Black-Box Optimization Benchmarking of the Multiobjective Optimizer adaptive IBEA (ϵ -indicator) with the COCO platform

Martin BAUW
Robin DURAZ
Jiaxin GAO
Hao LIU
Luca VEYRON-FORRER

ABSTRACT

This paper discusses the implementation of a multiobjective evolutionary algorithm (MOEA). We implemented the Multiobjective Optimizer IBEA (indicator-based evolutionary algorithm) with ϵ-indicator [9] in Python and benchmarked it using the COCO platform[7]. In addition, we compared the results obtained with this algorithm with those of NSGA 2 and Random Search.

KEYWORDS

Evolutionary algorithm, Benchmarking, Black-box optimization, Bi-objective optimization, Decision Vector

1 INTRODUCTION

The adaptive IBEA we implement is dedicated to the following improvements: it requires no diversity preservation techniques within its population evolution mechanism, and it aims at taking into account arbitrary preference information thanks to its ϵ additive binary quality indicators. It operates on a set of candidates in a decision space, which will be modified thanks to selection and variation mechanisms.

The parallel with evolution can be reminded here: selection in our algorithm could be associated with the competition for reproduction and resources in nature, while variation illustrates the ability of existing living creatures to create new living beings thanks to genetic recombination and mutation. Note that due to the randomness of the variation process some individuals may not be transformed in the evolutionary mechanism.

Since we are in the case of multiobjective optimization, it is possible that several optimal objective vectors co-exist: they would be trade-offs between the different objectives we are pursuing. As we will discover in the algorithm description, our IBEA could, as any general stochastic search algorithm, be divided into three main elements: a working memory (the population of decision space candidates), a selection module, and a variation module. Among the differences with single-objective optimization, the working memory can here consider several solutions at a time. In single-objective optimizatin no mating selection is required and variation translates into modifying the current solution candidate.

None, ,
None. ACM ISBN None.
https://doi.org/None

1.1 Why binary quality indicators ?

Unary quality measures have been proved to be theoretically limited: they do not allow to determine whether a Pareto set approximation is better than another one. This limitation is still valid for a finite combination of unary quality measures. Binary quality indicators overcome part of the limitations and can, for certain binary indicators, indicate whether a Pareto approximation set is better than another one. [10]

2 DESCRIPTION OF ALGORITHM

2.1 Description of the objects associated with the algorithm

The algorithm input consists of decision space vectors $x^i \in X$, an objective space Z and objective functions $f^j : X \to Z$. We suppose that $X \subseteq \mathbb{R}^l$ with $l \in \{2, 3, 5, 10, 20, 40\}$ and $Z \subseteq \mathbb{R}^2$.

The output of a MOEA is a set of incomparable decision vector, meaning that no member of the output set dominates another one. Domination between decision vectors is defined as follows: a decision vector x^1 is said to dominate another decision vector x^2 ($x^1 \succ x^2$), if $f_i(x^1) \leq f_i(x^2) \forall i \in \{1, ..., n\}$ and $\exists j \in \{1, ...n\}$ for which $f_j(x^1) < f_j(x^2)$.

This output will be our Pareto set approximation, and the space of Pareto set approximations will be noted as Ω. Our algorithm relies on a binary quality indicator $I : \Omega \times \Omega \to \mathbb{R}$ which associates a real number to k Pareto set approximations.

Our binary quality indicator is defined as the minimum distance by which a Pareto set approximation needs to be translated in each dimension in objective space so that another approximation (image of a decision space vector) is weakly dominated. The weak domination is defined as follows: decision vector x^1 weakly dominates x^2, written $x^1 \succeq x^2$, if x^1 dominates x^2 or the corresponding objective vectors are equal [9]. The mathematical translation of this definition consists in the following equation:

$$I_{\epsilon^+(A,B)} = min_\epsilon \{\forall x^2 \in B \, \exists x^1 \in A : f_i(x^1) - \epsilon \leq f_i(x^2) for \, i \in \{1, ..., n\}\} \quad (1)$$

2.2 Description of the algorithm adaptive IBEA

The algorithm steps is described in pseudo-code 1. The adaptive version of IBEA answers the potential issue of widely spread binary quality indicators values. Too spread out indicators values

complicate the task of determining a correct value for K, the scaling factor associated with our indicator. By adaptivaly scaling the binary quality indicators values back to a common $[-1;1]$ interval, we substantially suppress the need to adapt K to our different problems.

