Benchmarking the Pure Random Search on the Bi-objective BBOB-2016 Testbed

Anne Auger, Dimo Brockhoff, Nikolaus Hansen, Dejan Tušar, Tea Tušar, Tobias Wagner

Benchmarking the Pure Random Search on the Bi-objective BBOB-2016 Testbed

Anne Auger*
Dejan Tušar•
*Inria Saclay—Ile-de-France
TAO team, France
LRI, Univ. Paris-Sud
firstname.lastname@inria.fr

Dimo Brockhoff•
Tea Tušar•
•Inria Lille Nord-Europe
DOLPHIN team, France
Univ. Lille, CNRS, UMR 9189 – CRIStAL
firstname.lastname@inria.fr

Nikolaus Hansen*
Tobias Wagner◇
◇TU Dortmund University
Institute of Machining
Technology (ISF), Germany
wagner@isf.de

ABSTRACT

The Comparing Continuous Optimizers platform COCO has become a standard for benchmarking numerical (single-objective) optimization algorithms effortlessly. In 2016, COCO has been extended towards multi-objective optimization by providing a first bi-objective test suite. To provide a baseline, we benchmark a pure random search on this bi-objective `bbob-biobj` test suite of the COCO platform. For each combination of function, dimension n, and instance of the test suite, $10^6 \cdot n$ candidate solutions are sampled uniformly within the sampling box $[-5, 5]^n$.

Keywords

Benchmarking, Black-box optimization, Bi-objective optimization

1. INTRODUCTION

The pure random search, already discussed in the late 1950s by Brooks [4], is the simplest stochastic search algorithm and shall serve as a baseline algorithm in any benchmarking experiment. The algorithm samples each candidate solution independently and uniformly at random within a fixed search domain and returns the best solution found. In the case of numerical optimization $\min_{x \in \mathbb{R}^n} f(x)$ with $f : \mathbb{R}^n \mapsto \mathbb{R}^k$ and thus k objective functions, the search domain is typically a hyperrectangle $\mathcal{H} \subset \mathbb{R}^n$. Here, we assume all variables of equal scaling and the region of interest centered around the origin and choose a hypercube $\mathcal{H} = [-b, b]^n$ with $b \in \mathbb{R}$.

2. ALGORITHM PRESENTATION

For the numerical experiments, we employ the pure random search as implemented in the Matlab/Octave `examplexperiment`, provided by the COCO platform [6]. Algorithm 1 gives the pseudocode.

Algorithm 1 Pseudocode (based on the used Matlab/Octave code) of the pure random search (PRS). The parameters of the algorithm are a benchmark problem `problem` including its dimension n, the lower bounds `lb` $= -b \cdot$ `ones`$(1, n)$ and upper bounds `ub` $= b \cdot$ `ones`$(1, n)$ for each variable with $b \in \mathbb{R}$, and a `budget` in number of function evaluations.

```
1: procedure PRS(problem, n, lb, ub, budget)
2:     while budget > cocoGetEvaluations(problem) do
3:         x = lb + rand(1, n) ∗ (ub − lb);
4:         cocoEvaluateFunction(problem, x);
5:     end while
6: end procedure
```

2.1 Parameter Settings

Being a simple algorithm, only two parameters need to be set for running the pure random search: the upper and lower bounds $-b, b$ of the sampling hypercube and the overall runtime—typically linearly increasing with the search space dimension n.

As for the pure random search on the single-objective `bbob` test suite [2], we set here the sample hypercube to $[-5, 5]^n$ and perform $10^6 \cdot n$ function evaluations—although the region of interest for the `bbob-biobj` test suite has been suggested to be $[-100, 100]^n$ because it cannot be guaranteed that the entire Pareto set lies within the smaller hypercube of $[-5, 5]^n$, even if the single-objective optima are contained in it. The reason for sampling in the smaller hypercube here for the baseline algorithm is the *curse of dimensionality* which results in a much worse performance of the pure random search within $[-100, 100]^n$ than within $[-5, 5]^n$ as can be witnessed in the companion paper [1].

