Anytime Performance Assessment: Runtime Distributions Beyond Data Profiles

Nikolaus Hansen
Inria & École polytechnique, France

June 2024

OPTIMIZATION METHODS & SOFTWARE
2021, VOL. 36, NO. 1, 114–144
https://doi.org/10.1080/10556788.2020.1808977

Taylor & Francis
Taylor & Francis Group

COCO: a platform for comparing continuous optimizers in a black-box setting

Nikolaus Hansen[a,b], Anne Auger[a,b], Raymond Ros[c], Olaf Mersmann[d], Tea Tušar[e] and Dimo Brockhoff[a,b]

[a]Randopt Team, Inria, Palaiseau, France; [b]CMAP, CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, France; [c]LRI, Université Paris-Sud, Orsay, France; [d]Computational Statistics, TU Dortmund University, Dortmund, Germany; [e]Jožef Stefan Institute, Ljubljana, Slovenia

ABSTRACT

We introduce COCO, an open-source platform for Comparing Continuous Optimizers in a black-box setting. COCO aims at automatizing the tedious and repetitive task of benchmarking numerical optimization algorithms to the greatest possible extent. The platform and the underlying methodology allow to benchmark in the same framework deterministic and stochastic solvers for both single and multiobjective optimization. We present the rationals behind the (decade-long) development of the platform as a general proposition for guidelines towards better benchmarking. We detail underlying fundamental concepts of COCO such as the definition of a problem as a function instance, the underlying idea of instances, the use of target values, and runtime defined by the number of function calls as the central performance measure. Finally, we give a quick overview of the basic code structure and the currently available test suites.

ARTICLE HISTORY
Received 17 June 2019
Accepted 8 August 2020

KEYWORDS
Numerical optimization; black-box optimization; derivative-free optimization; benchmarking; performance assessment; test functions; runtime distributions; software

1. Introduction

We consider the continuous black-box optimization or search problem to minimize

$$f : X \subset \mathbb{R}^n \to \mathbb{R}^m \quad n, m \geq 1, \tag{1}$$

where the search domain X is typically a bounded hypercube or the entire continuous space.[1] More specifically, we aim to find, as quickly as possible, one or several solutions $\boldsymbol{x}$ in the search space X with *small* value(s) of $f(\boldsymbol{x}) \in \mathbb{R}^m$.

A continuous optimization algorithm, denoted as *solver*, addresses the above problem. In this paper, we only consider zero-order black-box optimization [19,57,58]: while the search domain $X \subset \mathbb{R}^n$ and its boundaries are accessible, no other prior knowledge about f is available to the solver.[2] That is, f is considered as a black-box, also known as an oracle, and the *only* way the solver can acquire information on f is by querying the value $f(\boldsymbol{x})$ of a solution $\boldsymbol{x} \in X$. Zero-order black-box optimization is thus a derivative-free optimization setting.[3] We generally consider 'time' to be the number of calls to the function f and will define 'runtime' correspondingly.

Anytime Performance Assessment in Blackbox Optimization Benchmarking

Nikolaus Hansen, Anne Auger, Dimo Brockhoff, and Tea Tušar

***Abstract*—We present concepts and recipes for the anytime performance assessment when benchmarking optimization algorithms in a blackbox scenario. We consider runtime—oftentimes measured in the number of blackbox evaluations needed to reach a target quality—to be a universally measurable cost for solving a problem. Starting from the graph that depicts the solution quality versus runtime, we argue that runtime is the *only* performance measure with a generic, meaningful, and quantitative interpretation. Hence, our assessment is solely based on runtime measurements. We discuss proper choices for solution quality indicators in single- and multi-objective optimization, as well as in the presence of noise and constraints. We also discuss the choice of the target values, budget-based targets, and the aggregation of runtimes by using simulated restarts, averages, and empirical cumulative distributions which generalize convergence graphs of single runs. The presented performance assessment is to a large extent implemented in the comparing continuous optimizers (COCO) platform freely available at https://github.com/numbbo/coco.**

***Index Terms*—Anytime optimization, benchmarking, blackbox optimization, performance assessment, quality indicator.**

I. Introduction

We present practical concepts and ideas for the performance assessment of optimization algorithms when benchmarked in a blackbox and anytime scenario. Going beyond a simple ranking of algorithms, we aim to provide a *quantitative* and *meaningful* performance assessment, which allows for conclusions like *algorithm A is seven times faster than algorithm B* in solving a given problem or in solving problems with certain characteristics. To achieve this end in a comparative and timeless manner, we argue that we should measure the *number of blackbox evaluations* to reach a predefined *quality indicator* value (a target). More generally, we argue to measure a cost that is defined on a ratio scale and is comparable across publications. We call this measure the *runtime* of the algorithm to reach a given target. Yet, our assessment methodology does not depend on any specific cost measure, as long as the costs are quantitative and comparable.[1]

In this article, we formalize the optimization goal by a so-called quality indicator. Its definition may heavily depend on the optimization scenario, e.g., the number of objectives or constraints. Broadly speaking, a quality indicator is based on the sequence of all so-far visited solutions. In the simplest case, it is the objective function value of the last visited solution.

Runtimes represent the cost of optimization. Compared to the quality indicator, the definition of costs depends to a lesser extent on the specific optimization scenario. To sustain reproducibility and comparability across publications, we recommend against CPU or wall-clock time as cost measure[2] (see also Hooker [22] for a further discussion on the unwanted consequences of benchmarking based on CPU time).

Benchmarking is usually computationally expensive and benchmarking for a *single* budget seems vastly inefficient by 1) addressing only one of many possible budget scenarios (scenarios heavily depend on the software and hardware environment) and 2) throwing away most of the data generated during the experiment. An anytime approach to benchmarking prevents these drawbacks. To allow for a budget-free performance assessment even for non-anytime algorithms that have a maximum or timeout budget as decisive or mandatory *input* parameter (decided by the user), we collect data with an any-budget *experimental procedure* that runs repeated experiments with increasing input budget [31].[3] Non-anytime algorithms that do not take a maximum budget as an input parameter can be accurately assessed only by the time of their final solution proposal.

In this article, we advocate to routinely use (anytime) *empirical runtime distributions* to assess the performance of optimization algorithms. We demonstrate how to directly compare the runtime distributions of algorithms that have

Manuscript received 18 September 2021; revised 16 February 2022 and 16 June 2022; accepted 14 September 2022. Date of publication 29 September 2022; date of current version 1 December 2022. This work was supported in part by the French National Research Agency (NumBBO) under Grant ANR-12-MONU-0009, and in part by the Slovenian Research Agency under Grant P2-0209 and Grant N2-0254. *(Corresponding author: Dimo Brockhoff.)*

Nikolaus Hansen, Anne Auger, and Dimo Brockhoff are with Inria and Ecole Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau, France (e-mail: dimo.brockhoff@inria.fr).

Tea Tušar is with the Department of Intelligent Systems, Jožef Stefan Institute, 1000 Ljubljana, Slovenia.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TEVC.2022.3210897.

[1]We are grateful to the anonymous reviewer pointing this out to us.

[2]An exploratory CPU timing experiment to get an estimate of the internal time complexity of the algorithm is still advisable, like it is prescribed in the comparing continuous optimizer (COCO) platform [20].

[3]We can stop the procedure when the last budget was not fully exhausted. Increasing the budget each time by a factor of $r > 1$ adds to the overall computational costs for the experimentation less than $\sum_{i=0}^{\infty} 1/r^i = r/(r-1)$ times the last consumed budget. For the performance assessment, always the data from the smallest eligible budget is used. The performance assessment will be too optimistic by tacitly assuming that the budget can be set properly without additional costs. On the other hand, runtimes may be overestimated (by less than a factor of r). A code example is provided in

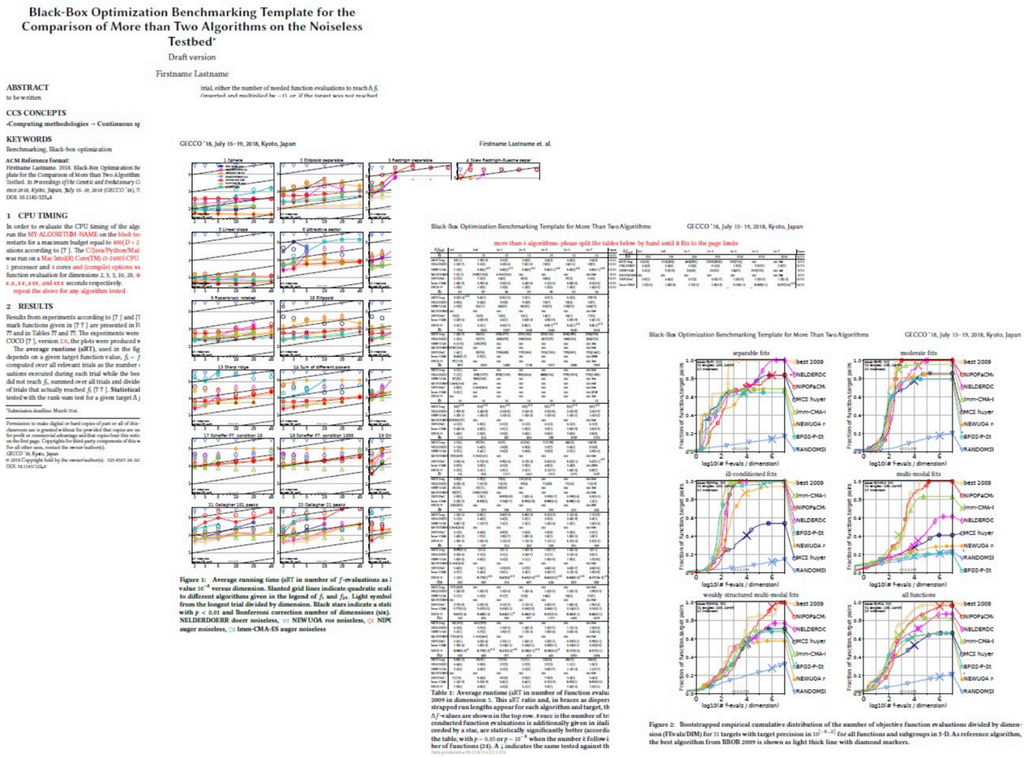
Overview

- The COCO platform
- Data profiles are runtime distributions
- What to aggregate?
- Target f-values
- Integrating out (low) success rates
- Runtime distributions versus convergence graphs

Benchmarking is a tedious and repetitive task

with surprisingly many traps and pitfalls

That's where COCO comes into play: semi-automatized benchmarking


Black-Box Optimization Benchmarking Template for the Comparison of More than Two Algorithms on the Noiseless Testbed*

Draft version

Firstname Lastname

ABSTRACT

CCS CONCEPTS

KEYWORDS

1 CPU TIMING

2 RESULTS

GECCO '18, July 15–19, 2018, Kyoto, Japan

Firstname Lastname et. al.

Figure 1: Average running time (aRT in number of f-evaluations as l

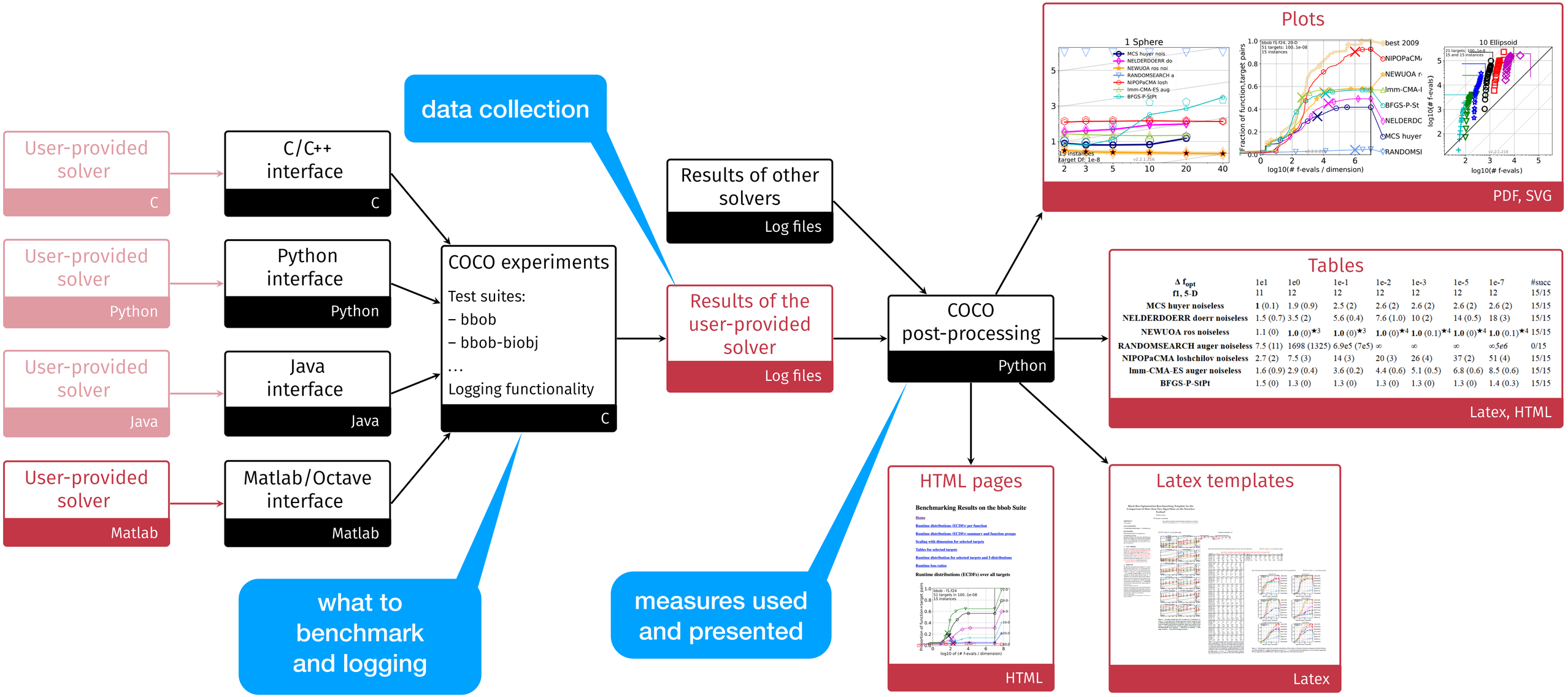
Table 1: Average runtime (aRT in number of function evalu

Figure 2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimen-

It becomes easy to give a comprehensive quantitative *context* of the presented work by using previous data...

https://dl.acm.org/doi/pdf/10.1145/1830761.1830790
https://github.com/numbbo/coco

COCO/bbob



Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f1, 5-D	11	12	12	12	12	12	12	15/15
MCS huyer noiseless	**1** (0.1)	1.9 (0.9)	2.5 (2)	2.6 (2)	2.6 (2)	2.6 (2)	2.6 (2)	15/15
NELDERDOERR doerr noiseless	1.5 (0.7)	3.5 (2)	5.6 (0.4)	7.6 (1.0)	10 (2)	14 (0.5)	18 (3)	15/15
NEWUOA ros noiseless	1.1 (0)	**1.0** (0)$^{\star 3}$	**1.0** (0)$^{\star 3}$	**1.0** (0)$^{\star 4}$	**1.0** (0.1)$^{\star 4}$	**1.0** (0)$^{\star 4}$	**1.0** (0.1)$^{\star 4}$	15/15
RANDOMSEARCH auger noiseless	7.5 (11)	1698 (1325)	6.9e5 (7e5)	∞	∞	∞	*∞5e6*	0/15
NIPOPaCMA loshchilov noiseless	2.7 (2)	7.5 (3)	14 (3)	20 (3)	26 (4)	37 (2)	51 (4)	15/15
lmm-CMA-ES auger noiseless	1.6 (0.9)	2.9 (0.4)	3.6 (0.2)	4.4 (0.6)	5.1 (0.5)	6.8 (0.6)	8.5 (0.6)	15/15
BFGS-P-StPt	1.5 (0)	1.3 (0)	1.3 (0)	1.3 (0)	1.3 (0)	1.3 (0)	1.4 (0.3)	15/15

Figure by Tea Tušar in Hansen et al (2021), COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1), 114-144.

