Benchmarking MATLAB's gamultiobj (NSGA-II) on the Bi-objective BBOB-2016 Test Suite

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

Benchmarking MATLAB's gamultiobj (NSGA-II) on the Bi-objective BBOB-2016 Test Suite

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

In this paper, we benchmark a variant of the well-known NSGA-II algorithm of Deb et al. on the biobjective `bbob-biobj` test suite of the Comparing Continuous Optimizers platform COCO. To this end, we employ the implementation of MATLAB's `gamultiobj` toolbox with its default settings and a population size of 100.

Keywords

Benchmarking, Black-box optimization, Bi-objective optimization

1. INTRODUCTION

NSGA-II [2] is arguably the most famous algorithm for multi-objective optimization. It is thus natural to be willing to benchmark it on the bi-objective test suite of the COCO framework [4, 7]. Because private implementations are arguably bug prone, we decided to use a "standard" implementation. We hence used the `gamultiobj` MATLAB implementation that is claimed to use "a controlled elitist genetic algorithm (a variant of NSGA-II)" where "An elitist GA always favors individuals with better fitness value (rank). A controlled elitist GA also favors individuals that can help increase the diversity of the population even if they have a lower fitness value" [6].

Throughout the paper, n denotes the dimension of the search space, such that all bi-objective problems, considered here, are mapping $\mathbb{R}^n$ to $\mathbb{R}^2$.

2. IMPLEMENTATION AND EXPERIMENTAL PROCEDURE

Our MATLAB implementation replaces the `my_optimizer` code within `exampleexperiment.m` of the COCO platform [4] by the function `my_gamultiobj` that is using `gamultiobj` and whose code is presented in Figure 1.

Table 1: Results of CPU timing experiment of MATLAB's `gamultobj` (NSGA-II) in runtime per function evaluation (in 10^{-4} seconds). Population size and number of generations of three variants are given in the first two columns.

		time per function evaluation (in 10^{-4}s)					
popsize	#gen	2-D	3-D	5-D	10-D	20-D	40-D
10	$50n$	2.8	2.7	2.5	2.5	2.7	3.1
50	$10n$	1.5	1.4	1.4	1.4	1.4	1.8
100	$5n$	1.3	1.3	1.3	1.2	1.4	1.7

The `gamultobj` (NSGA-II) algorithm was run with a population size of $N = 100$, as long as the remaining budget allows this N to be used. All other parameters were set according to the default values [6]. The initial population is a mixture of one solution generated according to a normal distribution with mean $\mathbf{0}$ and covariance matrix identity and $N - 1$ solutions generated by uniform sampling in between the smallest and largest value of interest ($l = -100$ and $u = 100$), as provided by the COCO platform for each problem. A tournament selection is used to select two parents for crossover from four random candidates. A ratio of 80% of the solutions is generated by intermediate crossover, whereas the remaining solutions are just copies of elite individuals. These individuals are mutated component per component using a Gaussian mutation with a standard deviation $u - l$ and a crossover probability of 0.01. The budget of $10^5 n$ function evaluations was used to determine the number of generations. No restarts were performed.

3. CPU TIMING

In order to evaluate the CPU timing of the algorithm, we have run the `gamultobj` (NSGA-II) without any restarts on the entire `bbob-biobj` test suite [7] for $500n$ function evaluations. To be more precise, we run the algorithm for three different population sizes to see its impact on the runtime. The MATLAB/Octave code was run under MATLAB 2015a on a Linux Intel(R) Xeon(R) CPU X5650 @ 2.67GHz with 6 cores. Table 1 shows the results. We observe that the time per function evaluation slightly increases with dimension and that a larger population size—and thus less internal computations for the same budget—is more time efficient.

```
function my_gamultiobj(f, lower_bounds, upper_bounds, budget)
    n = length(lower_bounds);
    options = gaoptimset(@gamultiobj);
    options = gaoptimset(options, 'Display', 'off', 'PopulationSize',10, 'Generations',50*n);
    gamultiobj(@(x)cocoEvaluateFunction(f, x), n, [], [], [], [],lower_bounds, upper_bounds, options);
```

Figure 1: MATLAB code of the `my_gamultiobj` function implementing a variant of NSGA-II

4. RESULTS

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

When looking at the results, in particular at the empirical cumulative distribution functions (ECDFs) of the runtimes to reach 58 target precisions, two general observations can be made. With a low budget (until around $100n$ function evaluations), only very few targets can be hit. This scenario coincides with the initial random initialization of the algorithm's population (and a bit beyond) and a closer inspection shows that the graphs are parallel to the ECDFs of a pure random search within this budget range (results not shown here). However, `gamultobj` (NSGA-II) shows a shift towards better performance that can be attributed to the initial search point which is chosen with a smaller variance as the other 99 solutions in the initial population. This initial search point is therefore likely to be produced closer to the origin and, by construction of the problems, potentially closer to the Pareto set.