3. CPU TIMING

In order to evaluate the CPU timing of the algorithm, we have run the pure random search within $[-5, 5]^n$, denoted as RS-5, on the entire `bbob-biobj` test suite for $10^3 \cdot n$ function evaluations. The Matlab/Octave code was run with Octave 4.0.0 on a Windows 7 machine with Intel(R) Core(TM) i7-5600U CPU 2.60GHz with 1 processor and 2 cores. The time per function evaluation for dimensions 2, 3, 5, 10, 20, and 40 equaled $6.0 \cdot 10^{-5}$, $5.8 \cdot 10^{-5}$, $5.5 \cdot 10^{-5}$, $5.5 \cdot 10^{-5}$, $5.8 \cdot 10^{-5}$, and $6.8 \cdot 10^{-5}$ seconds respectively.

4. RESULTS AND DISCUSSION

Results of RS-5 from experiments according to [7], [5] and [3] and on the benchmark functions given in [8] are presented in Figures 1, 2, 3, and 4, and in Table 1. The experiments were performed with COCO [6], version 1.0.1, the plots were produced with version 1.1.

Overall, the performance of the pure random search on the `bbob-biobj` test suite is rather weak as expected. In particular for the larger dimensions, many problems cannot be solved to even small target precisions in the allocated budget. In 20-D, for example, 38 of the 55 problems cannot be solved by a single instance to a target precision of 0.1 or less. The function-wise empirical cumulative distribution functions of the algorithm's runtimes in Fig. 1, 2, and 3, show well the scaling with the problem dimension. With only a few exceptions, the runtimes increase with the problem dimension as expected. Exceptions to this rule, indicating most likely that the quality of the Pareto set approximation is not equal over dimension, can be seen on problems 10, 40, 42, and 43 (almost equal performance in 20-D and in 40-D), on problem 12 (3-D curve better than 2-D curve), on problems 18 and 44 (5-D results almost as good as for 3-D for smaller budgets), on problem 48 (almost equal performance between 5-D and 10-D) and for problems 50 and 55 (almost same performance for 10-D, 20-D, and 40-D).

The differences between the dimensions tested are large in terms of solved problems (interpreted as function/dimension/target precision tuples): while in 2-D, 60% or more of the target precisions can be reached within the budget of $10^6 \cdot n$ function evaluations on 26 of the 55 functions, on no function, the algorithm solves more than 20% of the targets in 40-D. The extremes in terms of the number of solved target precisions are function 11 (sep. Ellipsoid/sep. Ellipsoid, for which >80% of the targets are solved in 2-D and almost 20% in 40-D) on the one hand and function 14 (sep. ellipsoid/sharp ridge) and function 46 (Rastrigin/Rastrigin) in 2-D where only about 50% of the targets could be reached within the computational budget and functions 19, 34, and 54 where, in 40-D only a handful of targets were reached (and this already with the first evaluation).

Interestingly, the performance of the algorithm within the first $10^6 \cdot n$ function evaluations shows most often an almost linearly increasing empirical cumulative distribution function of the runtimes to reach all 58 specified targets with fewer differences among the functions than in the single-objective case.

5. CONCLUSIONS

Pure random search is the simplest of all stochastic optimization algorithms. Hence, it should serve as a baseline in all numerical benchmarking exercises as a lower bound on the performance that every reasonable algorithm should achieve. In this paper, we benchmarked the pure random search within the hypercube $[-5, 5]^n$ (with n the problem dimension) as a baseline algorithm on the new bi-objective `bbob-biobj` test suite of the COCO platform.

6. ACKNOWLEDGMENTS

This work was supported by the grant ANR-12-MONU-0009 (NumBBO) of the French National Research Agency.

7. REFERENCES

[1] A. Auger, D. Brockhoff, N. Hansen, D. Tušar, T. Tušar, and T. Wagner. The impact of search volume on the performance of RANDOMSEARCH on the bi-objective BBOB-2016 test suite. In *GECCO (Companion)*, 2016. DOI: 10.1145/2908961.2931709, to appear.

