Unbounded Population MO-CMA-ES for the Bi-Objective BBOB Test Suite

Oswin Krause
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
oswin.krause@di.ku.dk

Tobias Glasmachers
Institut für Neuroinformatik
Ruhr-Universität Bochum
Bochum, Germany
tobias.glasmachers@ini.rub.de

Nikolaus Hansen
INRIA, Research centre Saclay - Île-de-France
Orsay Cedex, France
hansen@lri.fr

Christian Igel
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
igel@di.ku.dk

ABSTRACT

The unbounded population multi-objective covariance matrix adaptation evolution strategy (UP-MO-CMA-ES) aims at maximizing the total hypervolume covered by all evaluated points. It adds all non-dominated solutions found to its population and employs Gaussian mutations with adaptive covariance matrices to also solve ill-conditioned problems. A novel recombination operator adapts the covariance matrices to point along the Pareto front. The UP-MO-CMA-ES is combined with a parallel exploration strategy and empirically evaluated on the bi-objective BBOB-biobj benchmark problems. Results show that the algorithm can reliably solve ill-conditioned problems as well as weakly-structured problems. However, it is less suited for the rugged multi-modal objective functions in the benchmark.

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, Multi-objective optimization, Covariance matrix adaptation evolution strategy

1. INTRODUCTION

A typical goal of multi-objective evolutionary algorithms (MOEAs) is to achieve the best possible approximation of the Pareto front with an a-priori fixed number of solutions. In this paper, we propose a novel MOEA for the different goal of approximating the Pareto front without any restriction on the cardinality of the solution set. Our approach is implemented for the special case of bi-objective optimization, however, it is not limited to two objectives.

The algorithm is based on the multi-objective covariance-matrix-adaptation evolution strategy (MO-CMA-ES, [**?**, **?**]), which is a state-of-the-art method for vector optimization. It stores a set of μ individuals, which are optimized to maximize the dominated hypervolume. Selection is a two-step process consisting of non-dominated sorting based on Pareto dominance (as proposed in [**?**]), as well as the hypervolume indicator as a secondary sorting criterion (similar to [**?**]). The MO-CMA-ES uses adaptation of the covariance matrices of all individuals to make successful steps more likely. This allows for efficiently solving non-separable and/or ill-conditioned problems.

It is common in MOEAs to just evaluate a final set of μ points as the result of the optimization process, where μ is chosen a priori and is small compared to the total number of objective function evaluations. However, a principal limitation of this approach is that in general a predefined fixed number of points cannot approximate a Pareto front arbitrarily well. Furthermore, in practice one may not want to loose any non-dominated point. Therefore, this paper explores the possibility of storing and exploiting all non-dominated solutions found so far as parent population. This is particularly inexpensive in bi-objective optimization, since testing whether a solution is dominated, inserting it into the population, and computing its hypervolume contribution can be accomplished in $\mathcal{O}(\log(\mu))$ time and $\mathcal{O}(\mu)$ memory, where μ is the (current) number of non-dominated solutions. This allows to approximate a Pareto front with high precision.

In addition, we propose a novel recombination operator. This operator does not affect the search point, but only the associated strategy parameters. It is designed to foster adaptation of the covariance matrices to the Pareto front by aligning the shape of mutation distributions along the front.

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

ACM 978-1-4503-4323-7/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2908961.2931699

2. ALGORITHM PRESENTATION

At its core, UP-MO-CMA-ES is a variant of the MO-CMA-ES [**?**, **?**]. Individuals are mutated by adding realizations of Gaussian random vectors, the covariance of which is adapted based on past successful steps.

The main difference of UP-MO-CMA-ES compared to existing variants of MO-CMA-ES (and other MOEAs) lies in the employed population model and the selection mechanism. A predominant paradigm for maintaining a fixed size population of size μ is indicator-based environmental selection, often based on the hypervolume indicator [**?**]. In fact, the need for two-stage sorting based on non-dominance rank and hypervolume contribution is an artefact of a fixed population size. In contrast, UP-MO-CMA-ES maintains *all* non-dominated individuals in the population, which also means that we discard all dominated points. As a result, the size of the parent population is dynamic. Its minimal size is one, and there is no upper bound.

This design has consequences for the goal of optimization. At first our algorithm must approach the Pareto front. However, once the front is reached it replaces the usual goal of finding the optimal distribution of a fixed number of μ points over the front with that of filling in the gaps. Hence, in a successful run the hypervolume does not converge to the optimal μ-distribution [**?**], but to the hypervolume covered by the actual Pareto front. Of course, this requires unlimited memory, so in practice the algorithm must stop (for the latest) when it runs out of memory. This is not a serious limitation, we were able to run the algorithm on standard hardware on all BBOB-biobj 2016 problem instances for 10^6 function evaluations.

When generating an offspring, we propose to select a parent individual based on its contribution to the dominated hypervolume

$$\delta\mathrm{Vol}_Y(\boldsymbol{y}) = \mathrm{Vol}(Y) - \mathrm{Vol}(Y \setminus \{\boldsymbol{y}\}) \ ,$$

where $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_\mu\}$ is the set of objective vectors corresponding to the parent population. This rule has the interesting consequence to change the roles of the two nested sorting criteria of non-dominance rank and hypervolume contribution. In UP-MO-CMA-ES the primary criterion of a low non-dominance rank is used for environmental selection, which is hence entirely based on the Pareto dominance relation. The secondary criterion of a high contribution to the dominated hypervolume is applied for mating selection.

