Benchmarking RM-MEDA on the Bi-objective BBOB-2016 Test Suite

Anne Auger[⋆] Dimo Brockhoff[•] Nikolaus Hansen[⋆]
Dejan Tušar[•] Tea Tušar[•] Tobias Wagner[⋄]

[⋆]Inria Saclay—Ile-de-France
TAO team, France
LRI, Univ. Paris-Sud
firstname.lastname@inria.fr

[•]Inria Lille Nord-Europe
DOLPHIN team, France
Univ. Lille, CNRS, UMR 9189 – CRIStAL
firstname.lastname@inria.fr

[⋄]TU Dortmund University
Institute of Machining
Technology (ISF), Germany
wagner@isf.de

ABSTRACT

In this paper, we benchmark the Regularity Model-Based Multiobjective Estimation of Distribution Algorithm (`RM-MEDA`) of Zhang et al. on the bi-objective `bbob-biobj` test suite of the Comparing Continuous Optimizers (COCO) platform. It turns out that, starting from about 200 times dimension many function evaluations, `RM-MEDA` shows a linear increase in the solved hypervolume-based target values with time until a stagnation of the performance occurs rather quickly on all problems. The final percentage of solved hypervolume targets seems to decrease with the problem dimension.

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

Keywords

Benchmarking, Black-box optimization, Bi-objective optimization

1. INTRODUCTION

Multi-objective optimization differs from single-objective optimization most importantly in the type of the desired approximation. In the single-objective case, a single solution with a function value as small as possible is sought. In the multi-objective case, one is interested in finding an approximation of the *set* of Pareto-optimal solutions. The quality of this approximation is given explicitly or implicitly by an indicator function, such as the hypervolume indicator [1].

Despite this difference of aiming to converge either to a set or to a single solution, only few efforts have been pursued to specifically develop variation or solution generation approaches for the multi-objective case. In most cases, still the established variation operators from single-objective optimization are used. One proposal of such a specific variation method is the Regularity Model-Based Multiobjective Estimation of Distribution Algorithm (`RM-MEDA`) by Zhang et al. We will benchmark this approach on the bi-objective `bbob-biobj` test suite [6] of the Comparing Continuous Optimizers platform COCO [4].

Throughout the paper, n will denote the problem dimension.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GECCO'16 Companion, July 20 - 24, 2016, Denver, CO, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4323-7/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908961.2931707

2. ALGORITHM

The main idea behind `RM-MEDA` is to approximate the Pareto set of a multi-objective problem by an $(n-1)$-dimensional manifold represented by a piecewise linear model. This model is used to sample candidate solutions in the vicinity of the current approximation, where the selected solutions are in turn used to improve the sampling model. The number of segments or clusters is a parameter of the algorithm and the respective submodels are learned through the application of a linear principle component analysis (LPCA). The sampling of the piecewise linear model is performed by randomly picking a segment of the model. The probability of picking a segment is relative to the segments' volume[1]. On the segment, a point is chosen uniformly at random and a random perturbation in terms of an isotropic n-dimensional normal distribution is added. The variance or step size of the perturbation depends on the variation of the solutions of the current population assigned to the chosen segment.

For a more detailed algorithm description of `RM-MEDA`, we refer the interested reader to the original publication [7].

2.1 Parameter Settings

The MATLAB implementation of `RM-MEDA`[2] was run with the default parameters [7]. The code was slightly adjusted to cope with the assumptions regarding the vector and variable representation mady by the COCO platform. A population size $N = 100$ was set. The generator of the distribution for sampling new solutions was based on a linear principal component analysis (LPCA) with 5 clusters, 50 training steps, and an extension rate of 0.25. The budget of $10^5 n$ was used

[1]In the bi-objective case, the lengths of the segments are used.

[2]http://cswww.essex.ac.uk/staff/qzhang/code/RM-MEDA-Matlab v0.1.zip

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 `RM-MEDA` with restarts on the entire `bbob-biobj` test suite for $100n$ function evaluations. The COCO 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 $4.7 \cdot 10^{-4}$, $5.0 \cdot 10^{-4}$, $5.2 \cdot 10^{-4}$, $5.8 \cdot 10^{-4}$, $7.0 \cdot 10^{-4}$, and $8.7 \cdot 10^{-4}$ seconds respectively.

