Benchmarking of the Indicator-based evolution algorithm using the Bi-Objective BBOB Test Suite

Final report *

Karim Kouki Ahmed Mazari Daro Ozad
Mihaela Sorostinean Aris Tritas

ABSTRACT

This project set as objectives to study, implement and benchmark the Indicator Based Evolutionary Algorithm (IBEA) using the Comparing Continuous Optimizer (COCO) platform. IBEA uses as performance measure for the optimization goal an indicator of the relative quality of two sets of solution vectors with respect to the Pareto front. Our group's focus was on the epsilon indicator. The report is outlined as follows: first we present the specificity of the indicator used, as well as the thought process behind the evolution strategy. Then we describe our experimental methodology along with the results obtained for different parametrizations and variation operators. Finally, those results are discussed and compared with related multi-objective approaches.

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.

In our work we proposed a python implementation of the IBEA algorithm based on the steps presented by the authors in (paper nr). We then used this implementation to benchmark the algorithm for various categories of problems through the COCO platform. Section 3 of this paper provides a brief description of the IBEA algorithm. In section 4 we present the implementation steps as well as other considerations regarding it, while in section 5 we detail the experiments we have conducted. Finally section 6 presents the results of these experiments and following discussions, while section 7 provides conclusions and ideas for future work.

2. EPSILON INDICATOR

Give the formula of the epsilon indicator and the fitness, get into a little more detail about the dominance preserving relation and the values of the fitness function (to explain why it needs to be scaled as well).

3. 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

*Submission deadline: October 21st.

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

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 P' 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. 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.

4. IMPLEMENTATION

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.

Python dictionary, grid search

5. EXPERIMENTS

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.

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.

We intended to:

1. Experiment with self-adaptive strategies for the mutation step, and

2. Implement Self-Adaptive SBX which seeks an optimal index for the approximation distribution used

6. EXPERIMENTS

The proposed implementation of the IBEA algorithm was tested with the collection of benchmark problems comprised in the COCO platform. We performed several experiments with various combinations of parameters in order to evaluate the influence of specific parameters on the results for different categories of benchmark problems. Therefore we varied the following parameters:

- The budget - various values between 300, 500, 750, 850, 1000
- Population size - varied between 50, 60, 80, 100, 200
- Number of offspring - 20, 30 or 40
- Mutation probability - 0.7, 0.8, 0.9, 1.0
- Recombination probability - or probability of crossover for the recombination step - 0.4, 0.6, 0.8, 0.9, 1.0
- Variance - 1.3 , 2.5, 5, 10, 15
- Offspring : - 20, 30, 35, 40
- Dimension : 2, 5, 10, 20, 40
- Max iterations: 10^6, 10^9
- Distribution index for the SBX operator: 2, 5 or 20 (the value suggested by the authors)
- Operators : Isotropic, Derandomized

Since the number of variable parameters was considerable, carrying experiments with all combinations between them was impossible, so we selected a limited number of combinations which we considered might give significant results. The sets of parameters that we used in our experiments are illustrated in table 1. Running a bi-objective algorithm in a laptop is not an easy task. ItâĂŹs is both time and RAM consuming. The execution takes from 5 to 23 hours to finish with a budget of 1000. The timing experiment is intrinsically associated with the size of the population and the dimension. For that reason, we tried only one configuration for 40 dimensions that took almost 48h with a moderate budget of 300. In order to try several parameterizations of the algorithm we were obliged to make non stopping running of the algorithm on different machine with different parameter in order to evaluate how the algorithm performs under different parameterizations. We conclude from this empirical study that a high variance and low variance impacts negatively the performance of the algorithm. The trick is to find a trade-off between the size of population, variance and the recombination values.

6.1 Recombination

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.

In this case, the Simulated Binary Operator was chosen for the recombination part.

6.2 Variation

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. a polynomial distribution was used for the mutation part.

7. CPU TIMING

(Ubuntu 16.04 64 bits i5 and i7, MAC os, Windows) In order to evaluate the CPU timing of the algorithm, we have run the IBEAwith restarts on the entire bbob-biobj test suite [4] 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 $1.7e{-}3$, $1.9e{-}3$, $2.0e{-}3$, $2.2e{-}3$, $2.4e{-}3$ seconds respectively.