Important Characteristics of Benchmarking with COCO/bbob

- Functions are instantiated multiple times with
  - different locations of the optimum in x- and f-space
  - different search space rotations
  - several monotonous local transformations to obfuscate simple regularities

  making inadvertent exploitation of artificial regularities (aka overfitting) much less likely

- Functions are (generally) scalable with dimension

- Performance evaluation via (generalized) data profiles

  first and foremost

- Provision of ~300 data sets to directly compare with

Important Characteristics of Benchmarking with COCO/bbob

- Functions are instantiated multiple times with
  - different locations of the optimum in x- and f-space
  - different search space rotations
  - several monotonous local transformations to obfuscate simple regularities

  making inadvertent exploitation of artificial regularities (aka overfitting) much less likely

- Functions are (generally) scalable with dimension

- Performance evaluation via (generalized) data profiles

  first and foremost

- Provision of ~300 data sets to directly compare with

COCO (COmparing Continuous Optimizers) Impact Numbers

Hansen et al 2021. OMS…

Table 1. Visibility of COCO.

Data sets online	`bbob` suite	227
	`bbob-noisy` suite	45
	`bbob-biobj` suite	32
	`bbob-largescale` suite	11
	`bbob-mixint` suite	4
BBOB workshop papers using COCO		143
Unique authors on the workshop papers		109 from 28 countries
Papers in Google Scholar found with the search phrase *'comparing continuous optimizers' OR 'black-box optimization benchmarking (BBOB)'*		559
Citations to the COCO documentation including [16,17,28,35–37,39–42]		1455
Citations to Hansen et al 2021. COCO: A platform for comparing… OMS 36,1.		491

Any `` `cocopp.archiving.create()` ``-ed data sets that are provided under an URL can be loaded and processed.

Data Profiles

Moré and Wild 2009. *SIAM J. OPTIM.* 20,1.

"data profiles […] have been designed to provide […] the percentage of problems that can be solved (for a given tolerance τ) with a given number of function evaluations [μ_f]."

1. Create a set of recorded sequences f_t, $t = 1,2,\ldots,\mu_f$

   for different algorithms and different problems/functions

2. Define for each problem a *target* f-value based on the data (aka *convergence test*)

   based on the smallest observed f-value f_L and a tolerance parameter $\tau \in \{10^{-1},10^{-3},10^{-5},10^{-7}\}$:

   $$f_{\text{target}} = f_L + \tau \cdot (f(x_0) - f_L)$$

3. Find the "runtimes" t, as number of f-evaluations, when, for the first time, $f_t \leq f_{\text{target}}$ was met

   for all problems and each algorithm and each tolerance τ

4. Plot the empirical cumulative distribution of $\dfrac{t}{n+1}$ over all problems for each algorithm and each τ

   different tolerances τ are usually shown in different plots

Moré and Wild 2009. *SIAM J. OPTIM.* 20,1.

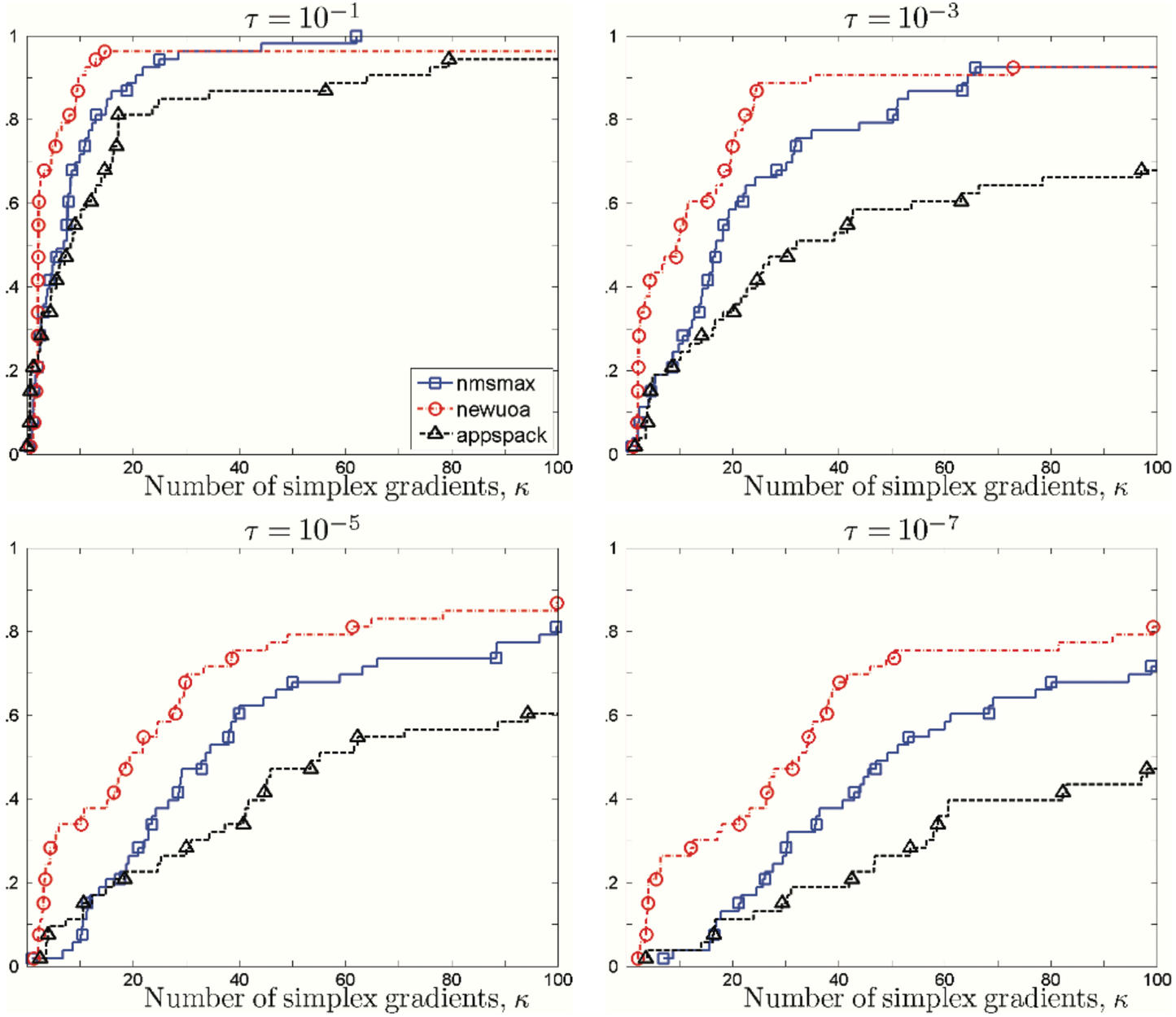


FIG. 5.1. *Data profiles $d_s(\kappa)$ for the smooth problems $\mathcal{P}_S$ show the percentage of problems solved as a function of a computational budget of simplex gradients.*

Remarks/Discussion

- The measured values t can be *any* cost value that is comparable between all problems.

  the values are aggregated in the same graph, hence they should have the same unit

- The target value f_{target} is based on (the same) empirical data

  even for the same problem, performance results are not directly comparable between publications

- Aggregation is
  - done over different problems and different dimensions
  - not done over different algorithms or different tolerances τ

Remarks/Discussion

- The measured values t can be *any* cost value that is comparable between all problems.

  the values are aggregated in the same graph, hence they should have the same unit

- The target value f_{target} is based on (the same) empirical data

  even for the same problem, performance results are not directly comparable between publications

- Aggregation is
  - done over different problems and different dimensions
  - not done over different algorithms or different tolerances τ

Our Insights

1. Aggregation over a wide range of dimensions is not conducive

   dimension can (and should) be used as algorithm selection decision parameter!

2. Aggregation over various tolerance values is possible and seems useful

   arguably, a problem is only defined by a function and a target
   then we still just aggregate over all problems

3. Restricting the budget of the *benchmarked* algorithms has no methodological advantage

   at least none I am aware of
   our measured data are runtimes (budgets)

Generalizing the setting of f_{target}

- We can separate the set of algorithms that determine $f_{\text{target}} = f_L + \tau \cdot (f(x_0) - f_L)$ (by determining f_L) from the set of benchmarked algorithms.

  Thereby, because we can keep f_L constant, data and graphs become comparable across publications

- Then, instead of setting and varying the f-tolerance τ, we can vary the budget μ_f to get different target values (AKA budget-based or runlength-based targets)

  the meaning of budgets is somewhat easier to understand
  we don't need to change the budget of the benchmarked algorithms

- In COCO, we (usually) set f_{target}-values based on the known optimal f-value

  thereby defining the objective extrinsically/absolute rather than relative

- We use a budget-free experimental setup

  increasing the "time-out" budget does not affect results for smaller budgets,
  hence results remain compatible
  larger budgets allow for a better quantification of performance losses

Restarted Algorithm and Simulated Runtimes

- The runtime (RT) of a (randomized) restarted algorithm is

Price 1997, Auger & Hansen 2005

$$\mathrm{RT} = \sum_{i=1}^{N_{\mathrm{unsuc}}} t_i^{\mathrm{unsuc}} + t^{\mathrm{succ}}$$

- The expected RT is (where t is a random variable)

$$E[\mathrm{RT}] = \frac{1 - p_{\mathrm{succ}}}{p_{\mathrm{succ}}} E[t^{\mathrm{unsuc}}] + E[t^{\mathrm{succ}}]$$

$$= \frac{(1 - p_{\mathrm{succ}}) E[t^{\mathrm{unsuc}}] + p_{\mathrm{succ}} E[t^{\mathrm{succ}}]}{p_{\mathrm{succ}}}$$

- An estimator from the data is

$$\hat{E}[\mathrm{RT}] = \frac{\sum t^{\mathrm{unsuc}} + \sum t^{\mathrm{succ}}}{N_{\mathrm{succ}}} = \frac{\text{overall costs}}{\#\text{successes}}$$

aka ERT, Enes, SP2, aRT.

Implications:

- if we have at least one successful run, we can simulate runtimes, estimate the expected runtime, and don't need to negotiate success vs speed
- if we have no successful run, we can estimate a lower bound for the expected runtime $\hat{E}[\mathrm{RT}] > \sum t^{\mathrm{unsuc}}$

Restarted Algorithm and Simulated Runtimes

- The runtime (RT) of a (randomized) restarted algorithm is

Price 1997, Auger & Hansen 2005

$$\mathrm{RT} = \sum_{i=1}^{N_{\mathrm{unsuc}}} t_i^{\mathrm{unsuc}} + t^{\mathrm{succ}}$$

- The expected RT is (where t is a random variable)

$$E[\mathrm{RT}] = \frac{1 - p_{\mathrm{succ}}}{p_{\mathrm{succ}}} E[t^{\mathrm{unsuc}}] + E[t^{\mathrm{succ}}]$$

$$= \frac{(1 - p_{\mathrm{succ}}) E[t^{\mathrm{unsuc}}] + p_{\mathrm{succ}} E[t^{\mathrm{succ}}]}{p_{\mathrm{succ}}}$$

- An estimator from the data is

$$\hat{E}[\mathrm{RT}] = \frac{\sum t^{\mathrm{unsuc}} + \sum t^{\mathrm{succ}}}{N_{\mathrm{succ}}} = \frac{\text{overall costs}}{\#\text{successes}}$$

aka ERT, Enes, SP2, aRT.

Implications:

- if we have at least one successful run, we can simulate runtimes, estimate the expected runtime, and don't need to negotiate success vs speed
- if we have no successful run, we can estimate a lower bound for the expected runtime $\hat{E}[\mathrm{RT}] > \sum t^{\mathrm{unsuc}}$

Restarted Algorithm and Simulated Runtimes

- The runtime (RT) of a (randomized) restarted algorithm is

Price 1997, Auger & Hansen 2005

$$\mathrm{RT} = \sum_{i=1}^{N_{\mathrm{unsuc}}} t_i^{\mathrm{unsuc}} + t^{\mathrm{succ}}$$

- The expected RT is (where t is a random variable)

$$E[\mathrm{RT}] = \frac{1 - p_{\mathrm{succ}}}{p_{\mathrm{succ}}} E[t^{\mathrm{unsuc}}] + E[t^{\mathrm{succ}}]$$

$$= \frac{(1 - p_{\mathrm{succ}}) E[t^{\mathrm{unsuc}}] + p_{\mathrm{succ}} E[t^{\mathrm{succ}}]}{p_{\mathrm{succ}}}$$