4. RESULTS

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

On almost all problems, three phases can be distinguished in the ECDF plots of Figures 1, 2, and 3. The first phase of initialization and learning of the piecewise representation takes about 100..200n function evaluations. In this phase, none or almost none of the target precisions are reached. It coincides with the performance of a pure random search within the region of interest $[-100, 100]^n$ (comparison results with pure random search not shown here). The second phase shows a linear convergence in which the performance displayed in the ECDFs does not differ much for changing dimension (ECDFs for different dimensions are almost parallel). In this phase, the algorithm successfully exploits the sampling model. The last phase is characterized by a stagnation of the approximation quality. The sampling model cannot further be refined and the number of targets achieved remains constant. The actual percentages of solved problems at the end of the runs depend on the dimension of the problems.

For the linear convergence phase, on some functions, the algorithm seems to be quicker for smaller problem dimensions (on the four functions 11, 12, 16, and 23) while on most, the algorithm shows an increased performance with higher dimension (on the 35 functions 1–10, 17, 21, 26, 28–30, 32–40, 44–50, 52, 54, 55, and on almost all function groups). This observation, however, depends on the relative quality of the reference sets and the corresponding hypervolume reference values underlying the performance assessment of COCO. This limitation must be taken into account before making more general statements.

This also holds for the performance of `RM-MEDA` in the last phase, but to a smaller degree: it is expected that algorithm performance is decreasing when the problem dimension increases as this is the case also for `RM-MEDA` on all function groups and most functions. Exceptions are f26 (Attractive sector/Schwefel), and functions for which pairs of dimensions show similar performance. Here, it is most likely that the reference sets have a larger impact on the display than the actual effect of the dimension.

5. CONCLUSION

After a short initialization phase, the Regularity Model-Based Multiobjective Estimation of Distribution Algorithm (`RM-MEDA`) of Zhang et al. showed a linear increase in the solved hypervolume-based target values on the bi-objective `bbob-biobj` test suite of the Comparing Continuous Optimizers (COCO) platform. However, after some time, a stagnation of the performance occurred. The final percentage of solved hypervolume targets seems to decrease with the problem dimension.

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, J. Bader, D. Brockhoff, and E. Zitzler. Hypervolume-based Multiobjective Optimization: Theoretical Foundations and Practical Implications. *Theoretical Computer Science*, 425:75–103, 2012.

[2] 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.

[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] 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.

[7] Q. Zhang, A. Zhou, and Y. Jin. RM-MEDA: A regularity model based multiobjective estimation of distribution algorithm. *IEEE Transactions on Evolutionary Computation*, 12(1):41–63, 2008.