IBEA can be faster than other algorithms since it only compares pair of decision vectors rather than complete approximation sets.

2.3 Details regarding mating selection

Mating selection aims at picking promising solutions for variation and usually is performed in a randomized fashion. Perform binary tournament selection with replacement on P in order to fill a temporary mating pool (the variable $P\_$ in our implementation). The binary tournament consists in keeping the candidate with the best fitness value from the random pick.

2.4 Details regarding mutation

The mutation operator modifies individuals by changing small parts in the associated vectors according to a given mutation rate. The mutation rate determines how much of the considered population we will use to generate new decision space vectors. This process reminds how related evolutionary algorithms seem related to biological evolution, mimicking natural hereditary relationships. For mutation we use a polynomial mutation operator.

The operations involved in one single mutation are as follows. A random individual is picked from the population P. According to a uniform probability pick, the mutation is realized using one mathematical transformation applied to its coordinates in the decision space. Note that the transformation change from one coordinate to another, since the uniform probability pick is repeated for each. In our case, there are two forms of transformations on coordinates. In the following equations, u is uniformly picked in $[0,1]$, $ind[j]$ the $j-th$ coordinate of the already existing picked individual, $p_{mut}[j]$ the $j-th$ coordinate of the newly generated individual, Up the biggest existing coordinate value, Lo the lowest. The two latters are each defined for each decision space dimension.

- if the uniform probability pick ≤ 0.5:

$$\sigma_L = (2u)^{\frac{1}{\mu+1}} - 1 \tag{2}$$

$$p_{mut}[j] = ind[j] + \sigma_L(ind[j] - Lo) \tag{3}$$

- else:

$$\sigma_R = (2(1-u))^{\frac{1}{\mu+1}} \tag{4}$$

$$p_{mut}[j] = ind[j] + \sigma_R(Up - ind[j]) \tag{5}$$

This is implemented thanks to a loop on individuals, with a nested loop treating each decision space dimension coordinate. Mathematical formulas involving hyperparameters are based on [3].

2.5 Details regarding recombination

The recombination operator takes a certain number of parents and creates a predefined number of children by combining parts of the parents. For recombination we use a simulated binary crossover (SBX) operator. To mimic the stochastic nature of evolution, a crossover probability is associated with this operator. In our case, two parents are selected among the current population. A uniform

Algorithm 1 Adaptive IBEA

Input:
α (population size)
N (maximum number of generations)
K (fitness scaling factor)

Output:
A (Pareto set approximation)
Step 1. Initialization
generate initial population of size α
set generation counter m to 0
end Step
Step 2. Fitness assignment
for all objective function f_i **do**
lower bound $\underline{b_i} = min_{x \in P} f_i(x)$
upper bound $\overline{b_i} = max_{x \in P} f_i(x)$
end for
for all objective function f_i **do**
$f_i^{'}(x) = \frac{f_i(x) - \underline{b_i}}{\overline{b_i} - \underline{b_i}}$
end for
calculate all indicator values $I(x^1, x^2)$ with $f_i^{'}$
determine max. indicator $c = max_{x^1, x^2 \in P} |I(x^1, x^2)|$
for all $x^1 \in P$ **do**
$F(x^1) = \sum_{x^2 \in P \setminus \{x^1\}} -e^{-\frac{I(\{x^1\},\{x^2\})}{ck}}$
end for
end Step
Step 3. Environmental selection
while population P $\geq \alpha$ **do**
choose x^* such that $F(x^*) \leq F(x)$ for all $x \in P$.
remove x^* from the population
update remaining individuals fitness values, *ie* $\forall x \in P$:
$F(x) = F(x) + e^{-\frac{I(\{x^*\},\{x\})}{ck}}$
end while
end Step
Step 4. Termination
If $m \geq N$ or another stopping criterion then the output A is defined as the set of nondominated decision vectors in P
end Step
Step 5. Mating selection
Binary tournament selection with replacement on P in order to fill a temporary mating pool P'
end Step
Step 6. Variation
Apply recombination and mutation operators to P'
Add the resulting offspring to P
Increment the counter ($m = m + 1$) and go to Step 2
end Step