- An estimator from the data is

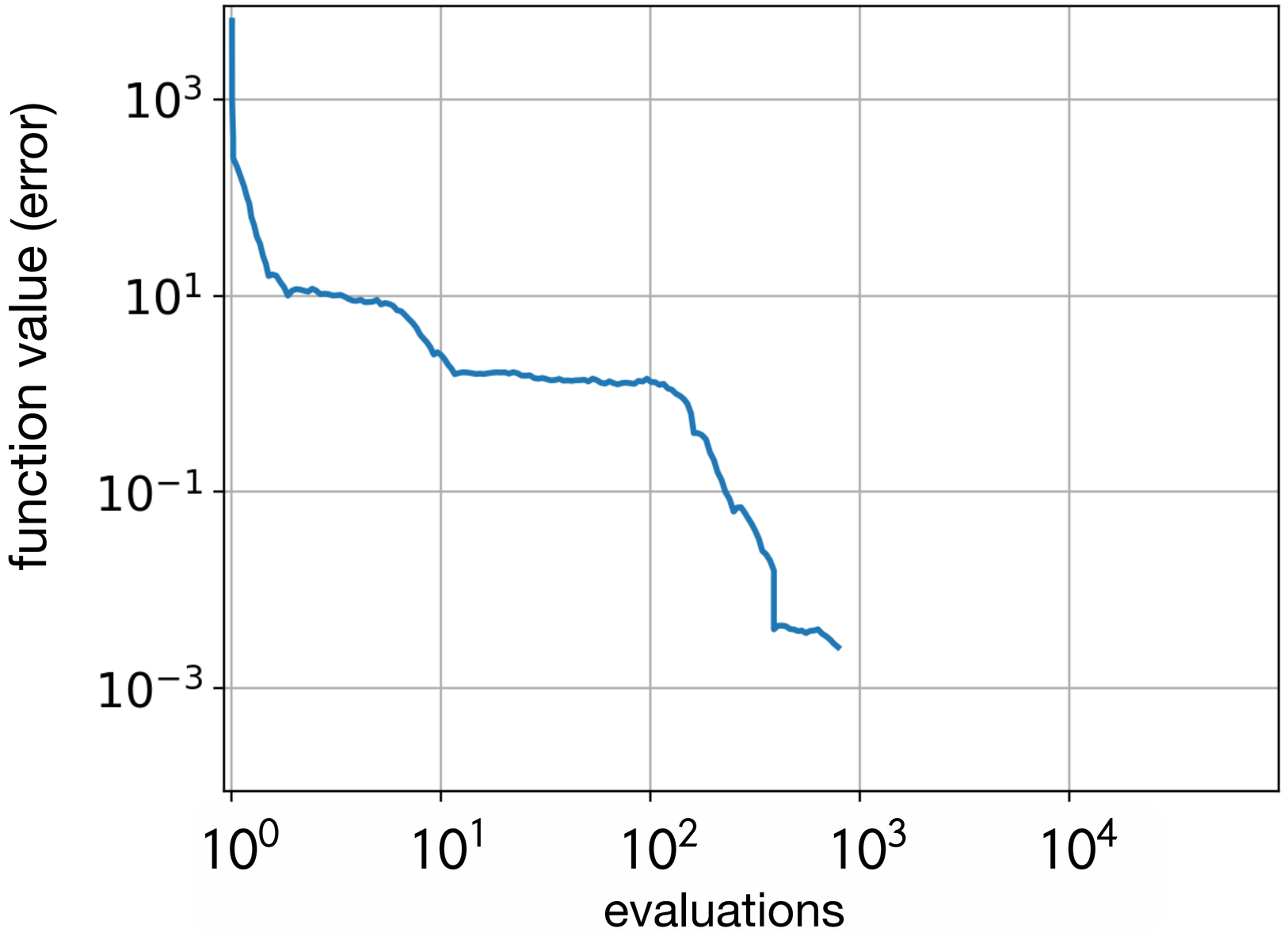
$$\hat{E}[\mathrm{RT}] = \frac{\sum t^{\mathrm{unsuc}} + \sum t^{\mathrm{succ}}}{N_{\mathrm{succ}}} = \frac{\text{overall costs}}{\#\text{successes}}$$

  aka ERT, Enes, SP2, aRT.

Implications:

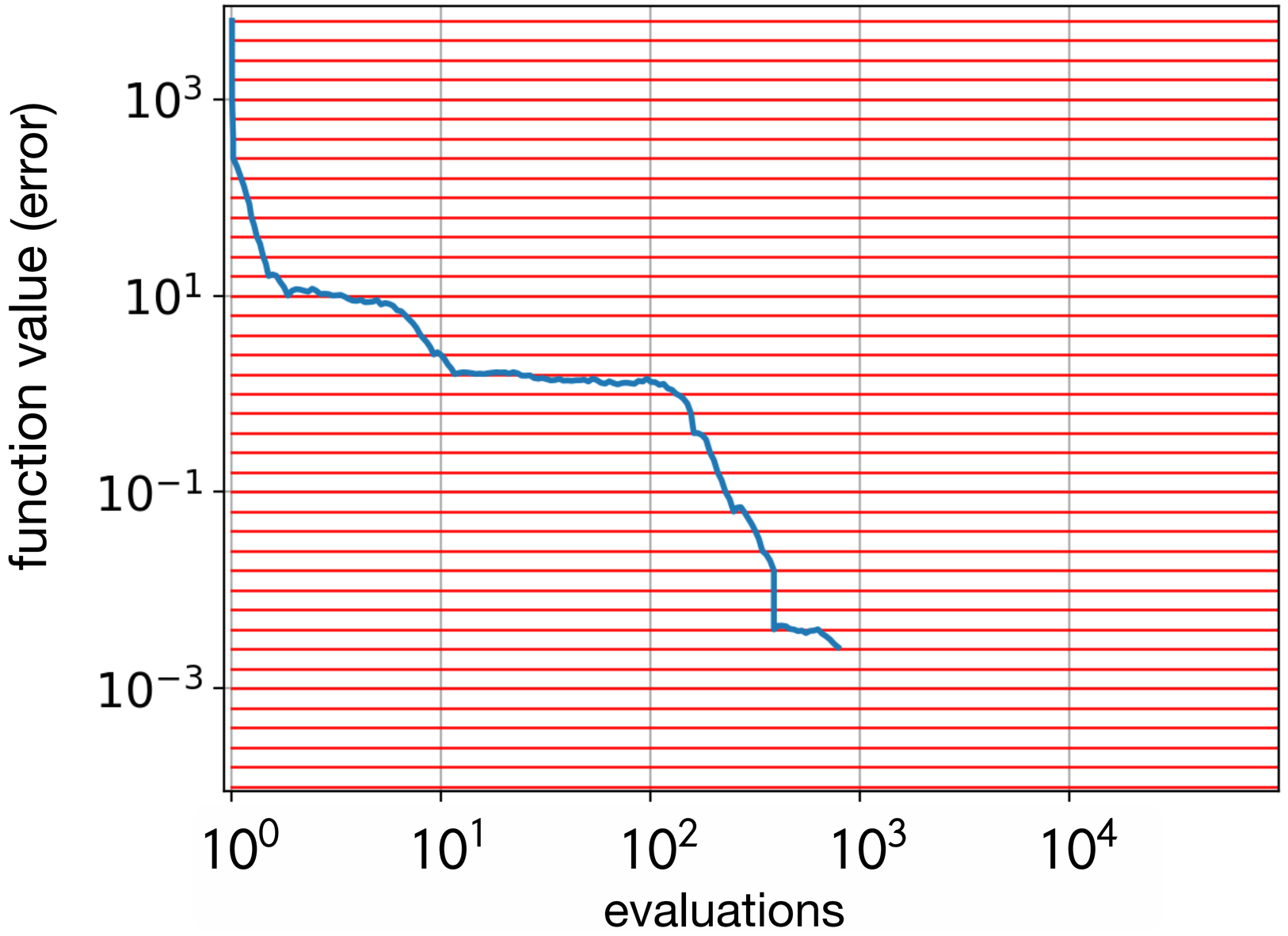
- if we have at least one successful run, we can simulate runtimes, estimate the expected runtime, and don't need to negotiate success vs speed
- if we have no successful run, we can estimate a lower bound for the expected runtime $\hat{E}[\mathrm{RT}] > \sum t^{\mathrm{unsuc}}$

From a Convergence Graph to the Empirical Runtime Distribution



Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

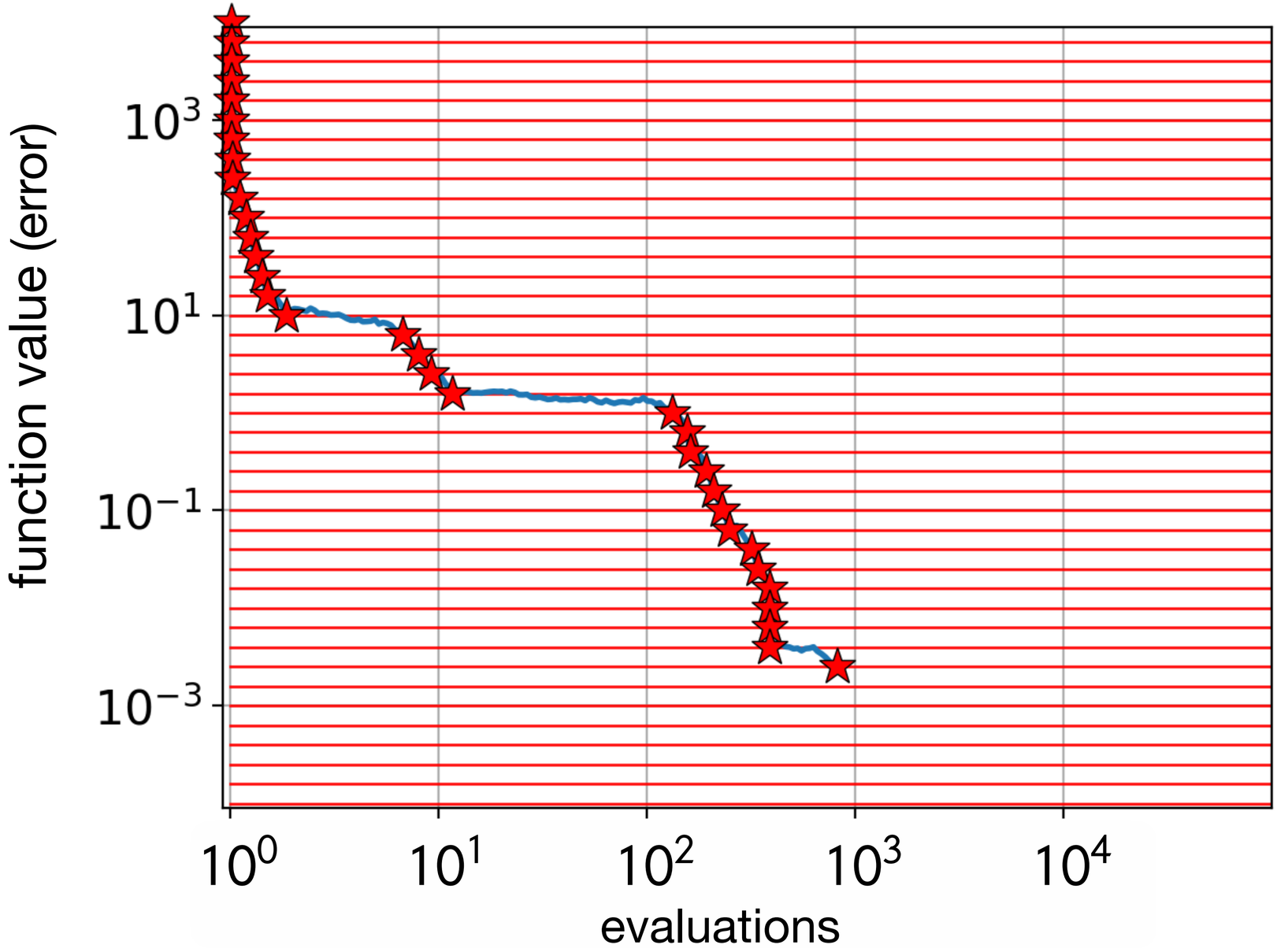
From a Convergence Graph to the Empirical Runtime Distribution



equidistance
“target” values

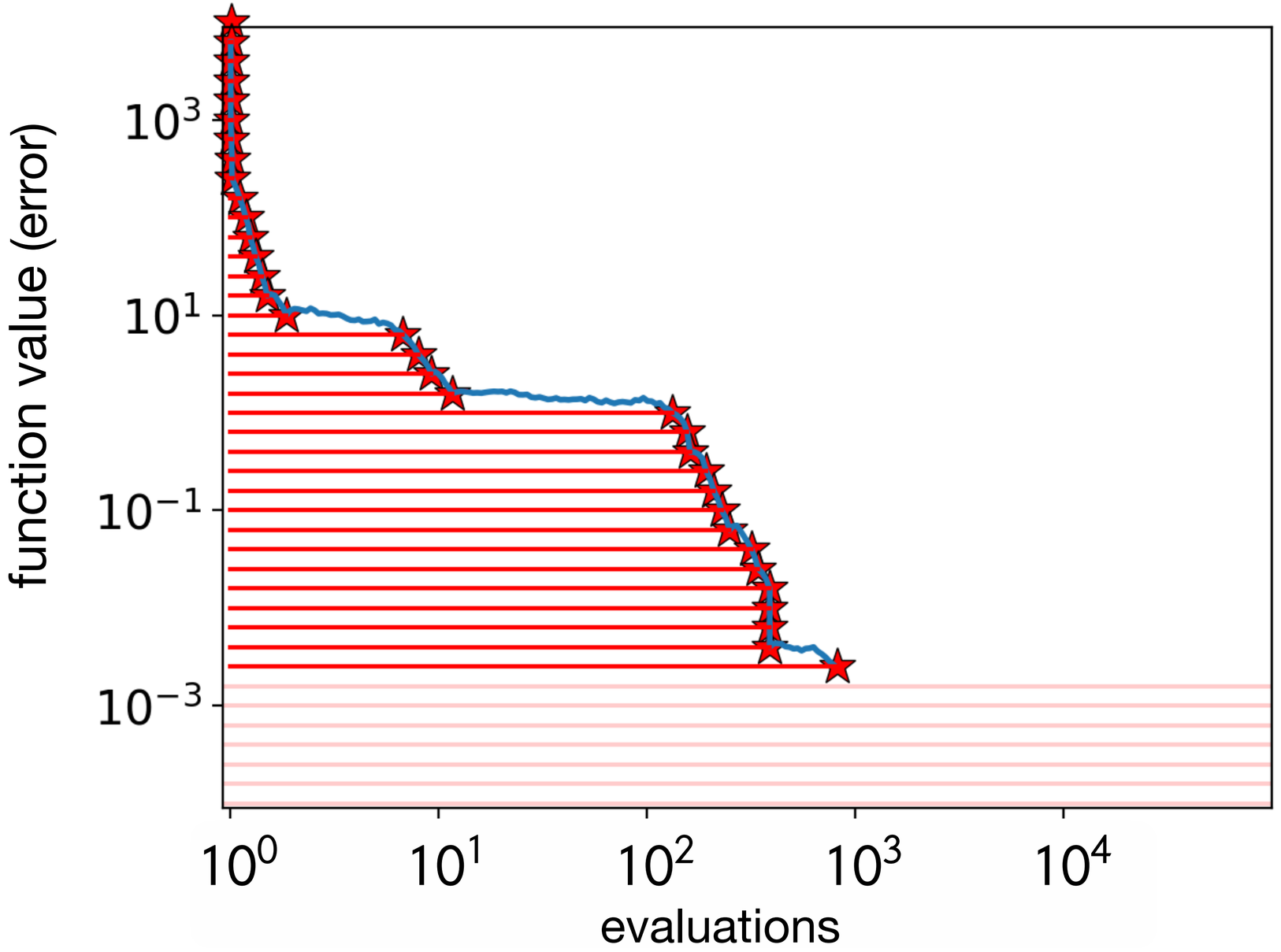
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