Once the initial population is filled and selection takes place, the performance starts to improve. The ECDFs displaying the performance for different dimensions thereby show a wide spread and most of the time are degrading with the problem dimension. While `gamultobj` (NSGA-II) can solve 60% of the targets for all functions but the Sharp Ridge/Sharp Ridge function (f35) and in 18 of the 55 functions, even 80% or more of the target precisions can be reached within the given budget in 2-D, the algorithm reaches less than 20% of the target precisions in the given budget in 40-D for 36 of the 55 `bbob-biobj` functions. A few functions show an, at first sight, counterintuitive anomalie against this trend: on f7, f20, and f26, the difficulty of the problem seems not monotonously increasing with the problem dimension. Instead, the ECDFs related to higher dimensions are sometimes above the ECDFs related to lower dimensions. This can be best explained by the fact that the shown performance is relative to the best known approximations of the Pareto set which are used to define the absolute hypervolume reference targets for the performance assessment and which might be of different quality in the different dimensions. Note that the objective functions of the problems for which the non-degrading performance with dimension is the most pronounced (Sphere/Rastrigin (f7), attractive sector/attractive sector (f20), and attractive sector/Schwefel (f26)), are highly multi-modal or asymmetrical (except for the sphere in f7).

5. ACKNOWLEDGMENTS

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

6. REFERENCES

[1] 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.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

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

[4] 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.

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

[6] MathWorks. *gaoptimset – Create genetic algorithm options structure*. MathWorks, 2014a edition, 2014.