The individuals are stored in an AVL-tree [**?**], which keeps them sorted by the first objective value in increasing order, and in decreasing order by the second objective value. This allows for efficient updates of the hypervolume contribution of a point in the tree, as neighbors on the front can be obtained in logarithmic time. As in the MO-CMA-ES, the ith individual consists of a search point $\boldsymbol{x}_i \in \mathbb{R}^D$, the corresponding objective vector $\boldsymbol{y}_i = f(\boldsymbol{x}_i) \in \mathbb{R}^2$, a step size $\sigma_i > 0$, and a covariance matrix $C_i \in \mathbb{R}^{D\times D}$. Unlike in the MO-CMA-ES, no evolution paths are maintained.

In bi-objective optimization only the hypervolume contributions of the extreme points (the current best w.r.t. one criterion) depend on the chosen reference point. These values are generally incomparable to the contributions of "interior" points. Therefore, parent individuals are selected in a two-step process. An extreme point is picked with a fixed probability p_{extreme}. If no extreme point is picked or the picked point seems to have converged (i.e., its step size σ is below some threshold $\sigma_{\min}$), then the ith interior point is picked from the tree with probability

$$p_i = \frac{\delta\mathrm{Vol}_Y(\boldsymbol{y}_i)^\alpha}{\sum_j \delta\mathrm{Vol}_Y(\boldsymbol{y}_j)^\alpha} \ .$$

This again can be achieved in logarithmic time by storing the cumulative contributions of each subtree in the AVL-tree. The constant $\alpha > 1$ gives more weight to points with large contributions.

After the individual i is selected as a parent, we create a new offspring point by first applying a recombination operation to the covariance matrix of the mutation distribution. For this, the two nearest neighbors on the Pareto front are picked. Let those have indices $i-1$ and $i+1$. The covariance matrix of the offspring C is then updated by

$$C = (1-c_r)C_i + \frac{c_r}{2}\left(\frac{\boldsymbol{x}_{i-1}-\boldsymbol{x}_i}{\sigma_i}\right)\left(\frac{\boldsymbol{x}_{i-1}-\boldsymbol{x}_i}{\sigma_i}\right)^{\mathrm{T}} + \frac{c_r}{2}\left(\frac{\boldsymbol{x}_{i+1}-\boldsymbol{x}_i}{\sigma_i}\right)\left(\frac{\boldsymbol{x}_{i+1}-\boldsymbol{x}_i}{\sigma_i}\right)^{\mathrm{T}}$$

The rationale of this operation is to enable the sampling scheme to fill in gaps between individuals of a population that has already reached the Pareto front. This new recombination operator is generic in nature and hence applicable to other variants of MO-CMA-ES. For an alternative recombination scheme for covariance matrices in the MO-CMA-ES we refer to [**?**].

The new point $\boldsymbol{x}$ is sampled from $\mathcal{N}(\boldsymbol{x}_i, \sigma_i^2 C)$. The update of the covariance matrix of a successful individual is the same as in MO-CMA-ES with evolution path learning-rate 1 (i.e., $c_c = 1$ in [**?**]) and thus C is adapted according to

$$C \leftarrow (1-c_{\mathrm{cov}})C + c_{\mathrm{cov}}\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{\sigma_i}\right)\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{\sigma_i}\right)^{\mathrm{T}} \ .$$

Unlike in MO-CMA-ES the same update is also applied to the parent.

The step size is adapted as in the MO-CMA-ES [**?**], however, the target success rate is increased to $1/2$ and there is no upper limit for the success rate.

As the update of the covariance matrix can directly be applied to its Cholesky factor in $\mathcal{O}(D^2)$ operations, a full iteration of the algorithm runs in $\mathcal{O}(\log(\mu)+D^2)$ time and requires $\mathcal{O}(\mu D^2)$ storage, see [**?**]. This complexity is tractable on standard PC hardware for moderate search space dimensions and even large numbers of individuals. In high-dimensional search spaces the covariance matrices can be stored on disk to reduce the memory requirements to $\mathcal{O}(\mu D)$. This is feasible because only two covariance matrices (one old and one new matrix) are accessed per iteration. For extremely high-dimensional problems one could even store the search point to disk to reduce the memory cost to $\mathcal{O}(\mu)$. However, the AVL tree should always be maintained in fast random access memory. The tuned parameter values of UP-MO-CMA-ES are given in Table 1.

constant	value
p_{extreme}	$\frac{1}{5}$
$\sigma_{\min}$	10^{-20}
α	3
c_{cov}	$\frac{2}{d^{2.1}+3}$
c_r	$\frac{c_{\text{cov}}}{2}$

Table 1: Constants used in UP-MO-CMA-ES.

3. EXPERIMENTAL PROCEDURE

We ran the UP-MO-CMA-ES on the full BBOB-biobj 2016 benchmark in dimensions $D \in \{2, 3, 5, 10, 20\}$ with a total budget of $10^6 D$ function evaluations. To tackle multi-modal functions, we start multiple instances of the algorithm in a parallel exploration phase. Here, we first started $k = 100$ independent instances of the algorithm each using an initial design of D points sampled uniformly in $[-5, 5]^D$. The instances were running in a round-robin fashion for a total budget of $10^4 D$ function evaluations, including the initial design.

After this exploration phase, we merge the fronts found by the k instances into one large front of non-dominated points. A single instance of the algorithm is initialized with this front and runs until the total budget is exhausted. In this setup, the initial exploration phase consumed 1 % of the total budget.

