Revisiting COCO with automated benchmarking in mind

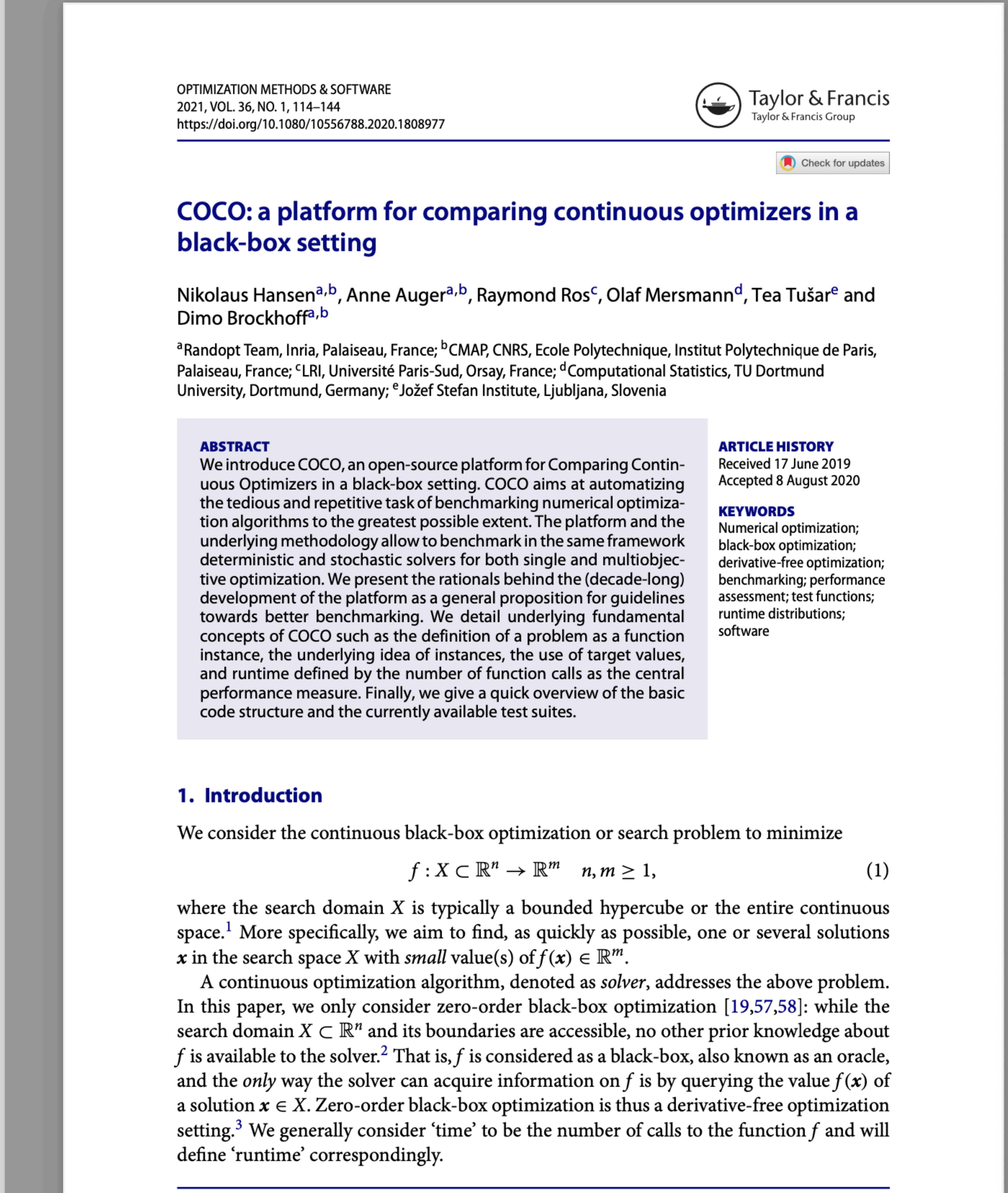
Nikolaus Hansen
Inria
CMAP, CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, France

Presented as invited talk at *BENCH@GECCO25 - Workshop on Good Benchmarking Practices for Evolutionary Computation*, July 2025

feel free…

…feel free to ask questions…

papers…



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

Taylor & Francis
Taylor & Francis Group

Check for updates

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

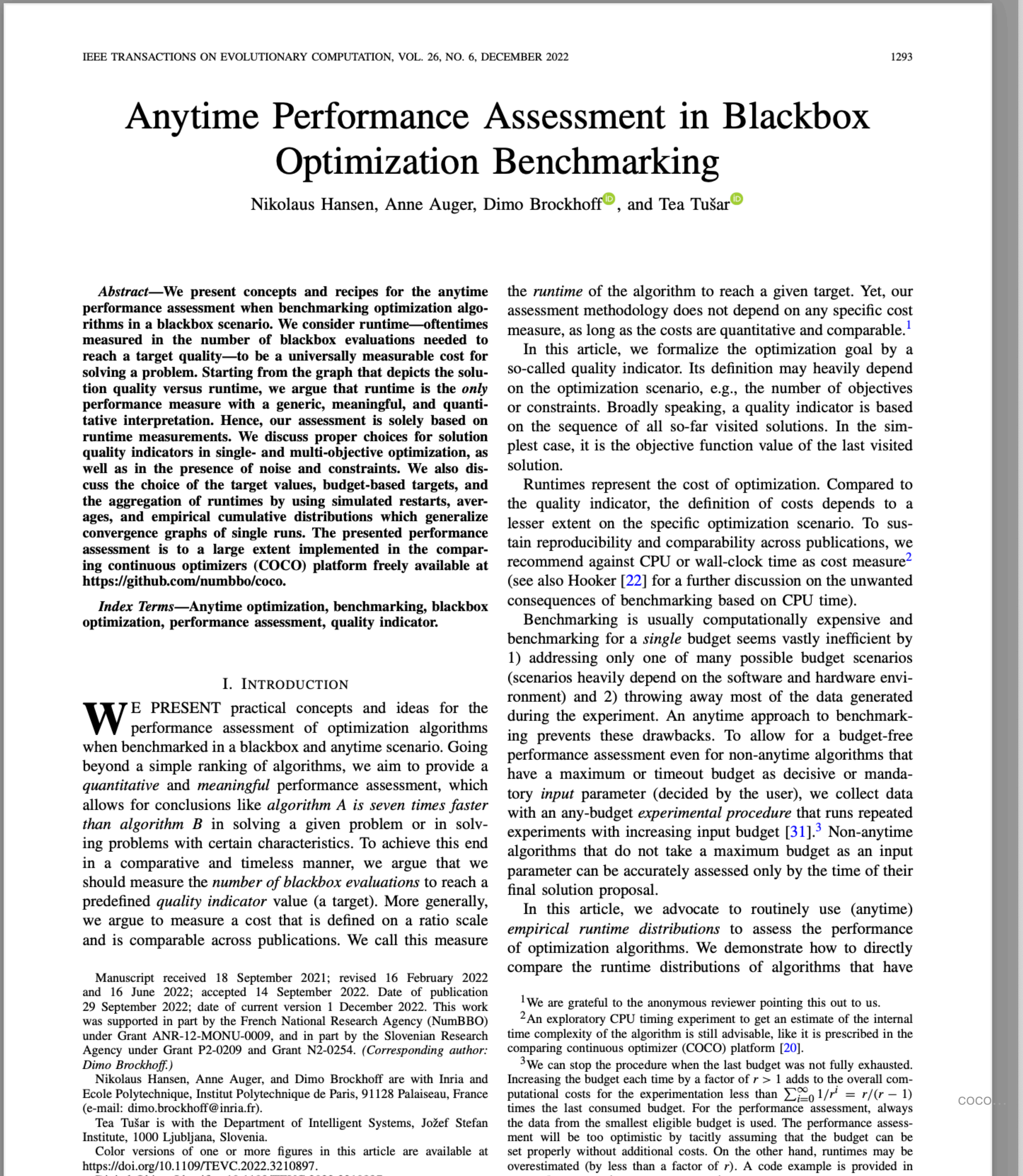
1. Introduction

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

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

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.



IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 26, NO. 6, DECEMBER 2022 1293

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

COCO/BBOB Flow Chart

User-provided solver — C → C/C++ interface — C

User-provided solver — Python → Python interface — Python

User-provided solver — Java → Java interface — Java

User-provided solver — Matlab → Matlab/Octave interface — Matlab

COCO experiments

Test suites:
- bbob
- bbob-biobj

…

Logging functionality

C

Results of other solvers — Log files

Results of the user-provided solver — Log files

COCO post-processing — Python

Plots — PDF, SVG

Tables — Latex, HTML

Δ f_{opt} f1, 5-D	1e1 11	1e0 12	1e-1 12	1e-2 12	1e-3 12	1e-5 12	1e-7 12	#succ 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)★3	**1.0** (0)★3	**1.0** (0)★4	**1.0** (0.1)★4	**1.0** (0)★4	**1.0** (0.1)★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

HTML pages — HTML

Latex templates — Latex

data collection

what to benchmark and experimental setup

measures used and presented

1 Sphere
MCS huyer nois
NELDERDOERR do
NEWUOA ros noi
RANDOMSEARCH a
NIPOPaCMA losh
lmm-CMA-ES aug
BFGS-P-StPt
15 instances
target Df: 1e-8
v2.2.1.216
2
3
5
10
20
40
1
3
5
bbob f1-f24, 20-D
51 targets: 100..1e-08
15 instances
Fraction of function,target pairs
log10(# f-evals / dimension)
best 2009
NIPOPaCM
NEWUOA r
lmm-CMA-I
BFGS-P-St
NELDERDC
MCS huyer
RANDOMSI
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
10 Ellipsoid
21 targets: 100..1e-8
15 and 15 instances
log10(# f-evals)
2
3
4
5
Benchmarking Results on the bbob Suite
Home
Runtime distributions (ECDFs) per function
Runtime distributions (ECDFs) summary and function groups
Scaling with dimension for selected targets
Tables for selected targets
Runtime distribution for selected targets and f-distributions
Runtime loss ratios
Runtime distributions (ECDFs) over all targets
bbob - f1-f24
51 targets in 100..1e-08
15 instances
Proportion of function+target pairs
log10 of (# f-evals / dimension)

Figure by Tea Tušar, in Hansen et al (2020), COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, published online, Aug 2020.

goals…

Benchmarking: Possible Goals

- Understanding algorithms

- Regression testing

- Measuring performance in a systematic way (a performance “profile”)

- Running a competition → ranking (or selection)

- Algorithm selection (or ranking), could be automated

revisiting…

Automated Benchmarking: Possible Goals

- ~~Understanding algorithms~~
- ~~Regression testing~~
- Measuring performance in a systematic way (a performance "profile")
- Running a competition → ranking (or selection)
- Algorithm selection (or ranking), could be automated

how…

Benchmarking: The Global Picture

Two *surprisingly* *(but not completely)* *independent* *issues* to address

1. **What to benchmark**? For example, which collection of test problems?

2. **How to assess performance**?

- experimental setup

- data collection

- measures used

suites…

What to benchmark: COCO/BBOB Test Suite(s)

- Functions are
  - Based on known (analytical) functions, modelling a “known” difficulty
  - Comprehensible

    aiding human interpretation, however, in science, comprehensible is invariably superior to incomprehensible
  - Scalable
  - Difficult (also: non-separable)

    compared to the “typical standard” (at that time)
  - Quasi-randomized as instances

    with arbitrary shifts and smallish irregularities to avoid artificial exploits and mitigate overfitting, emulates repetition of experiments
  - Come sometimes in pairs

    to observe the effect of a single property
- Require to define target values (function instance + target = problem)

  natural targets in the discrete search domain are known fitness levels and the global optimum, we may need experiments to define useful targets

revisiting…

What to automated benchmark

- Functions should be

  ✓ Difficult (also: non-separable)

  compared to the “typical standard” (at that time)

  ✓ Quasi-randomized as instances

  with arbitrary shifts and smallish irregularities to avoid artificial exploits and mitigate overfitting, emulates repetition of experiments

  - Scalable

  - ~~Based on known (analytical) functions, modelling a “known” difficulty~~

  - ~~Comprehensible~~

  aiding human interpretation, however, in science, comprehensible is invariably superior to incomprehensible

  - ~~Come sometimes in pairs~~

  to observe the effect of a single property

- Require to define target values (function instance + target = problem)

  natural targets in the discrete search domain are known fitness levels and the global optimum, we may need experiments to define useful targets

representative …

✓ Difficult (also: non-separable)

compared to the “typical standard” (at that time)

✓ Quasi-randomized as instances

with arbitrary shifts and smallish irregularities to avoid artificial exploits and mitigate overfitting, emulates repetition of experiments

- Scalable

- ~~Based on known (analytical) functions, modelling a “known” difficulty~~

- ~~Comprehensible~~

aiding human interpretation, however, in science, comprehensible is invariably superior to incomprehensible

- ~~Come sometimes in pairs~~

to observe the effect of a single property

- Require to define target values (function instance + target = problem)

natural targets in the discrete search domain are known fitness levels and the global optimum, we may need experiments to define useful targets

