Black-Box Optimization Benchmarking Template for the Bi-Objective BBOB Test Suite

Intermediate report *

Karim Kouki Ahmed Mazari Daro Ozad
Mihaela Sorostinean Aris Tritas

ABSTRACT

The purpose of our project is to implement and benchmark the Indicator Based Evolutionary Algorithm (IBEA) using the Comparing Continuous Optimizer (COCO) platform.

Keywords

Benchmarking, Black-box optimization, Bi-objective optimization

1. INTRODUCTION

In the context of a multi-objective optimization, the main goal is to find a good approximation of the set of Pareto-optimal solutions. IBEA defines an optimization goal using a performance measure, in our case and indicator of the relative quality of two sets of solution vectors with respect to the Pareto front. The fitness of an approximation set is defined as a dominance-preserving relation, and is computed with the epsilon indicator function. Since defining what a good approximation means is highly user dependent, the authors propose an algorithm which adapts to arbitrary preference information and optimization scenarios while removing the need to add diversity preservation techniques. As the authors noted in the paper, the main advantages of IBEA are:

- Generalization: the population size can be arbitrary, and
- Speed: the comparisons are done between pairs of individuals and not between sets of candidate solution vectors.

In turn, Adaptive-IBEA scales the objective function values as well as the range of values taken by the indicator function. This scheme decreases the need parameter tuning in face of problem and indicator function diversity.

Also, the authors reported a significant improvement in the quality of the resulted Pareto-set approximation with respect to the optimization goal.

*Submission deadline: October 10th.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. ALGORITHM PRESENTATION

The input of the algorithm is the size of the population (α), a maximum number of generations (corresponding to the budget) set as termination criterion and a fitness scaling factor (κ). The output of the algorithm is an approximation of the Pareto-set.

1. **Initialization:** consists in randomly generating an initial population P of the given size and setting a generation counter m to 0.

2. **Fitness assignment:** is represented by the computation and assignment of a fitness value for each individual in P; this fitness value is a measure of usefulness of each individual in regard to the optimization goal, so the algorithm tries to maximize it.

3. **Environmental selection:** consists of iteratively detecting and removing the individual which has the smallest fitness value form the population and then updating the fitness values for all remaining individuals until the current size of the population P does not exceed α.

4. **Termination:** after the environmental selection is performed, the termination criterion of the algorithm is checked; if the maximum number of generations is reached or another termination criterion is met, the algorithm returns the set of decision vectors A.

5. **Mating Selection:** consists in creating as temporary mating pool PâĂŹ which is filled with individuals from P by performing binary tournament selection with replacement on P.

6. **Variation:** finally the variation step consists applying recombination and mutation operators to the previously created mating pool. In this case, the operator SBX-20 was chosen for the recombination part, while a polynomial distribution was used for the mutation part. The offspring resulted after applying these operators is added to P, while also incrementing the generation counter. The algorithm is then performed again from step 2, until a termination criterion is met.

2.1 Recombination

2.2 Variation

3. EXPERIMENTAL PROCEDURE

For the first part of the project our milestone was to have a working version of the algorithm, and potentially reasonable results on at least some groups of functions.

Needless to say, to begin with, a thorough understanding of the problem's setting is required. As such, the first step was to have both individual and group readings of the paper.

As far as the implementation in Python is concerned we chose to represent the population in a compact data structure and did all numerical computation with NumPy. The steps of the algorithm were inlined in a single optimization function, except for the recombination and mutation operators which were implemented in separate functions to achieve modularity during testing.

We tried intermediate weighting and discrete recombination before implementing the Simulated Binary Crossover operator. However, results using these operators were not encouraging. The reason for that may be a rather naive exploration of the search space.

For the mutation step, we simply added isotropic Gaussian noise with fixed variance to the produced offspring. However, it is well known that fixing variance does not speed up search optimally. Without further changes to the original strategy, our algorithm does not seem to reach the optimal Pareto set of solution vectors. We tried applying recombination and mutation with high and low probabilities respectively, following the literature. This did not yield improved results, which leads us to think that either our implementation lacks robustness, or has a subtle bug.