[2] A. Auger and R. Ros. Benchmarking the pure random search on the BBOB-2009 testbed. In F. Rothlauf, editor, *GECCO (Companion)*, pages 2479–2484. ACM, 2009.

[3] D. Brockhoff, T. Tušar, D. Tušar, T. Wagner, N. Hansen, and A. Auger. Biobjective performance assessment with the COCO platform. *ArXiv e-prints*, arXiv:1605.01746, 2016.

[4] S. H. Brooks. A discussion of random methods for seeking maxima. *Operations Research*, 6(2):244–251, 1958.

[5] N. Hansen, A. Auger, D. Brockhoff, D. Tušar, and T. Tušar. COCO: Performance assessment. *ArXiv e-prints*, arXiv:1605.03560, 2016.

[6] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *ArXiv e-prints*, arXiv:1603.08785, 2016.

[7] N. Hansen, T. Tušar, O. Mersmann, A. Auger, and D. Brockhoff. COCO: The experimental procedure. *ArXiv e-prints*, arXiv:1603.08776, 2016.

[8] T. Tušar, D. Brockhoff, N. Hansen, and A. Auger. COCO: The bi-objective black-box optimization benchmarking (bbob-biobj) test suite. *ArXiv e-prints*, arXiv:1604.00359, 2016.

5-D

Δf	1e+0	1e-1	1e-2	1e-3	1e-4	1e-5	#succ
$\mathbf{f_1}$	1	4369(6682)	3.0e6(7e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_2}$	3.0(2)	26338(31254)	3.8e6(5e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_3}$	1	16311(17395)	5.1e6(8e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_4}$	1	3184(4861)	1.4e6(2e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_5}$	1	9353(5847)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_6}$	1	2941(3608)	4.2e6(8e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_7}$	1	4.8e5(6709)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_8}$	3.8(6)	1.5e6(1e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_9}$	2.4(4)	7503(606)	2.4e6(3e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{10}}$	1	29022(27019)	2.3e7(2e7)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{11}}$	1	2301(5514)	4.5e5(1e6)	2.2e6(3e6)	2.3e7(2e7)	∞ *5.0e6*	0/5
$\mathbf{f_{12}}$	1	9589(22148)	2.1e6(3e6)	3.6e6(2e6)	2.1e7(1e7)	∞ *5.0e6*	0/5
$\mathbf{f_{13}}$	2.8(4)	1068(1102)	1.3e5(2e5)	4.4e6(4e6)	∞	∞ *5.0e6*	0/5
$\mathbf{f_{14}}$	1	1.9e5(2e5)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{15}}$	6.4(7)	42729(63166)	8.4e6(1e7)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{16}}$	2.0(1)	1.8e5(3e5)	2.2e7(3e7)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{17}}$	1702(2126)	1.9e6(4e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{18}}$	1	1145(2450)	4.2e5(1e6)	9.7e6(1e7)	∞	∞ *5.0e6*	0/5
$\mathbf{f_{19}}$	1.8(2)	3281(1534)	5.5e5(5e5)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{20}}$	1	5694(6774)	1.6e6(3e6)	2.0e7(3e7)	∞	∞ *5.0e6*	0/5
$\mathbf{f_{21}}$	1	14450(16358)	5.7e6(1e7)	2.1e7(2e7)	∞	∞ *5.0e6*	0/5
$\mathbf{f_{22}}$	1	5793(4162)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{23}}$	1	2888(2777)	3.8e6(6e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{24}}$	1.4(0.5)	1.6e5(1e5)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{25}}$	3.0(2)	1.3e6(3e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{26}}$	3.6(6)	3706(2966)	1.7e6(4e5)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{27}}$	2.0(2)	4181(1072)	1.7e6(2e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{28}}$	1	4415(5346)	7.4e5(9e5)	2.4e7(2e7)	∞	∞ *5.0e6*	0/5
$\mathbf{f_{29}}$	1	1.3e6(1e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{30}}$	2.0(2)	8334(10080)	3.8e6(9e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{31}}$	1	5.1e5(1e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{32}}$	3.8(4)	8.8e5(2e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{33}}$	4.6(4)	1838(3348)	24433(39303)	4.7e6(1e7)	∞	∞ *5.0e6*	0/5
$\mathbf{f_{34}}$	1.6(0.5)	2309(1604)	1.6e6(3e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{35}}$	1	46951(73104)	2.3e7(1e7)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{36}}$	1	58385(47482)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{37}}$	1	4.1e6(9e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{38}}$	1	3.0e6(3e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{39}}$	1	14594(15590)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{40}}$	1	1.4e5(64766)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{41}}$	3.8(4)	9123(5418)	3.6e6(3e6)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{42}}$	3.0(2)	1.9e6(4e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{43}}$	2.2(2)	7.9e5(45617)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{44}}$	1	5159(9974)	2.9e6(4e6)	2.1e7(1e7)	∞	∞ *5.0e6*	0/5
$\mathbf{f_{45}}$	3.8(3)	60621(67165)	2.4e7(2e7)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{46}}$	1	5.5e6(8e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{47}}$	1	2.4e6(5e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{48}}$	5.6(5)	2.2e6(2e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{49}}$	2.0(1)	1.5e6(3e6)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{50}}$	1	3.8e5(7e5)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{51}}$	1.2(0.5)	1.6e5(2e5)	∞	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{52}}$	1.4(0.5)	1.3e5(2e5)	2.1e7(3e7)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{53}}$	20(47)	167(100)	17538(40461)	1.3e6(1e6)	∞	∞ *5.0e6*	0/5
$\mathbf{f_{54}}$	1	4300(2270)	9.9e5(5e5)	∞	∞	∞ *5.0e6*	0/5
$\mathbf{f_{55}}$	1	48997(56444)	2.2e7(2e7)	∞	∞	∞ *5.0e6*	0/5

