e+2*:79	*4.0e+1*:211	*2.5e+0*:1724	15/15
	0.81(0.5)	**0.66**(0.2)$^{\downarrow 4}$	**0.84**(0.1)$^{\downarrow 2}$	1.4(0.4)	∞*2000*	0/15
$\mathbf{f_{14}}$	*2.5e+1*:15	*1.6e+1*:42	*1.0e+1*:75	*1.6e+0*:219	*6.3e-4*:1106	15/15
	2.0(2)	3.3(7)	19(19)	∞	∞*2000*	0/15
$\mathbf{f_{15}}$	*6.3e+2*:15	*4.0e+2*:67	*2.5e+2*:292	*1.6e+2*:846	*1.0e+2*:1671	15/15
	1.1(0.8)	2.9(1)	2.8(4)	∞	∞*2000*	0/15
$\mathbf{f_{16}}$	*4.0e+1*:26	*2.5e+1*:127	*1.6e+1*:540	*1.6e+1*:540	*1.0e+1*:1384	15/15
	2.4(3)	1.5(1)	0.78(0.4)	0.78(0.4)	0.76(0.5)	14/15
$\mathbf{f_{17}}$	*1.6e+1*:11	*1.0e+1*:63	*6.3e+0*:305	*4.0e+0*:468	*1.0e+0*:1030	15/15
	0.52(1)	0.92(1)	15(16)	61(72)	∞*2000*	0/15
$\mathbf{f_{18}}$	*4.0e+1*:116	*2.5e+1*:252	*1.6e+1*:430	*1.0e+1*:621	*4.0e+0*:1090	15/15
	0.31(0.2)$^{\downarrow 2}$	8.3(11)	20(21)	22(26)	∞*2000*	0/15
$\mathbf{f_{19}}$	*1.6e-1*:2.5e5	*1.0e-1*:3.4e5	*6.3e-2*:3.4e5	*4.0e-2*:3.4e5	*2.5e-2*:3.4e5	3/15
	∞	∞	∞	∞	∞*2000*	0/15
$\mathbf{f_{20}}$	*1.6e+4*:38	*1.0e+4*:42	*2.5e+2*:62	*2.5e+0*:250	*1.6e+0*:2536	15/15
	0.25(0.1)$^{\downarrow}$	**0.46**(0.2)$^{\downarrow 4}$	0.90(0.2)	∞	∞*2000*	0/15
$\mathbf{f_{21}}$	*6.3e+1*:36	*4.0e+1*:77	*2.5e+1*:93	*1.6e+1*:456	*4.0e+0*:1094	15/15
	7.5(6)	4.2(3)	5.5(11)	2.7(4)	5.2(6)	4/15
$\mathbf{f_{22}}$	*6.3e+1*:45	*4.0e+1*:68	*2.5e+1*:155	*1.6e+1*:231	*6.3e+0*:1219	15/15
	10(22)	7.3(15)	3.3(6)	4.2(5)	2.0(2)	7/15
$\mathbf{f_{23}}$	*6.3e+0*:29	*4.0e+0*:118	*2.5e+0*:306	*1.6e+0*:534	*1.0e+0*:1614	15/15
	1.6(2)	5.0(8)	46(52)	∞	∞*2000*	0/15
$\mathbf{f_{24}}$	*2.5e+2*:208	*1.6e+2*:918	*1.0e+2*:6628	*6.3e+1*:9885	*4.0e+1*:31629	15/15
	0.65(0.6)	10(10)	∞	∞	∞*2000*	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 (preceded by the target Δf-value in *italics*) in the first. #succ is the number of trials that reached the target value of the last column. 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.

[3] 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.

[4] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proc. of CEC-96*, pages 312–317. Morgan Kaufmann, 1996.

[5] F. Hutter, H. H. Hoos, and K.Leyton-Brown. Parallel algorithm configuration. pages 55–70, 2012.

[6] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Bayesian optimization with censored response data. In *NIPS workshop on Bayesian Optimization, Sequential Experimental Design, and Bandits*, 2011. Published online.

[7] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. pages 507–523, 2011.

[8] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy. An experimental investigation of model-based parameter optimisation: SPO and beyond. pages 271–278, 2009.

[9] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy. Time-bounded sequential parameter optimization. volume 6073, pages 281–298, 2010.

[10] G. A. Jastrebski and D. V. Arnold. Improving Evolution Strategies through Active Covariance Matrix Adaptation. In *IEEE Congress on Evolutionary Computation – CEC 2006*, pages 2814–2821, 2006.

[11] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, October 1993.

[12] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[13] J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimisation*, 2:117–129, 1978. North Holland, Amsterdam.

[14] K. Price. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*, pages 153–157, 1997.

[15] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.