Going forward, we intend to:

1. Further test and validate the robustness of the current implementation
2. Experiment with self-adaptive strategies for the mutation step, and potentially
3. Implement Self-Adaptive SBX which seeks an optimal index for the approximation distribution used

4. CPU TIMING

In order to evaluate the CPU timing of the algorithm, we have run the IBEAwith restarts on the entire bbob-biobj test suite [**?**] for $2D$ function evaluations. The Python code was run on a Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz with 4 processors and 8 cores. The time per function evaluation for dimensions 2, 3, 5, 10, 20 equals $1.7e-03$, $1.9e-03$, $2.0e-03$, $2.2e-03$, $2.4e-03$ seconds respectively.

5. RESULTS

Results of IBEA from experiments according to [**?**], [**?**] and [**?**] on the benchmark functions given in [**?**] are presented in Figures **??**, **??**, **??**, and **??**, and in Table **??**. The experiments were performed with COCO [**?**], version 1.0.1, the plots were produced with version 1.0.4.

5.1 Comparison of isotropic and derandomized mutation operators

Derandomized mutation performs better on the following functions: (initial variance is 5, mutation prob is 0.8, crossover is SBX-5 and Pr(0.8))

- Rosenbrock function 28.
- Function 23
- Much better on functions 26 & 34
- Function 2

By using $\alpha = 80$ and offsprings 30, better results were obtained for the following functions:

- function 13 (as well as function 12)
- function 16
- function 19
- 2-moderate 4-multi-modal: no good results in function group overall except for a slight improvement with this configuration on function 25

Function 15: Isotropic : Variance of 5 better than variance of 10, Derandomized: Smaller population as good as isotropic with small variance

1-separable 2-moderate (3d): low mutation (0.1) is better for functions 12 and 13 and/or high recombination probability (>=0.8) is bad (why?) smaller initial variance (3) is better for f13

1-separable 4-multi-modal : high variance (10) is bad

1-separable 5-weakly-structured function 18: High recombination probability (0.7-0.9) High mutation probability (around 0.7-0.8) (along with isotropic pop 100) High variance is bad

2-moderate 2-moderate: Variance is a critical factor. Mutation with probability 1 without adaptation of the variance gives worse results. Function 20 gives good results with mut0.3 in 5d - best mut0.3 recomb0.7 28-2d as good for pop100 offs20 mut0.1 recomb0.7 var5.0 as for pop100 offs20 mut0.8 recomb0.6

2-moderate 5-weakly-structured Better results with low mutation probability (0.1-0.3) Function 33: good results with relatively higher variance. Improved for a smaller population and moderate variance 26, 33: best with mut0.8 recomb0.6

3-ill-conditioned 3-ill-conditioned: variance is not the determinating factor. Isotropic a little worse.

Function 42: Derandomized and/or Higher population are better than pop=60

(3-ill-conditioned 5-weakly-structured: High variance doesn't hurt w.r.t a smaller one. - Can't say much though because the problems are unstructured and hard) f44-5d: mut0.8 recomb0.7 var5.0derandomized sbx2

4-multi-modal 4-multi-modal : Derandomized improves a tiny bit on isotropic. 4-multi-modal 5-weakly-structured: Function 48: high variance does not perform worse, on other functions of derandomized is better

Functions 54, 55: Can't say much. Derandomized/Smaller variance better.

Cases in which low mutation probability works: 2-moderate 5-weakly-structured: best is 0.3 3-ill-conditioned 5-weakly-structured: function 44 has good results

For 5-d space: Function 53 performs very well with pop80, mut0.8, recombination1.0, derandomized On the other hand it performs well with low mutation probability (0.1-0.3) on 3d Find solutions really quick with var 3.

5.2 aRT

ibea pop100 offs20 mut0.1 recomb0.7 var5.0derandomized sbx5 max gen200 on bbob-biobj budget1000xD gets to 10-3 on f12