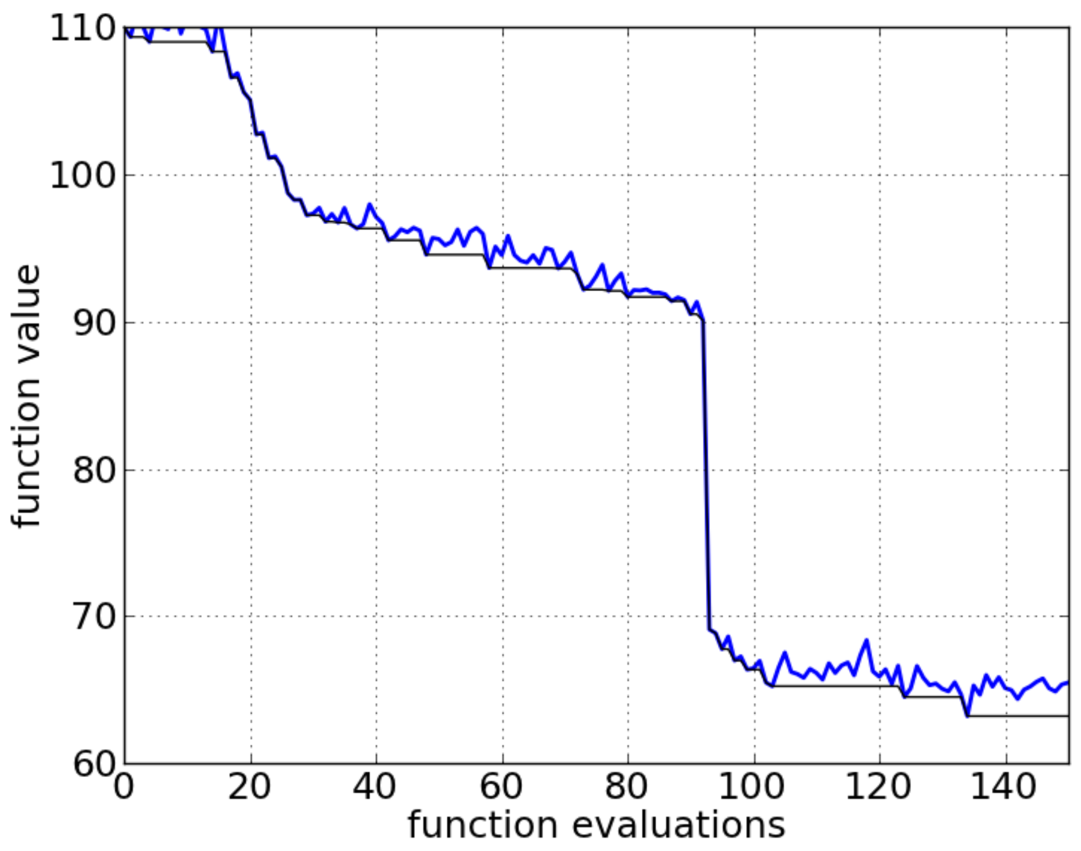
- **Benchmark suites “*ultimately define the anticipated purpose*” of the benchmarked algorithms and hence should (must?) be representative**

assessment…

COCO/BBOB: Performance Assessment

“quality indicator” versus “time”
convergence graphs

is all we have



time means…

Specifically

- **time**: we use number of function evaluations

the number is comparable across publications and invariant under changes of computer hardware, programming language, compiler, OS,…

- **quality indicator**:
  - single-objective: affine transformation of the function value (to be minimized)

different for each instance

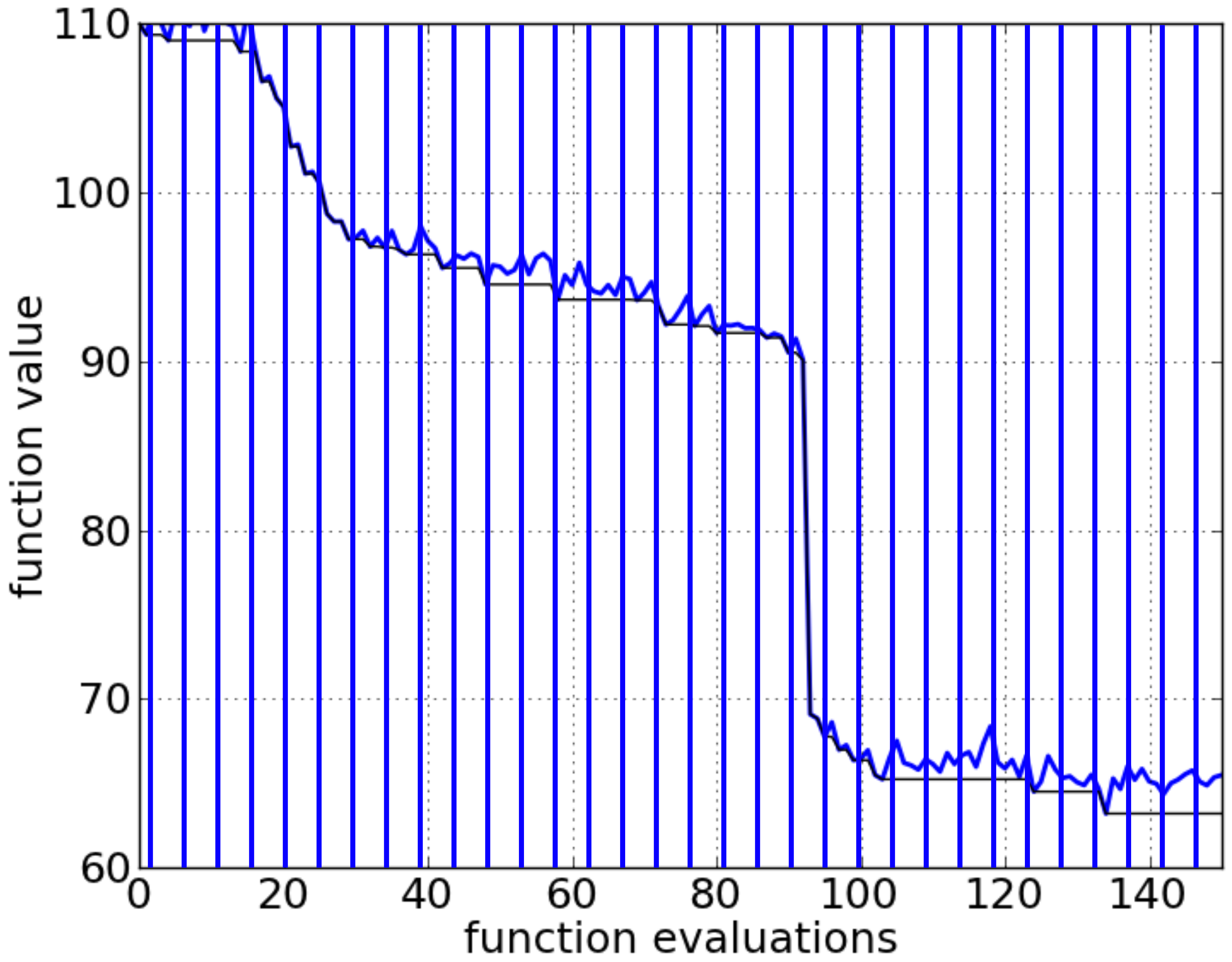
  - multi-objective: negative hypervolume value after objective-wise affine transformation (to be minimized)

Affine transformations are considered as part of the function definition (benchmark suite definition)

they also affect the target values that define a problem: target precisions are defined identical for all functions in a suite

two possibilities to collect comparable…

Discretization: Two Possibilities



- a convergence graph
- lower envelope (a monotonous graph)

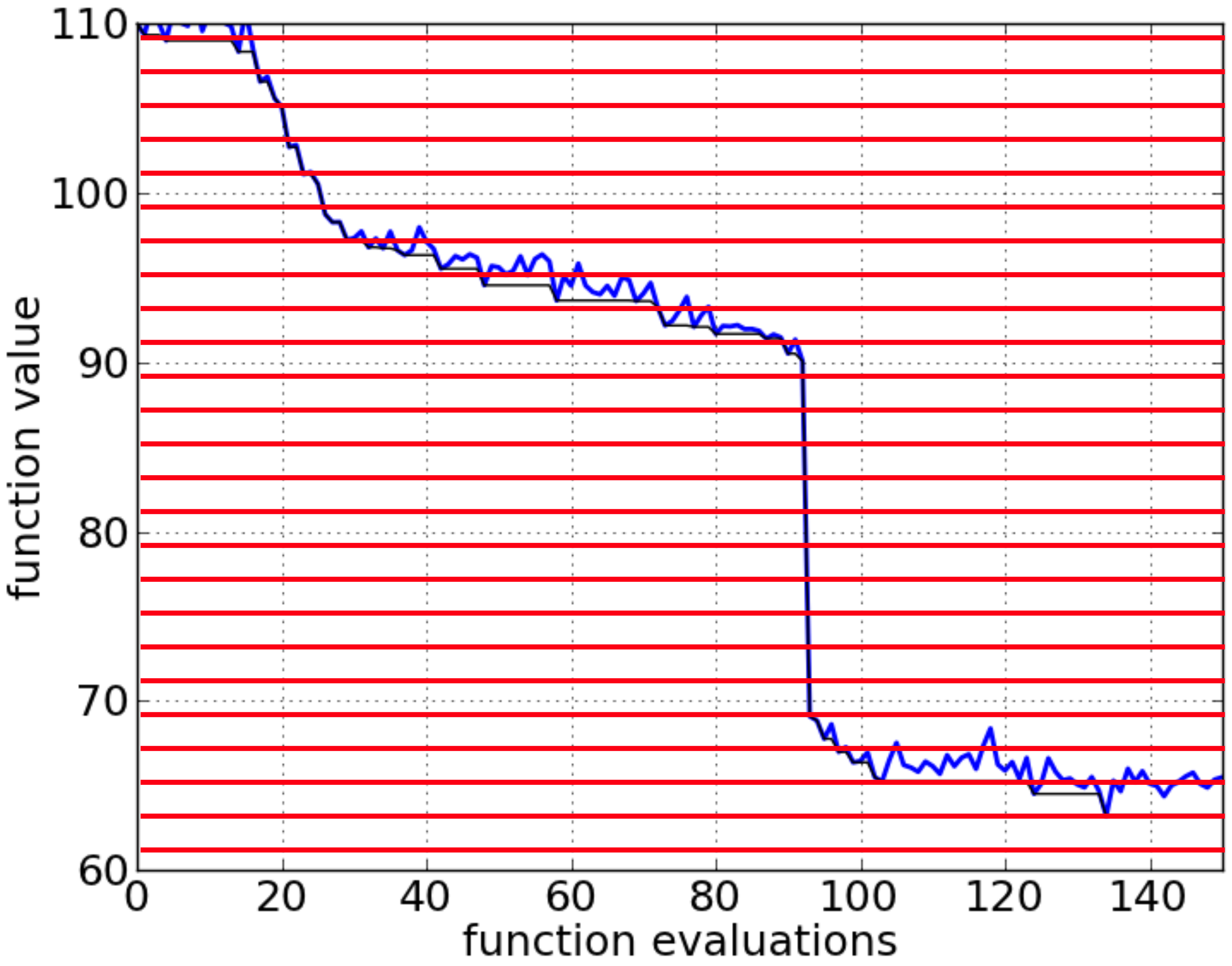
- **vertical** (by evaluation) is a natural discretization

for wall clock or CPU time we would need to determine discretization intervals

- $\Longrightarrow$ evaluations are the independent variable

function value (or its attainment fraction) is the dependent variable, the measurement

Discretization: Two Possibilities



- a convergence graph
- lower envelope (a monotonous graph)

- **horizontal** appears to be an “unnatural” discretization

we need to determine discretization values

- function “target” values (or attainment fractions) are the independent variable

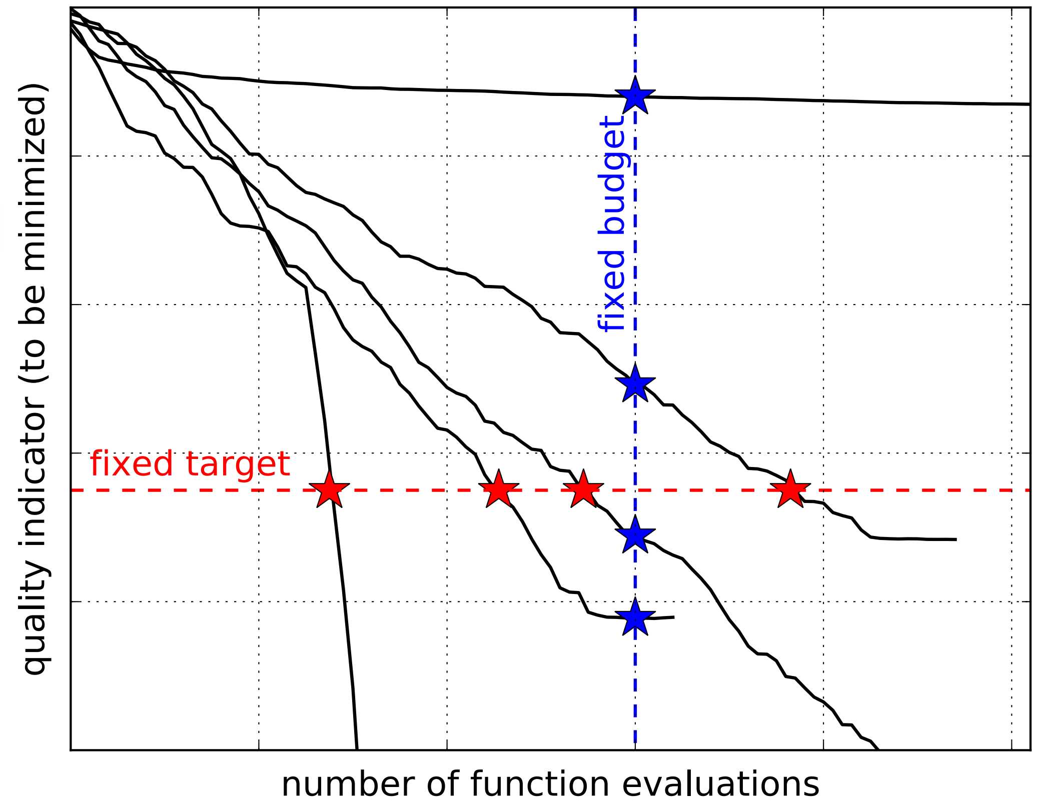
time is the dependent variable, the measurement