[7] 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	1023(128)	3457(2127)	1.3e5(64522)	∞	∞ *5.0e5*	0/5
$\mathbf{f_2}$	171(220)	1124(225)	2656(848)	2.0e5(37483)	∞	∞ *5.0e5*	0/5
$\mathbf{f_3}$	128(318)	2101(635)	13658(14136)	4.9e5(5e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_4}$	1	1005(232)	13526(26236)	97095(77360)	∞	∞ *5.0e5*	0/5
$\mathbf{f_5}$	1	1314(164)	24457(21174)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_6}$	1	1290(685)	7830(6164)	4.5e5(3e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_7}$	1	4286(3693)	3.2e5(3e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_8}$	152(378)	2311(616)	1.1e6(1e6)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_9}$	124(154)	3767(6943)	6832(6627)	81554(27237)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{10}}$	1	1712(249)	85178(84823)	7.4e5(6e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{11}}$	1	632(253)	1681(1616)	54088(1e5)	2.3e6(2e6)	∞ *5.0e5*	0/5
$\mathbf{f_{12}}$	54(66)	1199(1088)	6969(11161)	1.6e5(34444)	7.6e5(1e6)	1.0e6(3e5)	2/5
$\mathbf{f_{13}}$	1	824(179)	7307(7935)	17725(20169)	24586(35596)	5.8e5(1e6)	3/5
$\mathbf{f_{14}}$	1	1698(446)	51925(23083)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{15}}$	287(401)	3750(1654)	12574(20998)	3.1e5(2e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{16}}$	1	1617(840)	9.6e5(1e6)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{17}}$	277(558)	5538(6836)	3.2e5(3e5)	2.2e6(1e6)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{18}}$	1	881(345)	1371(102)	4.5e5(4e5)	2.3e6(4e6)	∞ *5.0e5*	0/5
$\mathbf{f_{19}}$	153(132)	1746(618)	59130(40483)	1.4e5(2e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{20}}$	117	7664(8023)	12988(16638)	2.0e6(2e6)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{21}}$	1	1348(234)	1.3e5(4e5)	2.2e5(2e5)	2.1e6(8e5)	∞ *5.0e5*	0/5
$\mathbf{f_{22}}$	1	1504(466)	37989(40248)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{23}}$	1	1318(396)	7494(4422)	9.6e5(1e6)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{24}}$	105(260)	3609(2818)	1.0e6(2e6)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{25}}$	83	16356(17904)	9.3e5(1e6)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{26}}$	126(156)	2668(2001)	4265(1518)	77754(58608)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{27}}$	4.4	4138(6840)	3.5e5(5e5)	2.3e6(2e6)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{28}}$	1	944(141)	1698(674)	42400(39492)	6.2e5(7e5)	∞ *5.0e5*	0/5
$\mathbf{f_{29}}$	1	1405(171)	40102(49078)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{30}}$	1	4890(8663)	18065(21102)	2.6e5(3e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{31}}$	1	1865(1091)	2.3e5(2e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{32}}$	130	10042(10552)	1.1e6(2e6)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{33}}$	124(306)	847(148)	1054(136)	4352(3190)	2.4e5(3e5)	∞ *5.0e5*	0/5
$\mathbf{f_{34}}$	1	2817(3377)	1.3e5(48830)	7.2e5(5e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{35}}$	1	1343(470)	1.0e5(1e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{36}}$	1	1431(296)	1.4e5(5e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{37}}$	1	4898(4619)	5.8e5(5e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{38}}$	1	10888(10367)	2.2e6(4e6)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{39}}$	1	1494(484)	21866(13051)	1.1e6(4e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{40}}$	1	2382(872)	32741(15023)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{41}}$	110(272)	1454(972)	4646(2865)	4.3e5(5e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{42}}$	122(152)	3976(3110)	1.0e6(7e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{43}}$	152(376)	7363(733)	4.6e5(3e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{44}}$	1	914(123)	1586(358)	80902(1e5)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{45}}$	1	1641(340)	52093(21540)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{46}}$	1	44412(44167)	∞	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{47}}$	1	47106(1e5)	2.1e6(2e6)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{48}}$	1	48161(1e5)	8.2e5(2e6)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{49}}$	1	48112(2179)	1.1e6(9e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{50}}$	1	8103(14312)	8.8e5(2e6)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{51}}$	120(148)	4122(6372)	2.5e5(6e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{52}}$	1	3996(2984)	3.4e5(3e5)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{53}}$	256(458)	867(118)	1561(1154)	1.3e5(3e5)	1.4e5(1e5)	2.4e6(4e6)	1/5
$\mathbf{f_{54}}$	1	1221(165)	49247(56700)	3.7e5(4e5)	2.4e6(2e6)	∞ *5.0e5*	0/5
$\mathbf{f_{55}}$	1	9836(17764)	1.5e5(1e5)	9.6e5(1e6)	∞	∞ *5.0e5*	0/5

20-D

Δf	1e+0	1e-1	1e-2	1e-3	1e-4	1e-5	#succ
$\mathbf{f_1}$	1	99267(1e5)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_2}$	1	1.4e5(2e5)	8.4e6(9e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_3}$	1	1.6e5(2e5)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_4}$	1	1.0e5(59908)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_5}$	1	1.5e5(80869)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_6}$	1	65759(9606)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_7}$	1	8.1e6(7e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_8}$	1	9.2e6(7e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_9}$	1	40038(30460)	3.1e6(5e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{10}}$	1	1.0e6(1e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{11}}$	1	5535(5265)	1.1e6(2e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{12}}$	1	5169(7392)	96953(28656)	8.9e6(1e7)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{13}}$	1	18098(20263)	1.5e6(1e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{14}}$	1	9.6e5(1e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{15}}$	1	19578(11590)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{16}}$	1	3.6e6(4e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{17}}$	123(305)	8.0e6(1e7)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{18}}$	1	3325(1811)	90659(61807)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{19}}$	122(302)	1.1e6(3e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{20}}$	1	2064(224)	8.6e5(7e5)	4.6e6(4e6)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{21}}$	1	6933(6128)	3.4e6(3e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{22}}$	1	1.6e6(3e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{23}}$	1	67372(91698)	9.4e6(6e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{24}}$	1	8.0e6(1e7)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{25}}$	1	3.2e6(3e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{26}}$	1	2018(443)	52116(46974)	1.8e6(4e6)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{27}}$	1	3.4e6(3e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{28}}$	1	18889(14934)	1.2e6(2e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{29}}$	1	2.8e5(2e5)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{30}}$	1	88480(2e5)	9.4e6(5e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{31}}$	1	8.2e6(9e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{32}}$	1	3.1e6(2e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{33}}$	1	22972(17846)	6.0e5(1e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{34}}$	1	9.4e5(1e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{35}}$	1	9.1e5(1e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{36}}$	1	2.1e6(2e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{37}}$	1	∞	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{38}}$	1	∞	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{39}}$	1	1.7e6(3e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{40}}$	1	3.8e6(5e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{41}}$	1	2.1e5(4e5)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{42}}$	1	∞	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{43}}$	1	9.7e6(8e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{44}}$	1	73988(9778)	3.3e6(7e6)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{45}}$	1	3.9e6(4e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{46}}$	1	∞	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{47}}$	1	∞	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{48}}$	1	8.5e6(1e7)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{49}}$	1	∞	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{50}}$	1	∞	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{51}}$	1	3.2e6(5e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{52}}$	1	∞	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{53}}$	1	4868(8452)	29104(16048)	2.8e5(2e5)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{54}}$	1	1.1e6(2e6)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{55}}$	1	8.2e6(6e6)	∞	∞	∞	∞ *2.0e6*	0/5