From a Convergence Graph to the Empirical Runtime Distribution



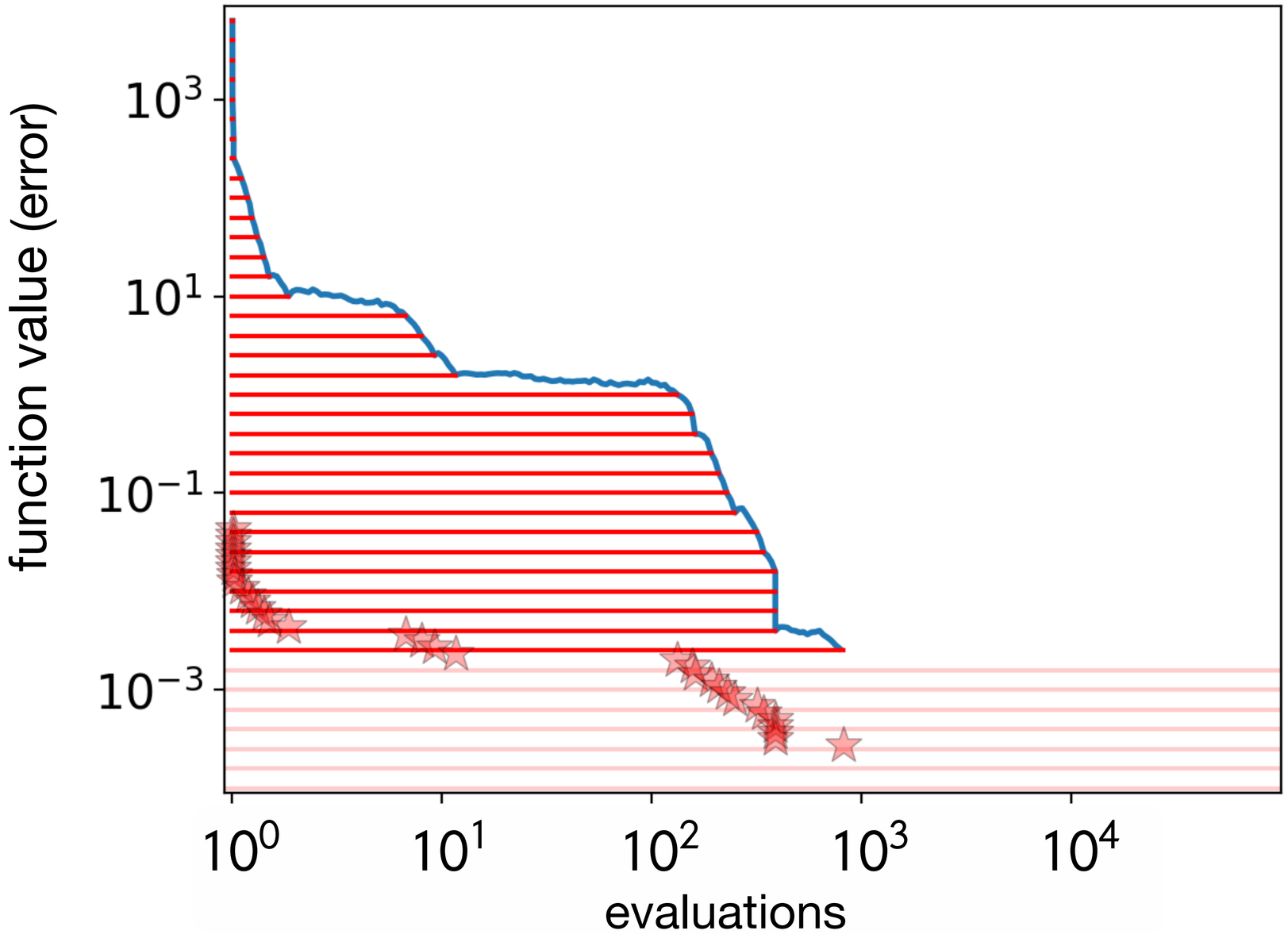
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

From a Convergence Graph to the Empirical Runtime Distribution



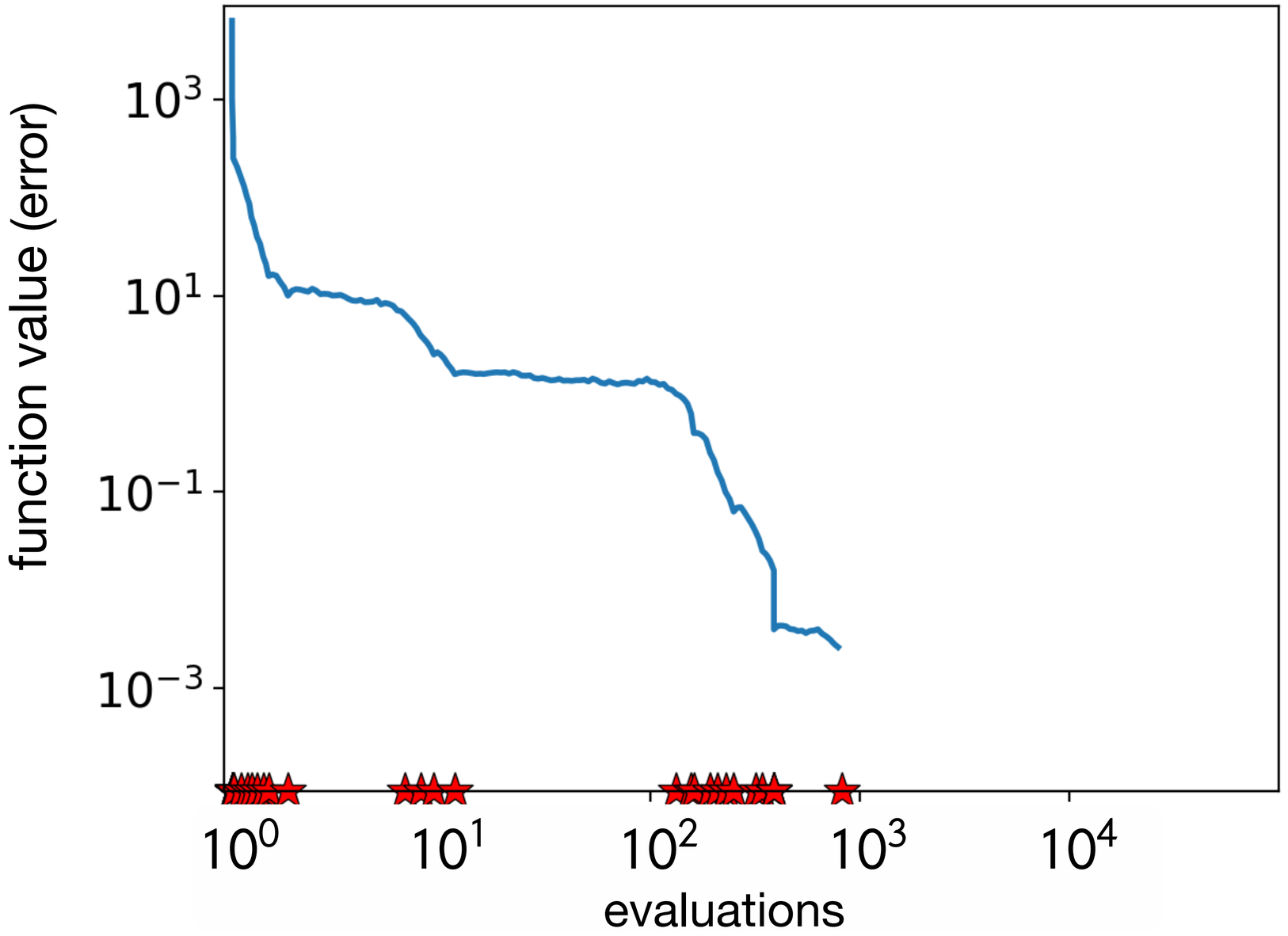
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

From a Convergence Graph to the Empirical Runtime Distribution



Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

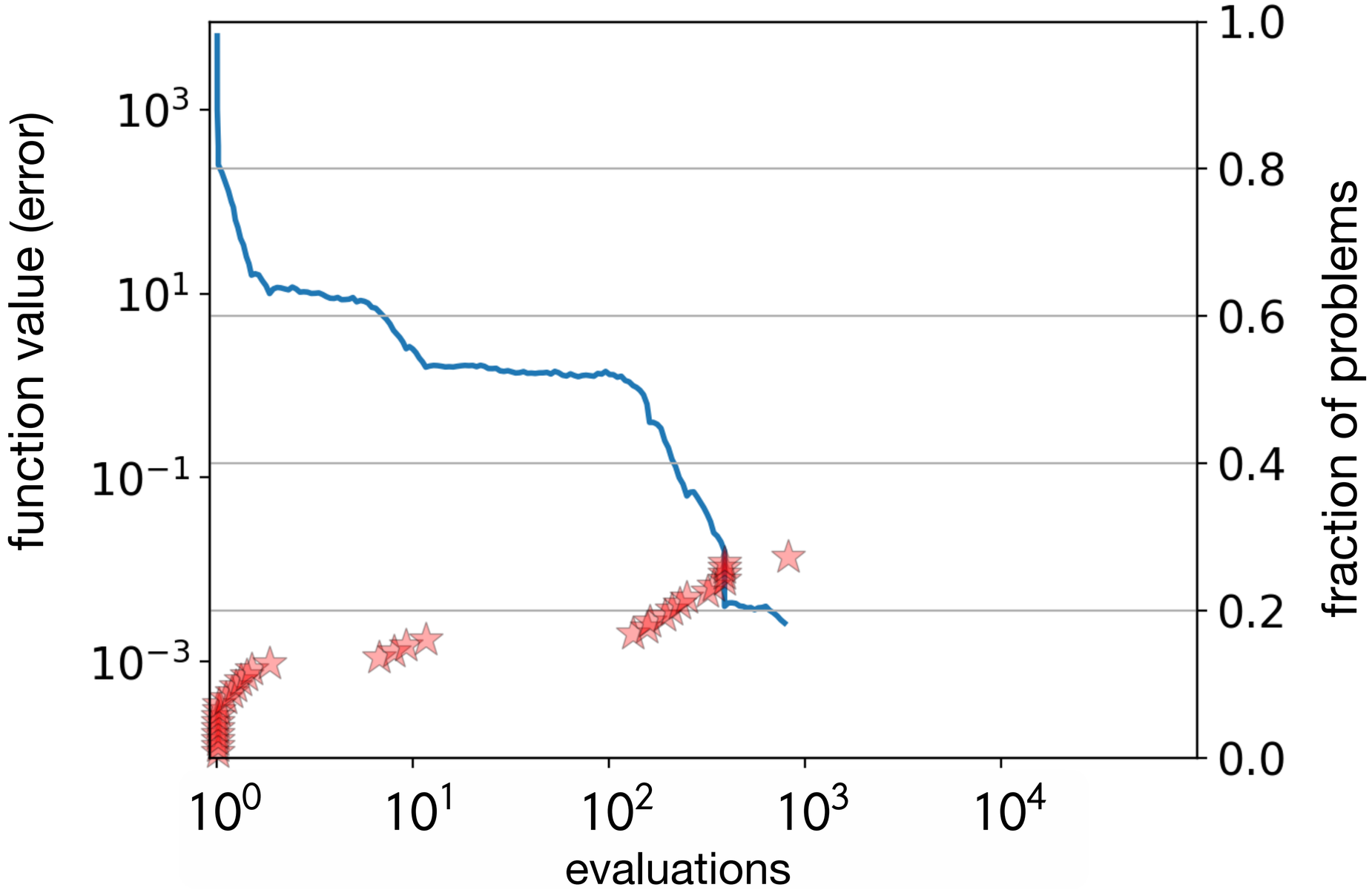
From a Convergence Graph to the Empirical Runtime Distribution



for the remaining construction, we could use any runtimes, for example, from different runs or different functions

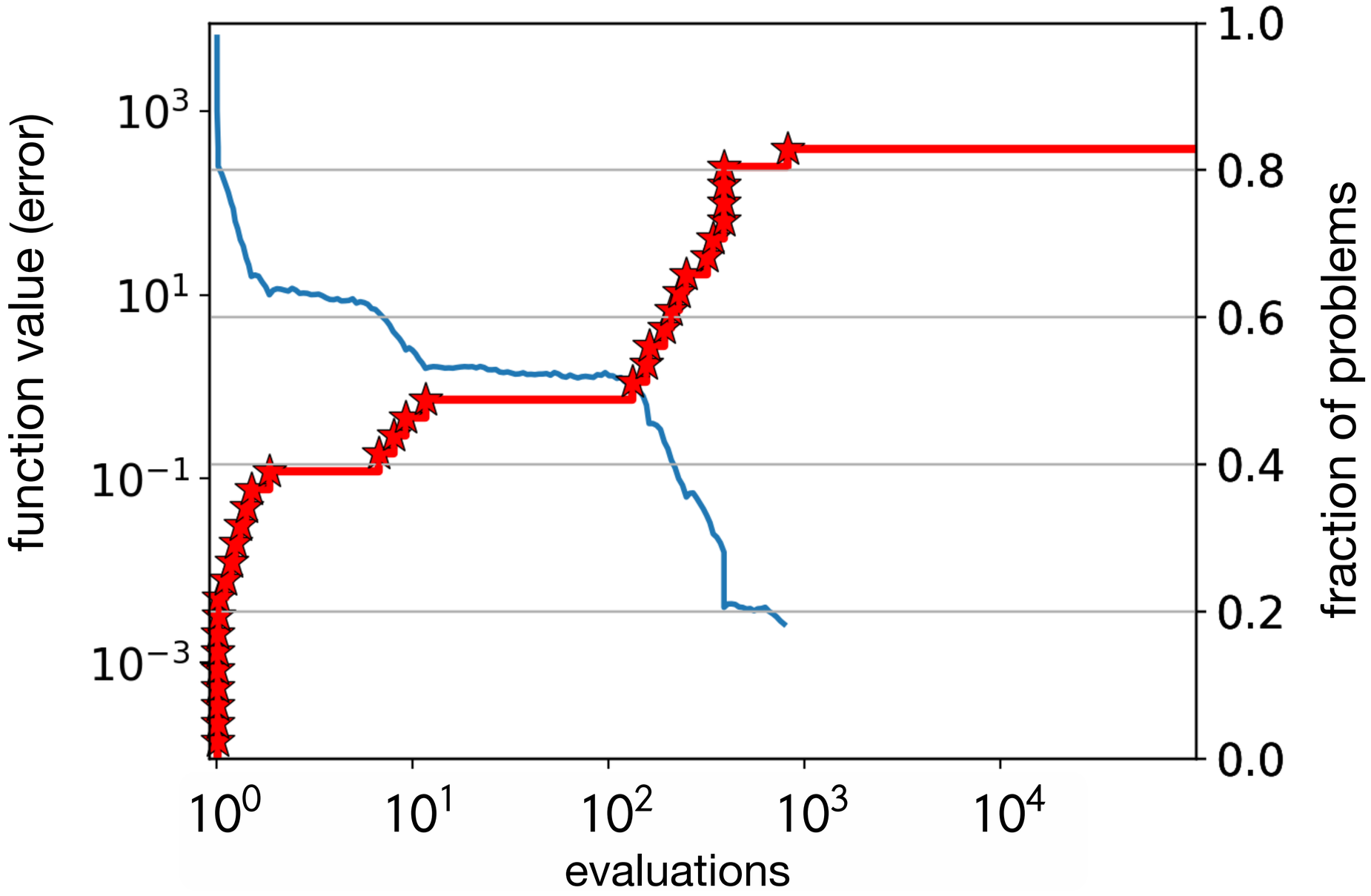
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