- both discretizations recover the original data (in the limit)

a time measurement for each discretization function value, these measurements can be plotted as ECDF

resulting measurements…

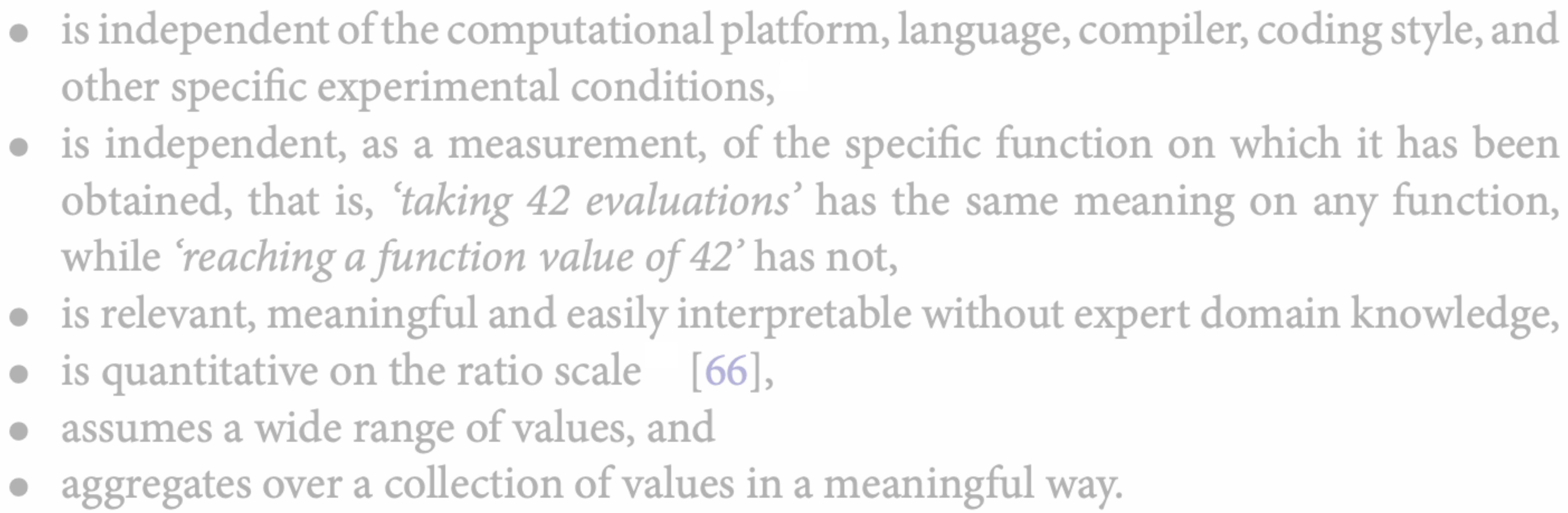
COCO/BBOB: Fixed Target(s) versus Fixed Budget(s)



The resulting measurement

- Vertical discretization = fixed budget design **measures *quality indicator values*** *and/or their attainment fractions*
- Horizontal discretization = fixed target design **measures *evaluations***, aka *runtime*

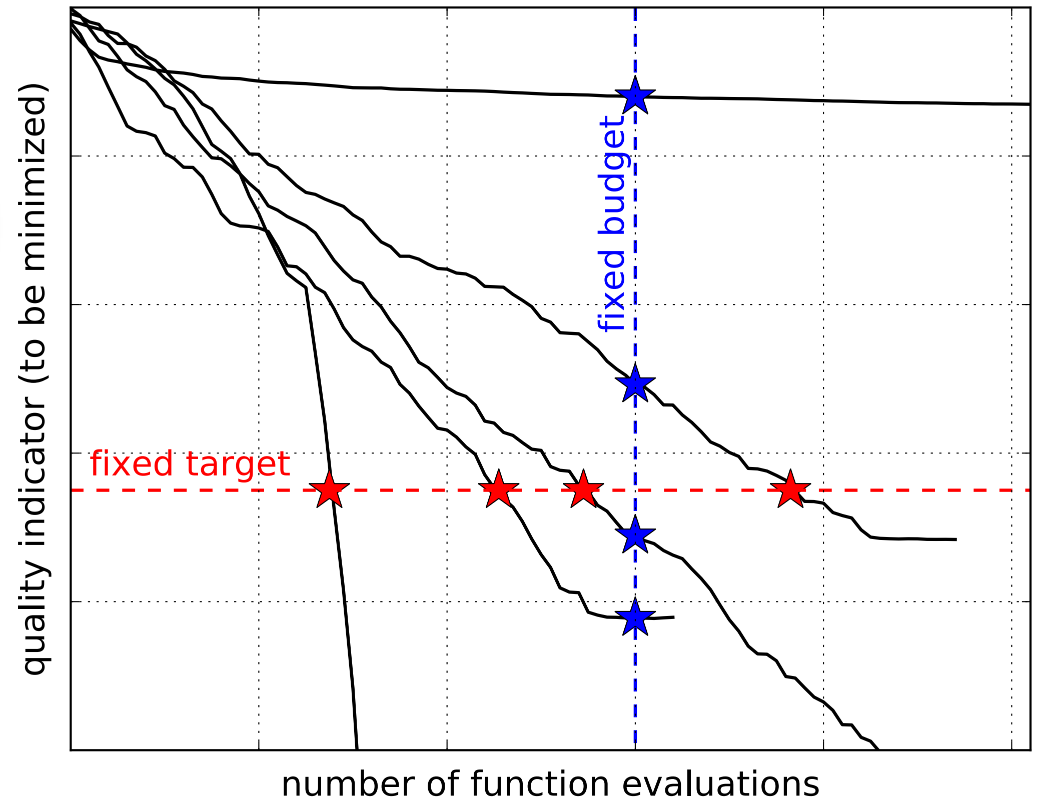
Runtime, when measured in number of evaluations,



- is independent of the computational platform, language, compiler, coding style, and other specific experimental conditions,
- is independent, as a measurement, of the specific function on which it has been obtained, that is, *'taking 42 evaluations'* has the same meaning on any function, while *'reaching a function value of 42'* has not,
- is relevant, meaningful and easily interpretable without expert domain knowledge,
- is quantitative on the ratio scale [66],
- assumes a wide range of values, and
- aggregates over a collection of values in a meaningful way.

Source: Hansen et al. 2021

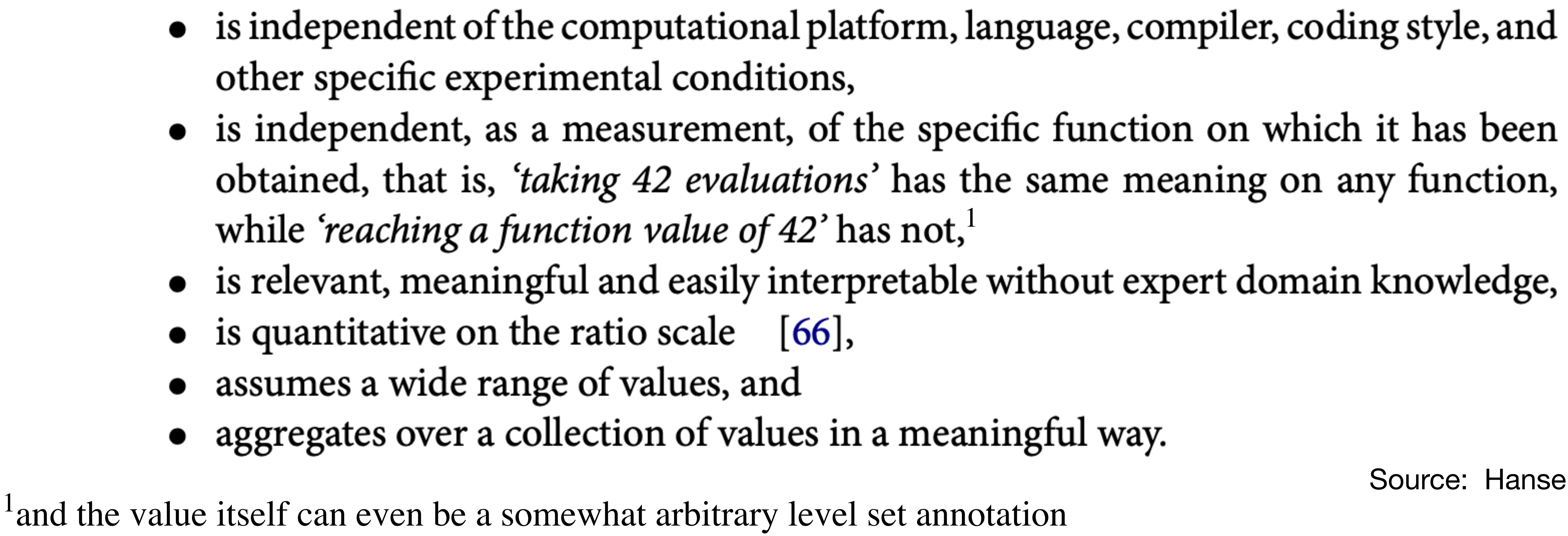
COCO/BBOB: Fixed Target(s) versus Fixed Budget(s)



The resulting measurement

- Vertical discretization = fixed budget design **measures *quality indicator values*** *and/or their attainment fractions*
- Horizontal discretization = fixed target design **measures *evaluations***, aka *runtime*

Runtime, when measured in number of evaluations,



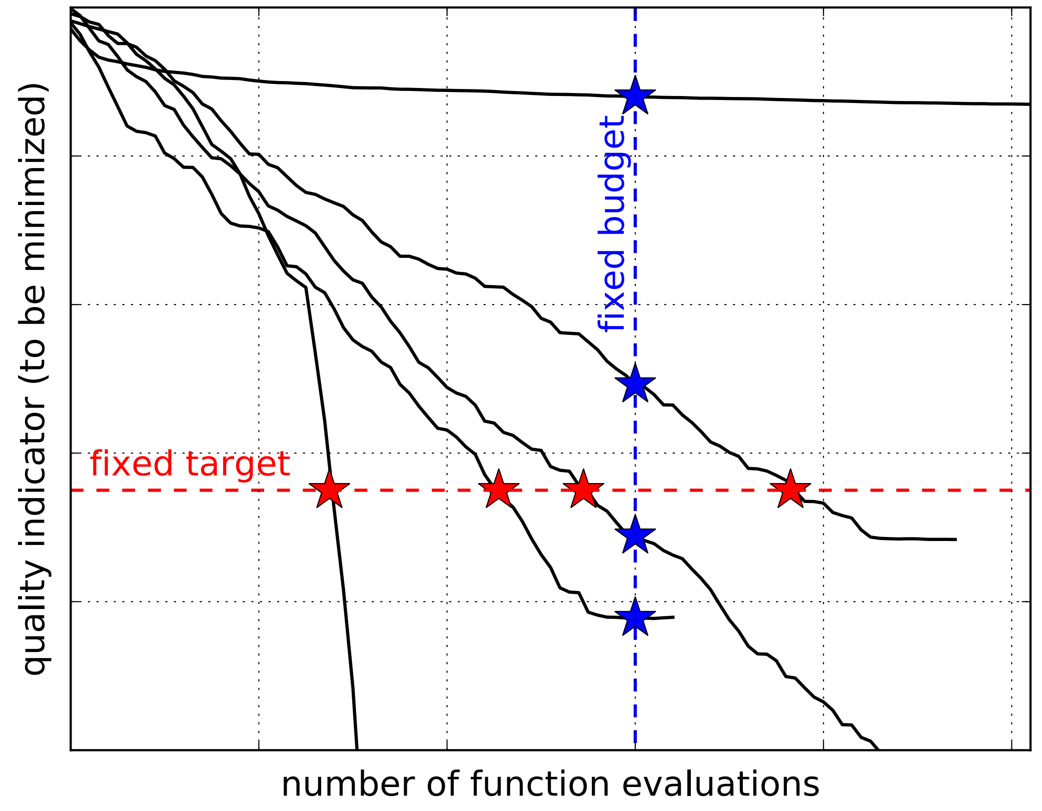
- is independent of the computational platform, language, compiler, coding style, and other specific experimental conditions,
- is independent, as a measurement, of the specific function on which it has been obtained, that is, *'taking 42 evaluations'* has the same meaning on any function, while *'reaching a function value of 42'* has not,[1]
- is relevant, meaningful and easily interpretable without expert domain knowledge,
- is quantitative on the ratio scale [66],
- assumes a wide range of values, and
- aggregates over a collection of values in a meaningful way.

[1] and the value itself can even be a somewhat arbitrary level set annotation

Source: Hansen et al. 2021

automated…

Automated BBOB: Fixed Target(s) versus Fixed Budget(s)

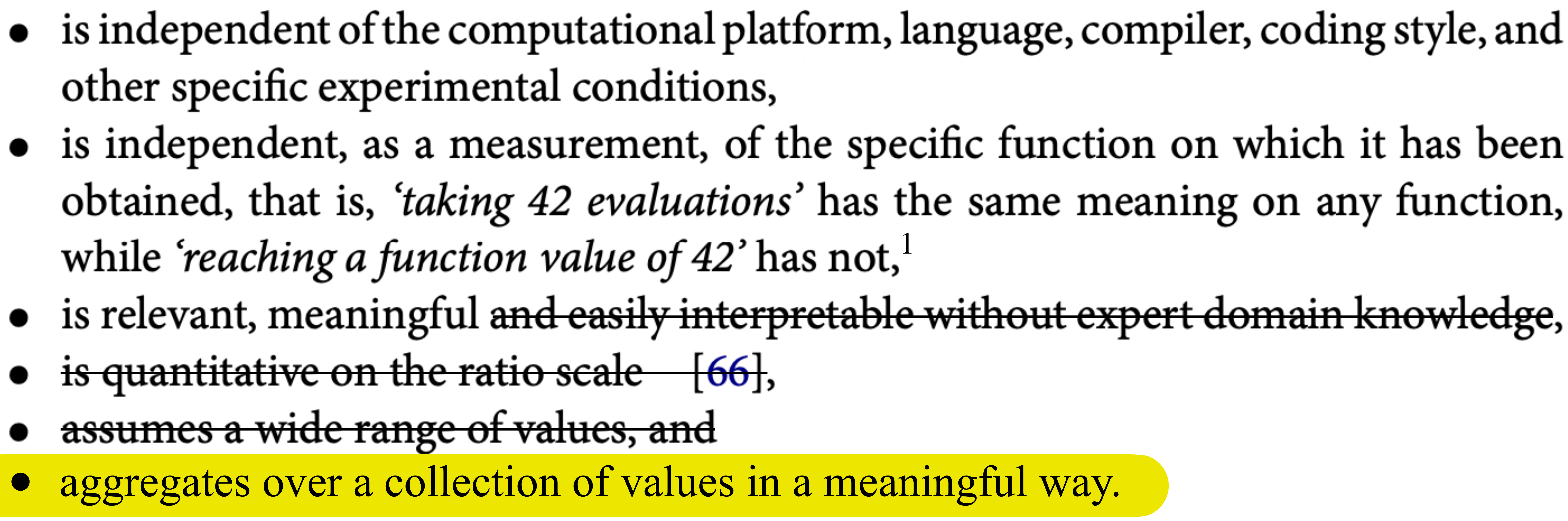


The resulting measurement

- Vertical discretization = fixed budget design **measures *quality indicator values*** *and/or their attainment fractions*