5-D

Δf	1e+0	1e-1	1e-2	1e-3	1e-4	1e-5	#succ
$\mathbf{f_{1}}$	419(312)	1873(189)	2740(212)	6019(217)	2.1e5(26314)	∞ *5.0e5*	0/5
$\mathbf{f_{2}}$	220(241)	3869(1299)	5774(1815)	12860(6950)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{3}}$	217(512)	2423(968)	3517(1569)	8617(4606)	4.5e5(*3e5*)	∞ *5.0e5*	0/5
$\mathbf{f_{4}}$	628(182)	2109(104)	3053(248)	5212(898)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{5}}$	3.8(4)	2197(126)	3792(914)	1.6e5(47390)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{6}}$	685(414)	2171(208)	3071(238)	6572(1218)	4.5e5(*3e5*)	∞ *5.0e5*	0/5
$\mathbf{f_{7}}$	359(305)	2644(662)	2.2e5(*3e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{8}}$	1108(121)	2867(852)	3.0e5(*2e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{9}}$	823(426)	1977(293)	2857(270)	4619(423)	8.2e5(*1e6*)	∞ *5.0e5*	0/5
$\mathbf{f_{10}}$	628(134)	2519(428)	1.3e5(*4e5*)	3.4e5(*1e6*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{11}}$	1.6(0.8)	7235(11517)	23168(20566)	1.6e5(*3e5*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{12}}$	91(111)	3371(2847)	6203(5462)	1.5e5(*4e5*)	7.6e5(*1e6*)	∞ *5.0e5*	0/5
$\mathbf{f_{13}}$	570(648)	2424(698)	3442(951)	6185(2855)	13040(7560)	4.4e5(*8e5*)	3/5
$\mathbf{f_{14}}$	10(10)	4477(541)	8068(1219)	1.6e5(*1e5*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{15}}$	933(1532)	4619(797)	7342(3738)	1.4e5(3512)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{16}}$	131(230)	5017(2023)	4.5e5(*1e6*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{17}}$	1488(206)	7646(4549)	44218(23987)	2.4e6(*3e6*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{18}}$	43(52)	3118(690)	5255(1988)	13463(5350)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{19}}$	321(268)	4560(999)	7976(1776)	7.6e5(*2e6*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{20}}$	150(370)	3192(1956)	5549(2672)	30923(27578)	5.8e5(*7e5*)	2.0e6(*4e6*)	1/5
$\mathbf{f_{21}}$	355(400)	2841(958)	1.3e5(3517)	1.3e5(*1e5*)	2.0e6(*2e6*)	∞ *5.0e5*	0/5
$\mathbf{f_{22}}$	6.2(12)	2409(394)	4674(1633)	15668(8626)	2.0e6(*4e6*)	∞ *5.0e5*	0/5
$\mathbf{f_{23}}$	13(18)	2423(474)	3702(578)	11018(9460)	9.0e5(*7e5*)	∞ *5.0e5*	0/5
$\mathbf{f_{24}}$	197(383)	4098(1030)	3.3e5(*5e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{25}}$	318(457)	3439(1448)	98545(*1e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{26}}$	468(408)	2376(706)	3353(822)	1.3e5(*3e5*)	9.4e5(*1e6*)	∞ *5.0e5*	0/5
$\mathbf{f_{27}}$	17(9)	2304(156)	7.5e5(*1e6*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{28}}$	378(296)	1989(268)	2979(458)	4519(1206)	3.8e5(*4e5*)	∞ *5.0e5*	0/5
$\mathbf{f_{29}}$	318(307)	2658(416)	5387(1641)	37993(10668)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{30}}$	974(390)	2210(366)	3360(370)	6456(1502)	2.3e6(*3e6*)	∞ *5.0e5*	0/5
$\mathbf{f_{31}}$	703(198)	3271(1177)	2.1e5(*2e5*)	2.3e6(*2e6*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{32}}$	1164(324)	2963(477)	1.8e5(*3e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{33}}$	842(593)	1775(437)	2236(431)	3810(1714)	7.6e5(*1e6*)	∞ *5.0e5*	0/5
$\mathbf{f_{34}}$	800(371)	2523(604)	7.5e5(*1e6*)	2.0e6(*9e5*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{35}}$	1	2489(356)	5261(1218)	2.3e5(*3e5*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{36}}$	115(50)	2648(281)	8546(4864)	2.0e6(*3e6*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{37}}$	86(114)	4173(1593)	3.1e5(*4e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{38}}$	641(294)	4459(2067)	9.9e5(*1e6*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{39}}$	132(36)	2361(206)	1.3e5(*1e5*)	1.4e5(*1e5*)	2.1e6(*4e6*)	∞ *5.0e5*	0/5
$\mathbf{f_{40}}$	141(97)	2757(320)	6782(4396)	2.1e6(*2e6*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{41}}$	842(315)	2566(362)	3822(824)	8131(2770)	1.0e6(*5e5*)	∞ *5.0e5*	0/5
$\mathbf{f_{42}}$	1044(294)	4269(2458)	9.3e5(*8e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{43}}$	1205(754)	3323(789)	1.7e5(*2e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{44}}$	1135(473)	2411(504)	3301(151)	4948(1096)	2.3e5(75384)	∞ *5.0e5*	0/5
$\mathbf{f_{45}}$	831(518)	2973(253)	6856(3190)	7.8e5(*1e6*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{46}}$	616(392)	13388(11192)	2.2e6(*2e6*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{47}}$	914(320)	10020(6517)	2.1e6(*2e6*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{48}}$	1127(156)	19081(37880)	3.7e5(*5e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{49}}$	743(374)	3812(500)	5.1e5(*2e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{50}}$	1066(262)	2933(826)	3.0e5(*3e5*)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{51}}$	967(197)	2500(831)	34425(32526)	∞	∞	∞ *5.0e5*	0/5
$\mathbf{f_{52}}$	887(179)	2723(881)	3.6e5(*1e6*)	2.3e6(*3e6*)	∞	∞ *5.0e5*	0/5
$\mathbf{f_{53}}$	786(865)	1769(45)	2141(118)	5153(6247)	1.4e5(372)	∞ *5.0e5*	0/5
$\mathbf{f_{54}}$	860(238)	2374(357)	1.3e5(*3e5*)	8.2e5(*1e6*)	1.2e6(*7e5*)	∞ *5.0e5*	0/5
$\mathbf{f_{55}}$	750(378)	2996(218)	3.4e5(*5e5*)	7.6e5(*2e6*)	8.4e5(*1e6*)	∞ *5.0e5*	0/5