5-D

Δf	1e+0	1e-1	1e-2	1e-3	1e-4	1e-5	#succ
$\mathbf{f_1}$	1267(1266)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_2}$	2120(3385)	11232(10007)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_3}$	1728(1546)	23316(21518)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_4}$	1	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_5}$	1	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_6}$	739	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_7}$	1	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_8}$	4435(10920)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_9}$	578(721)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{10}}$	4307(3797)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{11}}$	691(1390)	3795(4612)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{12}}$	3382(3568)	10798(3296)	22565(13923)	23834(51896)	∞	∞*5100*	0/5
$\mathbf{f_{13}}$	1766(6329)	7352(6140)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{14}}$	1763(2532)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{15}}$	4021(9828)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{16}}$	1267	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{17}}$	7596(16455)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{18}}$	1634(1784)	11347(10126)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{19}}$	3895(3482)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{20}}$	7596(10126)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{21}}$	7596(10126)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{22}}$	842(2102)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{23}}$	1	22875(37972)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{24}}$	7596(7594)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{25}}$	1267(3797)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{26}}$	9300(10565)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{27}}$	2953(4942)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{28}}$	3376(6329)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{29}}$	3376(7594)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{30}}$	582(1452)	23196(21518)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{31}}$	1267(1266)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{32}}$	4990(6217)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{33}}$	2962(2265)	24652(27846)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{34}}$	4553(7212)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{35}}$	1267(3797)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{36}}$	639	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{37}}$	1267(2532)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{38}}$	3376(7594)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{39}}$	460	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{40}}$	2447(1921)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{41}}$	1267(2532)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{42}}$	1267(2532)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{43}}$	7596(10126)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{44}}$	1	23759(21518)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{45}}$	7596(8860)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{46}}$	3376(5063)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{47}}$	2265(7327)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{48}}$	8838(9482)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{49}}$	9998(10061)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{50}}$	822(1026)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{51}}$	1	23876(25315)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{52}}$	3376(6329)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{53}}$	934(2333)	24374(17720)	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{54}}$	3494(2439)	∞	∞	∞	∞	∞*5100*	0/5
$\mathbf{f_{55}}$	3694(4402)	∞	∞	∞	∞	∞*5100*	0/5