probability pick in $[0, 1]$ written u determines the parameter used in computing the features (decision space coordinates) of the children. In the following equations, $child0 - 1[j]$ is the $j - th$ coordinate of the generated child decision vector, $parent0 - 1[j]$ the $j - th$ coordinate of the parent associated with the child.

- if the uniform probability pick ≤ 0.5:

$$\beta_q = (2u)^{\frac{1}{\mu+1}} \tag{6}$$

- else:

$$\beta_q = (\frac{1}{2(1-u)})^{\frac{1}{\mu+1}} \tag{7}$$

Thanks to this stochastic parameter we can compute the children's coordinates:

- first child:

$$child0[j] = 0.5((1+\beta_q)parent0[j] + (1-\beta_q)parent1[j]) \tag{8}$$

- second child:

$$child1[j] = 0.5((1-\beta_q)parent0[j] + (1+\beta_q)parent1[j]) \tag{9}$$

Regarding the hyperparameter μ, formulas were taken from [4]. In the article this parameter is referred to as η_c. A large η_c implies an offspring close to the parents in coordinates. For a smaller η_c, children solutions tend to differ more from their parents. This parameter is therefore essential to controlling the spread of the offspring.

3 DESCRIPTION OF IMPLEMENTATION

3.1 Asymptotic analysis

Pour nous permettre de faire fonctionner l'algorithme dans un temps de décent il faut analyser les complexités de chaque étape, posons, I le nombre d'époques de notre algorithme, P la population, d la dimention de l'espace, do le nombre de fonction objectif, r le taut de recombination, et v le taut de mutation, de plus la fin d'une époque on se obtient une population de taille $O(P(1+r+v(1+r))) = O(P)$ ceci ne change pas notre analyse asymptotique.

- step 1 : a un temps d'execution en $O(P * d)$ c'est la construction du nouvel ensemble
- step 2 : $O(P^2 * d)$ il faut P^2 opération pour calculer touts les $I(x_1, x_2)$ pour la fonction F et pour calculer $c = max|I|$
- step 3 : si P est représenté sous forme d'un ensemble (en hashmap) cette étape prend $O(P)$
- step 4 : s'exécutre en temps constant
- setp 5 : s'exécutre en $O(P)$
- step 6 : s'exécutre en $O(P * d)$

Nous remarquons que le nombre d'éléments choisis

Nous considérons le temps d'appel à la fonction I_ϵ en temps do car elles est utilisée uniquement élément par élément : $I_\epsilon(x_1, x_2)$, dans ce cas la le calcul revient à un calcul de max.

3.2 Code acceleration (non asymptotic)

Les améliorations suivantes ont permis de gagner un facteur 10 dans le temps d'exécution global de notre code. Avec les outils de profiling de python nous avons observé que la fonction la plus appelé était I_ϵ et de loin. Améliorer son temps de plusieurs manières possible nous a été grandement utilse. Pour cela que dans un premier temps nous avons chercher à ne jamais la recalculer cette fonction et de la stoker dans une hashmap. Puis poru améliorer le temps d'exeucution de I_ϵ nous avons recodé la fonction max en dur, au lieu d'utiliser celle fourni par les bibiotheques. Puis dans un souci d'otimisation nous avons fourni une implémentation pour uniquement 2 coordonés, effectivement les tests sur lesquells nous allons faire fonctionner notre algorithmme sont bbob-biobj ce qui nous a permis de gagner un temps conséquent.

Les fonctions objectif étant fournies par l'exterieur, et ne sachant rien à priori sur leurs temps d'exécution, nous avons aussi préféré ne jamais demander plus d'une fois leur valeurs, en les stoquant dans une hashmap.