20-D

Δf	1e+0	1e-1	1e-2	1e-3	1e-4	1e-5	#succ
$\mathbf{f_{1}}$	612(132)	4067(57)	6108(55)	44585(6122)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{2}}$	935(490)	6755(776)	10689(1774)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{3}}$	135(106)	5204(217)	7434(69)	57154(39353)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{4}}$	1677(207)	4239(142)	6154(242)	70895(*1e5*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{5}}$	1	4536(167)	8658(723)	4.0e5(*2e5*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{6}}$	1473(628)	5059(316)	8564(1133)	31063(7742)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{7}}$	743(400)	22951(2170)	8.6e6(*1e7*)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{8}}$	2059(226)	11729(8618)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{9}}$	1762(376)	4166(116)	6100(102)	11855(527)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{10}}$	1046(194)	5553(531)	1.3e6(*1e6*)	8.1e6(*9e6*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{11}}$	1	23952(9283)	6.4e5(*5e5*)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{12}}$	5.6(6)	11321(4858)	7.7e5(*1e6*)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{13}}$	62(76)	5684(988)	8909(1695)	3.5e6(*6e6*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{14}}$	13(15)	8927(1539)	26707(5312)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{15}}$	1657(1094)	14221(4316)	2.5e5(*5e5*)	8.7e6(*8e6*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{16}}$	287(88)	66577(37917)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{17}}$	1861(306)	46214(69280)	8.0e6(*5e6*)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{18}}$	1	7868(1739)	13156(3840)	2.1e6(*2e6*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{19}}$	1281(465)	5.1e5(*3e6*)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{20}}$	80(58)	7645(3776)	17398(10138)	2.0e5(*2e5*)	2.2e6(*3e6*)	∞ *2.0e6*	0/5
$\mathbf{f_{21}}$	490(221)	6430(1936)	11300(5847)	1.3e5(*2e5*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{22}}$	1	8823(2016)	24214(10872)	8.2e5(*2e6*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{23}}$	532(178)	30357(23130)	55152(33387)	8.5e5(*1e6*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{24}}$	177(352)	5.5e5(*1e6*)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{25}}$	743(128)	5.8e5(72574)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{26}}$	1255(1431)	6677(1548)	8853(2797)	47428(45488)	1.6e6(*3e6*)	∞ *2.0e6*	0/5
$\mathbf{f_{27}}$	581(299)	1.4e6(*2e6*)	3.0e6(*4e6*)	8.1e6(*1e7*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{28}}$	961(260)	4298(216)	6368(794)	53832(31698)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{29}}$	1029(147)	5442(50)	13456(2084)	8.6e6(*1e7*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{30}}$	2907(208)	5513(776)	9565(2364)	3.2e6(*7e6*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{31}}$	1843(223)	50772(60356)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{32}}$	2676(334)	19018(16230)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{33}}$	1051(484)	3865(142)	4993(309)	14662(8327)	9.9e6(*8e6*)	∞ *2.0e6*	0/5
$\mathbf{f_{34}}$	1965(119)	6285(702)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{35}}$	1	5101(359)	11282(3294)	3.4e6(*3e6*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{36}}$	79(130)	6305(370)	19967(6103)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{37}}$	148(140)	38277(37382)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{38}}$	1255(136)	39586(9720)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{39}}$	86(110)	5267(472)	12937(8766)	2.5e6(*2e6*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{40}}$	1.6(1)	5.1e5(*2e6*)	3.0e6(*8e6*)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{41}}$	3042(1116)	8872(1952)	19026(10856)	5.4e5(*4e5*)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{42}}$	2345(782)	6.8e5(*2e6*)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{43}}$	3461(248)	67015(86923)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{44}}$	2868(782)	5537(1058)	8726(1437)	17067(7840)	∞	∞ *2.0e6*	0/5
$\mathbf{f_{45}}$	1806(265)	7925(2045)	3.0e6(*2e6*)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{46}}$	793(228)	1.2e5(*1e5*)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{47}}$	2550(72)	1.3e5(38852)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{48}}$	981(381)	30028(20545)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{49}}$	1351(75)	3.2e6(*4e6*)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{50}}$	2704(176)	20957(14228)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{51}}$	2552(370)	8885(4466)	8.0e6(*6e6*)	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{52}}$	2151(156)	52339(86772)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{53}}$	521(26)	3949(128)	4922(233)	7598(1198)	62026(10936)	∞ *2.0e6*	0/5
$\mathbf{f_{54}}$	2014(120)	5991(598)	∞	∞	∞	∞ *2.0e6*	0/5
$\mathbf{f_{55}}$	1088(183)	1.4e6(*2e6*)	3.0e6(*6e6*)	∞	∞	∞ *2.0e6*	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.**