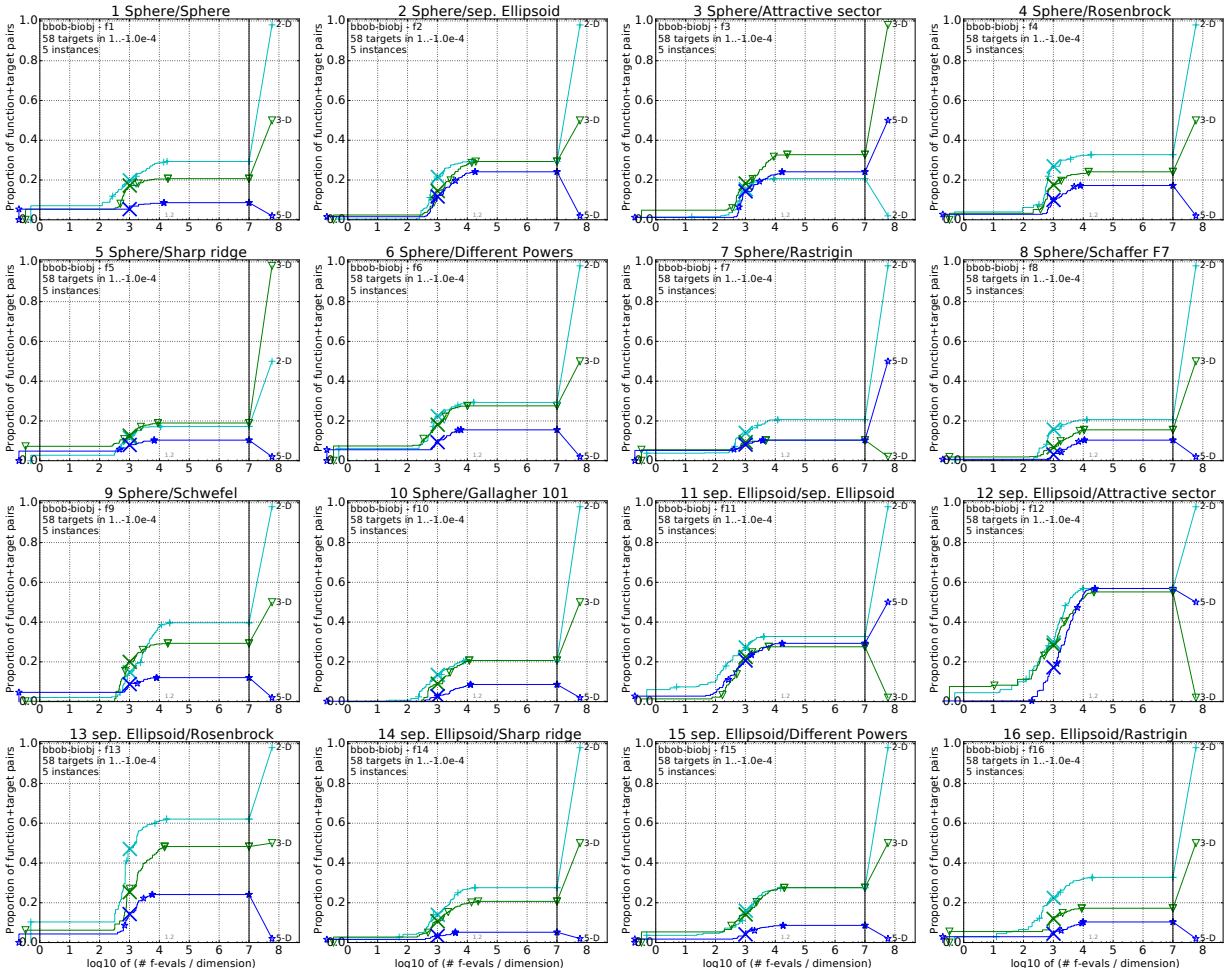

Figure 1: Empirical cumulative distribution of simulated (bootstrapped) runtimes in number of objective function evaluations divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots, 10^{-0.1},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