Before running the benchmark, the parameters were tuned on a subset of it. We used functions 1,2, 10 and 11 for this task. The parameters α and the target success rate were tuned on *Sphere/Sphere*, c_{cov} was tuned on *Sphere/Ellipsoid*, c_r on *Ellipsoid/Ellipsoid* and the number of restarts k on *Sphere/Galagher 101*. Tuning was not performed exhaustively and the optimal parameters are likely to be different.

4. CPU TIMING

In order to evaluate the CPU timing of the algorithm, we ran the UP-MO-CMA-ES on the entire bbob-biobj test suite for $1000D$ function evaluations. The code was implemented in C++ using the Shark library [**?**] and was run on an Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50GHz with 1 processor and 6 cores on a single thread. The time per function evaluation for dimensions 2, 3, 5, 10, and 20 was $3.7 \cdot 10^{-6}$, $4.2 \cdot 10^{-6}$, $5.3 \cdot 10^{-6}$, $7.8 \cdot 10^{-6}$, and $1.4 \cdot 10^{-5}$ seconds, respectively.

5. RESULTS AND DISCUSSION

Results of UP-MO-CMA-ES from experiments according to [**?**], [**?**] and [**?**] on the benchmark functions given in [**?**] are presented in Figures 1, 2, 3 , and 4, and in Table 2. The experiments were performed with COCO [**?**], version 1.0.1, the plots were produced with version 1.1. Note that in all Figures, the reference value of the hypervolume has been reached with high probability when the graph reaches the value 0.9.

As can be seen from Figures 1, 2 and 3, the UP-MO-CMA-ES performed well on all problems except those involving the multi-modal Rastrigin and Schaffer-F7 functions as objectives. In Figure 4, there are no strong differences between the function classes separable, moderate and ill-conditioned, indicating that the covariance matrix adaptation works as desired. The initial exploration strategy proofed to be successful for solving the category of weakly structured problems, often finding better values than the reference used in the benchmark. Only the class of structured multi-modal functions posed a problem to the algorithm. We observed drastically decaying performance in higher dimensions. This suggests that the chosen exploration strategy is not suitable for this type of problem.

It turns out that about half of the evaluated points are non-dominated at the end of the run. This indicates that the algorithm is very effective at filling in gaps between existing solutions. It also means that the memory requirements are substantial (about 1.7 GB for $5 \cdot 10^5$ non-dominated points and covariance matrices in $D = 20$ dimensions with double precision numbers), but bearable even on standard hardware.

6. CONCLUSION

We have presented a novel type of evolution strategy for multi-objective optimization. The algorithm abandons the established paradigm of maximizing the hypervolume dominated by a population of a-priori fixed size μ. Instead we maintain an unbounded population consisting of all non-dominated points. This allows us to apply a simple and straightforward selection using Pareto dominance for environmental selection and contributed hypervolume for parent selection. The algorithm bears the potential to converge to the true Pareto front, not only to an optimal μ-distribution.

The practical performance of UP-MO-CMA-ES appears to be promising, unless the algorithm is facing a highly multi-modal problem with millions of local optima. In this case it fails due to its inability to exploit the structure of the distribution of these optima. However, such a structure can be considered somewhat artificial. Hence, we believe that the proposed algorithm is of high practical value.

Acknowledgements

CI and OK gratefully acknowledge support from the Innovation Fund Denmark through the *Danish Center for Big Data Analytics Driven Innovation* (DABAI) and the project *Personalized Breast Cancer Screening*.