4 CPU TIMING

In order to evaluate the CPU timing of the algorithm, we have run the IBEA algorithm with restarts on the entire bbob-biobj test suite [8] for $2D$ function evaluations according to [7]. The Python code was run on a Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz with a quad core CPU having 16GB of RAM. The options were, for dimensions 2, 3, 5, 10, 20, batch 1 on 3 running alone, then batch 2 and 3 on 3 running at the same time. For 40 dimensions, we ran it alone on the whole test suite.
Parameters for the algorithm were :

- Population size : 100
- Maximum number of generation : 100
- Scaling factor : 0.05
- Mutation rate : 0.01
- Recombination and mutation mu : 1
- Population initialization in range (-5, 5)

Mean times for function evaluation were as described in table 1.

Table 1: Mean times for function evaluation

Batch \ Dimension	2	3	5
Batch 1 on 3	6.0e-04	6.3e-04	8.1e-04
Batch 2 and 3 on 3 run simultaneously	8.6e-04 8.3e-04	8.6e-04 8.4e-04	9.1e-04 8.9e-04

Batch \ Dimension	10	20
Batch 1 on 3	8.3e-04	1.1e-03
Batch 2 and 3 on 3 run simultaneously	1.1e-03 1.0e-03	1.3e-03 1.3e-03

Batch \ Dimension	40
Whole test suite	4.2e-03

5 DISCUSSION OF THE RESULTS

Results from experiments according to [7], [5] and [1] on the benchmark functions given in [8] are presented in Figures 3, 4 and 5. The experiments were performed with COCO [6], version 2.0, the plots were produced with version 2.0.

5.1 Results of our algorithm

We think that the results of our implementation of the adaptive IBEA algorithm needs not to be taken as it is. There are things that we couldn't do, because of time and equipment constraints, or we just failed to implement it. We couldn't use the suggested parameters of the paper, such as a maximum number of generations of 200. Going further with some parameters like the budget, the size of the initial population or the maximum number of generations would have probably improved our results. Furthermore, the initial population was randomly generated in the range -5 to 5, for which we failed to find a better solution. Nevertheless, the results of this implementation of an adaptive IBEA algorithm was still good, according to the benchmarking realized with the Coco platform. In the end, compared to the results of other algorithms visible in the coco archives at http://coco.gforge.inria.fr/ppdata-archive/bbob-biobj/2016-all/, our results are in the middle group, and comparatively better for higher dimensions than smaller ones.

5.2 Comparison with NSGA 2 and Random Search

NSGA-II is a very famous multi-objective optimization algorithm. It has three special characteristics: fast non-dominated sorting approach, fast crowded distance estimation procedure and simple crowded comparison operator. And it has been used successfully in lots of multi-objective optimization problems. We take the data obtained by NSGA-II algorithm from the following link: http://coco.gforge.inria.fr/data-archive/bbob-biobj/2016/NSGA-II-MATLAB_Auger_bbob-biobj.tgz

The basic idea of the Random search algorithm is to randomly select some points according to certain criteria, then put these points into the functions, and keep the best points from them for the next iteration. It is essentially a violent search mechanism, but when the number of evalutions is greatly increased, it can also achieve good results. We take the data obtained by Random search algorithm from the following link: http://coco.gforge.inria.fr/data-archive/bbob-biobj/2016/RANDOMSEARCH-5_Auger_bbob-biobj.tgz
The general results of the NSGA-II, random search algorithm and Adaptive IBEA are as seen in Figure 1 and Figure 2.

Table 2: Comparison of three algorithms

Algorithm \ Category	Log10(f-evals/ dimension)	Best for 2-D
Random search	6	0,62
NSGA-II	5	0,76
IBEA	2,4(40D)-3,6(2D)	0,58

Best for 40-D	Best evaluated function	Worst evaluated function
0,08	f13, f53	f37, f46
0,17	f19, f53	f22
0,12	f18, f53	f31, f47

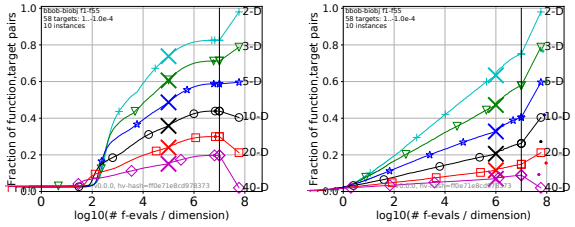

Figure 1: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for the two algorithms for all dimensions. NSGA-II is on the left, random search on the right.