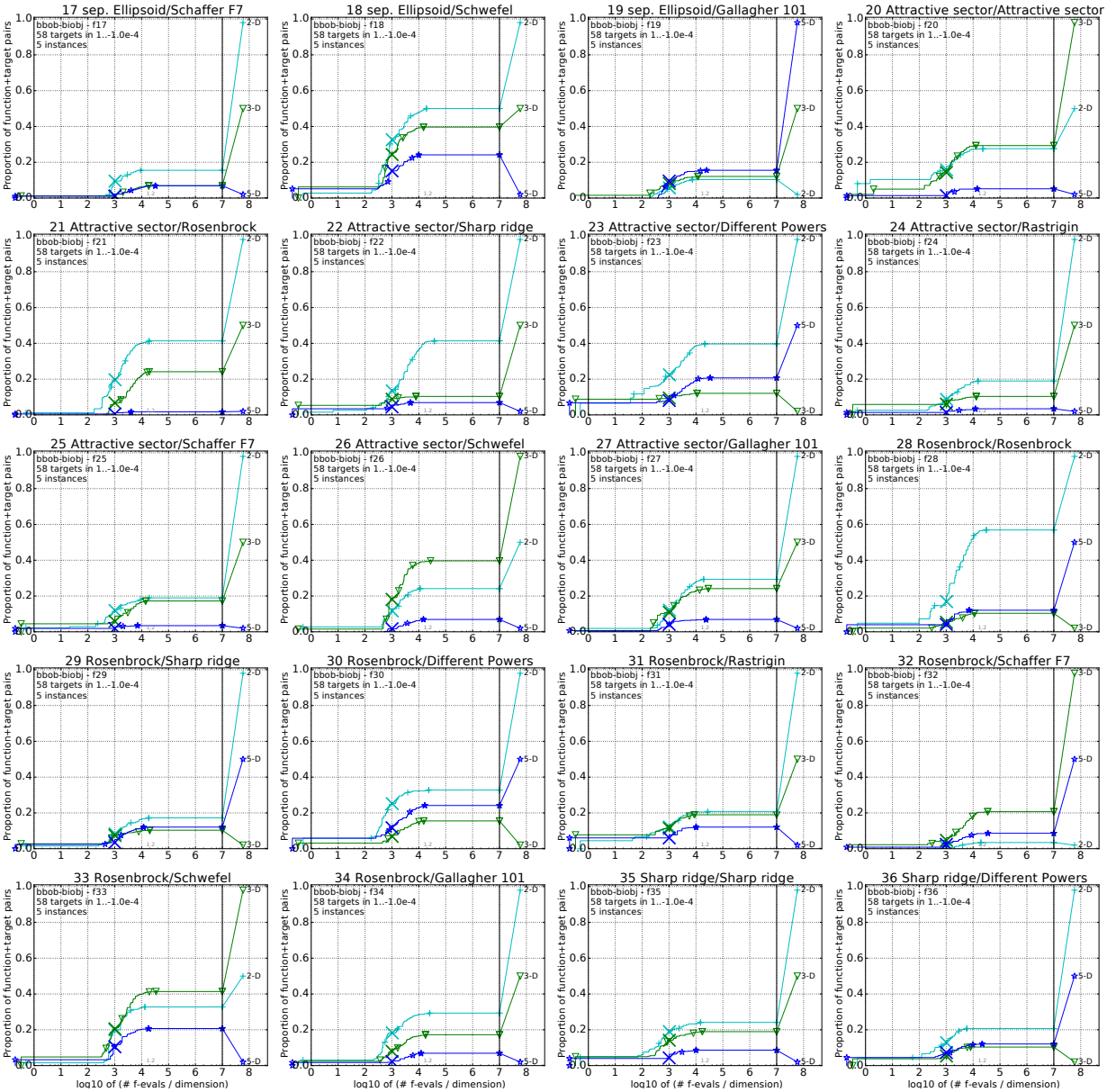

Figure 2: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. ?? for functions f_{17} to f_{36} and all dimensions.

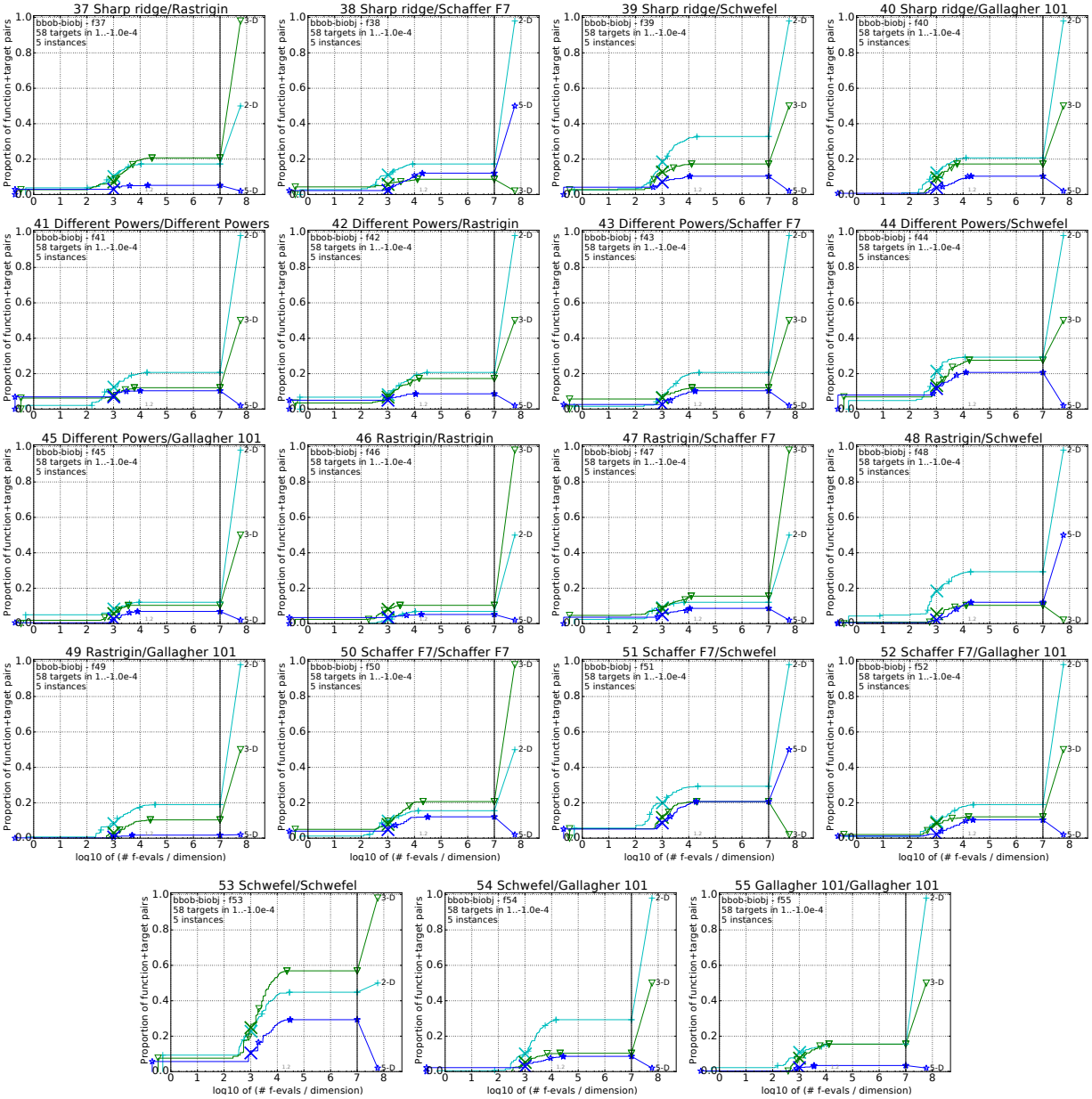

Figure 3: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. ?? for functions f_{37} to f_{55} and all dimensions.