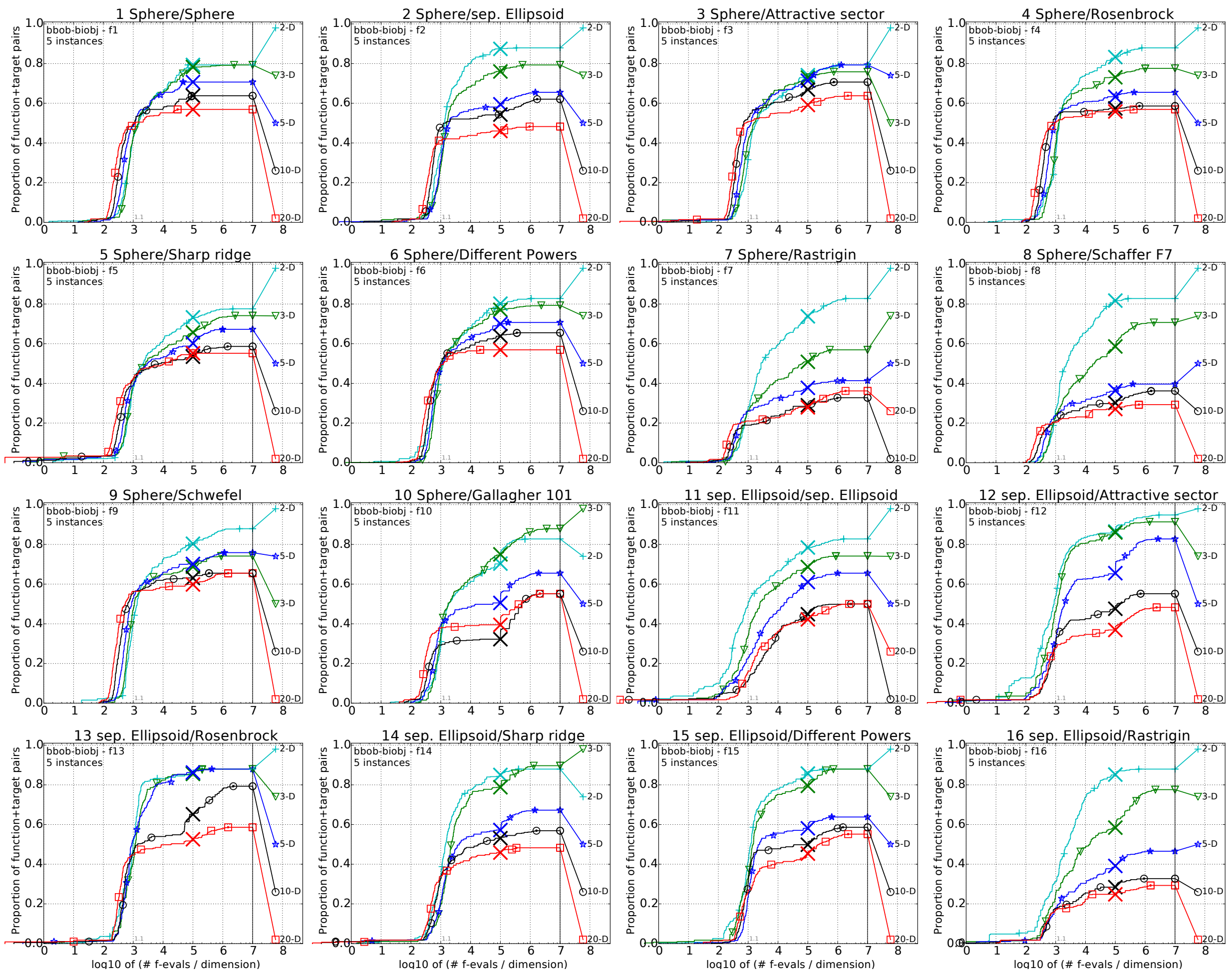

Figure 1: Empirical cumulative distribution of simulated (bootstrapped) runtimes in number of objective function evaluations divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots, 