From a Convergence Graph to the Empirical Runtime Distribution



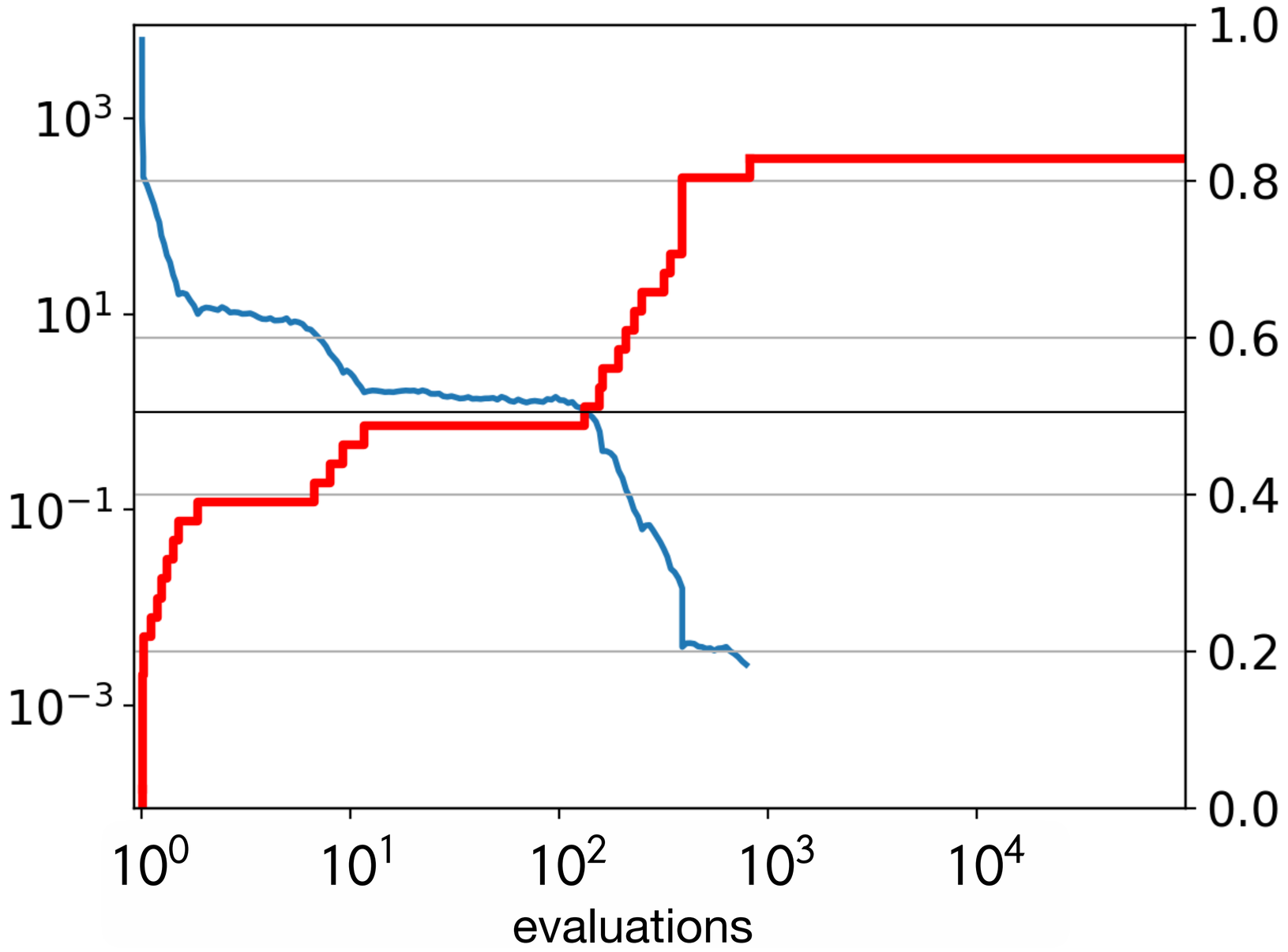
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

From a Convergence Graph to the Empirical Runtime Distribution



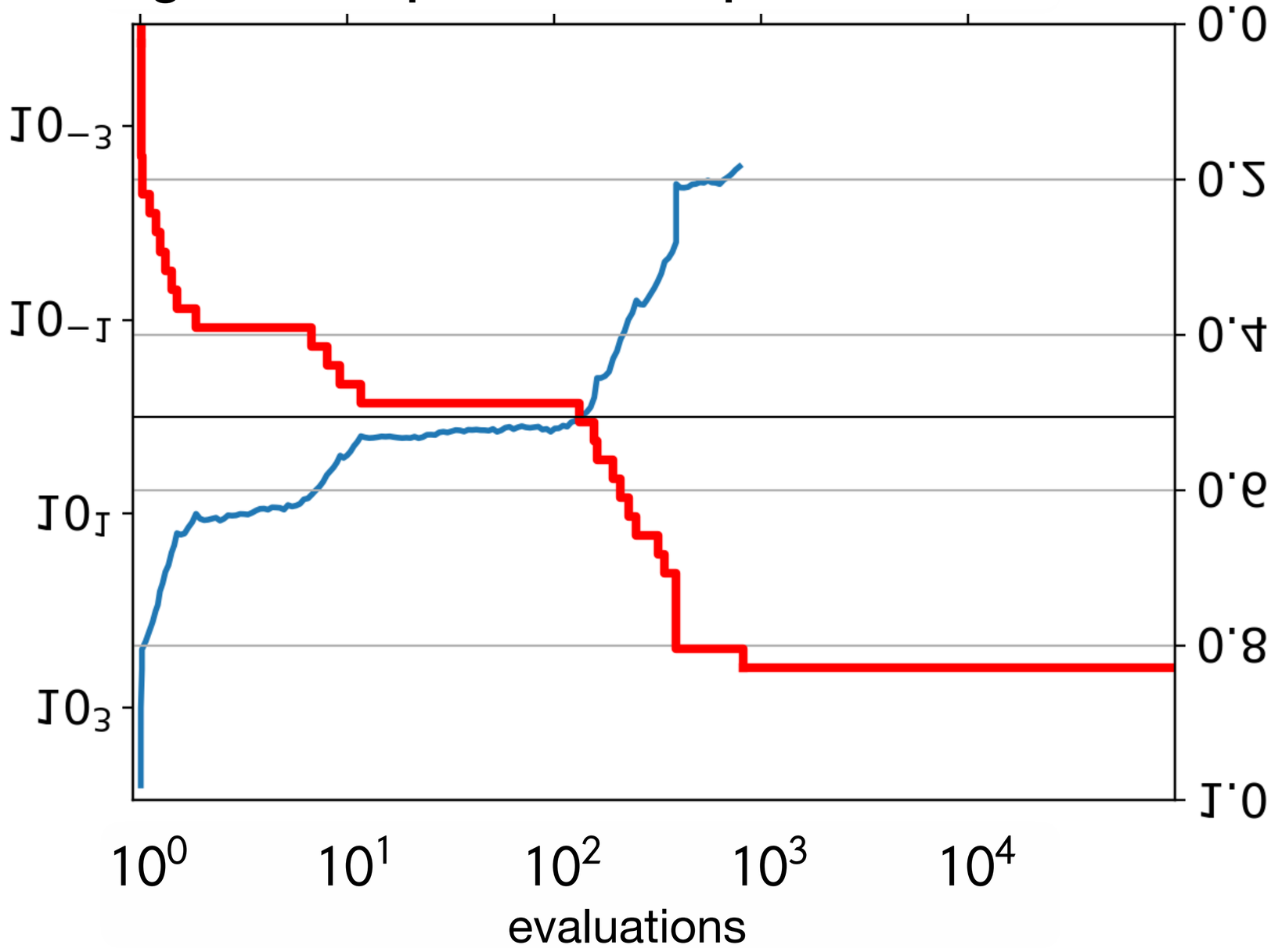
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

From a Convergence Graph to the Empirical Runtime Distribution



Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

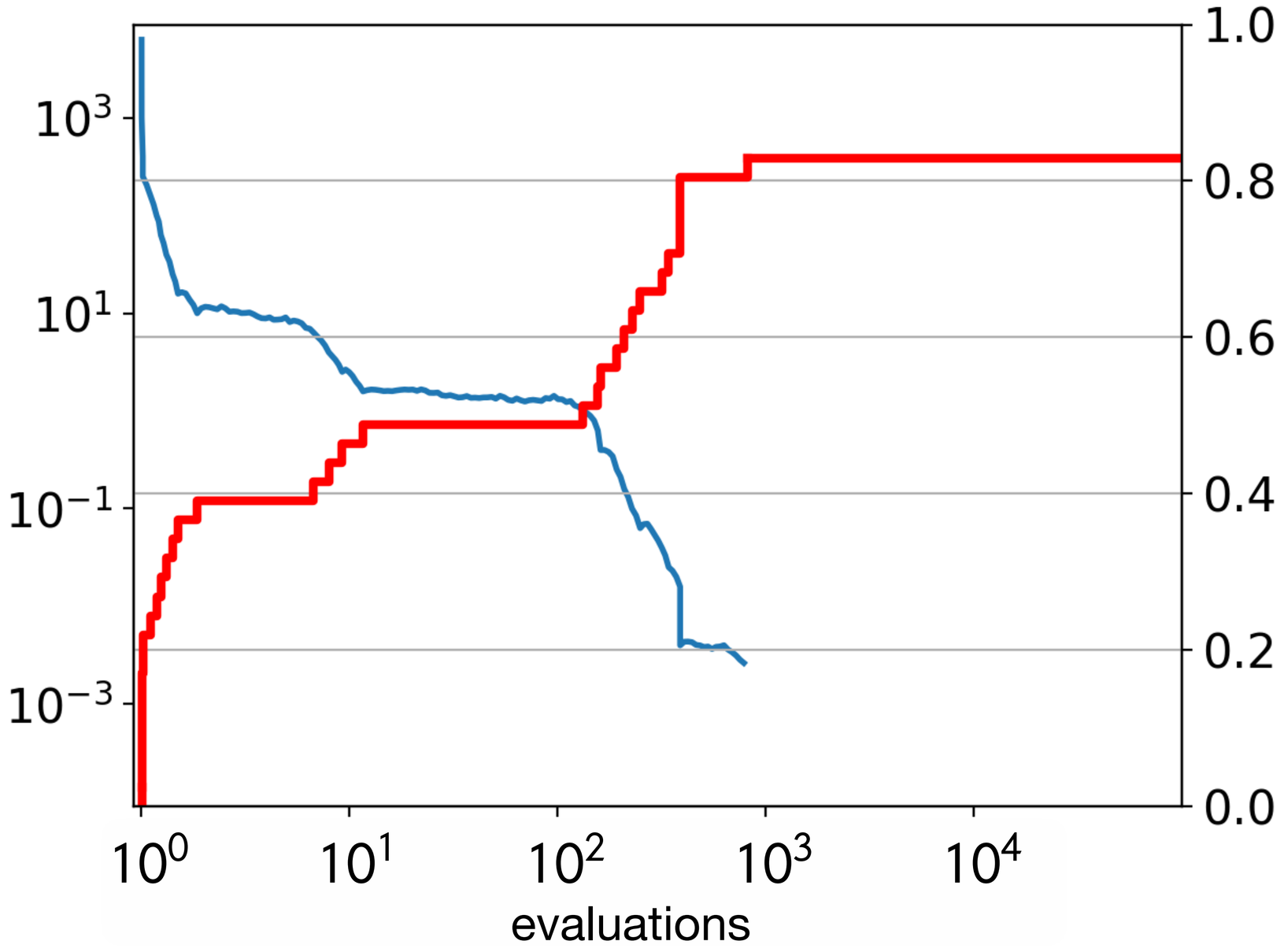
From a Convergence Graph to the Empirical Runtime Distribution



Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

From a Convergence Graph to the Empirical Runtime Distribution

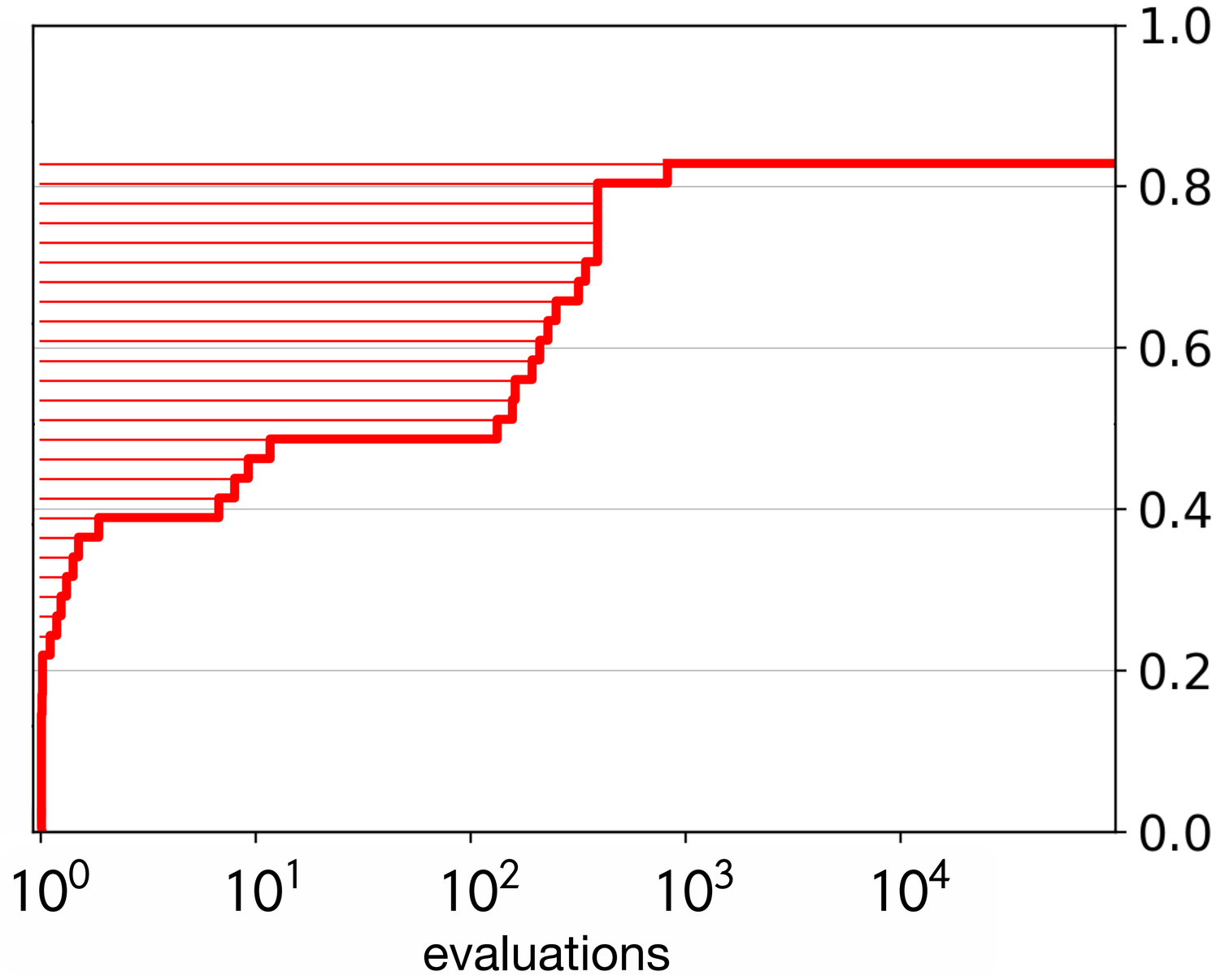
when we maximize (instead of minimize), the graph can be considered as an empirical runtime distribution *as is*



Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

From a Convergence Graph to the Empirical Runtime Distribution

when we maximize (instead of minimize), the graph can be considered as an empirical runtime distribution *as is*

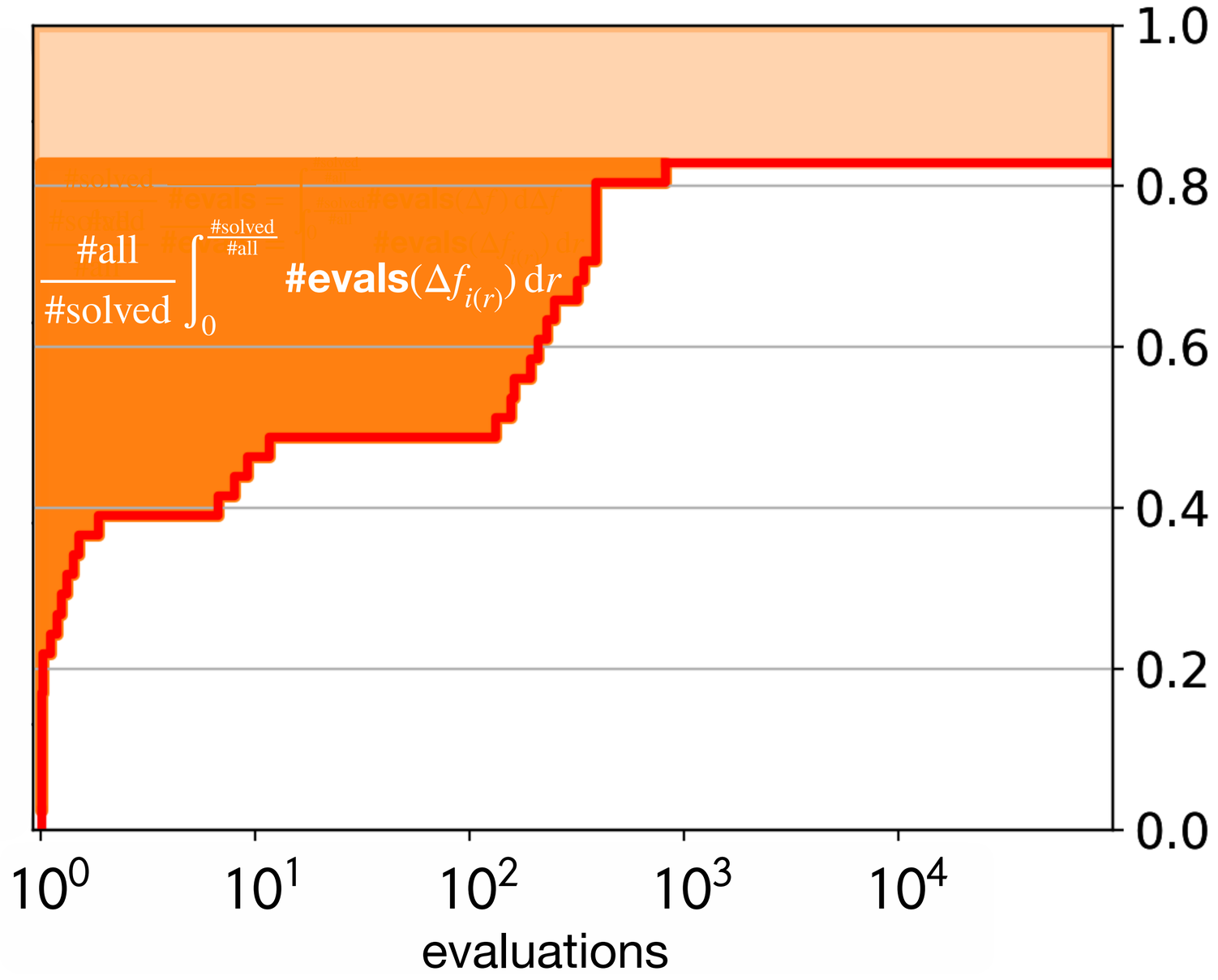


Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

Empirical Runtime Distribution and area above the curve

the area above the curve represent a (truncated) average runtime

When the x-axis is in log-scale, the area is the (truncated) geometric average

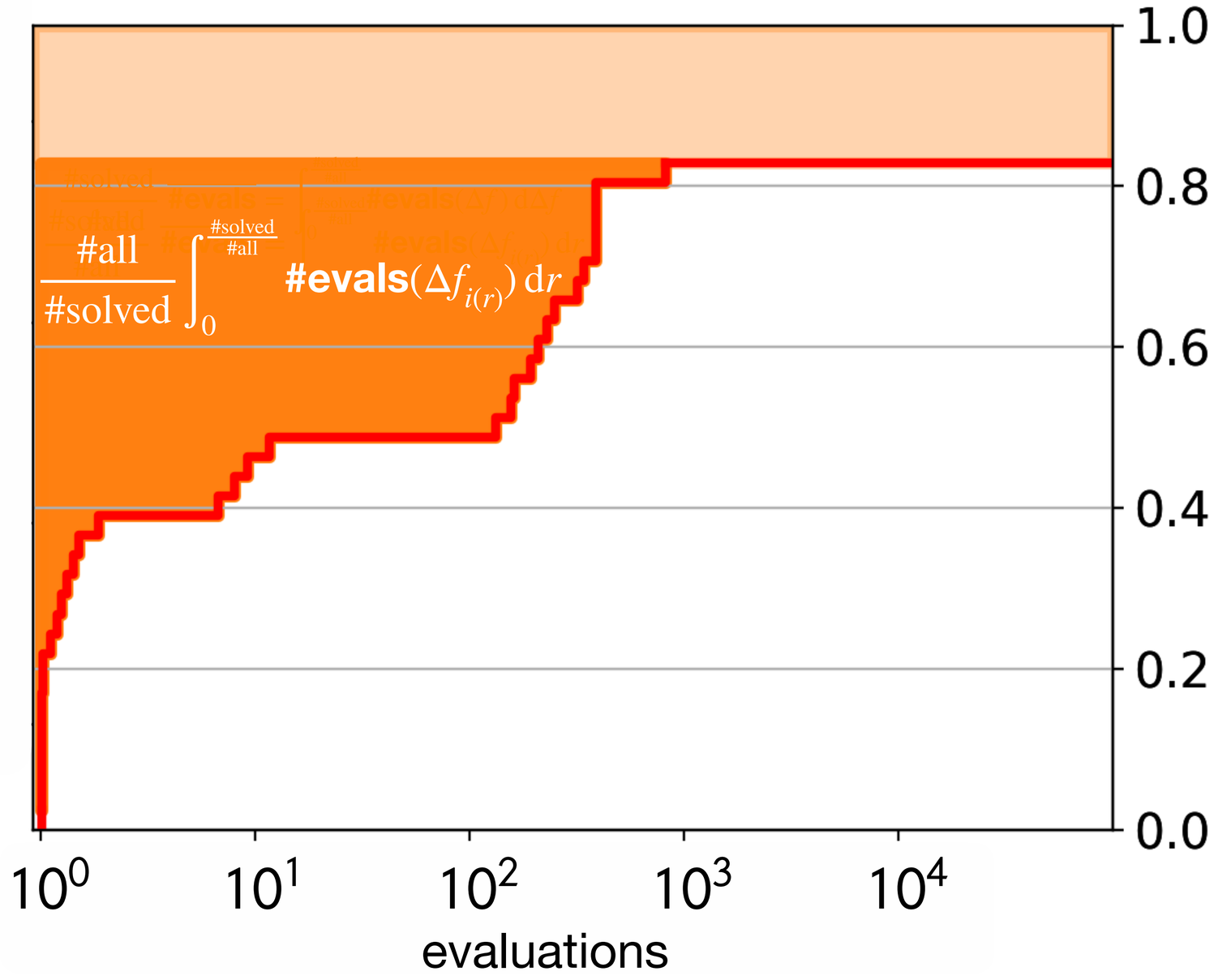


Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

Empirical Runtime Distribution and area above the curve

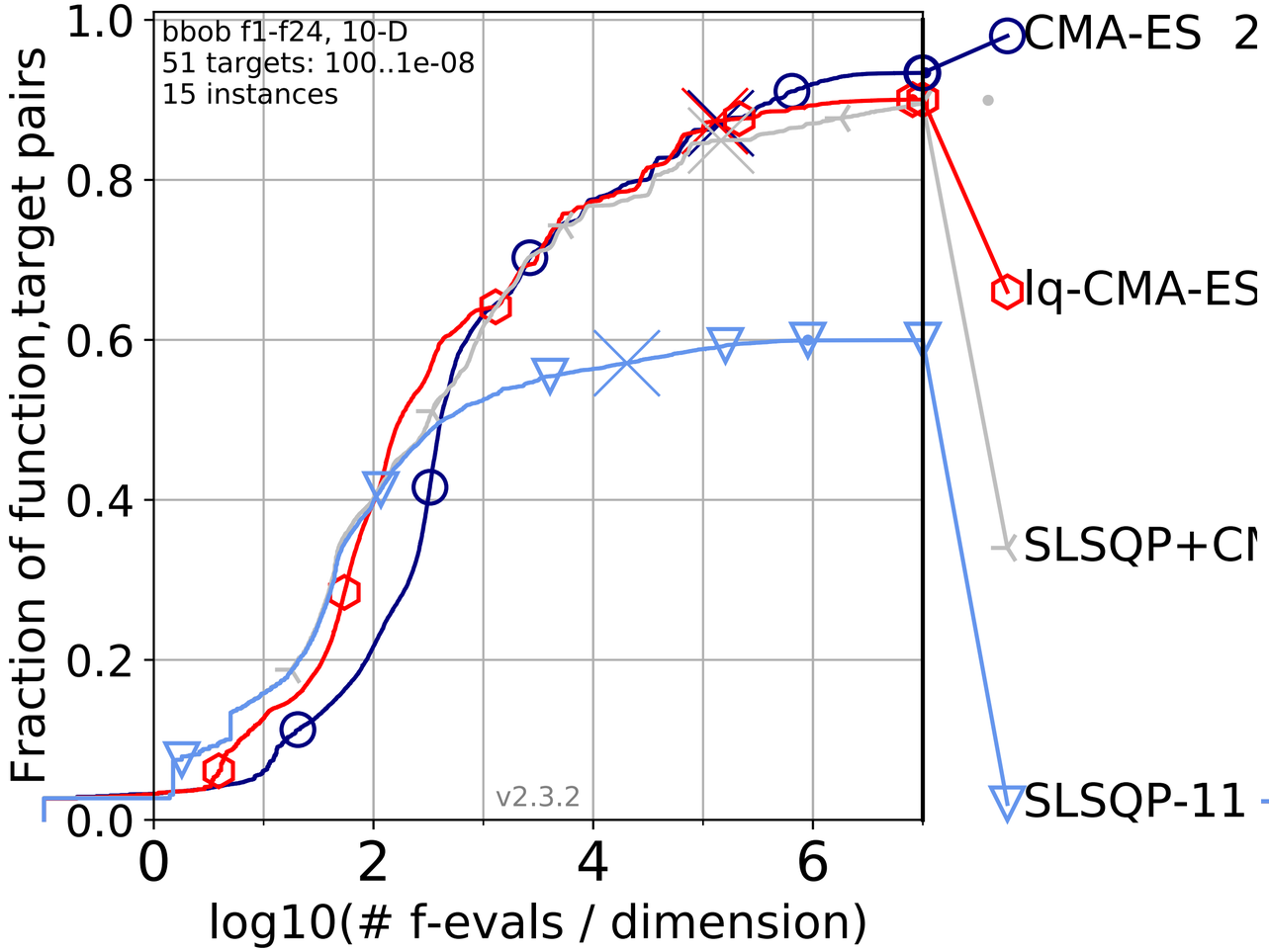
the area above the curve represent a (truncated) average runtime