Table 2: 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.

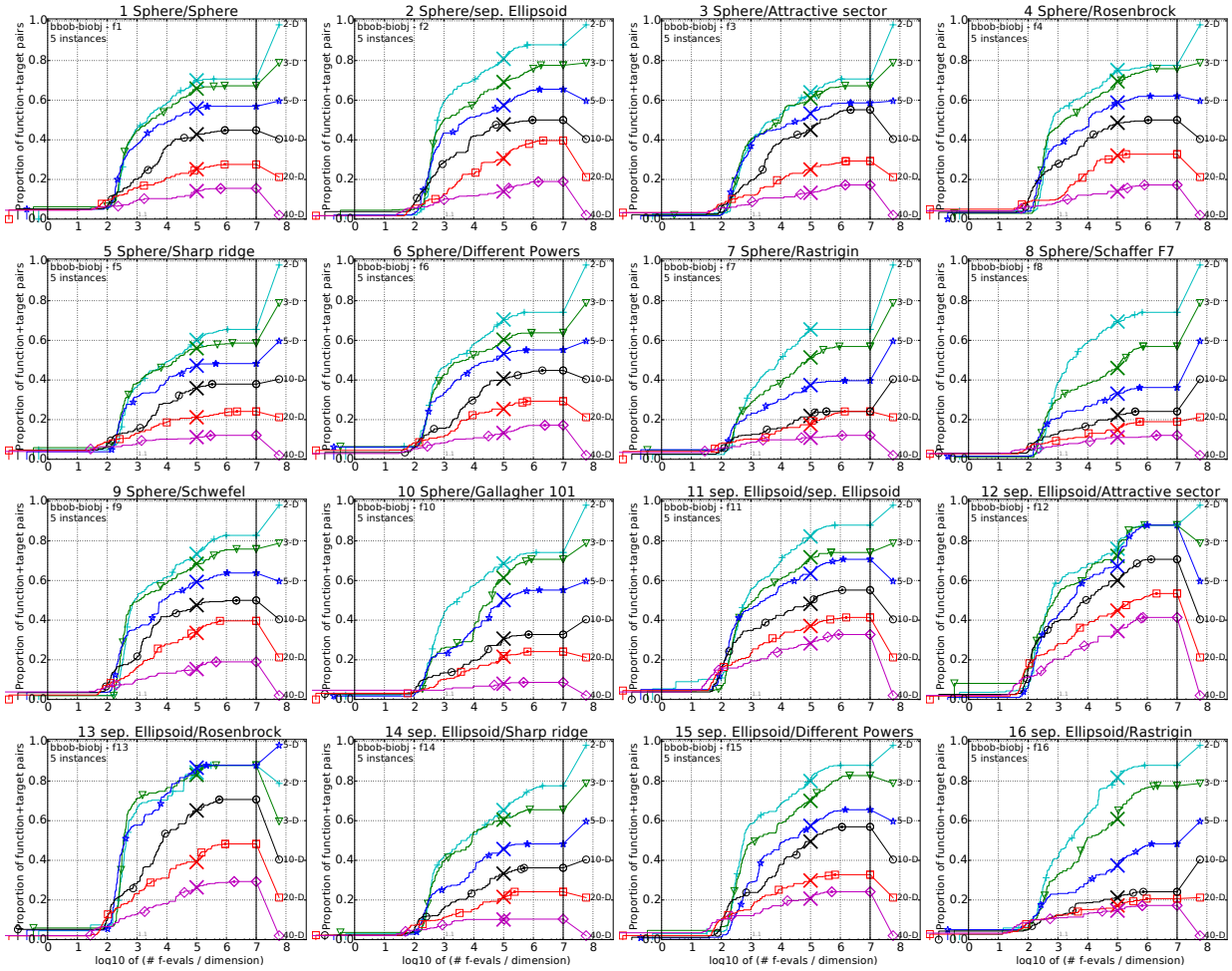

Figure 2: Empirical cumulative distribution of simulated (bootstrapped) runtimes in number of objective function evaluations divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots, 10^{-0.1}, 10^{0}\}$ for functions f_1 to f_{16} and all dimensions.