20-D

Δf	1e+0	1e-1	1e-2	1e-3	1e-4	1e-5	#succ
$\mathbf{f_1}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_2}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_3}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_4}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_5}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_6}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_7}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_8}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_9}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{10}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{11}}$	1	5.7e5(1e6)	8.6e7(2e8)	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{12}}$	1	7.3e6(1e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{13}}$	1	8.3e6(2e6)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{14}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{15}}$	1	3.5e7(3e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{16}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{17}}$	1.8(1)	8.0e7(1e8)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{18}}$	1	1.1e7(2e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{19}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{20}}$	1	8.3e7(1e8)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{21}}$	1	1.9e7(2e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{22}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{23}}$	1.2(0.2)	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{24}}$	1	8.3e7(4e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{25}}$	1.2(0.5)	8.1e7(2e8)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{26}}$	1	3.1e7(9e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{27}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{28}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{29}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{30}}$	1	3.8e7(9e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{31}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{32}}$	613(1526)	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{33}}$	1	3.4e7(3e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{34}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{35}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{36}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{37}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{38}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{39}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{40}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{41}}$	1.2(0.5)	3.7e7(5e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{42}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{43}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{44}}$	2.0(2)	8.1e7(1e8)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{45}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{46}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{47}}$	1.2(0.5)	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{48}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{49}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{50}}$	1.4(1)	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{51}}$	6.0(6)	8.7e7(8e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{52}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{53}}$	1	1.0e7(1e7)	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{54}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5
$\mathbf{f_{55}}$	1	∞	∞	∞	∞	∞ *2.0e7*	0/5

Table 1: Average runtime (aRT) to reach given targets, measured in number of function evaluations. For each function, the aRT and, in braces as dispersion measure, the half difference between 10 and 90%-tile of (bootstrapped) runtimes is shown for the different target Δf-values as shown in the top row. #succ is the number of trials that reached the last target $HV_{\text{ref}} + 10^{-5}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached.