- Horizontal discretization = fixed target design **measures *evaluations***, aka *runtime*

Runtime, when measured in number of evaluations,



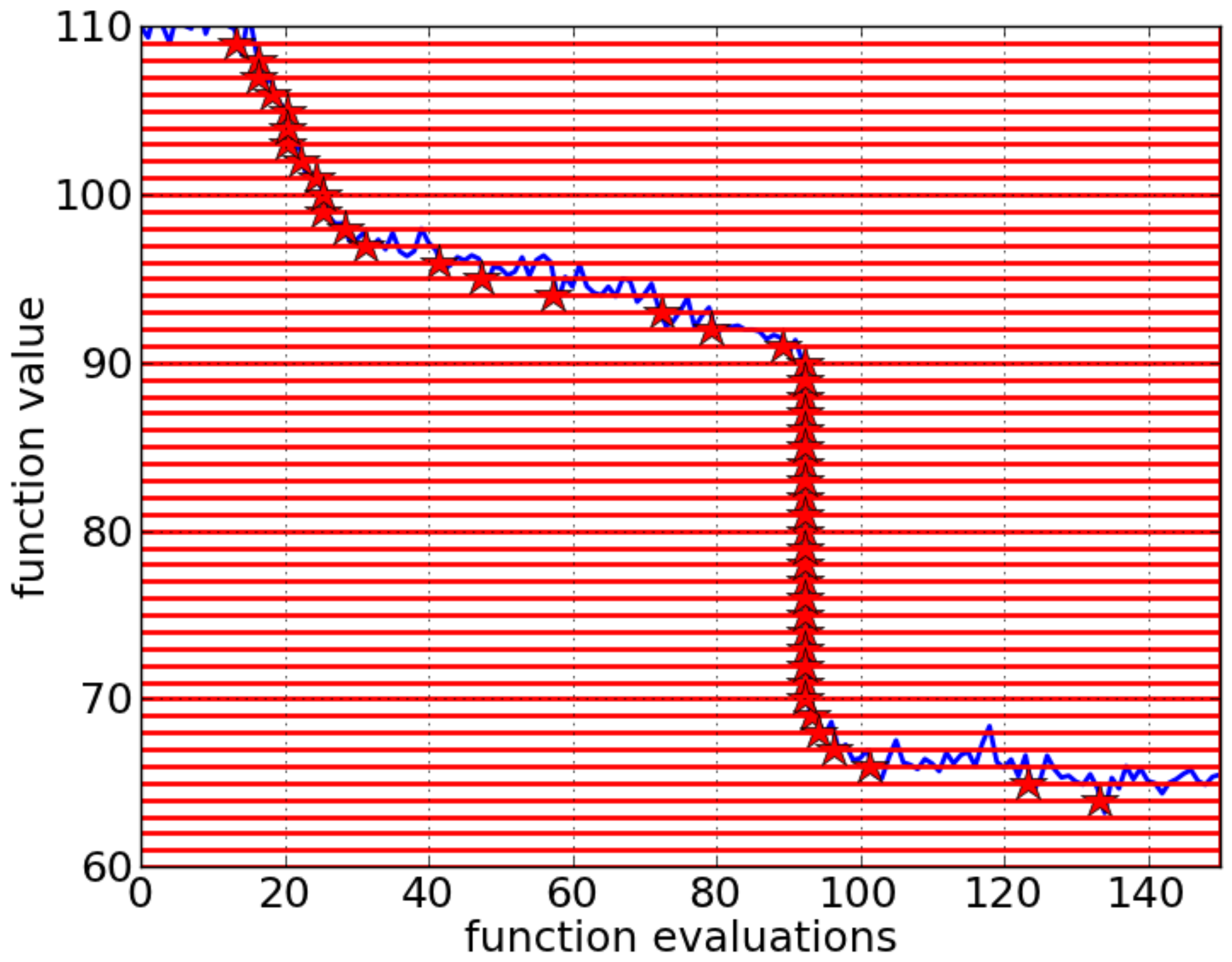
- is independent of the computational platform, language, compiler, coding style, and other specific experimental conditions,
- is independent, as a measurement, of the specific function on which it has been obtained, that is, *'taking 42 evaluations'* has the same meaning on any function, while *'reaching a function value of 42'* has not,[1]
- is relevant, meaningful ~~and easily interpretable without expert domain knowledge~~,
- ~~is quantitative on the ratio scale [66]~~,
- ~~assumes a wide range of values, and~~
- aggregates over a collection of values in a meaningful way.

Source: Hansen et al. 2021

[1] and the value itself can even be a somewhat arbitrary level set annotation

we only use…

COCO/BBOB

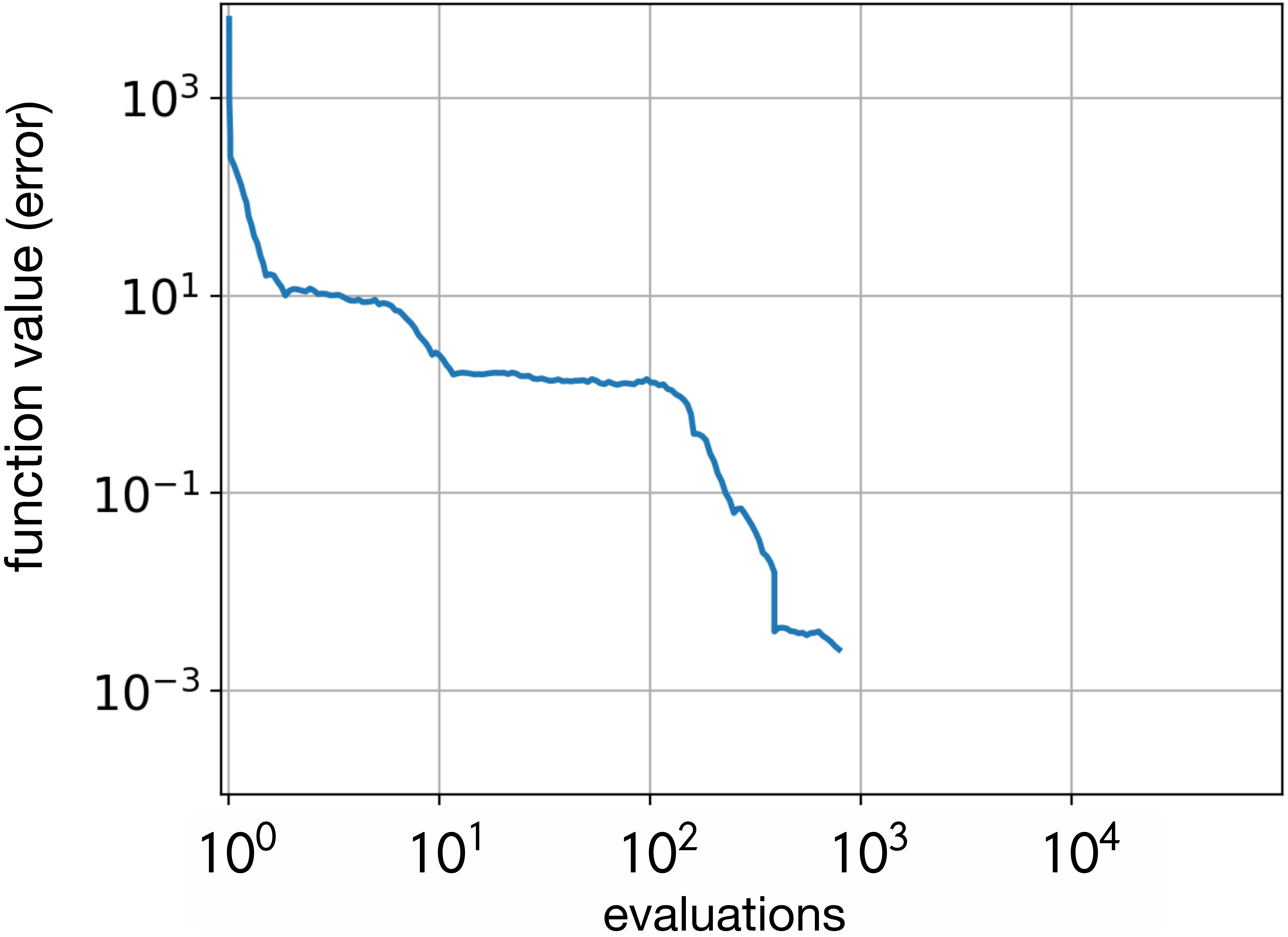


uses (almost) always

horizontal discretization

example…

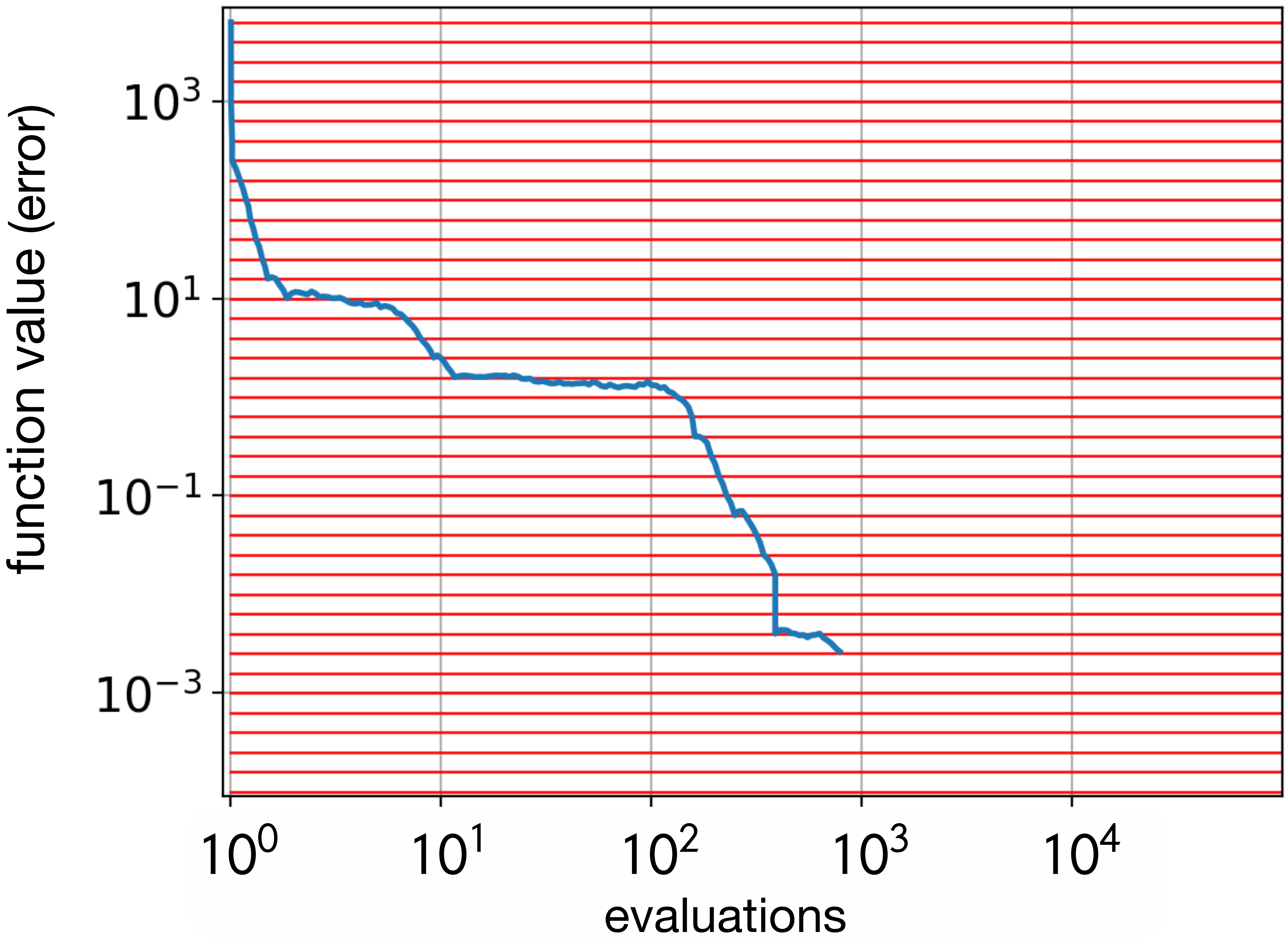
From a Convergence Graph to the Empirical Runtime Distribution