8. RESULTS

Results of IBEA from experiments according to [3] and [1] on the benchmark functions given in [4] are presented in Figures 1, 2, 3, and 4, and in Table **??**. The experiments were performed with COCO [2], version 1.0.1, the plots were produced with version 1.0.4.

8.1 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.

8.2 aRT

We observe with no surprise that our algorithm has a runtime of $1e{-}3$ iff the variance is adapted. gets to 10-3 on f12

We also observe that using a better indicator tends to find targets much faster.

9. RELATED WORK

Comparison with results stemming from the same approach was performed, and to an extent is it quite revealing. For lack of time, no comparison was done with the state-of-the-art. However, using the tools provided by COCO as well as the datasets shared by other groups, and after comparison with the implementation of IBEA-ϵ in the C language as well as results of groups studing IBEA-HV. We can safely say that the ϵ indicator may be overly simplistic for the wide range of optimization functions benchmarked.

Conclusion

Covariance matrix adaptation, different indicator function.

10. REFERENCES

[1] D. Brockhoff, T. Tušar, D. Tušar, T. Wagner, N. Hansen, and A. Auger. Biobjective performance assessment with the coco platform. *arXiv preprint arXiv:1605.01746*, 2016.

[2] 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.

[3] N. Hansen, T. Tusar, O. Mersmann, A. Auger, and D. Brockhoff. Coco: The experimental procedure. *arXiv preprint arXiv:1603.08776*, 2016.