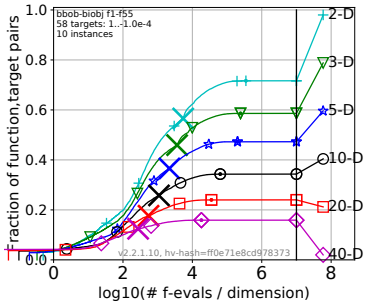

Figure 2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for all dimensions for Adaptive IBEA.

From the above table we can see that, in general, NSGA-II is the best algorithm, its evaluation is not too big, but it can get the best results. IBEA is also a good algorithm, it does not need too many evaluations, but it can also get good results. And according to the estimation of COCO, its best value can exceed Random search (if the algorithm does more evaluations). In contrast, the effect of Random search is not very good, it need too many evaluations to get a good value, and when the dimension of input becomes higher, the algorithm is not so effective. From the last two columns of the table, we can see that different algorithms have different optimization capabilities for each function, and f53 is friendly to all the algorithms.

6 CONCLUSION

In this paper, an indicator-based evolutionary algorithm (IBEA) specifically with ϵ indicator was studied and implemented. Since it can be adapted to the preferences of the user and does not require any additional diversity preservation mechanism. We benchmarked it with COCO platform, which enables us to compare the algorithm's performance with other well-knowing multiobjective evolutionary algorithms (MOEAs). In our experiments, we majorly focused on the comparison with NSGA-II and random search. In the original paper of IBEA, the author revealed that IBEA has been shown to generate significantly better results on six of eight benchmark problems in comparison to NSGA-II and SPEA2. However, we did not have a result as well as theirs. Overall, in our expriment IBEA outperforms Random search. A relatively good Pareto set

approximations was given by IBEA. But IBEA performs less well as NSGA-II. Nevertheless, this could be biased. Since potential factors that may impact performance need to be taken into account. Another way of initialization the original population, other methods of implementing mutation and recombination (other operators are used), or just abundant computing power, and all other hyperparameters impact directly algorithm's performance.

Implementing a well behaved optimisation algorithm is a very empirical and challenging work. It may take a lot expert's insight and years of experience. One problem that really confused us is about the choice of mutation index parameter (η_m). In the paper[2], Deb and Agrawal suggested that a value $\eta_m \in [20, 100]$ is adequate in most problems that they tried. To our surprise, when we changed η_m from 25 (by default) to 1, our result improved significantly. We do believe that further refinement of these hyperparameters will lead to a much desired result.