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

discretization…

From a Convergence Graph to the Empirical Runtime Distribution

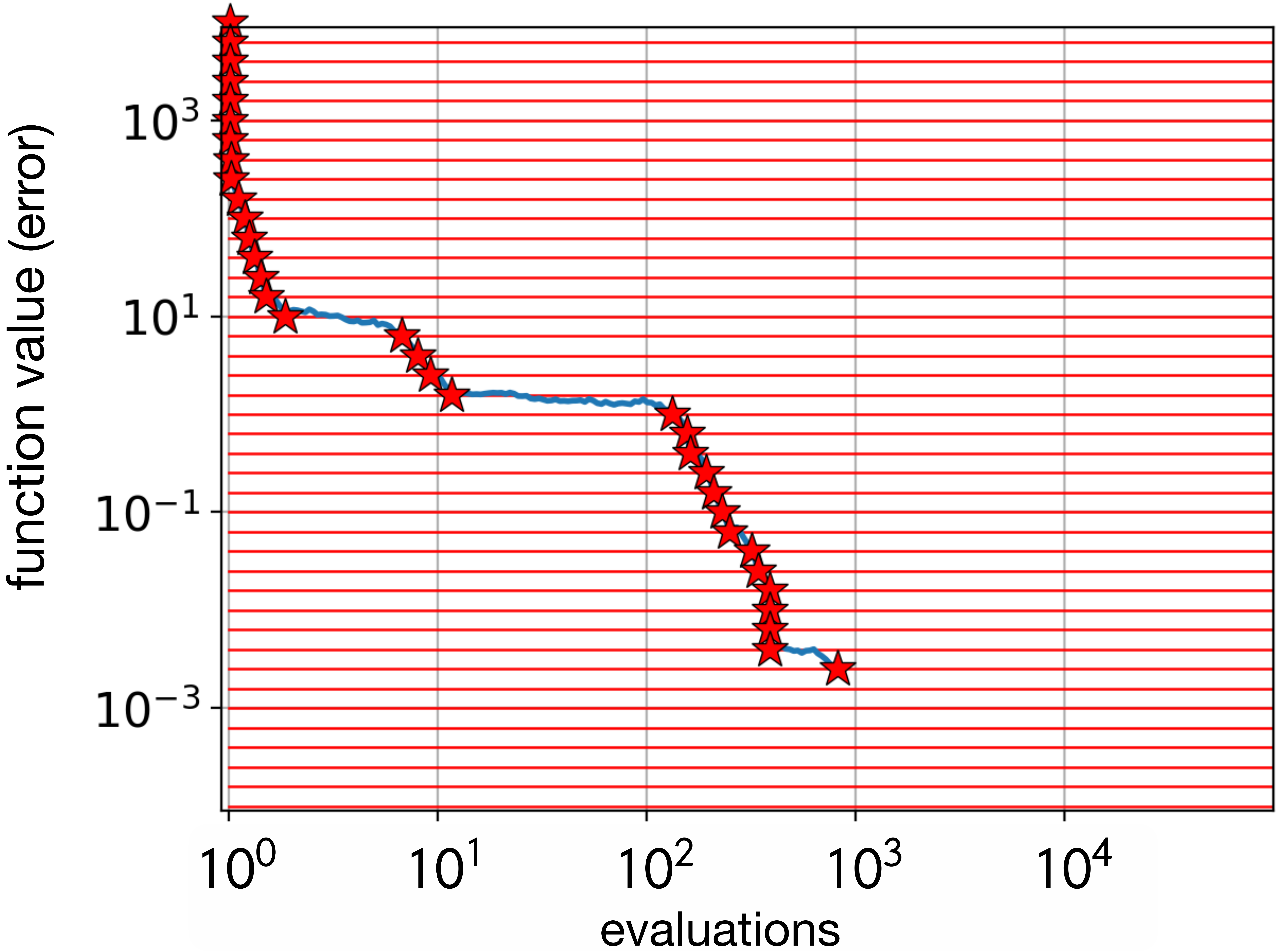


equidistance
“target” values

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

stars…

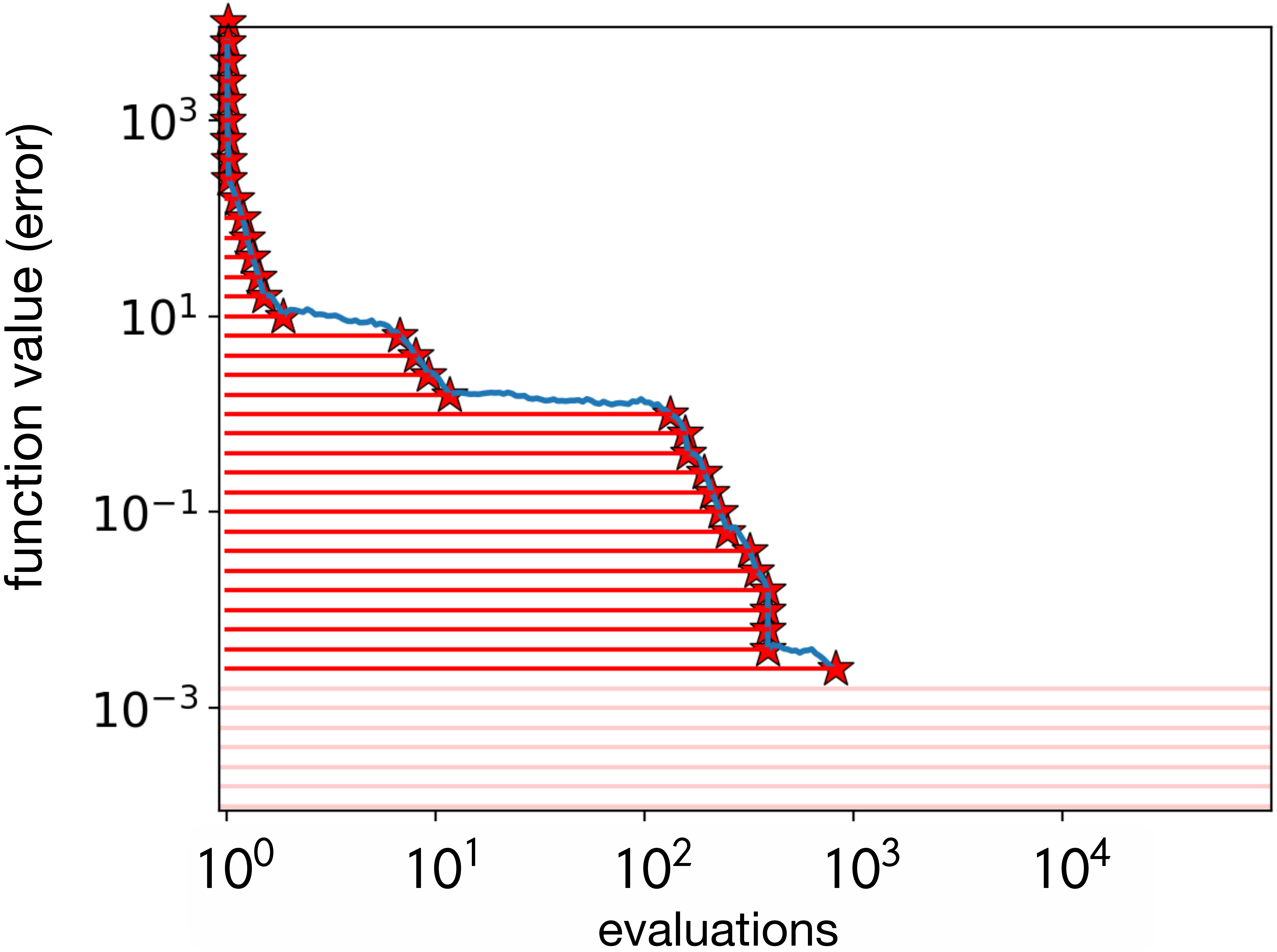
From a Convergence Graph to the Empirical Runtime Distribution



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

bars…

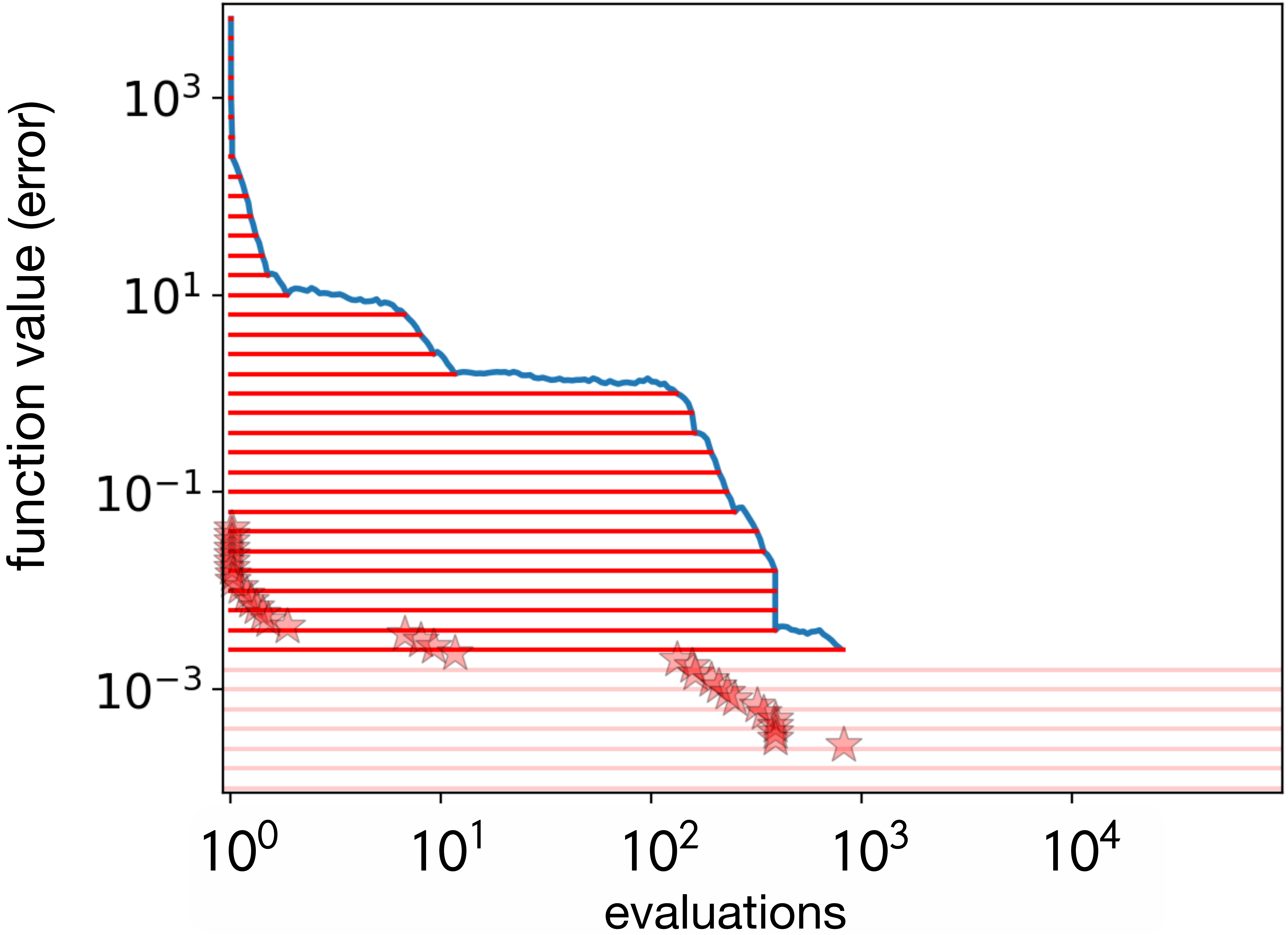
From a Convergence Graph to the Empirical Runtime Distribution



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

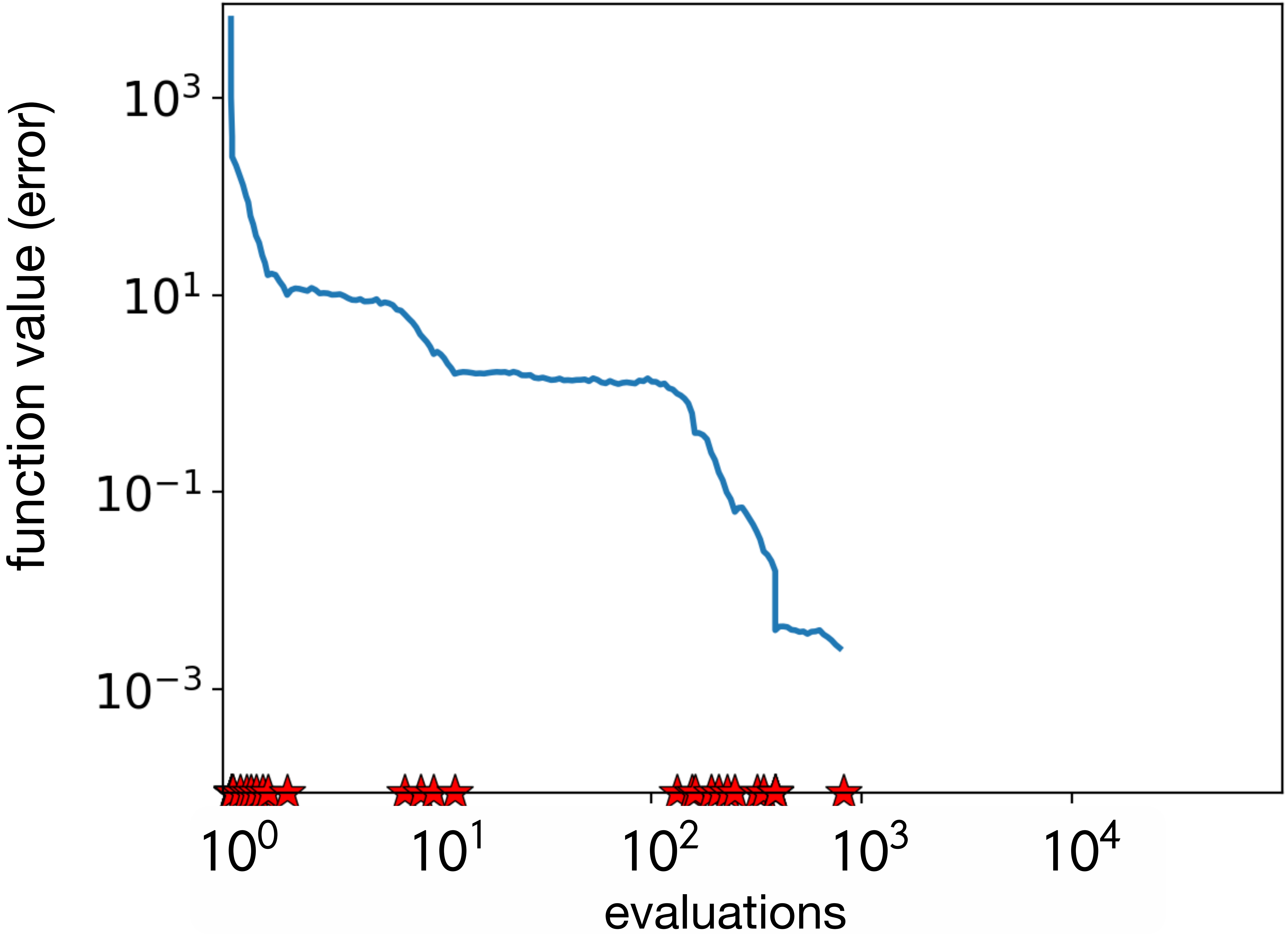
drop…

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).