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

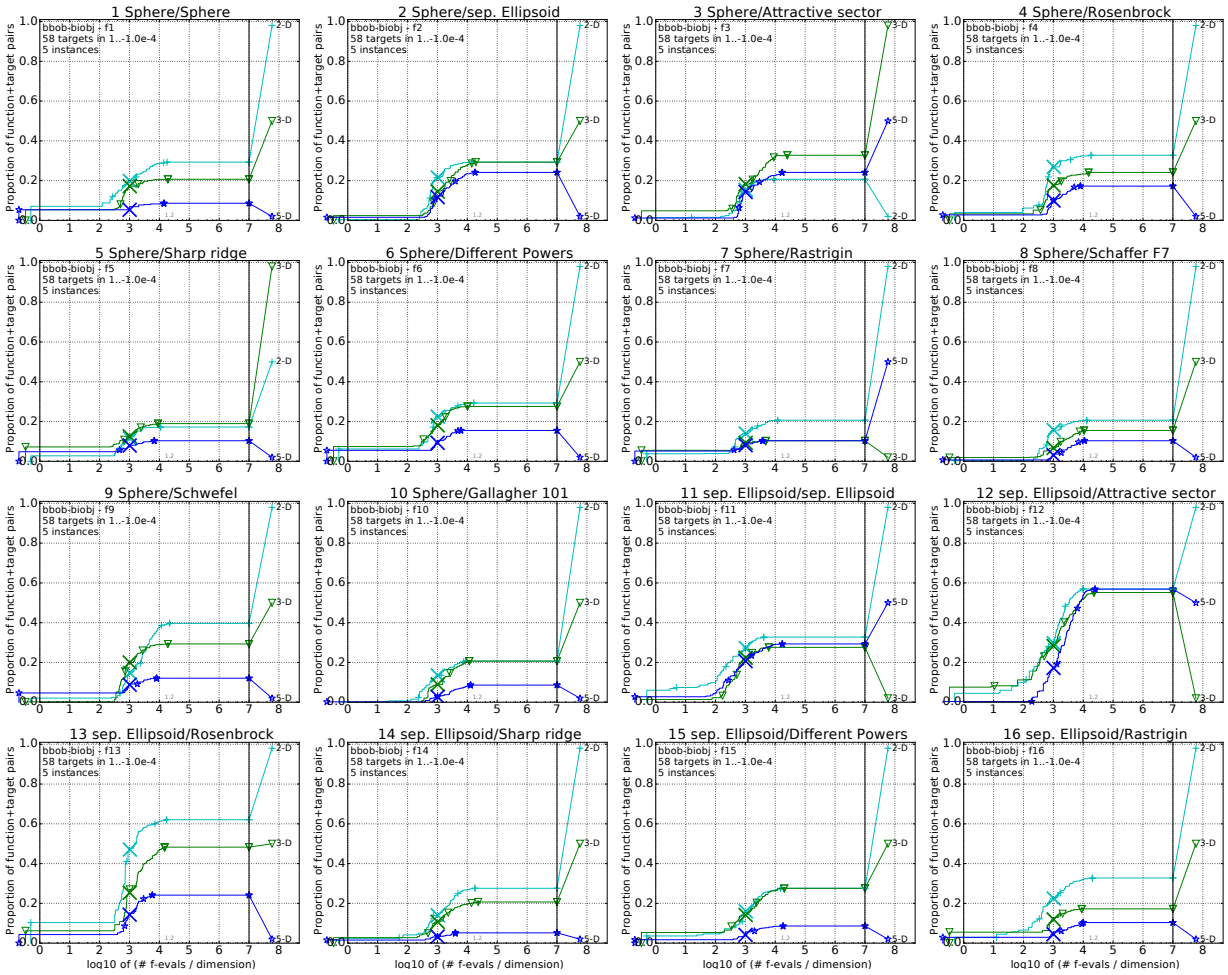

Figure 1: Empirical cumulative distribution of simulated (bootstrapped) runtimes in number of objective function evaluations divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots, 10^{-0.1},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

[4] T. Tusar, D. Brockhoff, N. Hansen, and A. Auger. Coco: The bi-objective black box optimization benchmarking (bbob-biobj) test suite. *arXiv preprint arXiv:1604.00359*, 2016.