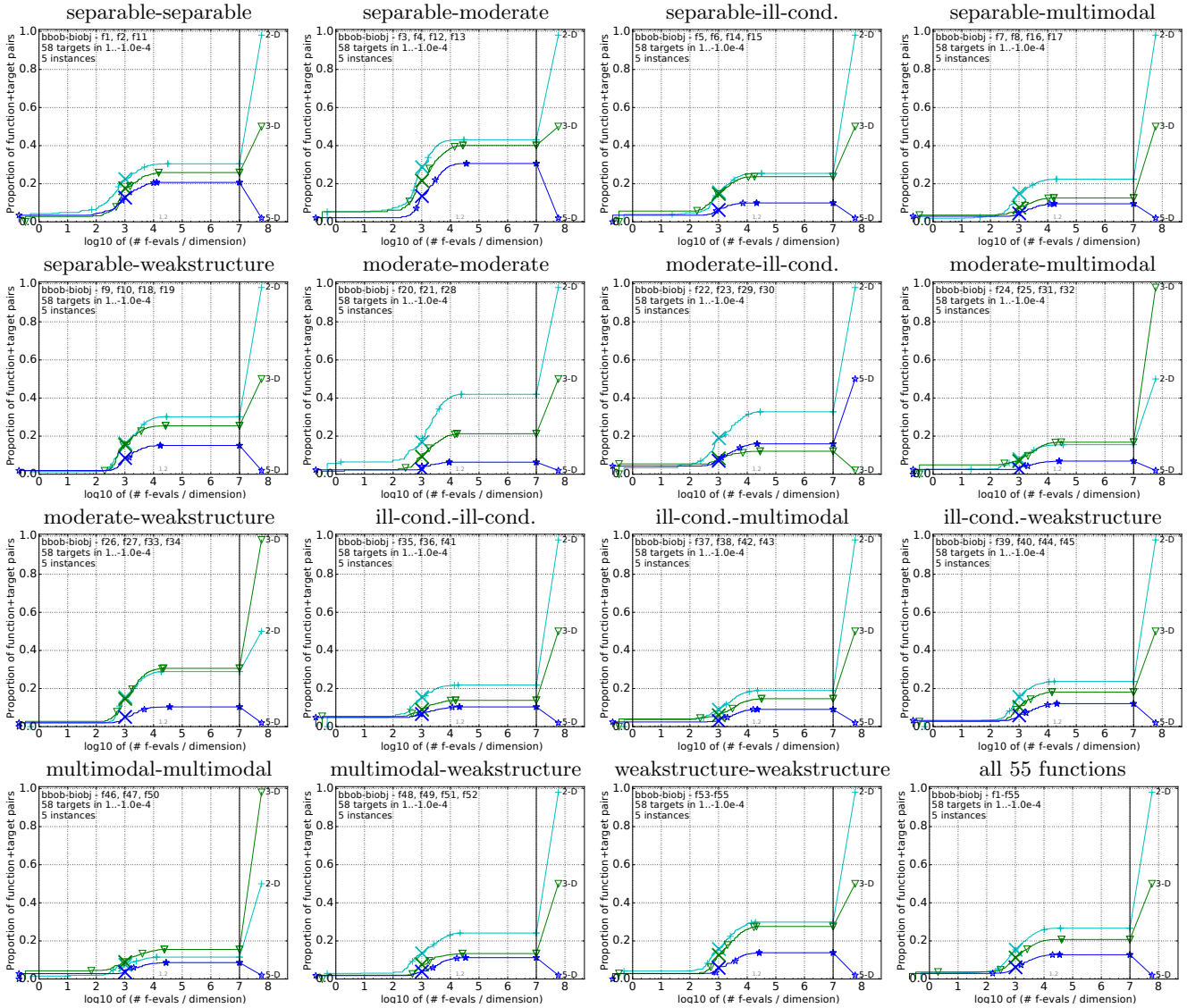

Figure 4: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots,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