10^{-0.1},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

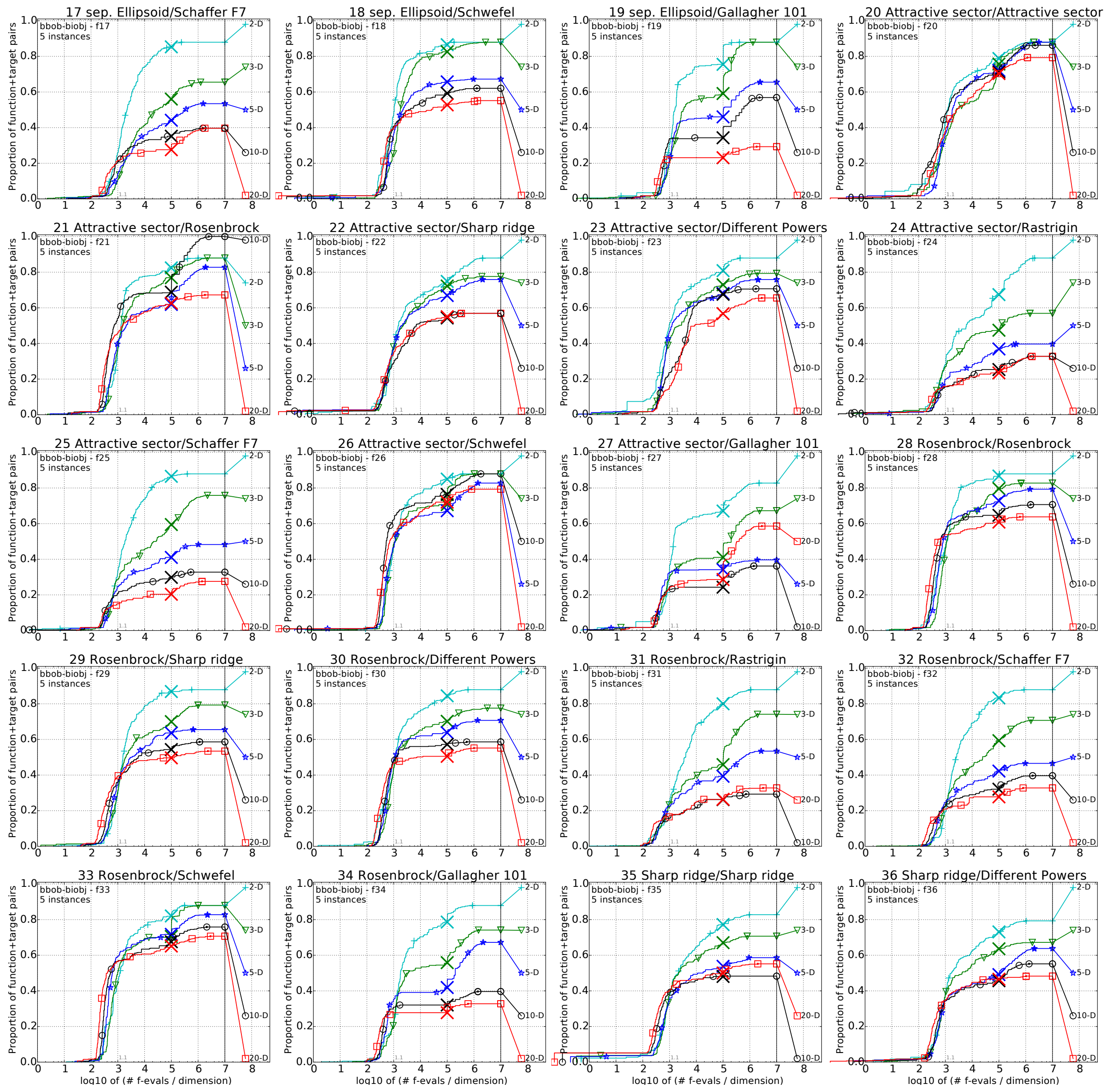

Figure 2: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 1 for functions f_{17} to f_{36} and all dimensions.