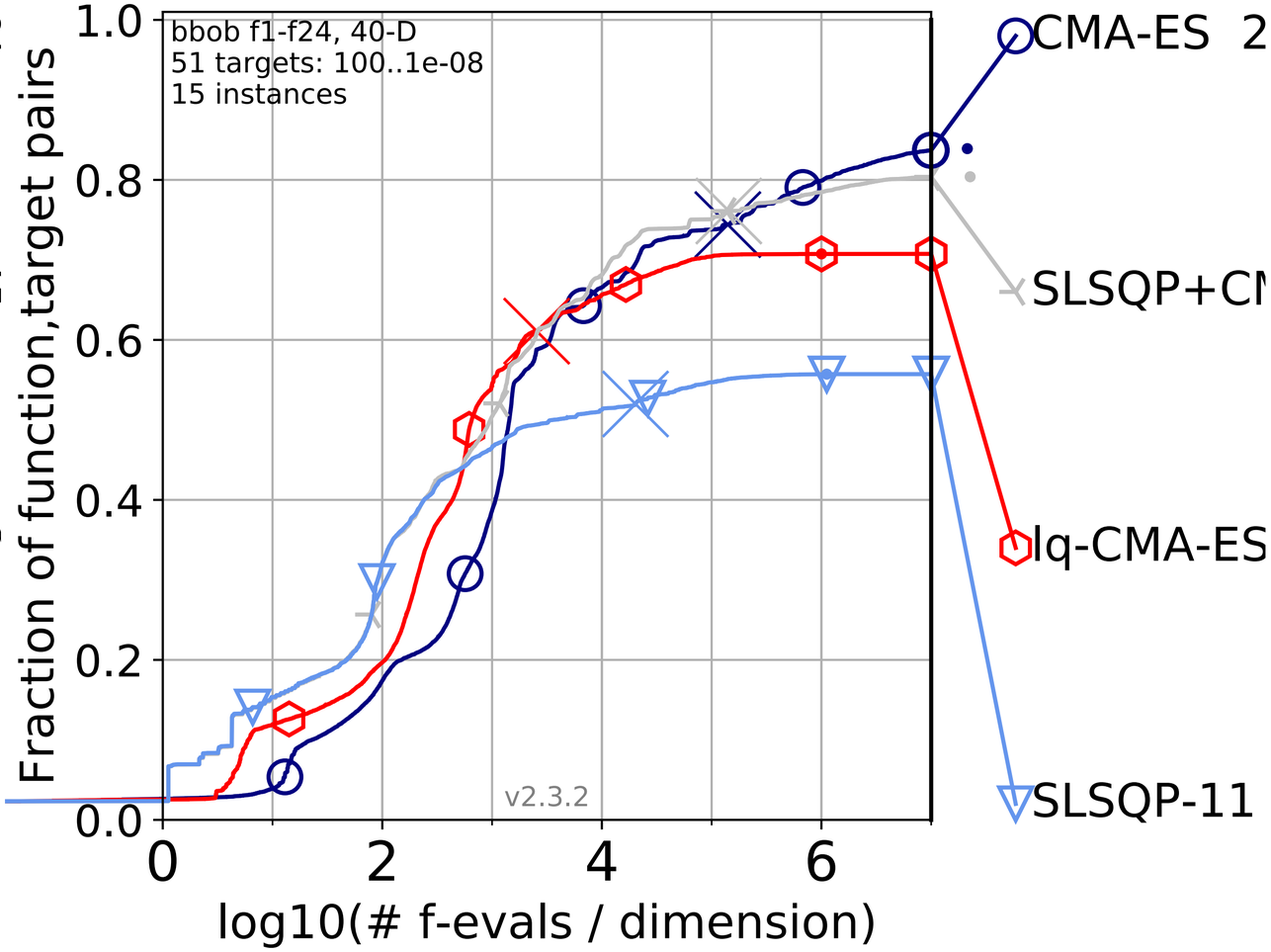
When the x-axis is in log-scale, the area is the (truncated) geometric average



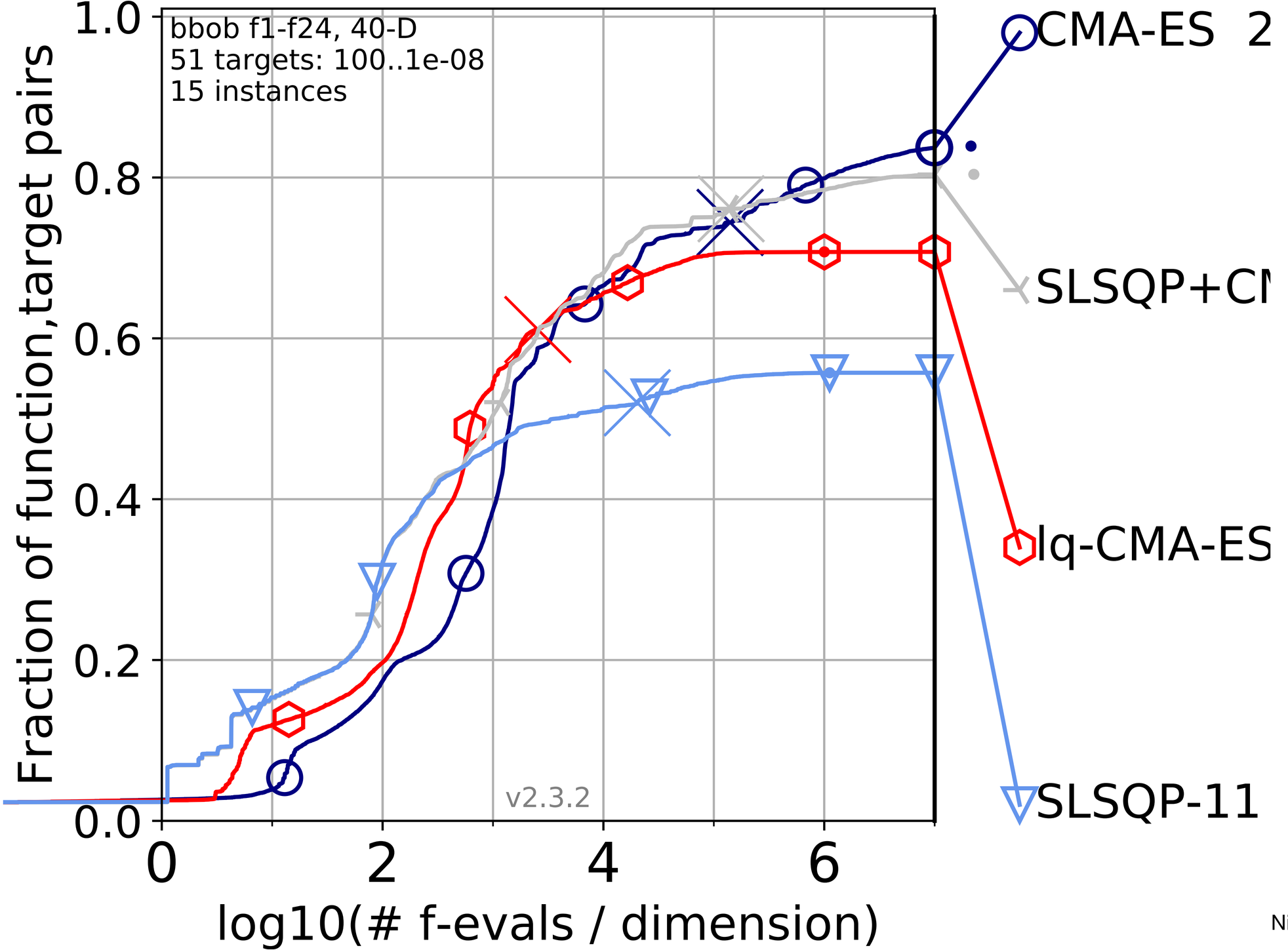
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