APPENDIX

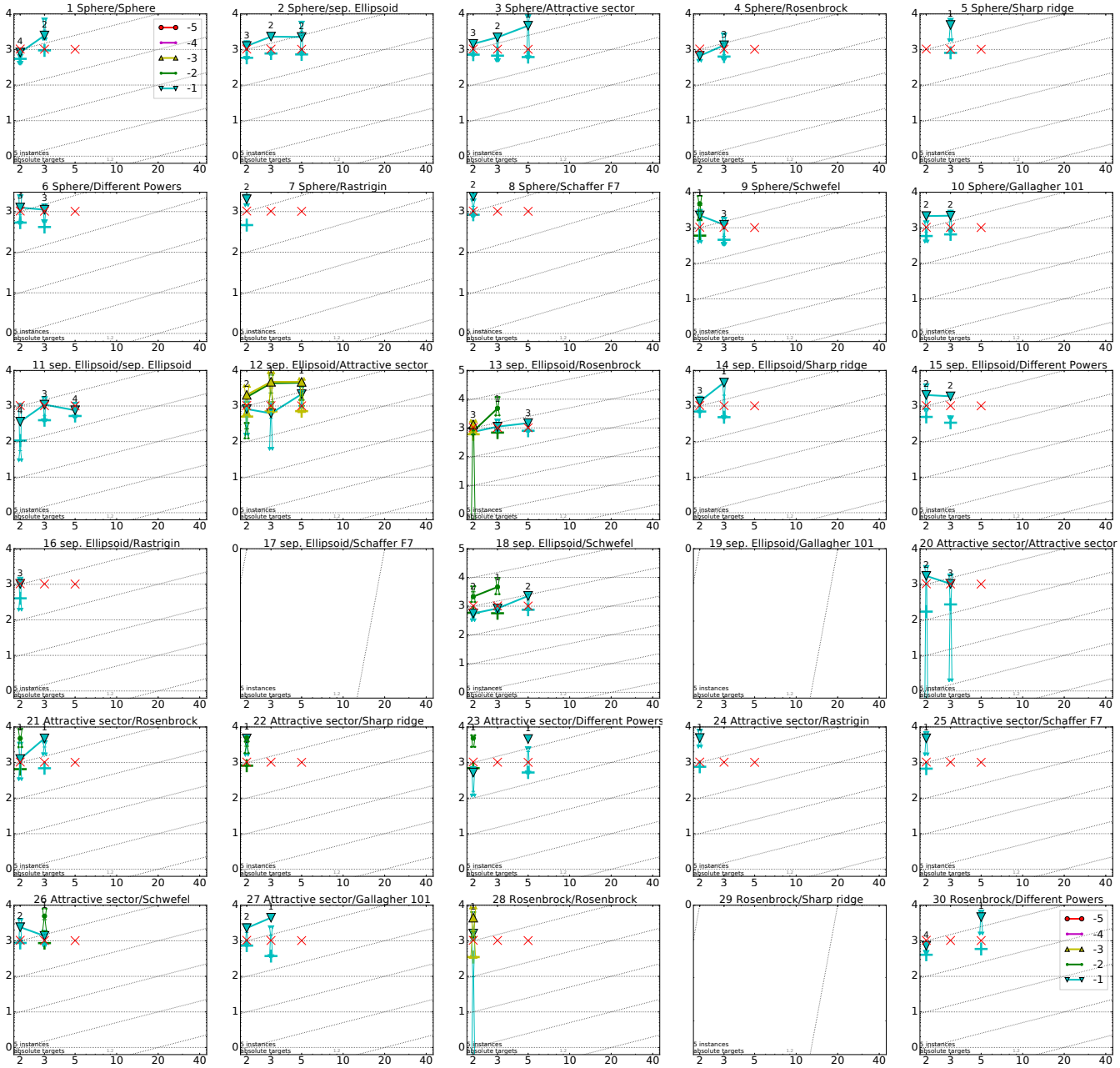

Figure 5: Scaling of runtime to reach $HV_{\mathrm{ref}} + 10^{\#}$ with dimension; runtime is measured in number of f-evaluations and $\#$ is given in the legend; Lines: average runtime (aRT); Cross (+): median runtime of successful runs to reach the most difficult target that was reached at least once (but not always); Cross ($\times$): maximum number of f-evaluations in any trial. Notched boxes: interquartile range with median of simulated runs; All values are divided by dimension and plotted as $\log_{10}$ values versus dimension. Numbers above aRT-symbols (if appearing) indicate the number of trials reaching the respective target. Horizontal lines mean linear scaling, slanted grid lines depict quadratic scaling.