5-D

Δf	1e+0	1e-1	1e-2	1e-3	1e-4	1e-5	#succ
f_1	1	1175(730)	14760(2241)	54736(186)	1.1e5(1476)	3.7e5(10644)	5/5
f_2	1.8(1)	1791(1843)	11880(6776)	48883(8687)	79219(15560)	2.8e5(94608)	5/5
f_3	1.8(1)	1583(1454)	14747(5046)	48725(8134)	87385(21462)	2.6e5(91156)	5/5
f_4	1.2(0.5)	706(214)	9331(3179)	49936(448)	75668(3538)	2.3e5(20340)	5/5
f_5	1	1903(699)	25050(4092)	59686(3159)	1.2e5(6879)	3.8e5(15152)	5/5
f_6	1	946(780)	12591(3864)	53281(1521)	94508(10653)	2.9e5(48731)	5/5
f_7	1	5370(8972)	1.3e6(1e6)	∞	∞	∞ *5.0e6*	0/5
f_8	6.6(10)	8076(1782)	7.6e6(1e7)	∞	∞	∞ *5.0e6*	0/5
f_9	1.6(2)	1230(412)	9120(2728)	47180(5848)	70863(5504)	1.6e6(3e6)	4/5
f_{10}	1	3224(1504)	19486(1301)	52609(988)	87005(7765)	2.6e5(36982)	5/5
f_{11}	1	329(569)	4320(4661)	37484(18858)	1.0e5(22766)	1.6e6(4e6)	4/5
f_{12}	1.4(0.5)	1728(1926)	7862(12485)	23542(25995)	42125(29803)	98686(70606)	5/5
f_{13}	1.8(2)	408(446)	2091(1678)	9415(5777)	31473(17348)	61032(8834)	5/5
f_{14}	1	4540(2780)	32266(6544)	55042(1556)	98142(12366)	2.1e5(30152)	5/5
f_{15}	13(14)	2065(1608)	13760(5238)	48001(5672)	73181(9258)	2.0e5(49650)	5/5
f_{16}	1.8(1)	4618(4344)	7.5e6(2e7)	∞	∞	∞ *5.0e6*	0/5
f_{17}	884(4)	7331(6666)	4.4e6(8e6)	∞	∞	∞ *5.0e6*	0/5
f_{18}	1	393(468)	3624(2686)	35555(22938)	65906(13880)	3.6e6(8e6)	3/5
f_{19}	3.8(6)	1542(1270)	14286(11745)	1.3e6(4e6)	1.3e6(3e6)	1.4e6(3e6)	4/5
f_{20}	1.2(0.2)	1038(444)	12945(8538)	45512(18471)	75486(22272)	1.9e5(1e5)	5/5
f_{21}	1	2302(3025)	13760(3439)	42741(883)	69170(19976)	2.0e5(1e5)	5/5
f_{22}	1	1870(710)	25311(8456)	54215(2100)	98341(25338)	1.6e6(5e6)	4/5
f_{23}	1.2(0.5)	1087(957)	13399(6388)	47699(7211)	82349(15916)	2.4e5(91536)	5/5
f_{24}	6.0(6)	5546(2149)	7.7e6(8e6)	∞	∞	∞ *5.0e6*	0/5
f_{25}	2.4(2)	6379(10943)	7.5e6(1e7)	2.0e7(3e7)	2.2e7(3e7)	∞ *5.0e6*	0/5
f_{26}	5.4(11)	827(700)	8516(8166)	37565(12237)	1.3e6(1e6)	3.6e6(9e6)	3/5
f_{27}	1.8(2)	1610(673)	24637(16112)	52565(10012)	91983(45811)	1.5e6(1e6)	4/5
f_{28}	1	577(362)	4418(978)	23782(11924)	54804(3969)	1.1e5(9010)	5/5
f_{29}	1	3658(2370)	30159(10312)	53961(1612)	95674(16799)	1.9e5(55706)	5/5
f_{30}	2.6(2)	1240(800)	11460(4878)	45057(9238)	71185(11518)	1.9e5(53702)	5/5
f_{31}	1.6(0.8)	4395(3814)	7.6e6(4e6)	∞	∞	∞ *5.0e6*	0/5
f_{32}	2.6(2)	5075(4395)	7.9e6(9e6)	∞	∞	∞ *5.0e6*	0/5
f_{33}	9.4(5)	404(308)	1474(334)	9841(4268)	1.3e6(1e6)	2.0e7(2e7)	1/5
f_{34}	1.4(1)	1180(1028)	16095(11334)	54675(32142)	81662(60424)	4.1e5(5e5)	5/5
f_{35}	1	2892(2346)	37744(10168)	62024(1007)	1.4e5(25480)	2.9e5(69148)	5/5
f_{36}	1	3252(1890)	32167(8712)	57518(5603)	1.1e5(17397)	2.8e5(77163)	5/5
f_{37}	1	10484(4041)	3.4e6(4e6)	∞	∞	∞ *5.0e6*	0/5
f_{38}	1	14348(7710)	2.0e7(2e7)	∞	∞	∞ *5.0e6*	0/5
f_{39}	1	2662(1228)	25927(5090)	52606(1174)	78422(9906)	2.4e5(1e5)	5/5
f_{40}	1	4224(1354)	31467(7377)	54740(2072)	1.0e5(16465)	3.0e5(1e5)	5/5
f_{41}	1.6(2)	1317(1696)	13611(7058)	46701(11140)	82951(19925)	2.3e5(62943)	5/5
f_{42}	1.4(1)	7070(4786)	1.4e6(94118)	∞	∞	∞ *5.0e6*	0/5
f_{43}	1.6	6621(6752)	3.6e6(6e6)	∞	∞	∞ *5.0e6*	0/5
f_{44}	1.6(2)	757(847)	8576(5360)	38396(15632)	64168(7027)	3.8e6(5e6)	3/5
f_{45}	3.0(4)	2917(1376)	21745(2272)	53955(1722)	88571(12472)	2.5e5(50842)	5/5
f_{46}	1	16592(11948)	∞	∞	∞	∞ *5.0e6*	0/5
f_{47}	2.2	17497(12794)	∞	∞	∞	∞ *5.0e6*	0/5
f_{48}	2.4(0.5)	12860(14273)	2.0e7(2e7)	∞	∞	∞ *5.0e6*	0/5
f_{49}	1.4(0.5)	1.3e6(3e6)	7.6e6(8e6)	∞	∞	∞ *5.0e6*	0/5
f_{50}	1.6(0.8)	7555(8273)	2.2e7(2e7)	∞	∞	∞ *5.0e6*	0/5
f_{51}	2.2(2)	4661(8623)	3.4e6(4e6)	∞	∞	∞ *5.0e6*	0/5
f_{52}	1	4658(4482)	1.7e6(3e6)	∞	∞	∞ *5.0e6*	0/5
f_{53}	5.8(8)	201(238)	1040(601)	1.3e6(1e6)	1.3e6(3e6)	∞ *5.0e6*	0/5
f_{54}	1.6(1)	933(525)	12674(5495)	43826(10858)	55476(2082)	1.7e5(51598)	5/5
f_{55}	1.2(0.5)	3919(4202)	16947(7455)	50911(684)	1.3e6(2e6)	1.4e6(46746)	4/5