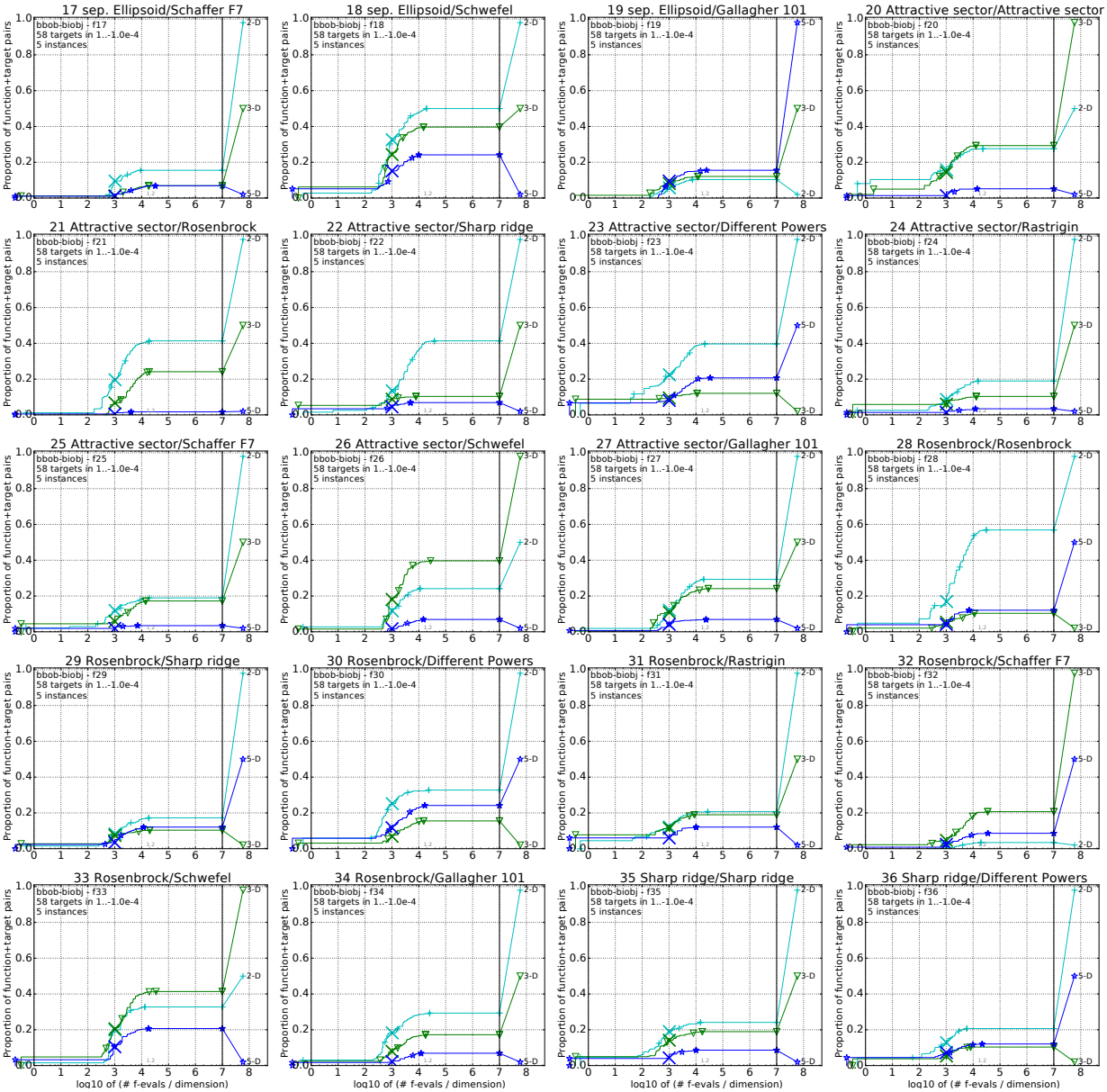

Figure 2: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 1 for functions f_{17} to f_{36} and all dimensions.