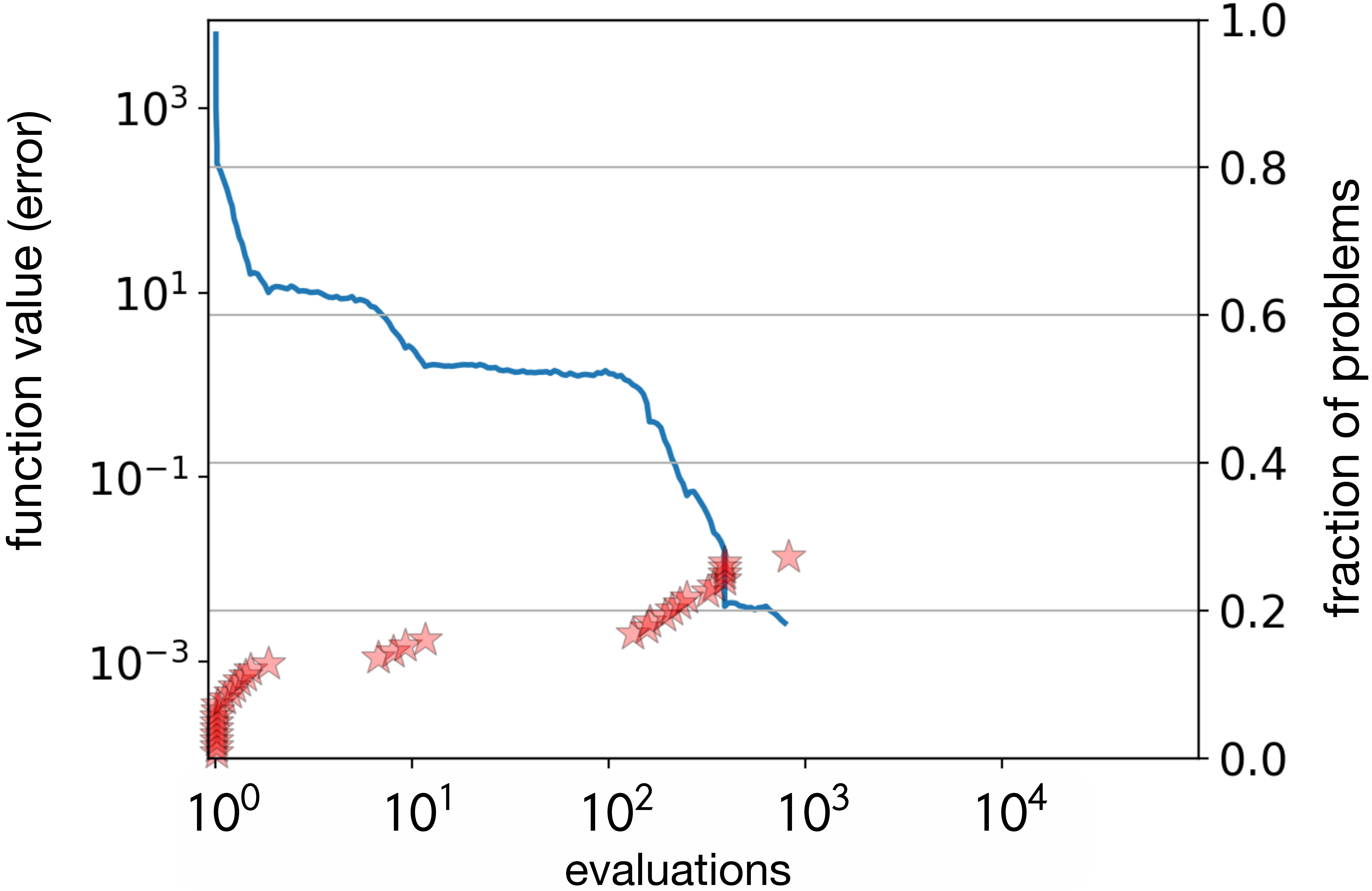
From a Convergence Graph to the Empirical Runtime Distribution



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

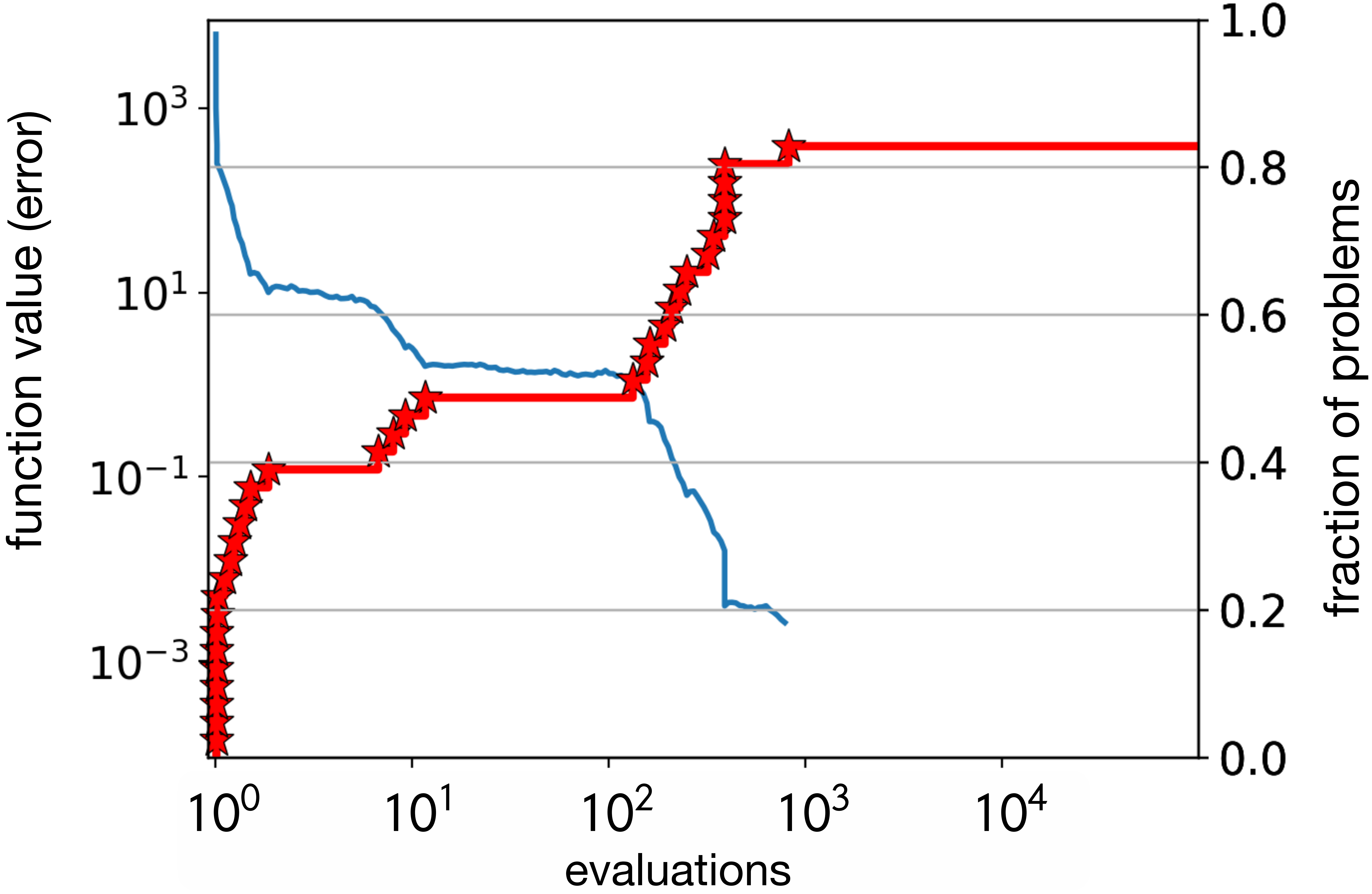
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

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).

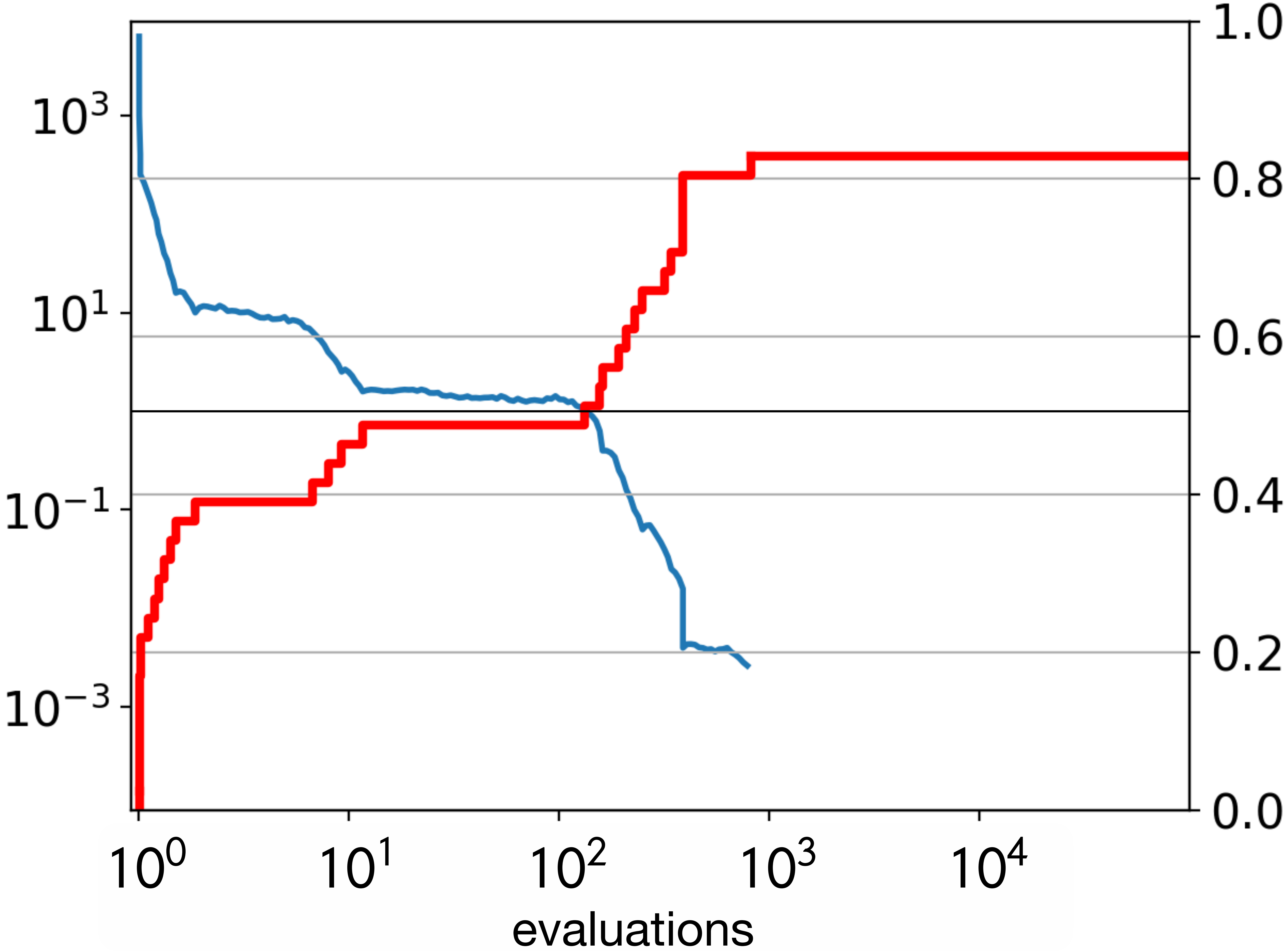
From a Convergence Graph to the Empirical Runtime Distribution



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

two curves…

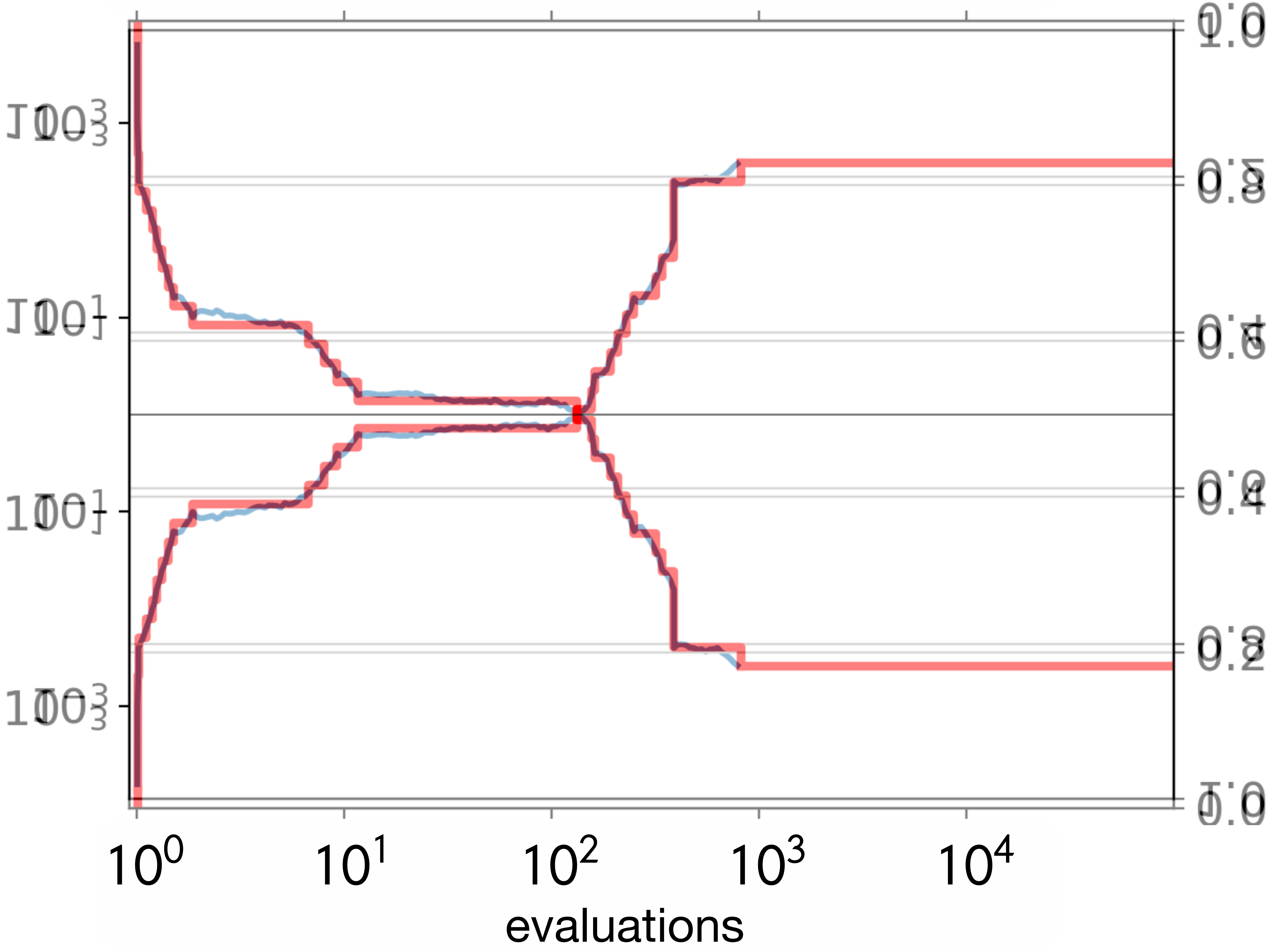
From a Convergence Graph to the Empirical Runtime Distribution