20-D

Δf	1e+0	1e-1	1e-2	1e-3	1e-4	1e-5	#succ
f_1	1	24250(2860)	2.0e5(241)	2.4e5(540)	4.9e5(2178)	2.0e6(20390)	5/5
f_2	1.2(0.5)	22992(5944)	1.1e5(6516)	2.2e5(4027)	3.7e5(39330)	1.6e6(6e5)	5/5
f_3	1	24523(4734)	1.5e5(41629)	2.2e5(7664)	3.7e5(50630)	1.0e6(1e5)	5/5
f_4	1	22012(2014)	1.2e5(9995)	2.2e5(1368)	3.8e5(15444)	5.1e6(4e6)	5/5
f_5	1	34530(4529)	2.0e5(228)	2.6e5(1782)	5.9e5(2882)	1.9e6(2e5)	5/5
f_6	1	22385(2062)	1.6e5(29220)	2.3e5(3179)	4.0e5(23338)	1.3e6(31132)	5/5
f_7	1	93172(54648)	8.0e7(1e8)	∞	∞	∞ *2.0e7*	0/5
f_8	1	6.6e5(1e6)	∞	∞	∞	∞ *2.0e7*	0/5
f_9	1	18113(2177)	85867(3488)	2.1e5(794)	2.9e5(5557)	1.5e6(6e5)	5/5
f_{10}	1	42870(5367)	2.0e5(689)	2.4e5(3757)	4.1e5(64228)	8.6e5(4e5)	5/5
f_{11}	1	4932(3425)	1.4e5(57823)	2.6e5(6850)	6.6e6(1e6)	3.4e7(2e7)	2/5
f_{12}	1	8120(2022)	42549(19727)	1.8e5(30804)	2.5e5(7652)	5.0e5(51219)	5/5
f_{13}	1	10011(4967)	64216(28804)	2.1e5(9396)	6.2e5(3e5)	3.6e6(5e6)	5/5
f_{14}	1	50430(10655)	2.0e5(709)	2.5e5(23899)	1.4e7(2e7)	3.3e7(2e7)	2/5
f_{15}	1	16023(8137)	1.1e5(39292)	2.1e5(4594)	3.4e5(33025)	1.2e6(8e5)	5/5
f_{16}	1	1.4e5(53060)	∞	∞	∞	∞ *2.0e7*	0/5
f_{17}	3.2(4)	3.0e7(5e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{18}	1	9458(2956)	61691(14623)	2.1e5(3955)	4.2e5(75919)	1.6e6(2e5)	5/5
f_{19}	1.4(1)	47426(10094)	2.0e5(14312)	1.4e7(4e7)	1.4e7(2e7)	1.4e7(2e7)	3/5
f_{20}	1	12687(2092)	50670(29424)	1.2e5(75502)	2.5e5(85638)	4.8e5(1e5)	5/5
f_{21}	1	10972(6694)	53804(15568)	2.0e5(25018)	1.8e6(4e6)	1.4e7(2e7)	3/5
f_{22}	1	43660(11777)	2.0e5(960)	2.4e5(9475)	4.6e5(32735)	6.0e6(1e5)	4/5
f_{23}	1.2(0.5)	19856(7274)	1.4e5(51657)	2.1e5(9996)	3.2e5(40171)	8.1e5(1e5)	5/5
f_{24}	1	82074(55678)	∞	∞	∞	∞ *2.0e7*	0/5
f_{25}	1	3.0e7(2e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{26}	1	10536(3902)	28379(13740)	1.0e5(82184)	5.2e6(1e7)	1.4e7(2e7)	3/5
f_{27}	1.2(0.2)	62458(23427)	2.4e5(89422)	1.4e7(2e7)	1.4e7(3e7)	1.5e7(2e7)	3/5
f_{28}	1	11364(2241)	44438(8929)	2.0e5(1201)	3.0e5(1e5)	7.2e6(1e7)	4/5
f_{29}	1	44621(6146)	2.0e5(198)	2.5e5(7304)	5.6e5(1e5)	6.6e6(2e7)	4/5
f_{30}	1.2(0.2)	21842(13058)	1.2e5(36436)	2.1e5(6409)	4.2e5(65750)	6.5e6(6e6)	4/5
f_{31}	1	5.5e6(1e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{32}	36(4)	3.0e7(2e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{33}	1	9150(1409)	33300(10430)	1.7e5(38073)	5.1e5(3e5)	7.8e6(2e7)	4/5
f_{34}	1	44812(7042)	2.0e5(6711)	2.2e5(4606)	3.0e5(37672)	5.5e5(4e5)	5/5
f_{35}	1	42446(9208)	2.0e5(635)	2.8e5(23055)	1.4e7(2e7)	1.5e7(2e7)	3/5
f_{36}	1	48049(3520)	2.0e5(1278)	2.6e5(4235)	5.6e6(2e7)	1.6e7(1e7)	3/5
f_{37}	1	1.3e5(54330)	∞	∞	∞	∞ *2.0e7*	0/5
f_{38}	1	5.2e6(1e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{39}	1	52059(11004)	5.2e6(2e7)	5.2e6(5e6)	5.5e6(2e7)	1.9e7(2e7)	3/5
f_{40}	1	53109(12046)	2.1e5(3474)	2.8e5(31684)	6.0e5(4e5)	5.8e6(1e7)	4/5
f_{41}	1	26271(19342)	1.6e5(49220)	2.2e5(11487)	3.7e5(80212)	1.1e6(5e5)	5/5
f_{42}	1	1.5e5(57428)	∞	∞	∞	∞ *2.0e7*	0/5
f_{43}	1	3.0e7(5e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{44}	2.2(2)	14631(7376)	84064(15472)	2.0e5(2334)	2.6e5(17827)	9.6e5(4e5)	5/5
f_{45}	1	46451(10626)	2.0e5(6442)	5.2e6(15073)	1.4e7(2e7)	1.4e7(3e7)	3/5
f_{46}	1	3.0e7(5e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{47}	1.2(0.5)	∞	∞	∞	∞	∞ *2.0e7*	0/5
f_{48}	1	5.1e6(1e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{49}	1	6.4e6(2e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{50}	1.2(0.2)	∞	∞	∞	∞	∞ *2.0e7*	0/5
f_{51}	2.2(1)	5.1e6(1e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{52}	1	6.0e6(1e7)	∞	∞	∞	∞ *2.0e7*	0/5
f_{53}	1	6370(1035)	20640(4125)	62587(35485)	1.9e5(37624)	3.0e7(4e7)	2/5
f_{54}	1	39216(9374)	1.8e5(28929)	5.3e6(1e7)	5.6e6(4e5)	1.4e7(2e7)	3/5
f_{55}	1	72011(25069)	5.2e6(1e7)	5.3e6(2e7)	1.4e7(2e7)	1.4e7(2e7)	3/5

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.