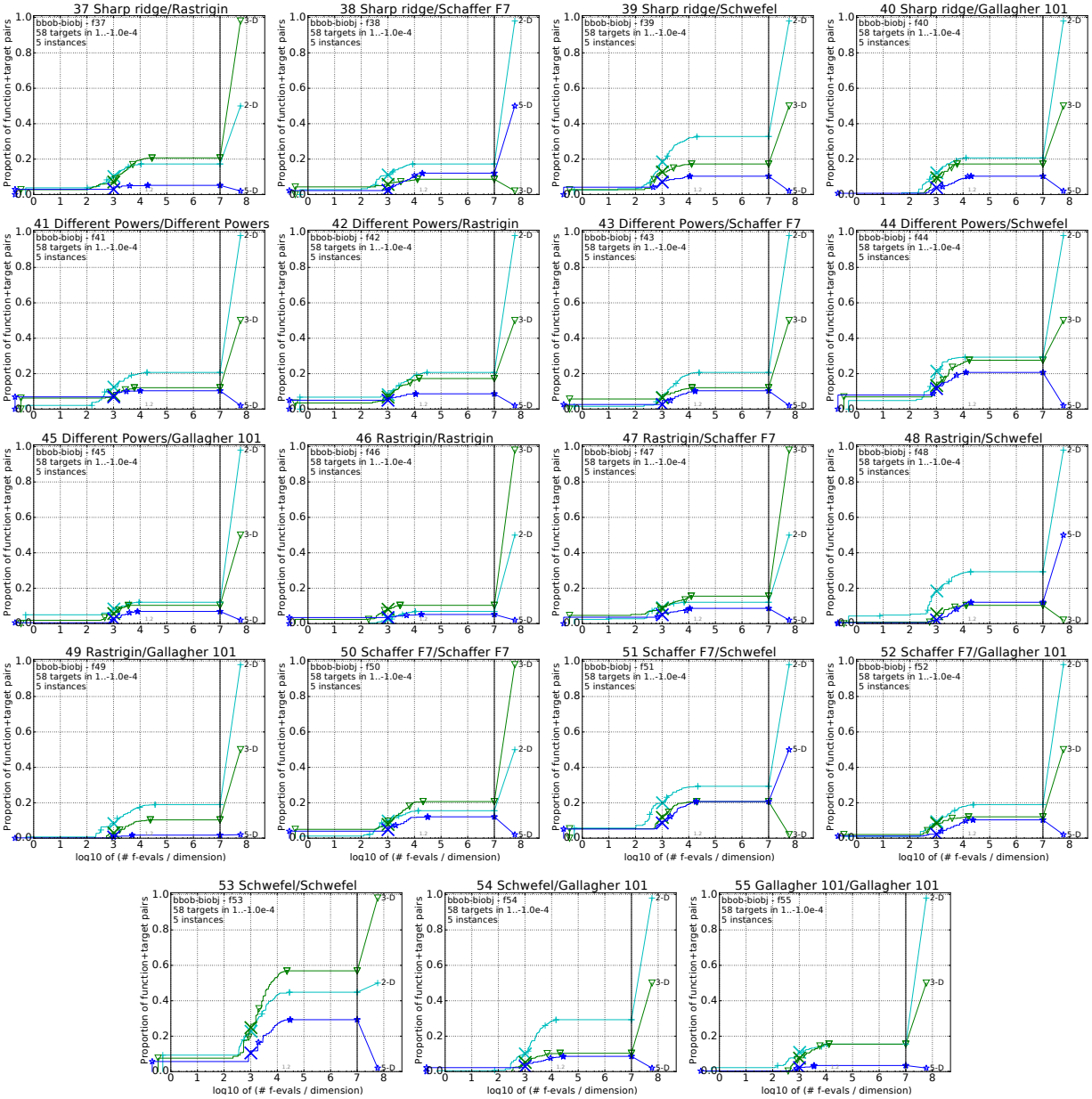

Figure 3: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the targets as given in Fig. 1 for functions f_{37} to f_{55} and all dimensions.

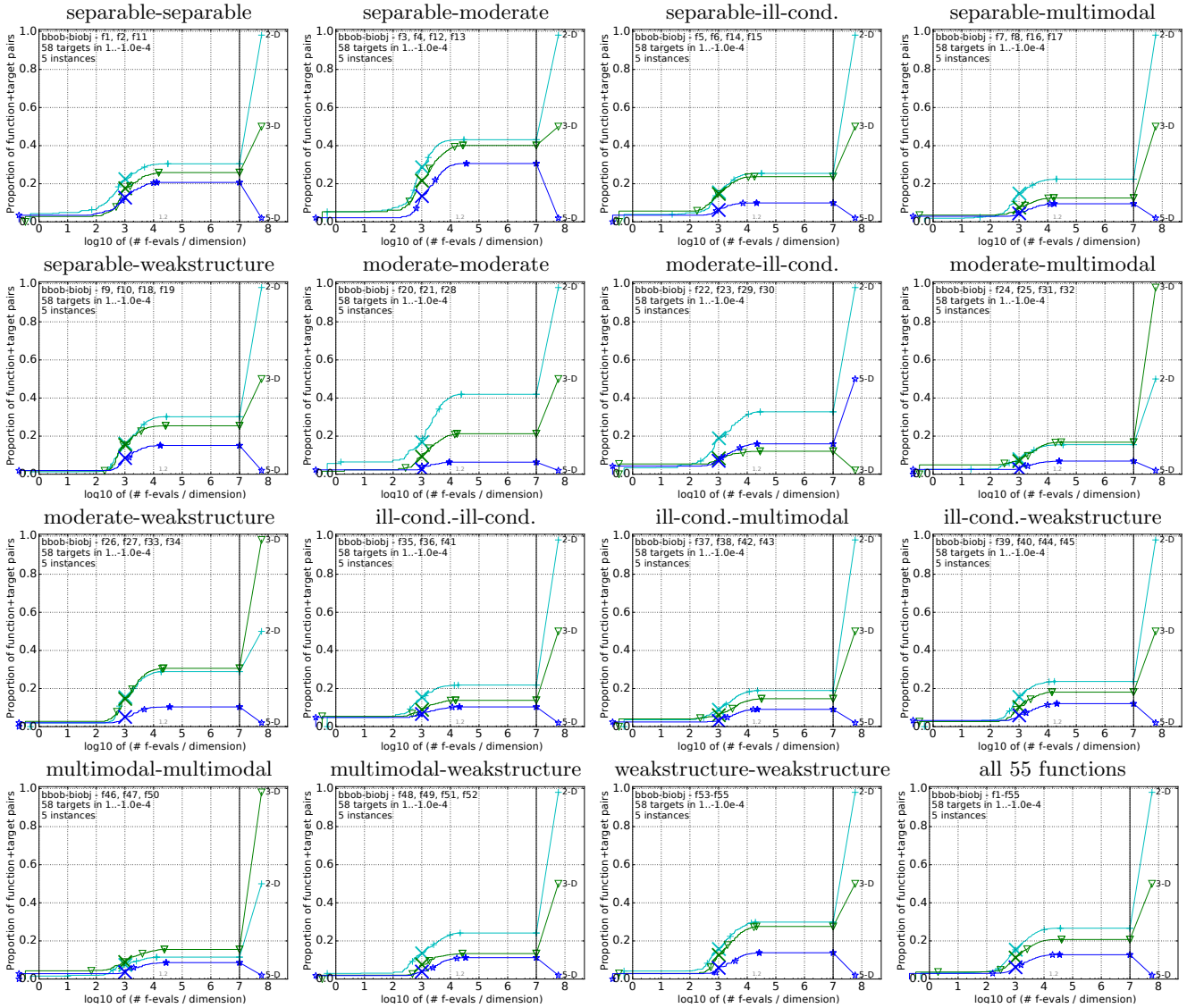

Figure 4: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots,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