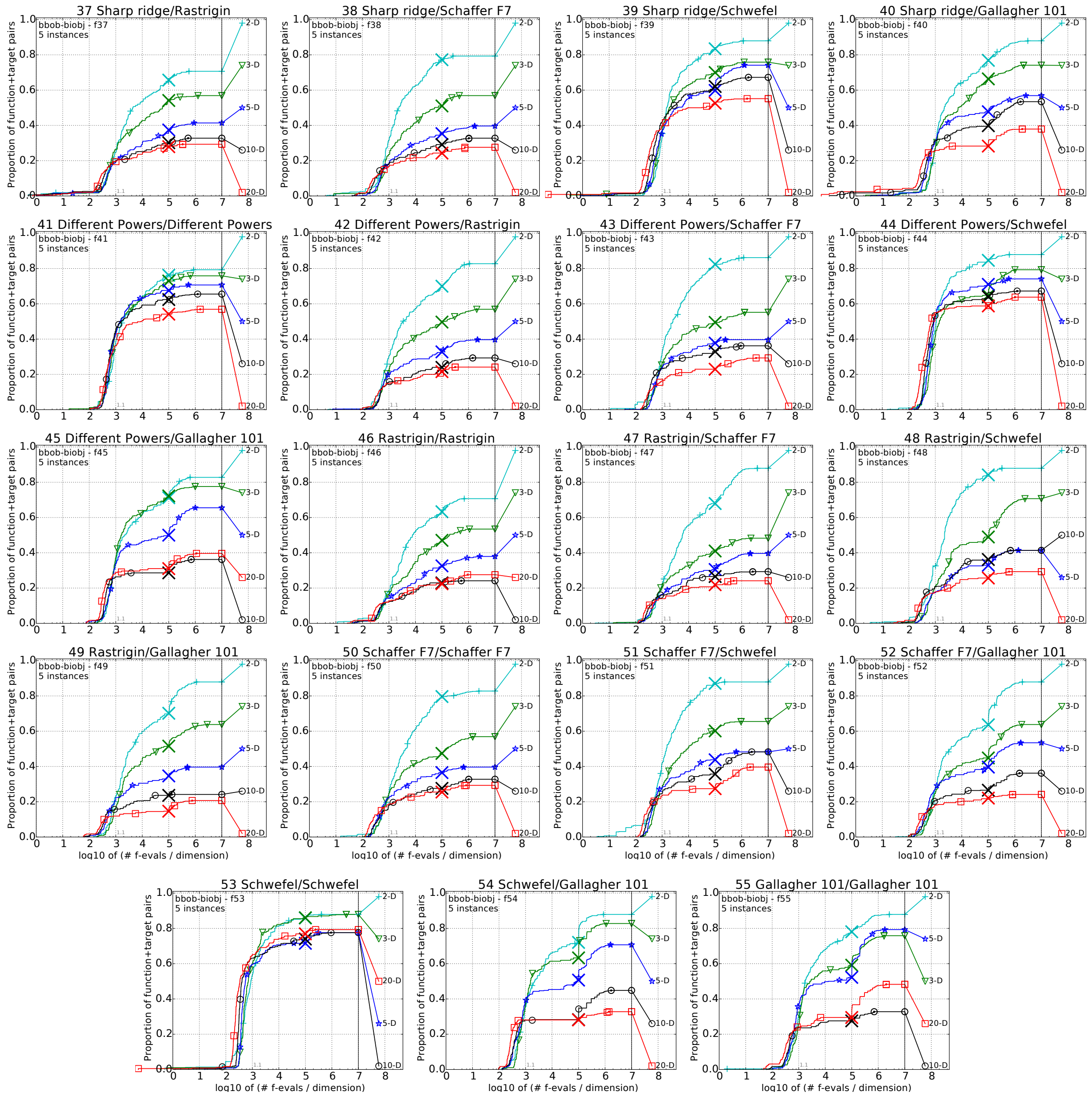

Figure 3: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 1 for functions f_{37} to f_{55} and all dimensions.