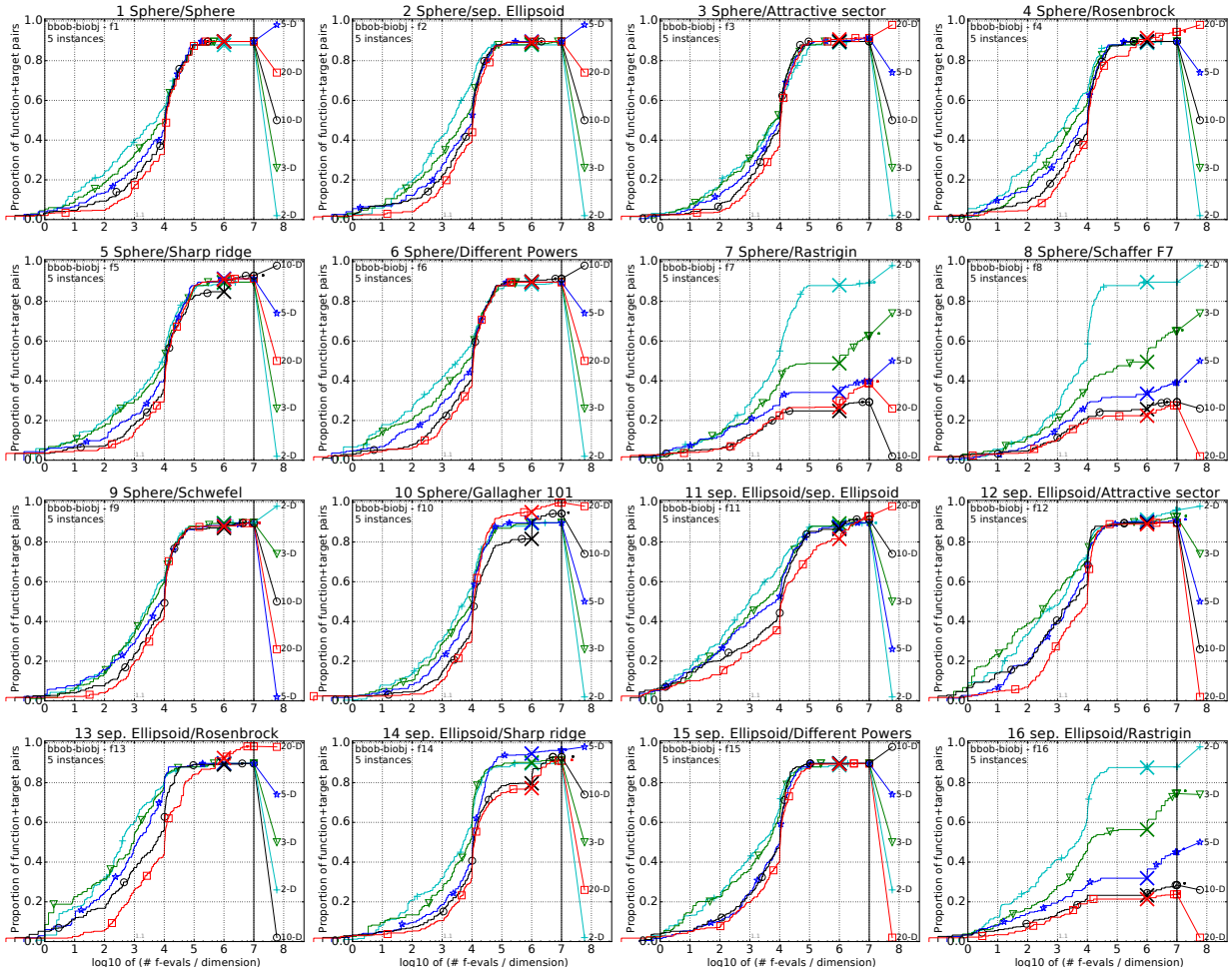

Figure 1: Empirical cumulative distribution of simulated (bootstrapped) runtimes in number of objective function evaluations divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots, 10^{-0.1},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