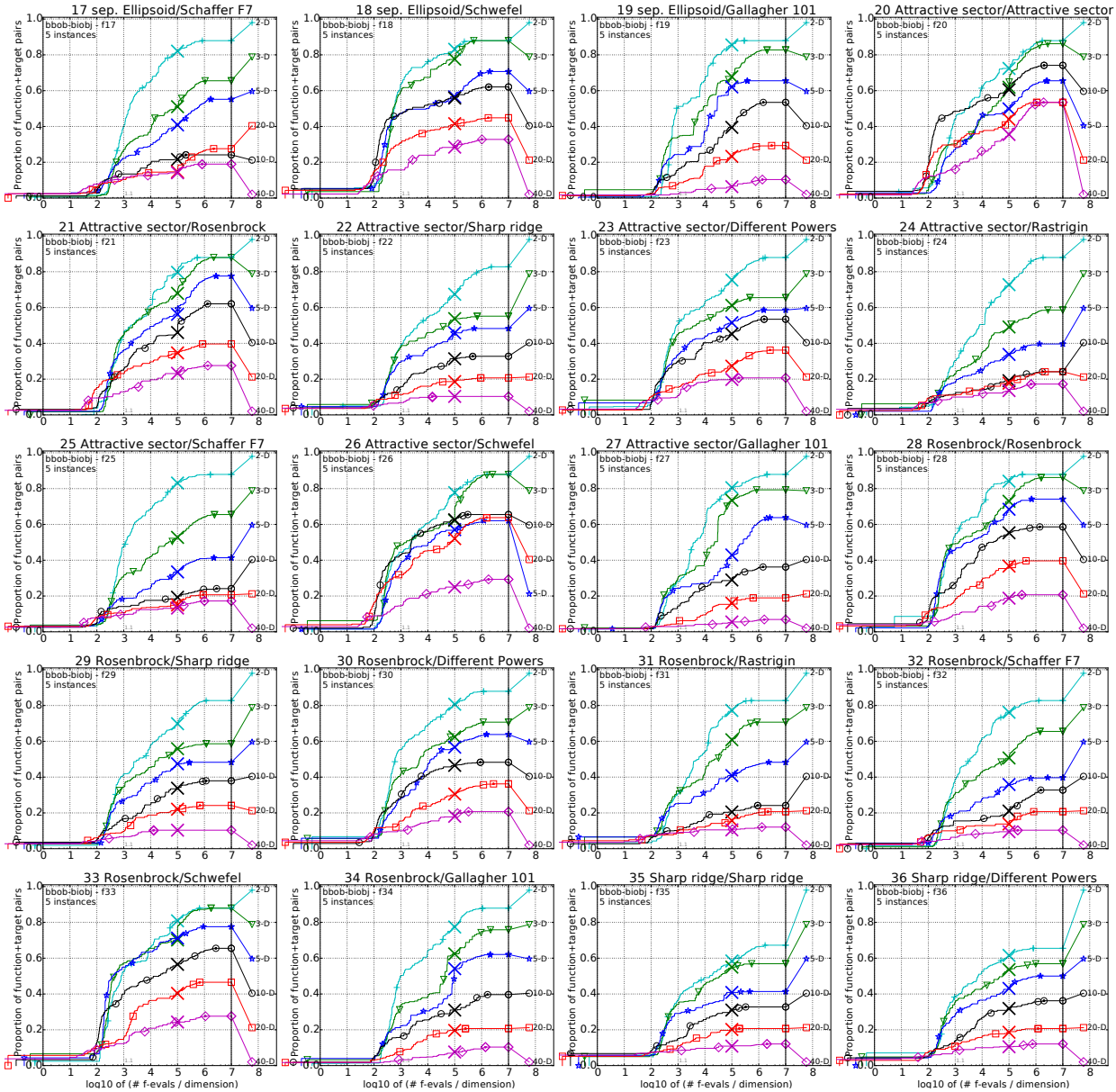

Figure 3: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 2 for functions f_{17} to f_{36} and all dimensions.