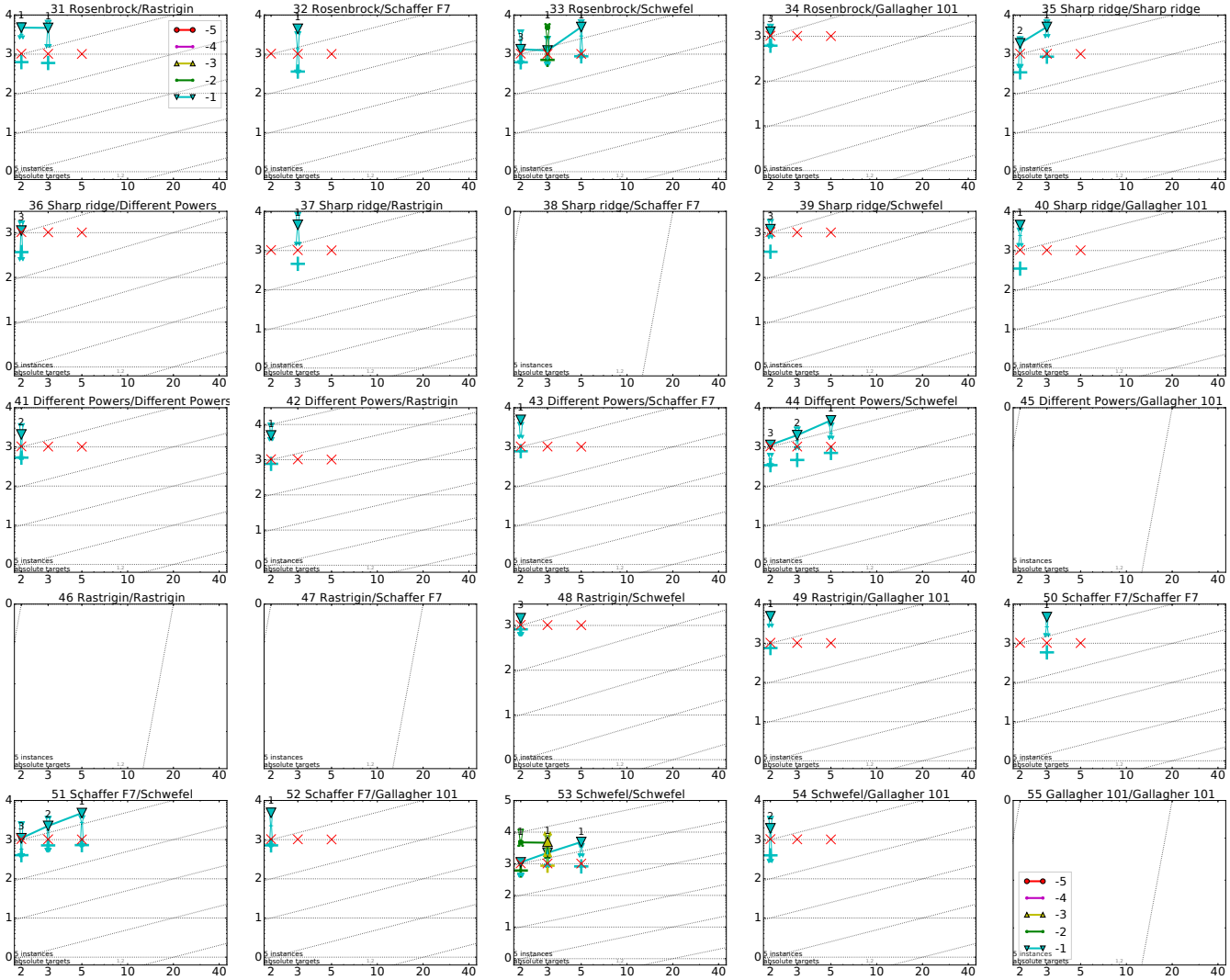

Figure 6: Runtime versus dimension as described in Fig. ??, here for functions f_{31} to f_{55}.