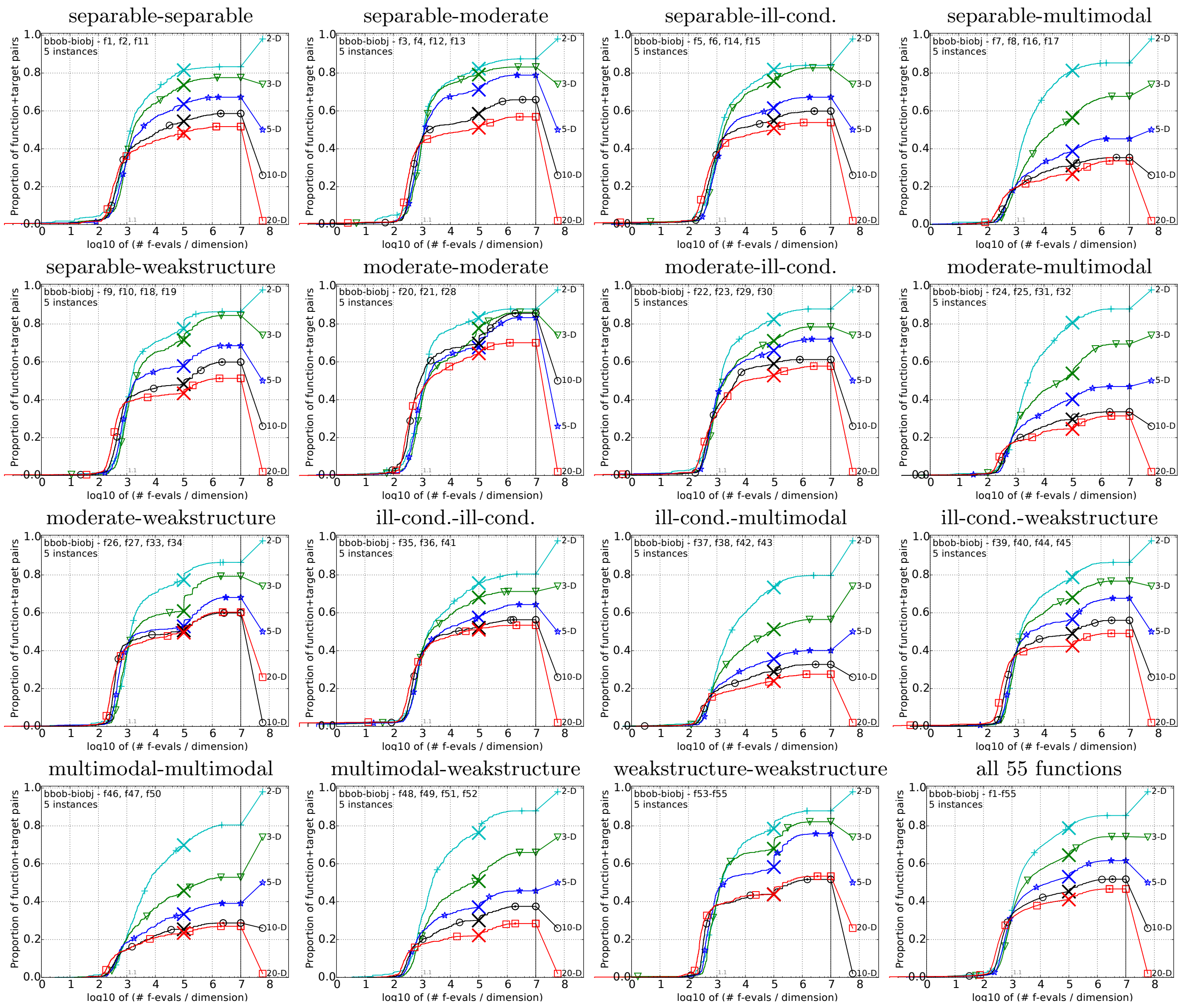

Figure 4: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots,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