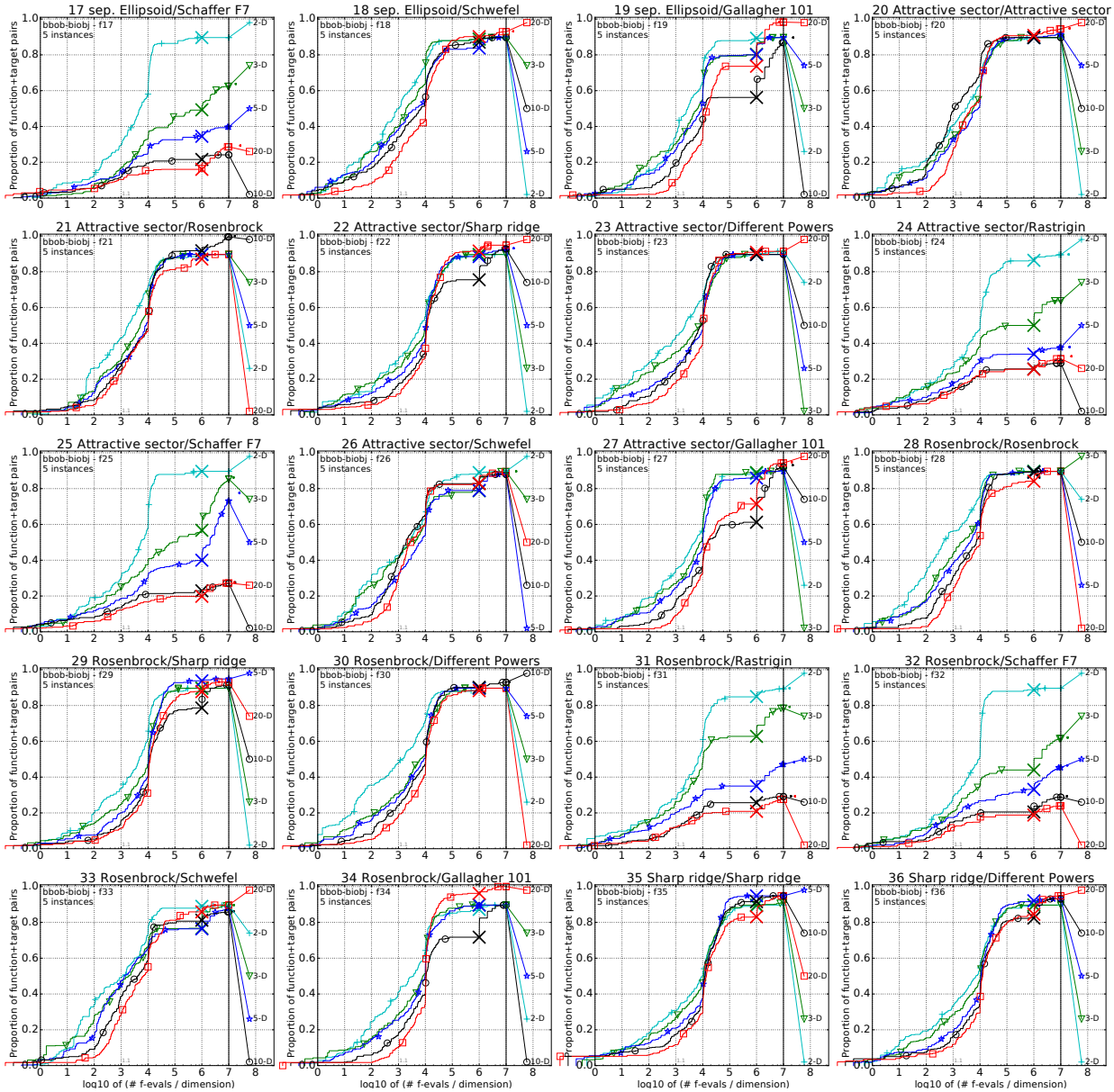

Figure 2: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 1 for functions f_{17} to f_{36} and all dimensions.