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

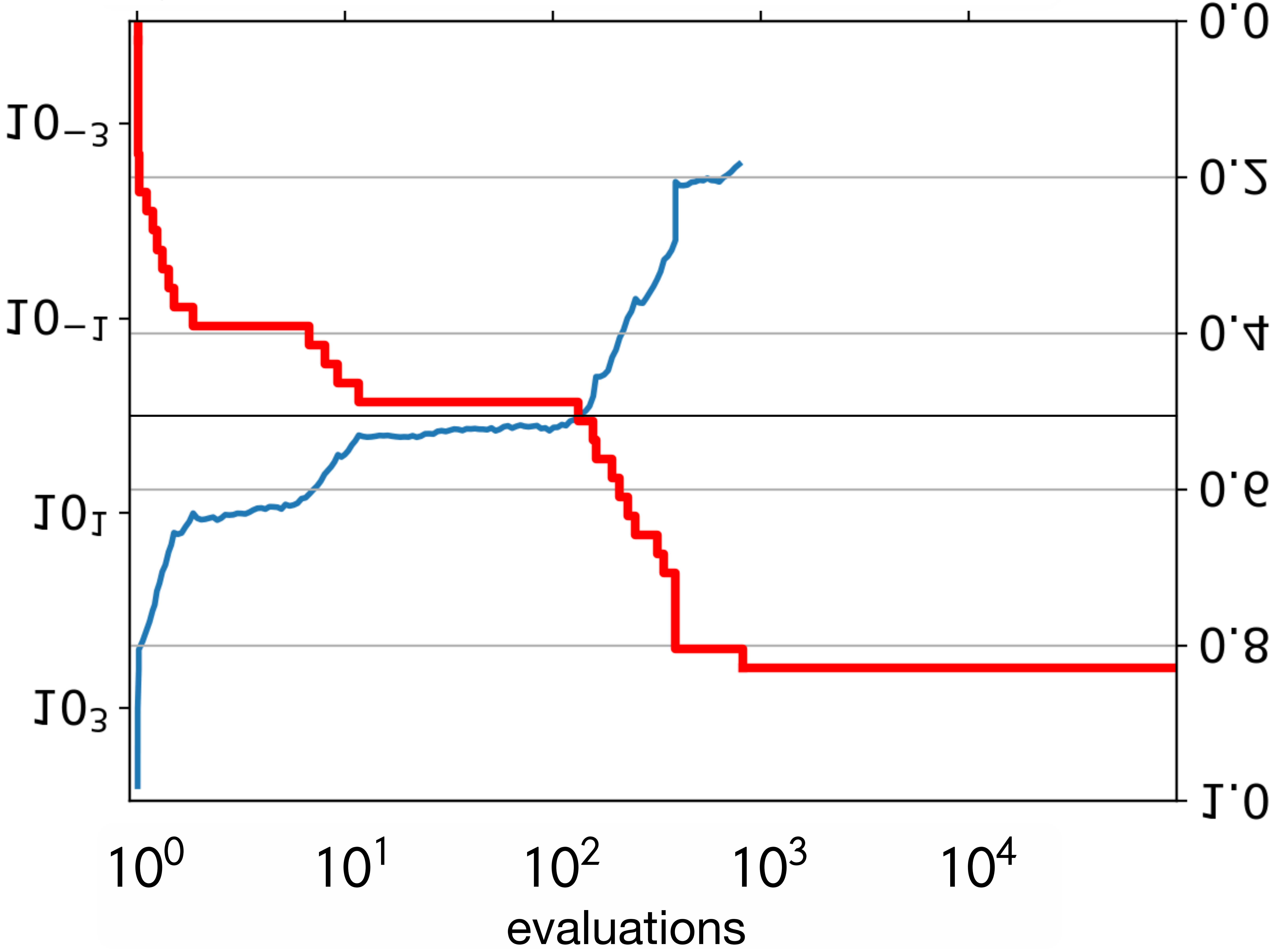
flip...

From a Convergence Graph to the Empirical Runtime Distribution



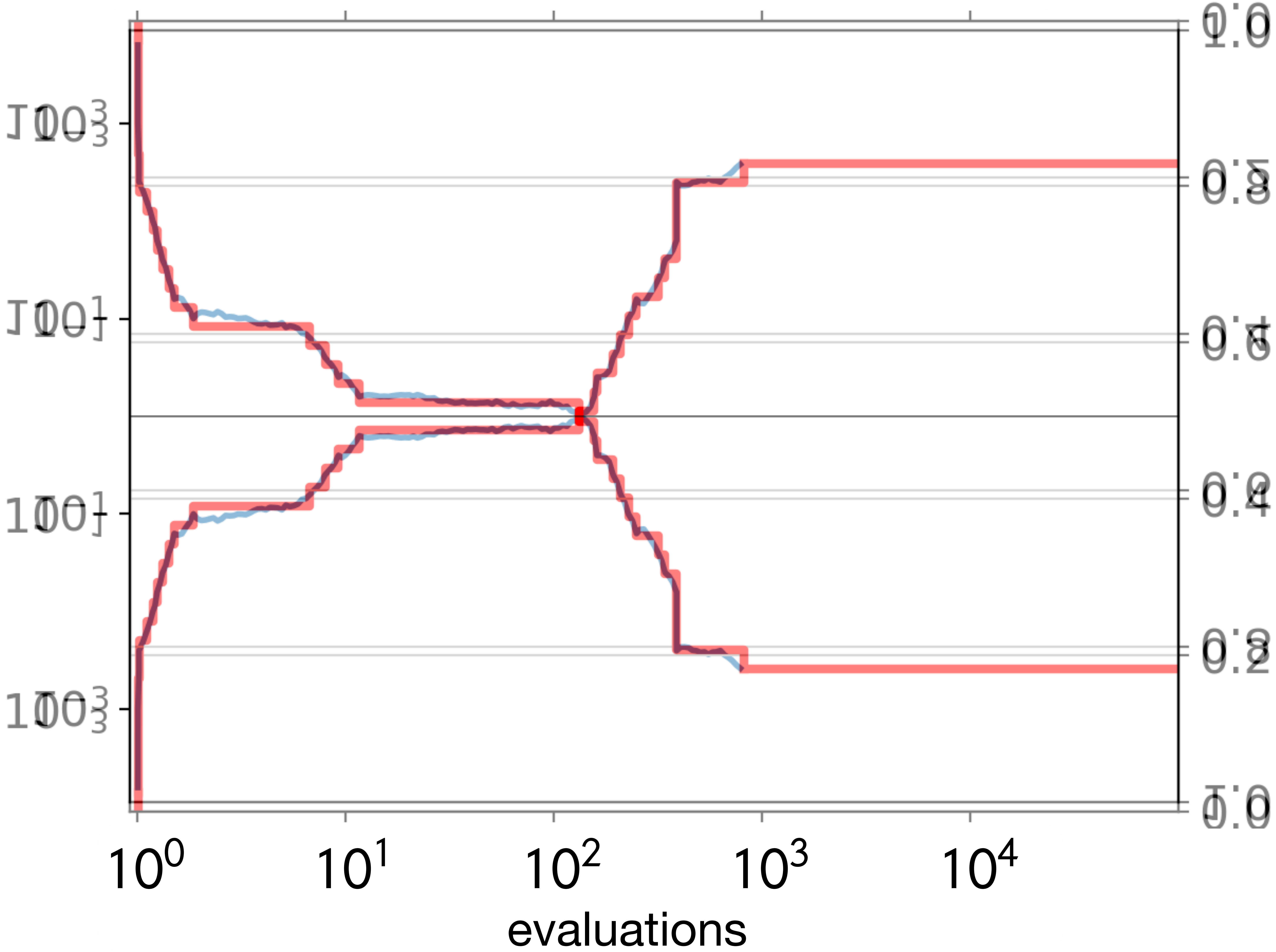
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

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).

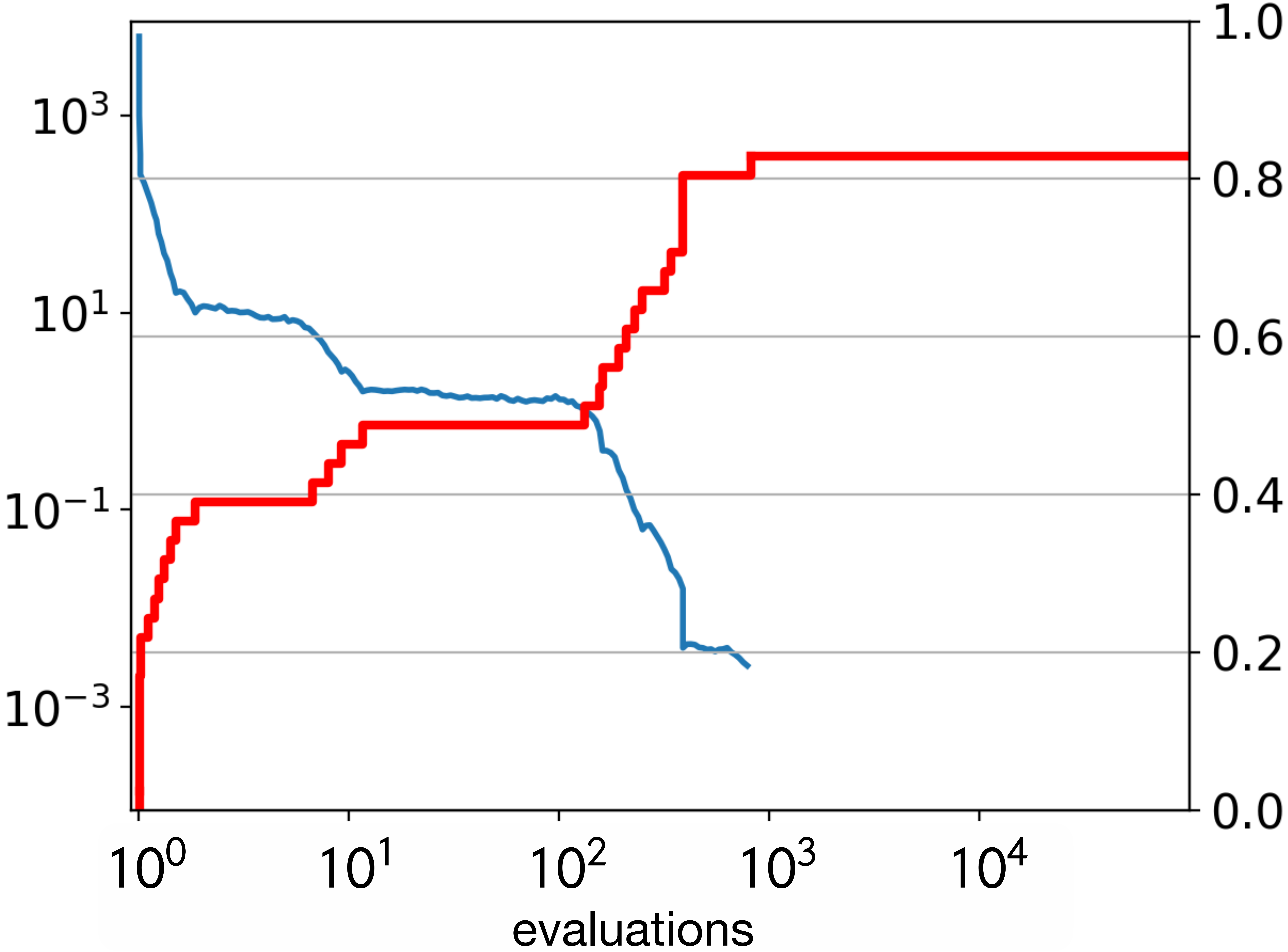
From a Convergence Graph to the Empirical Runtime Distribution



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*

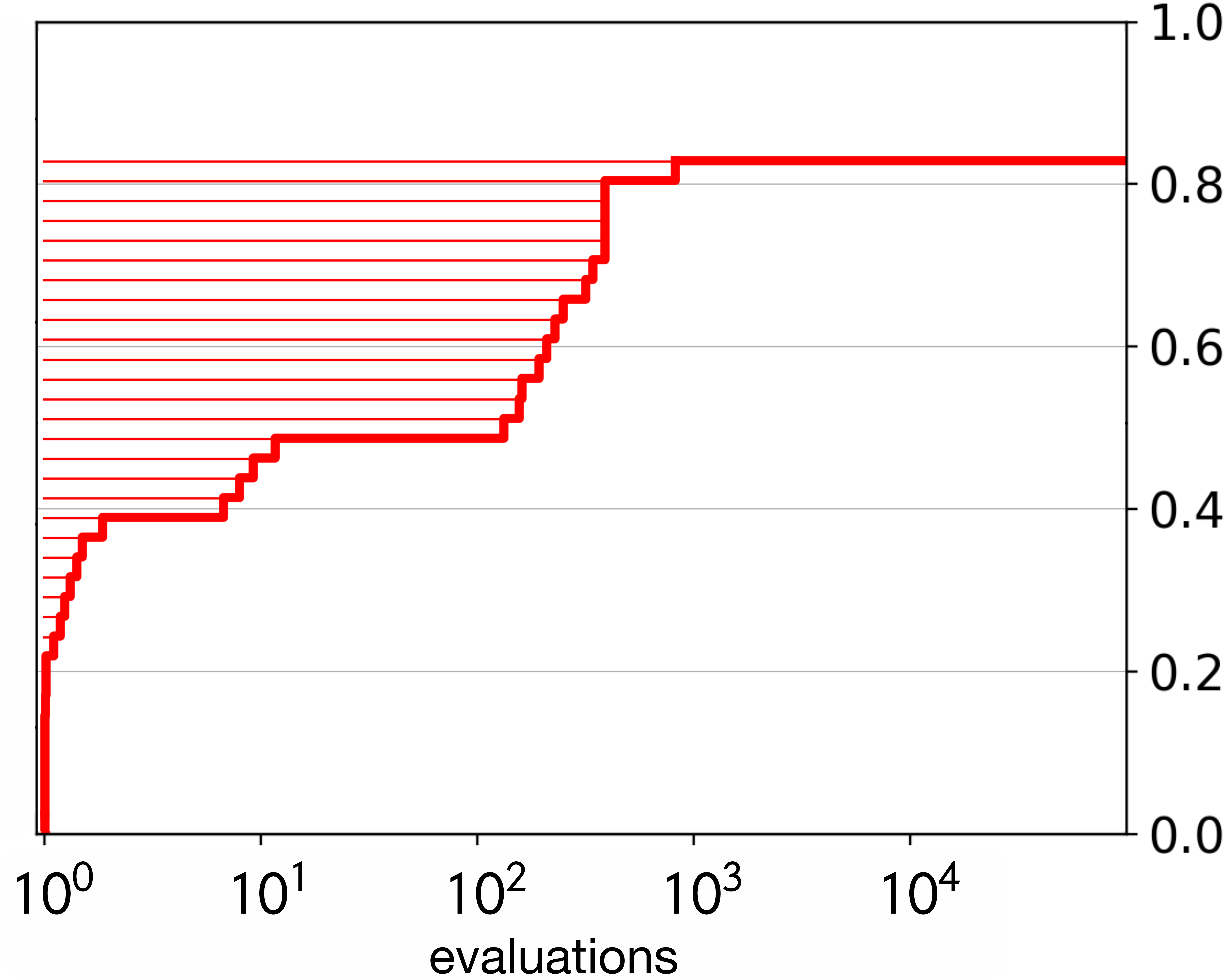


Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

horizontal bars…

From a Convergence Graph to the Empirical Runtime Distribution

the runtime distribution represents the ordered runtimes depicted as horizontal bars