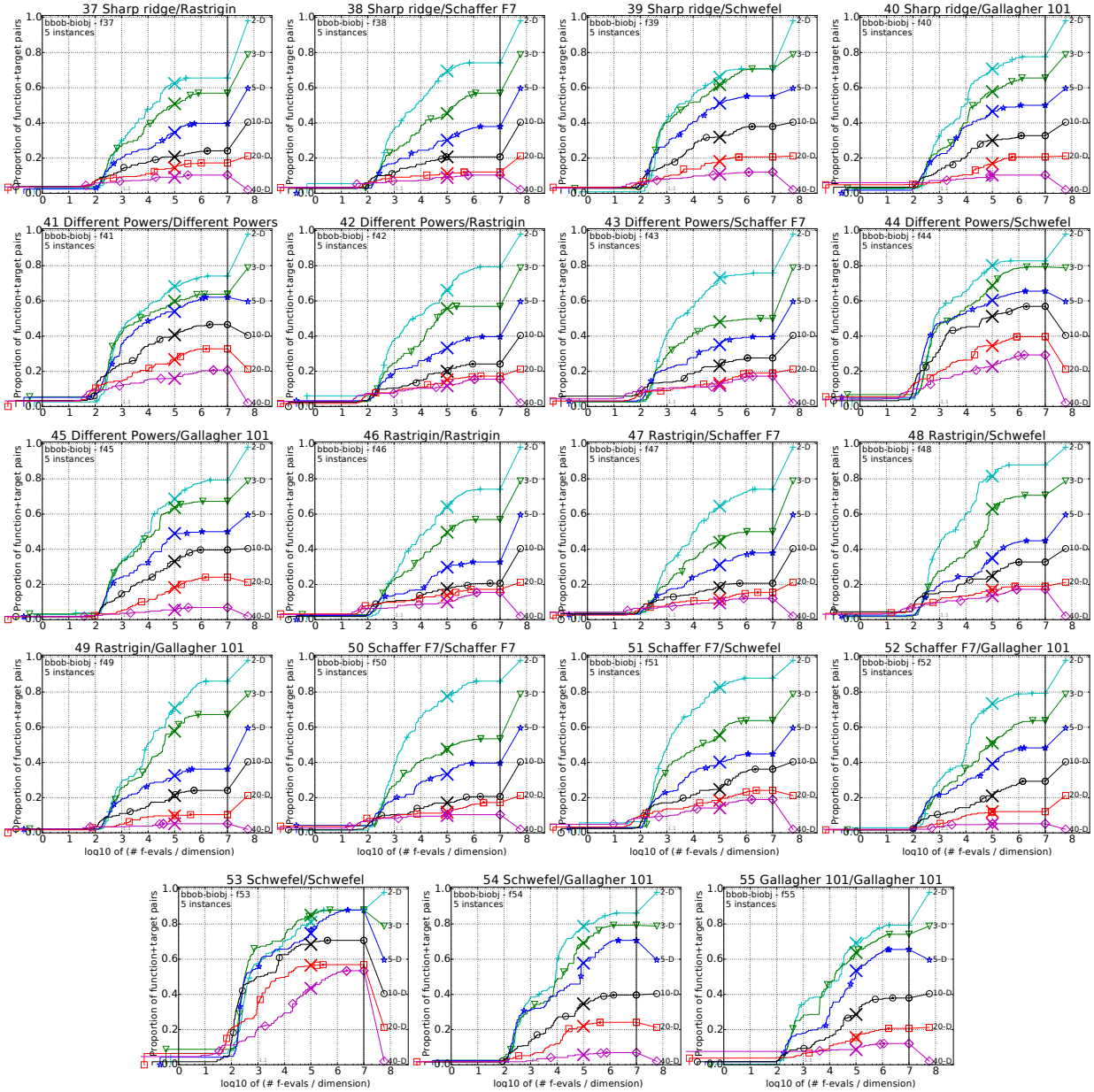

Figure 4: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 2 for functions f_{37} to f_{55} and all dimensions.