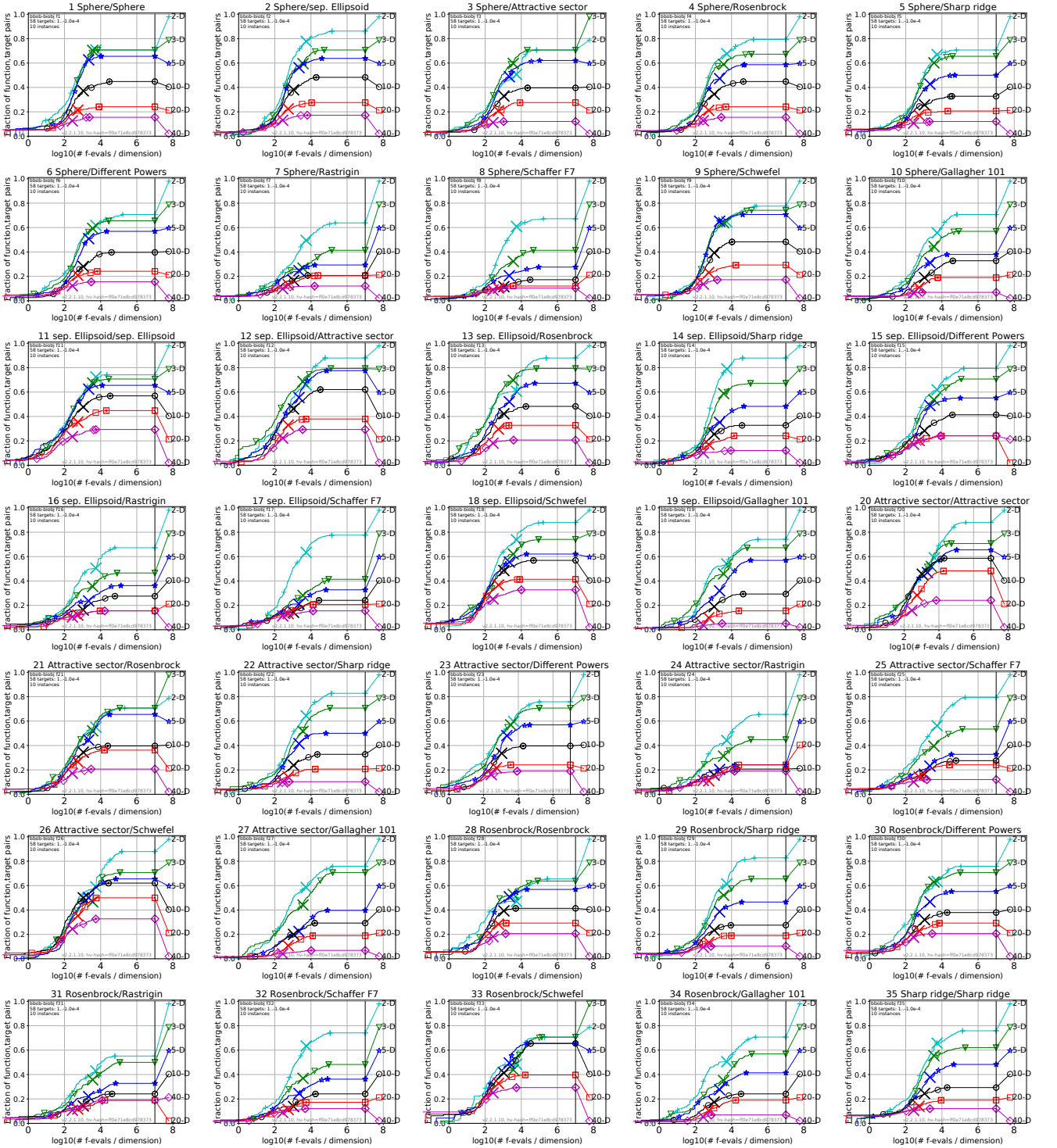

Figure 3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for functions f_1 to f_{35} in all dimensions.