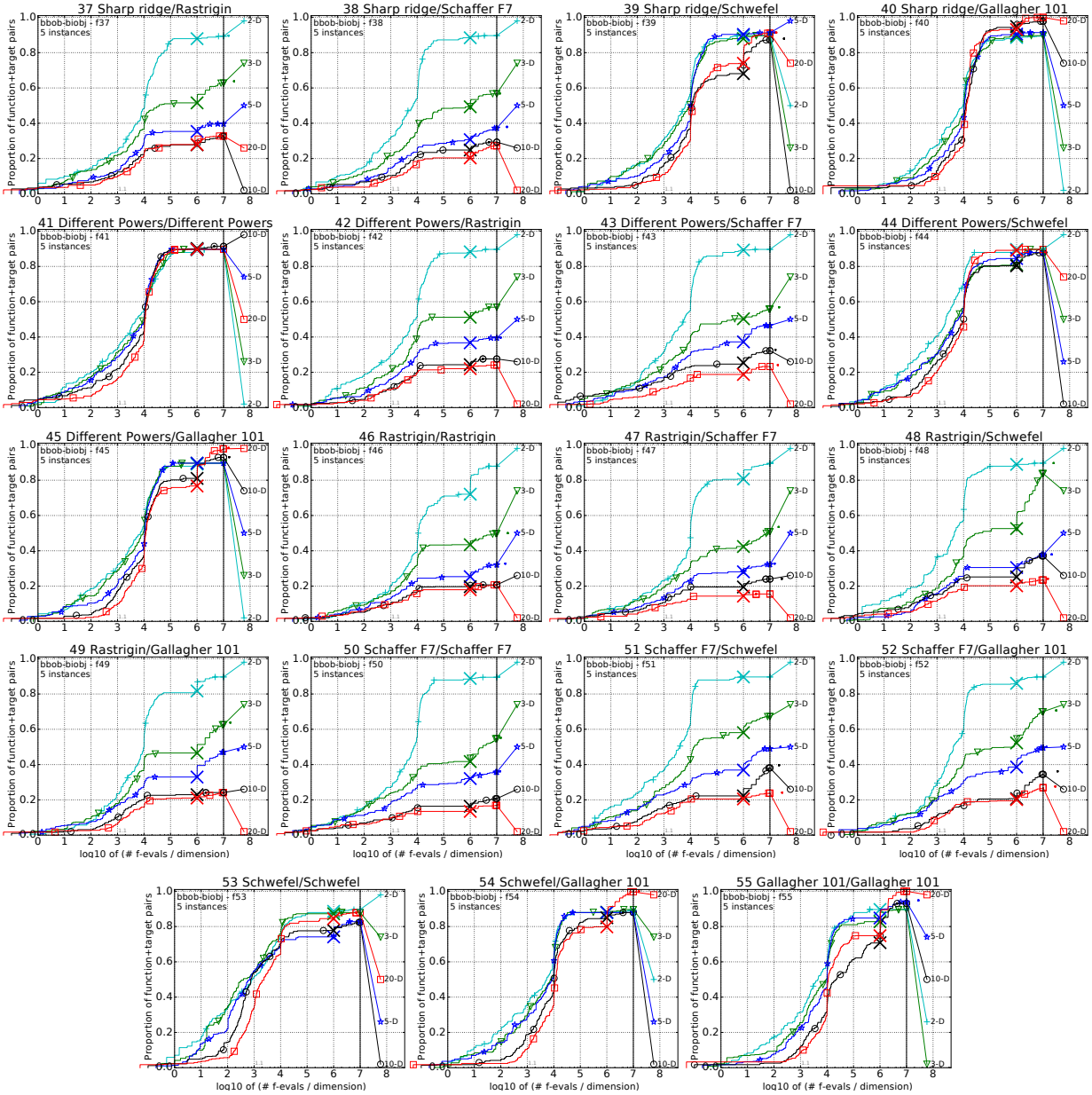

Figure 3: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 1 for functions f_{37} to f_{55} and all dimensions.

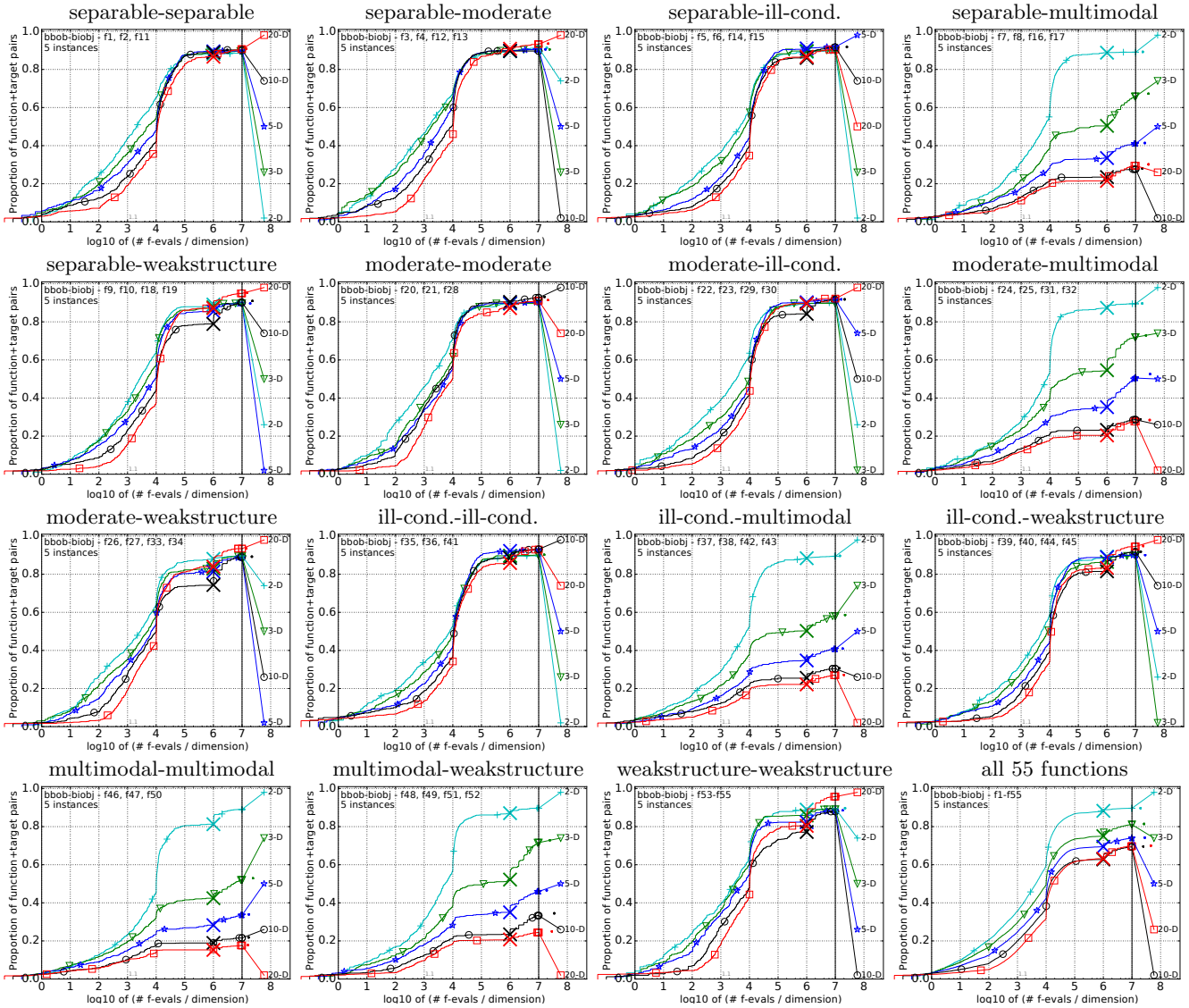

Figure 4: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots,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