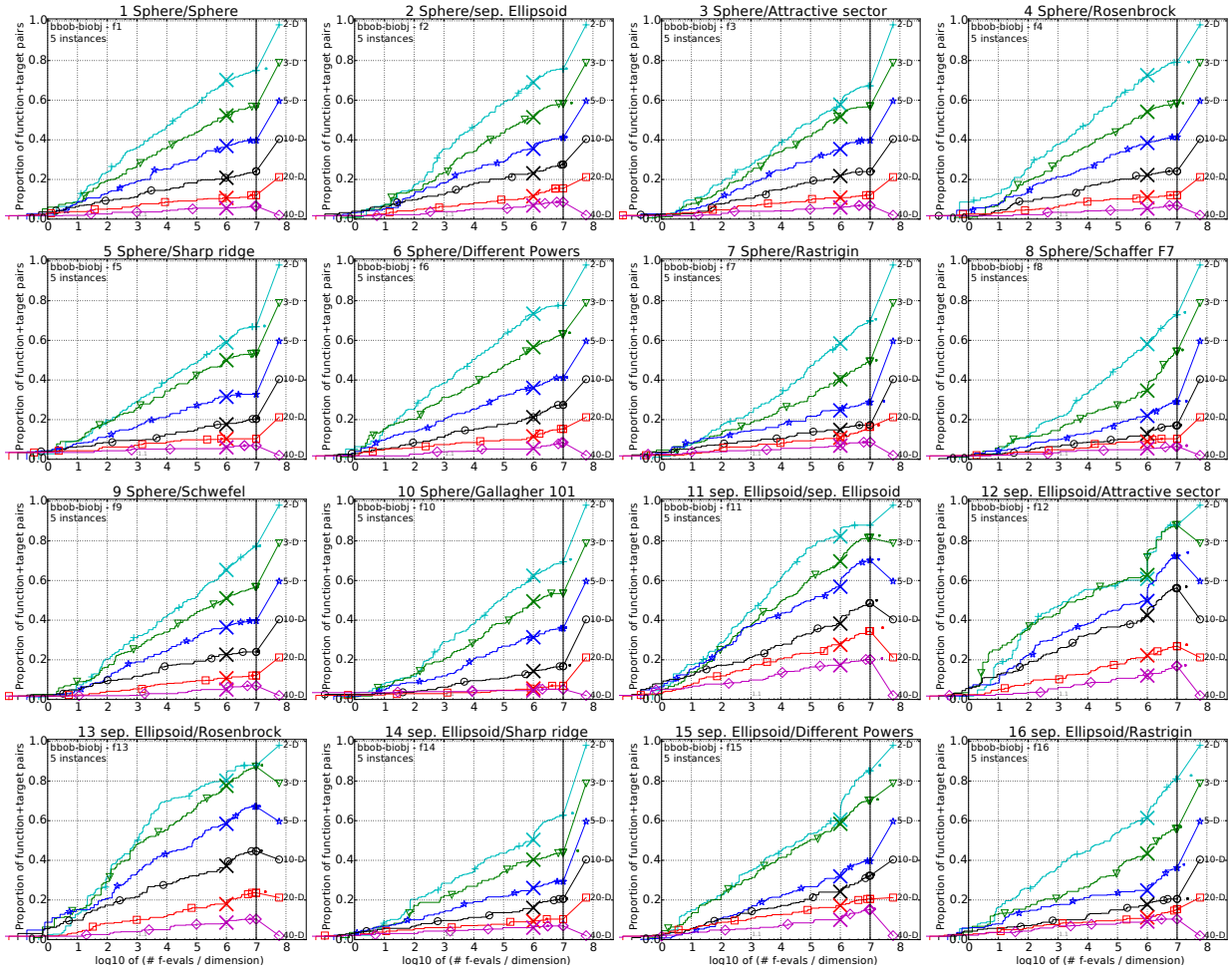

Figure 1: Empirical cumulative distribution of simulated (bootstrapped) runtimes in number of objective function evaluations divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots, 10^{-0.1}, 10^{0}\}$ for functions f_1 to f_{16} and all dimensions.