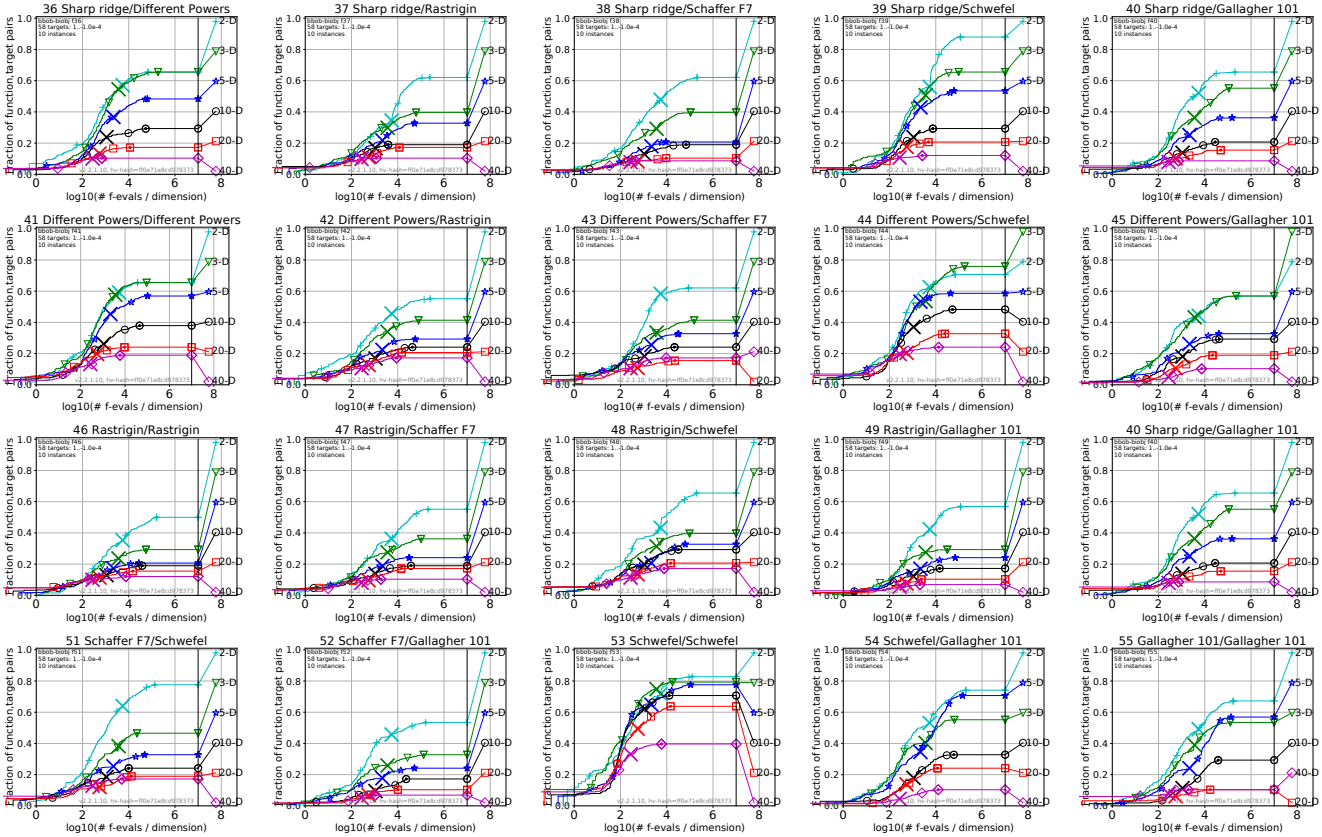

Figure 4: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) as in Fig. 3 but for functions f_{36} to f_{55} in all dimensions.