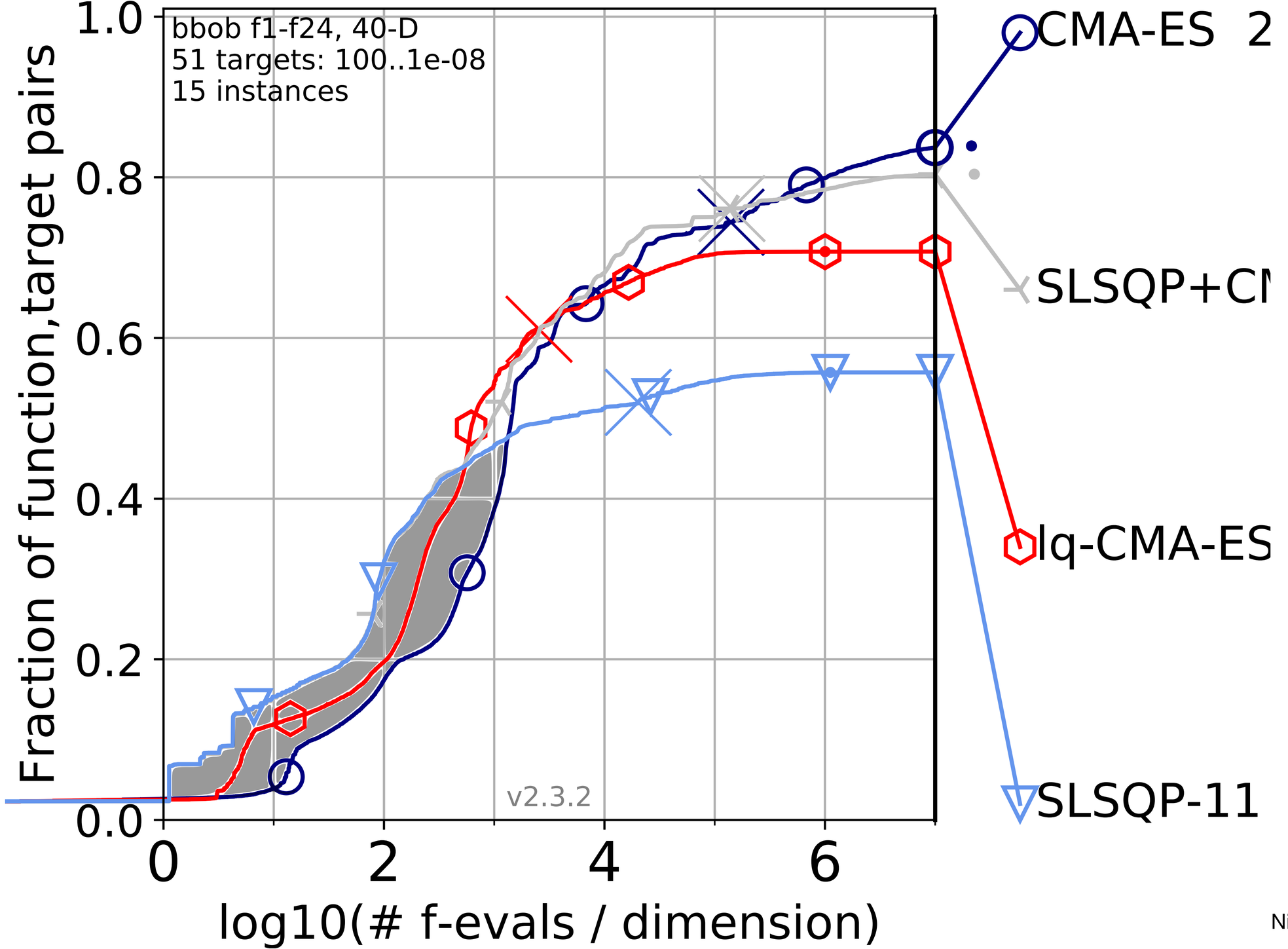
Aggregated Runtime Distributions

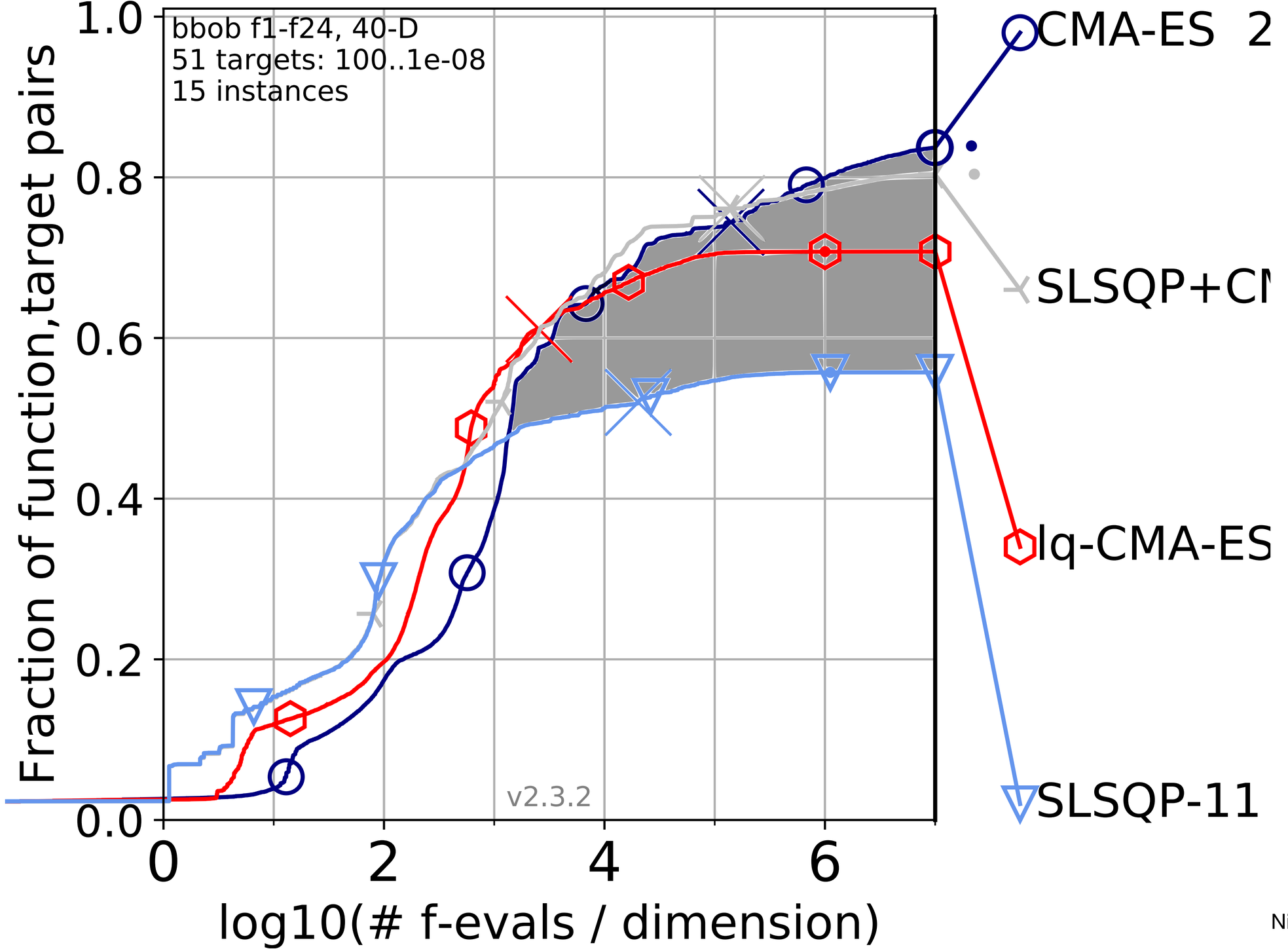


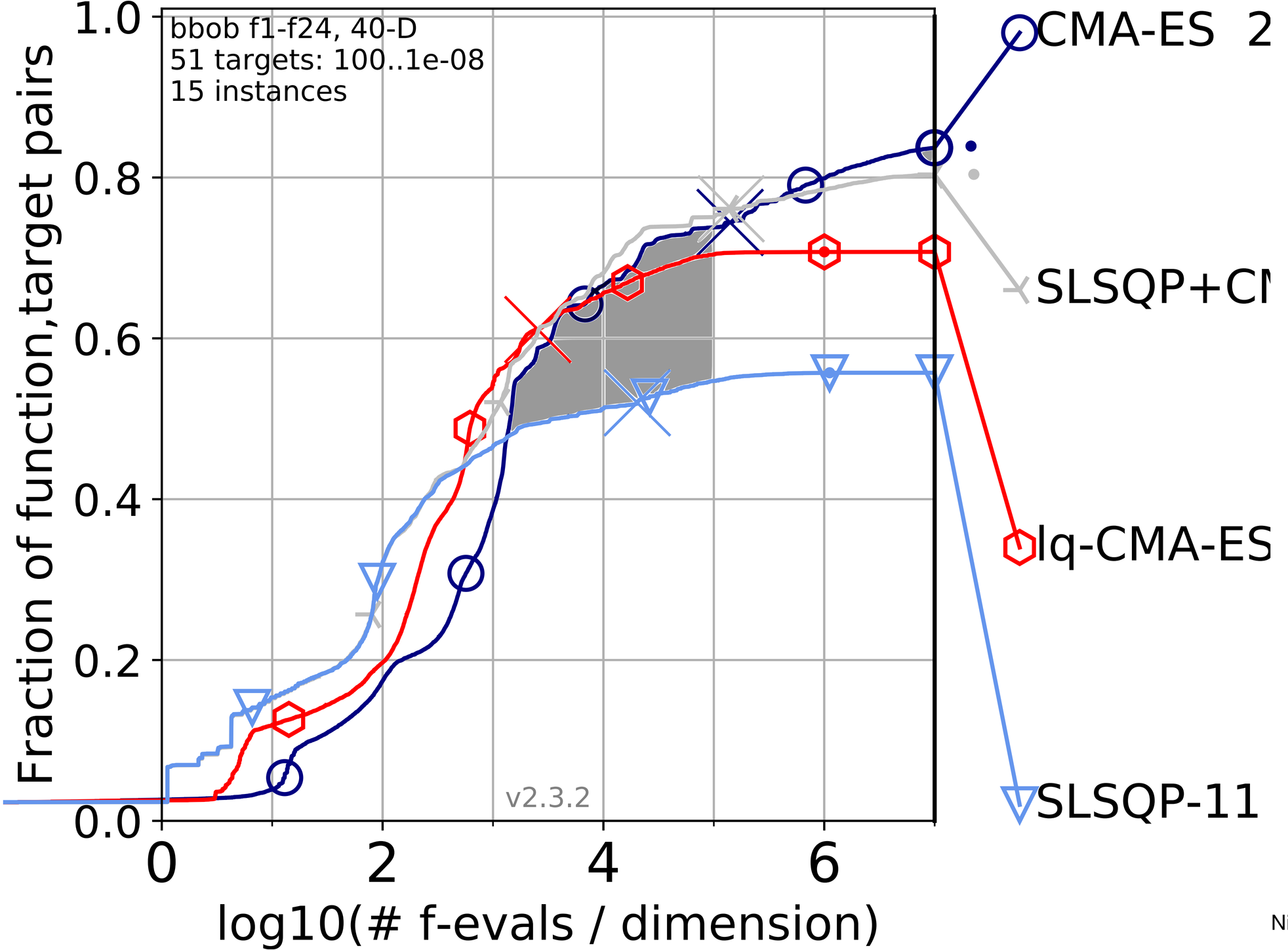


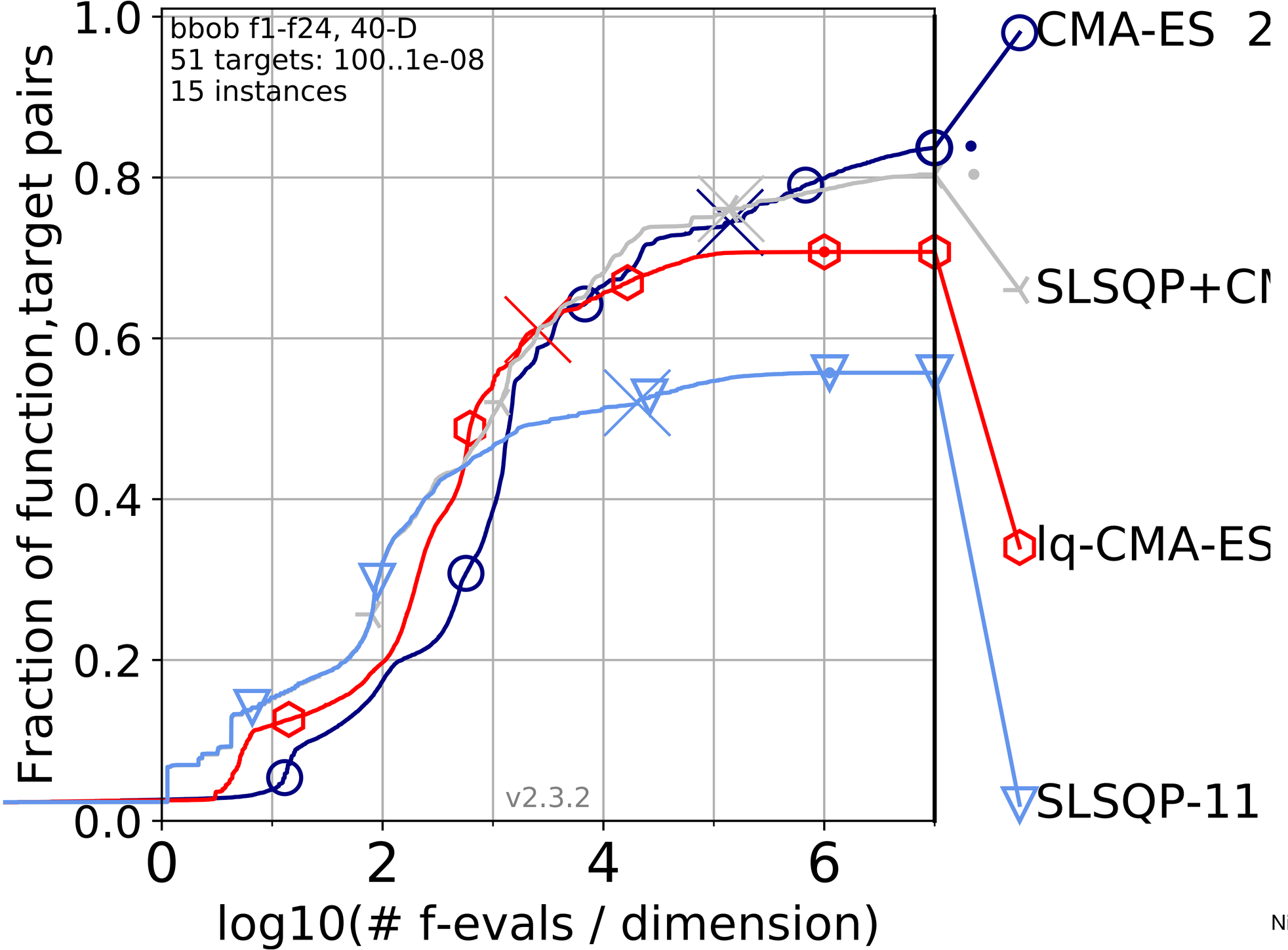
Hansen 2019. A Global Surrogate Assisted CMA-ES. *GECCO '19*.

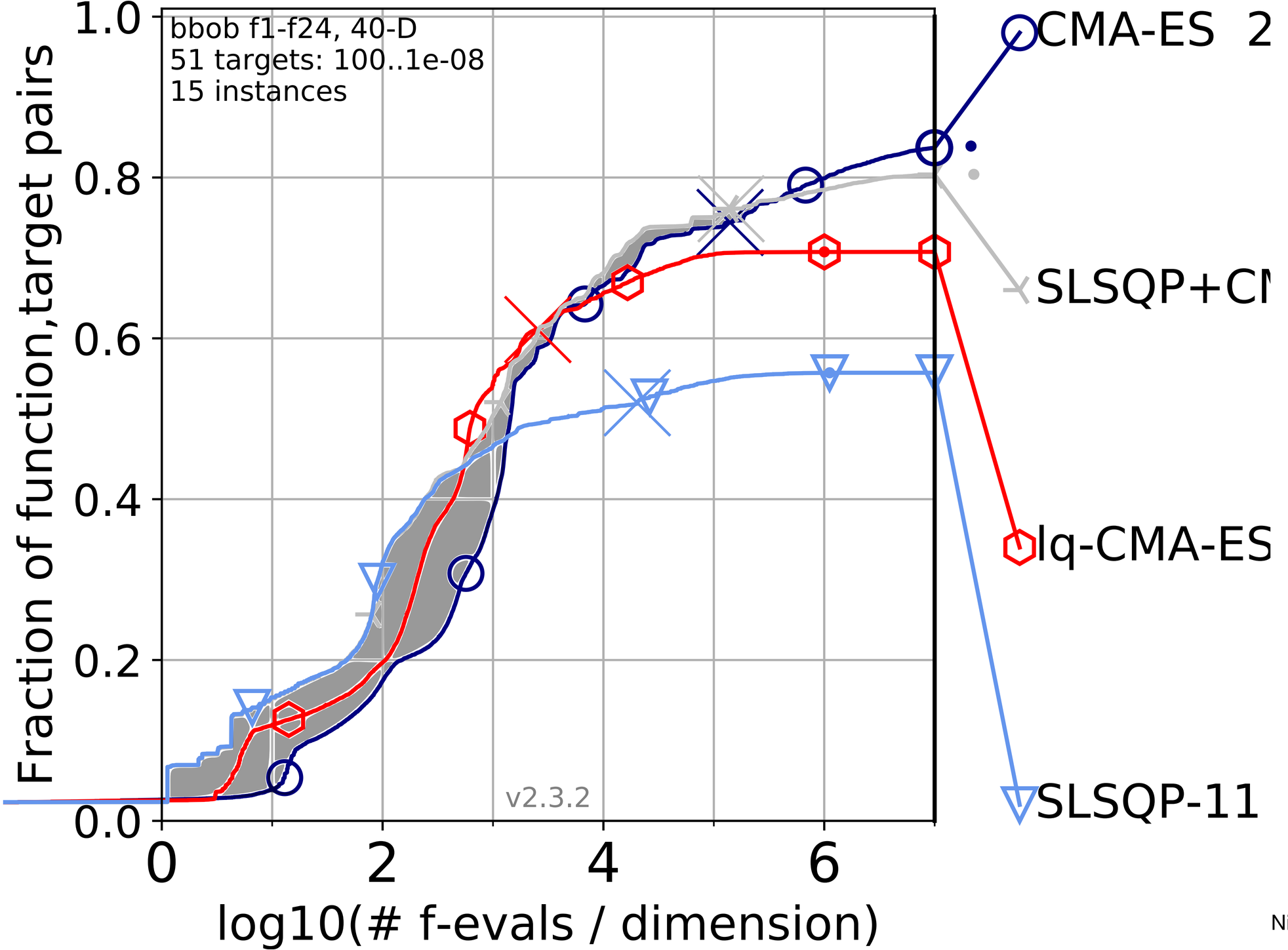

bbob f1-f24, 40-D
51 targets: 100..1e-08
15 instances
v2.3.2
Fraction of function,target pairs
log10(# f-evals / dimension)
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
CMA-ES 2
SLSQP+CM
lq-CMA-ES
SLSQP-11


bbob f1-f24, 40-D
51 targets: 100..1e-08
15 instances
v2.3.2
Fraction of function,target pairs
log10(# f-evals / dimension)
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
CMA-ES 2
SLSQP+CM
lq-CMA-ES
SLSQP-11


bbob f1-f24, 40-D
51 targets: 100..1e-08
15 instances
v2.3.2
Fraction of function,target pairs
log10(# f-evals / dimension)
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
CMA-ES 2
SLSQP+CM
lq-CMA-ES
SLSQP-11


bbob f1-f24, 40-D
51 targets: 100..1e-08
15 instances
v2.3.2
Fraction of function,target pairs
log10(# f-evals / dimension)
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
CMA-ES 2
SLSQP+CM
lq-CMA-ES
SLSQP-11


bbob f1-f24, 40-D
51 targets: 100..1e-08
15 instances
v2.3.2
Fraction of function,target pairs
log10(# f-evals / dimension)
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
CMA-ES 2
SLSQP+CM
lq-CMA-ES
SLSQP-11


bbob f1-f24, 40-D
51 targets: 100..1e-08
15 instances
v2.3.2
Fraction of function,target pairs
log10(# f-evals / dimension)
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
CMA-ES 2
SLSQP+CM
lq-CMA-ES
SLSQP-11

Stop writing “statistically significant”

- Wasserstein et al. 2019: “*We conclude, based on our review of the articles in this special issue and the broader literature, that it is* ***time to stop using the term “statistically significant” entirely****. Nor should variants such as “significantly different,” “p < 0.05,” and “nonsignificant” survive, […] however, we are not recommending that the calculation and use of continuous p-values be discontinued. Where* ***p-values*** *are used, they* ***should be reported as continuous quantities*** *(e.g., p = 0.08).*”

  Moving to a World Beyond “*p* < 0.05”. *The American Statistician*, 73, S1.

- Amrhein et al. + 800 signatories, 2019: *“We agree, and call for the entire* ***concept of statistical significance to be abandoned****. […] we are calling for a stop to the use of P values* ***in the conventional, dichotomous way*** *— to decide whether a result refutes or supports a scientific hypothesis.”*

  Retire statistical significance. Scientists rise up against statistical significance. *Nature*, 567(7748).

- Cockburn et al. 2020: “***misuse of statistical significance*** *as the standard of evidence for experimental success has been identified as* ***a key contributor in the replication crisis****.”*

  Threats of a replication crisis in empirical computer science. *Communications of the ACM*, 63(8).

How to use a p-value wisely

An observed p-value indicates by how much we should ***update*** our confidence in H_0 (not: how confident we should *be* in H_0)

$$\underbrace{\text{Odds}(H_0 \mid D)}_{\text{posterior odds}} \approx \underbrace{\text{Odds}(H_0)}_{\text{prior odds}} \times 2p$$

If we do not provide an estimate for the prior odds, we have no argument to reject H_0 (and that's perfectly fine too)

a small p stands on its own merits: we *can* conclude that the odds for H_0 have decreased by a factor of about $2p$

If we improved a well-established state-of-the-art algorithm or invented cold fusion or find a room-temperature superconductor at atmospheric pressures, the prior odds of H_0 are usually high, say, at least 10^4.

the higher the prior odds for H_0, the more exceptional or surprising is the scientific result to accept $\neg H_0$ with the same confidence we had in H_0 before, we need $p \approx P(\neg H_0)^2$

Summary

- Target definitions (the convergence test condition) can be separated from benchmarked algorithms

thereby, results can become comparable across publications

- We aggregate over different target values but not over dimension
- (simulated) restarts can integrate out success rates
- We use a budget-free experimental setup
- We read data profiles preferably as horizontal data (runtimes) rather then vertically data (success rates)
- Data profiles do not obscure the problem difficulty (as performance profiles do)

Summary

- Target definitions (the convergence test condition) can be separated from benchmarked algorithms

thereby, results can become comparable across publications

- We aggregate over different target values but not over dimension
- (simulated) restarts can integrate out success rates
- We use a budget-free experimental setup
- We read data profiles preferably as horizontal data (runtimes) rather then vertically data (success rates)
- Data profiles do not obscure the problem difficulty (as performance profiles do)