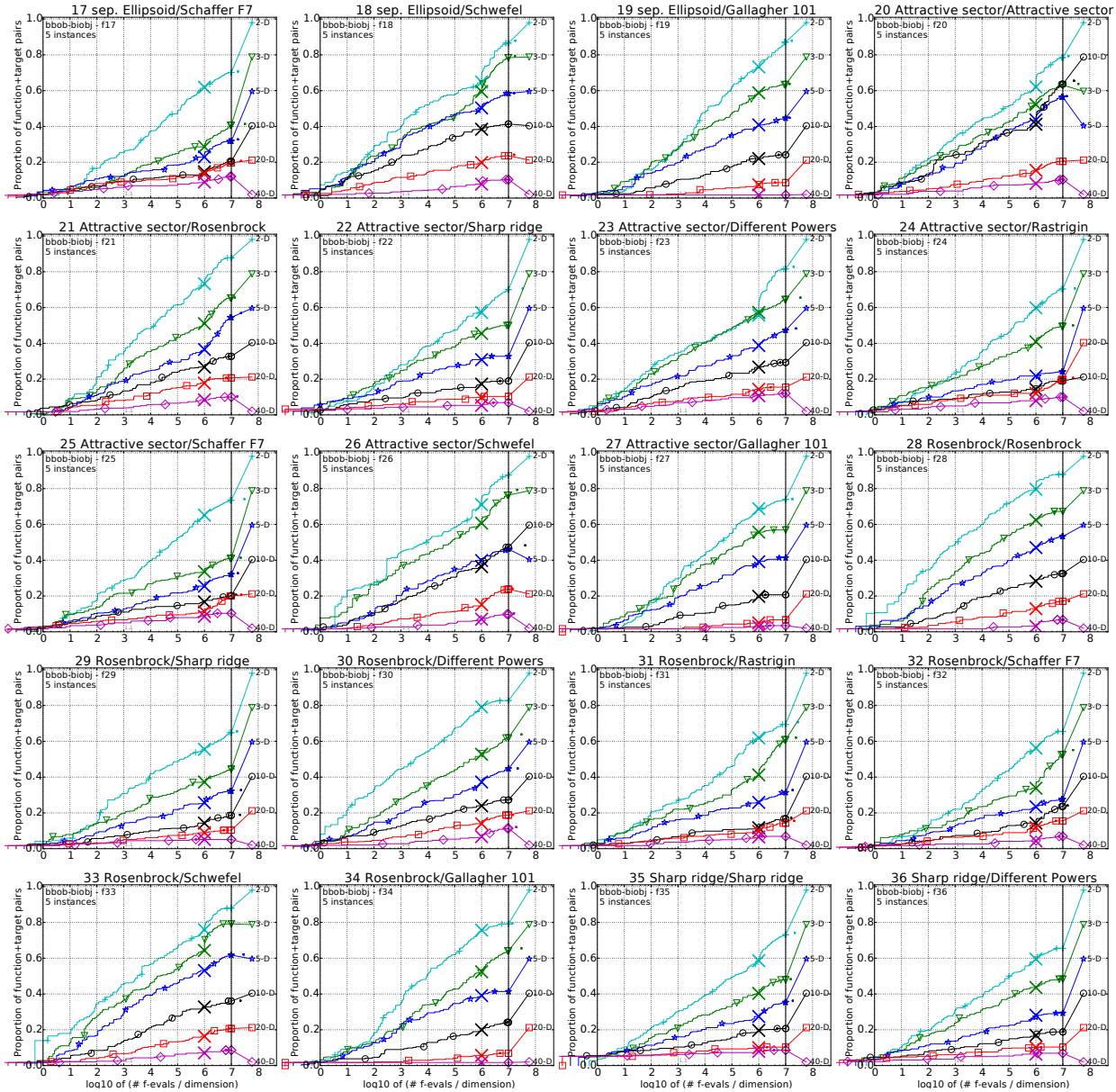

Figure 2: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 1 for functions f_{17} to f_{36} and all dimensions.