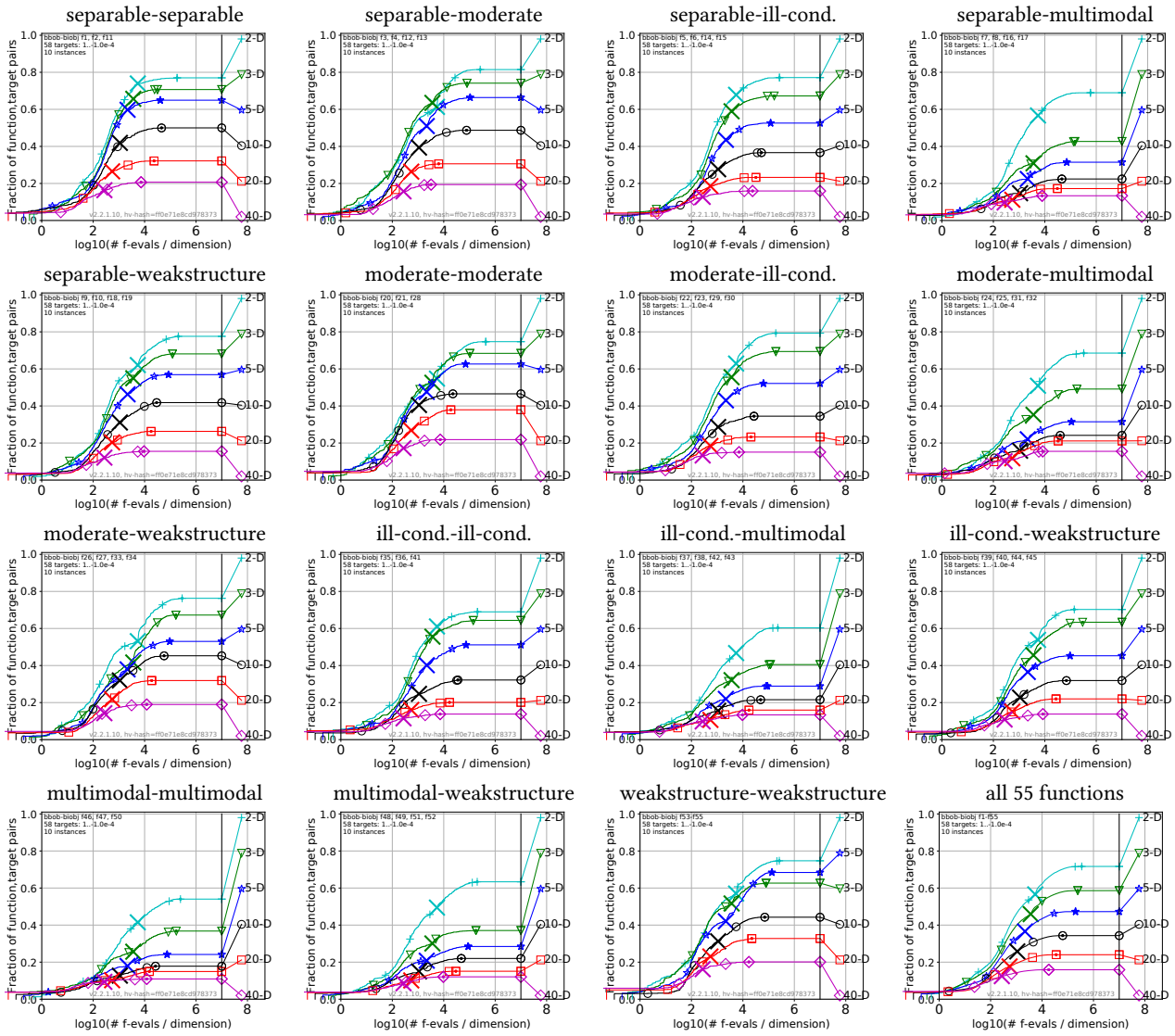

Figure 5: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 58 targets $\{-10^{-4}, -10^{-4.2}, -10^{-4.4}, -10^{-4.6}, -10^{-4.8}, -10^{-5}, 0, 10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots,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

REFERENCES

[1] D. Brockhoff, T. Tušar, D. Tušar, T. Wagner, N. Hansen, and A. Auger. 2016. Biobjective Performance Assessment with the COCO Platform. *ArXiv e-prints* arXiv:1605.01746 (2016).

[2] K. Deb and S. Agrawal. 1999. A niched-penalty approach for constraint handling in genetic algorithms. ICANNGA, 235–243.

[3] K. Deb and D. Deb. 2014. Analyzing Mutation Schemes for Real-Parameter Genetic Algorithms. *International Journal of Artificial Intelligence and Soft Computing* (2014).

[4] K. Deb, S. Karthik, and T. Okabe. 2007. Self-Adaptive Simulated Binary Crossover for Real-Parameter Optimization. *Genetic and Evolutionary Computation Conference* (2007).

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

[6] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *ArXiv e-prints* arXiv:1603.08785 (2016).

[7] N. Hansen, T. Tušar, O. Mersmann, A. Auger, and D. Brockhoff. 2016. COCO: The Experimental Procedure. *ArXiv e-prints* arXiv:1603.08776 (2016).

[8] T. Tušar, D. Brockhoff, N. Hansen, and A. Auger. 2016. COCO: The Bi-objective Black-Box Optimization Benchmarking (bbob-biobj) Test Suite. *ArXiv e-prints* arXiv:1604.00359 (2016).

[9] E. Zitzler and S. Künzli. 2004. Indicator-Based Selection in Multiobjective Search. *International conference on parallel problem solving from nature* (2004).

[10] E. Zitzler, M. Laumanns, and S. Bleuler. 2004. A Tutorial on Evolutionary Multiobjective Optimization. *Metaheuristics for Multiobjective Optimization* (2004).