APPENDIX

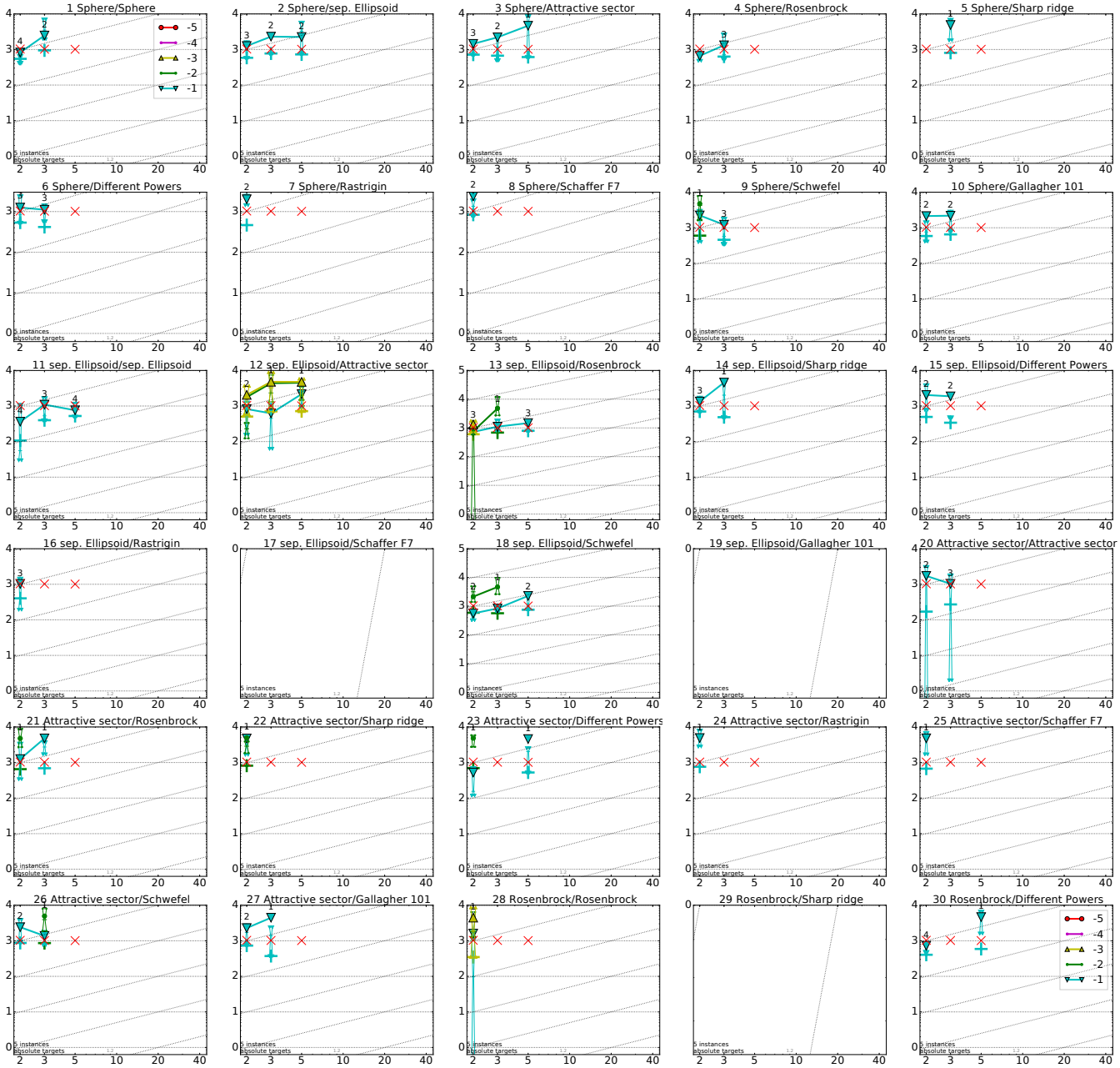

Figure 5: Scaling of runtime to reach $HV_{\mathrm{ref}} + 10^{\#}$ with dimension; runtime is measured in number of f-evaluations and $\#$ is given in the legend; Lines: average runtime (aRT); Cross (+): median runtime of successful runs to reach the most difficult target that was reached at least once (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.