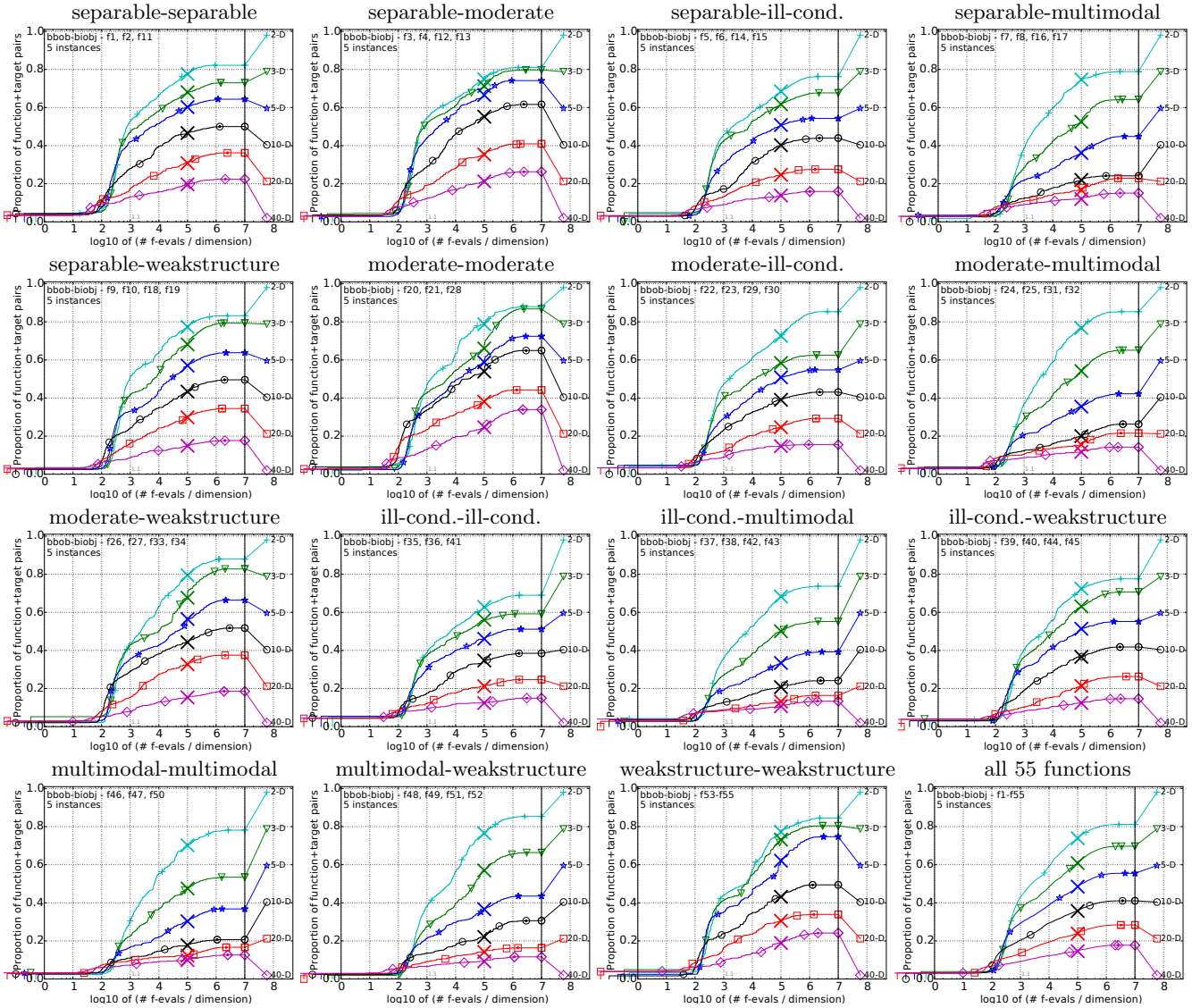

Figure 5: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots,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