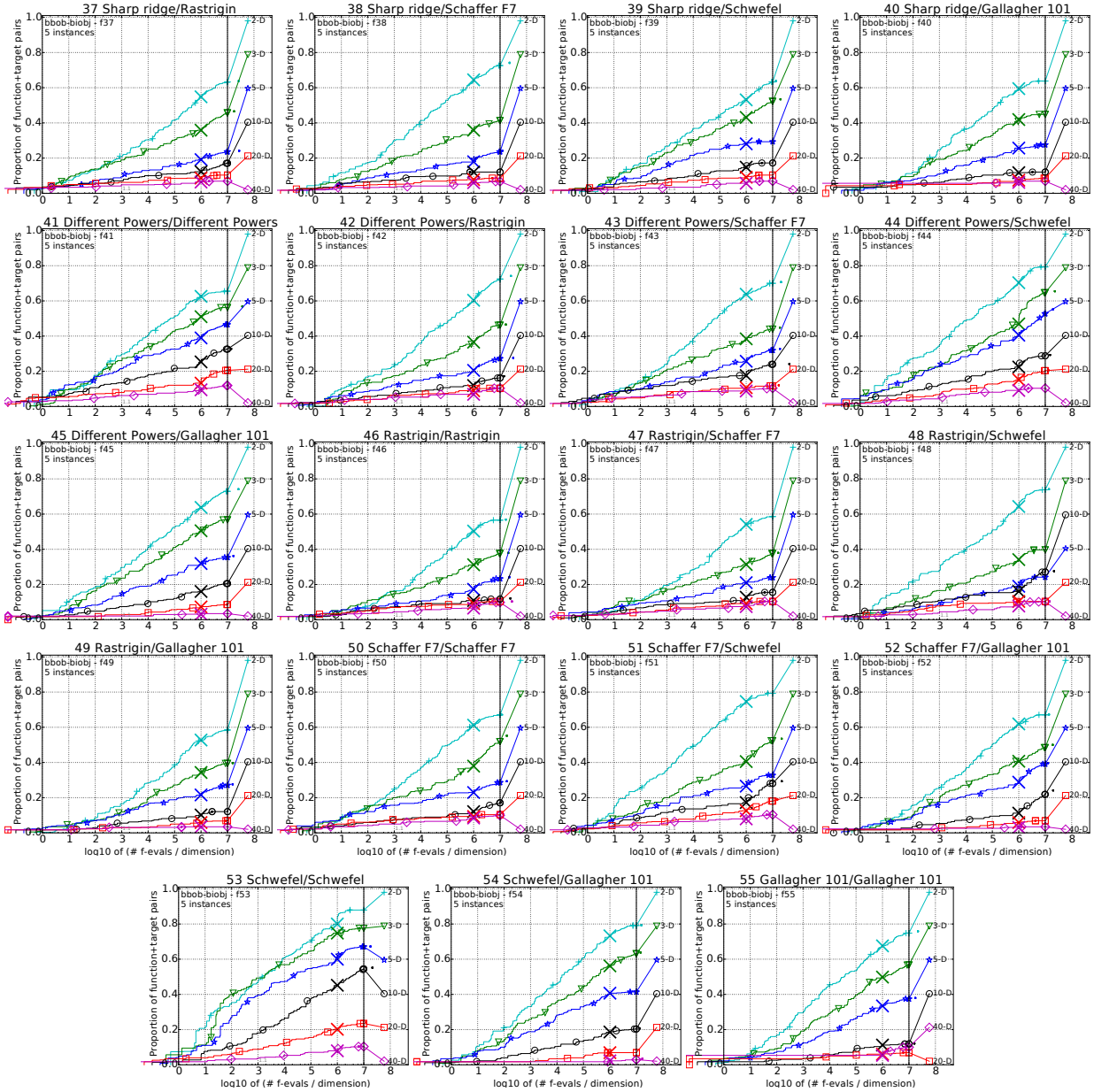

Figure 3: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 1 for functions f_{37} to f_{55} and all dimensions.