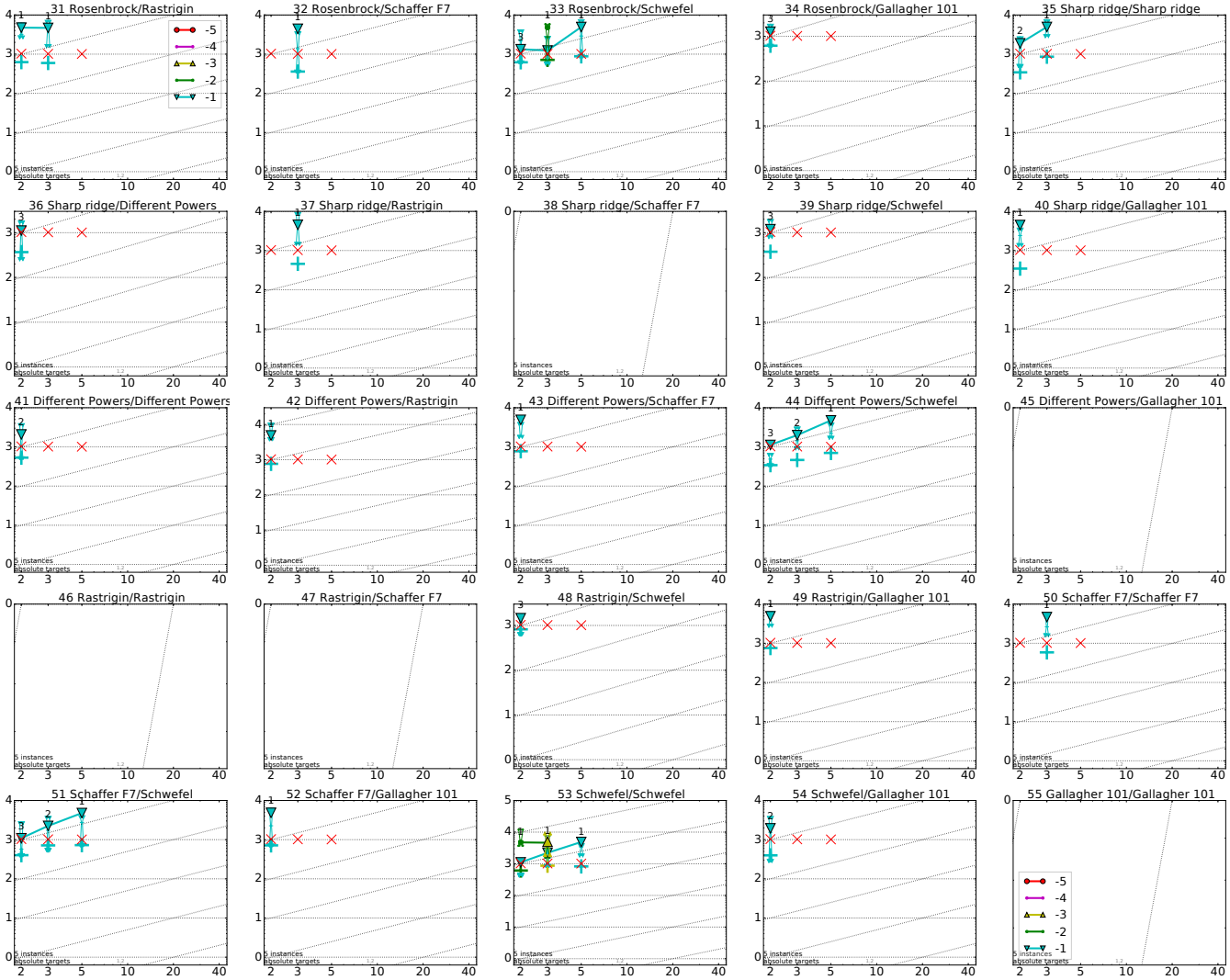

Figure 6: Runtime versus dimension as described in Fig. 5, here for functions f_{31} to f_{55}.