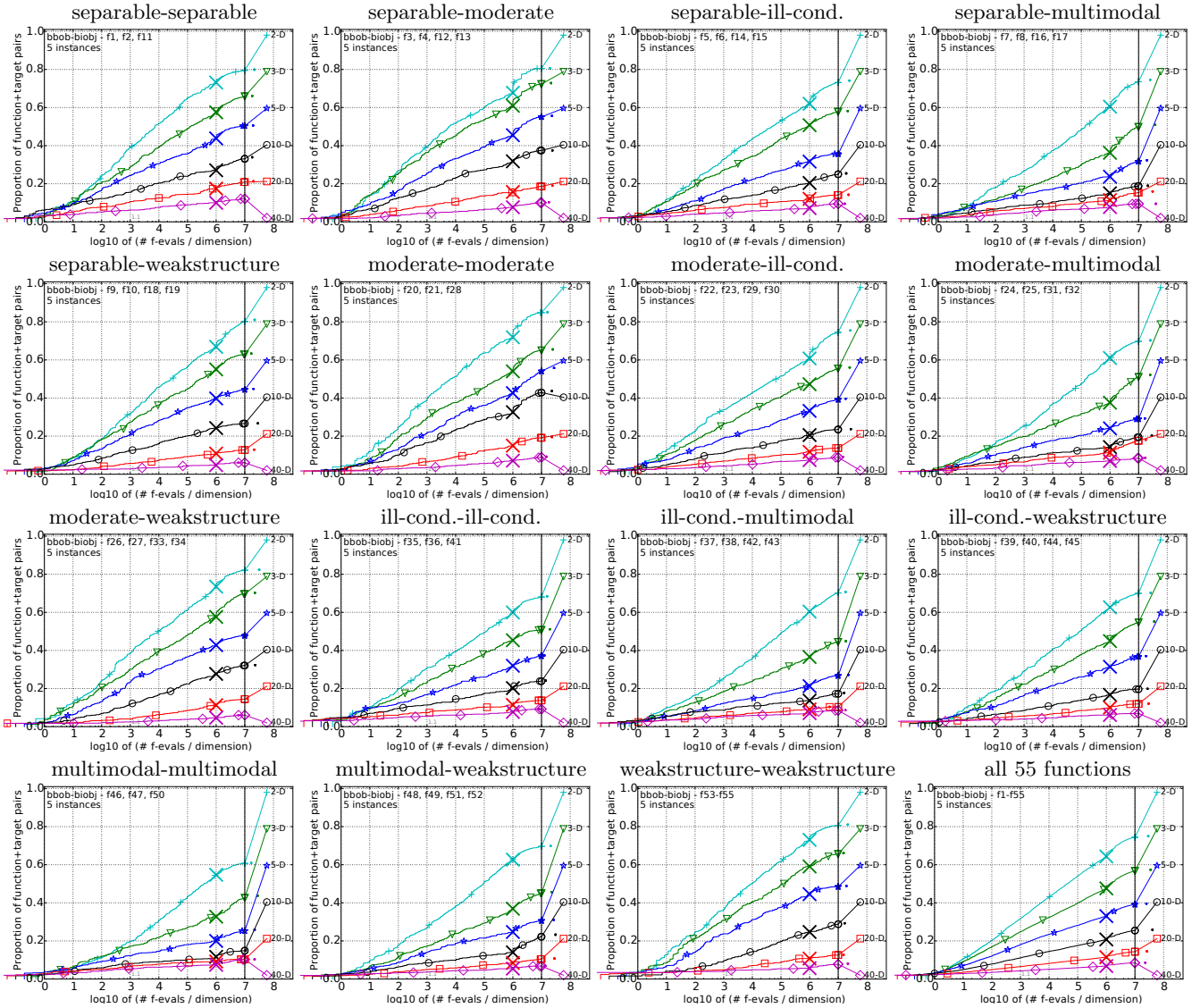

Figure 4: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots, 10^{-0.1}, 10^{0}\}$ for all function groups and all dimensions. The aggregation over all 55 functions is shown in the last plot.