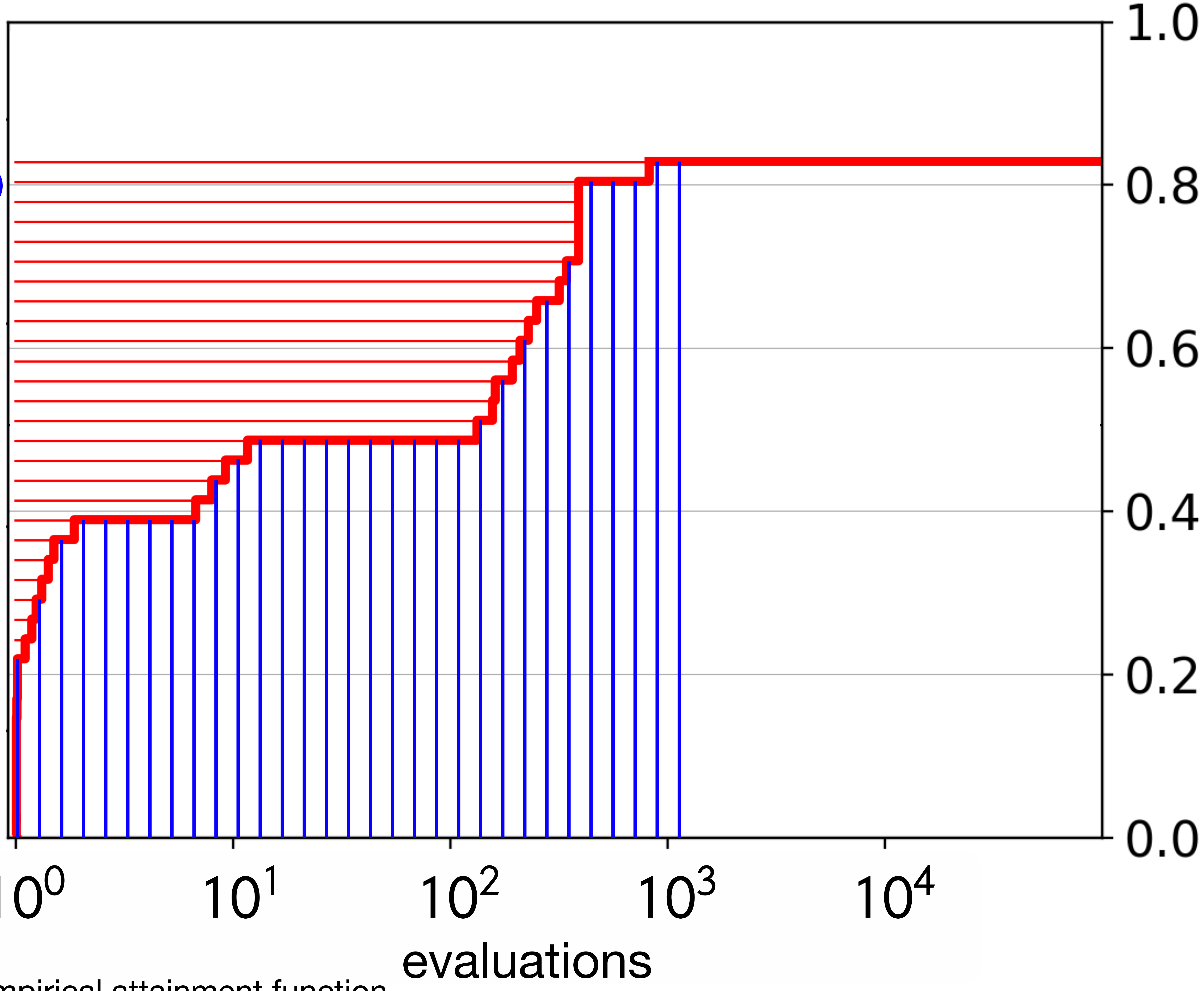
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

two readings…

From a Convergence Graph to the Empirical Runtime Distribution

Two readings of ECDFs (CDFs)
Conventional:
attainment fractions (probabilities)
My (strong) preference:
runtime values

the runtime distribution represents the ordered runtimes depicted as horizontal bars



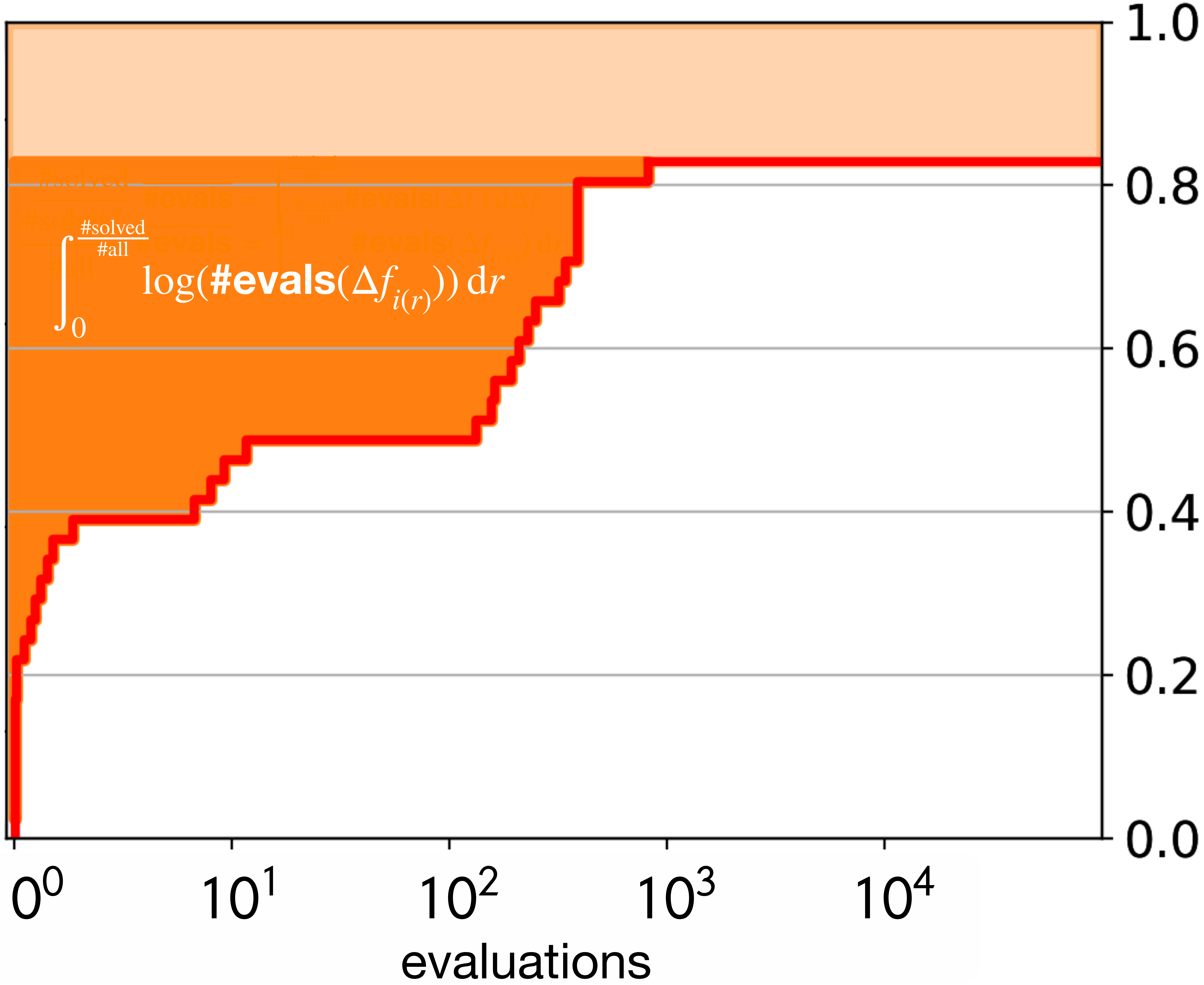
López-Ibáñez, et al. 2024. Using the empirical attainment function for analyzing single-objective black-box optimization algorithms. *IEEE Trans. on EC, 29(2)*.

area…

The Area Above the Empirical Runtime Distribution

the area above the runtime distribution represents a (truncated) average runtime

Specifically, the logarithm of the truncated geometric average equals the bright orange area divided by the truncation fraction



$$\int_0^{\frac{\#\text{solved}}{\#\text{all}}} \log(\textbf{\#evals}(\Delta f_{i(r)}))\,\mathrm{d}r$$

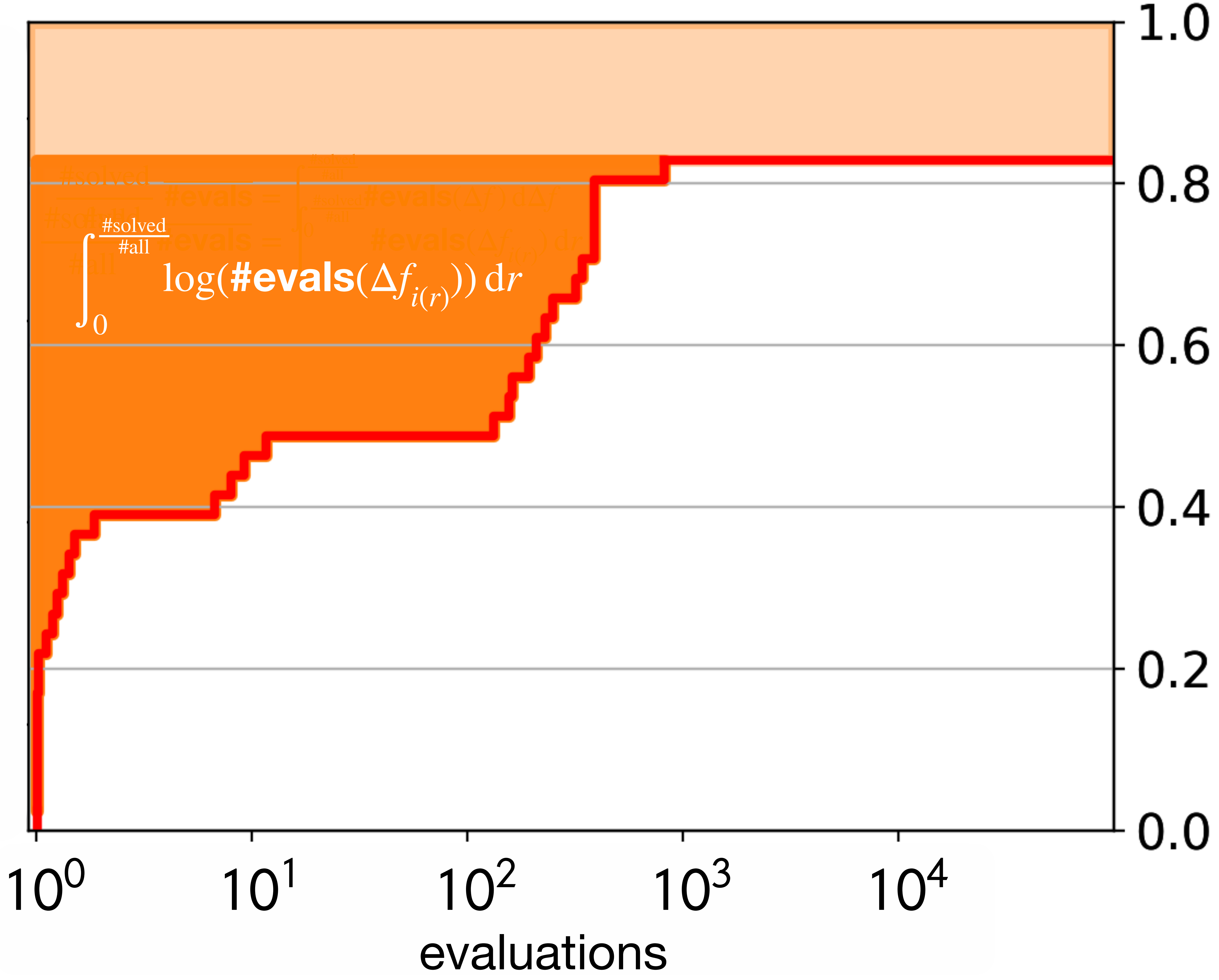
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

formula…

The Area Above the Empirical Runtime Distribution

the area above the runtime distribution represents a (truncated) average runtime

Specifically, the logarithm of the truncated geometric average equals the bright orange area divided by the truncation fraction



$$\int_0^{\frac{\#\text{solved}}{\#\text{all}}} \log(\textbf{\#evals}(\Delta f_{i(r)}))\,\mathrm{d}r$$

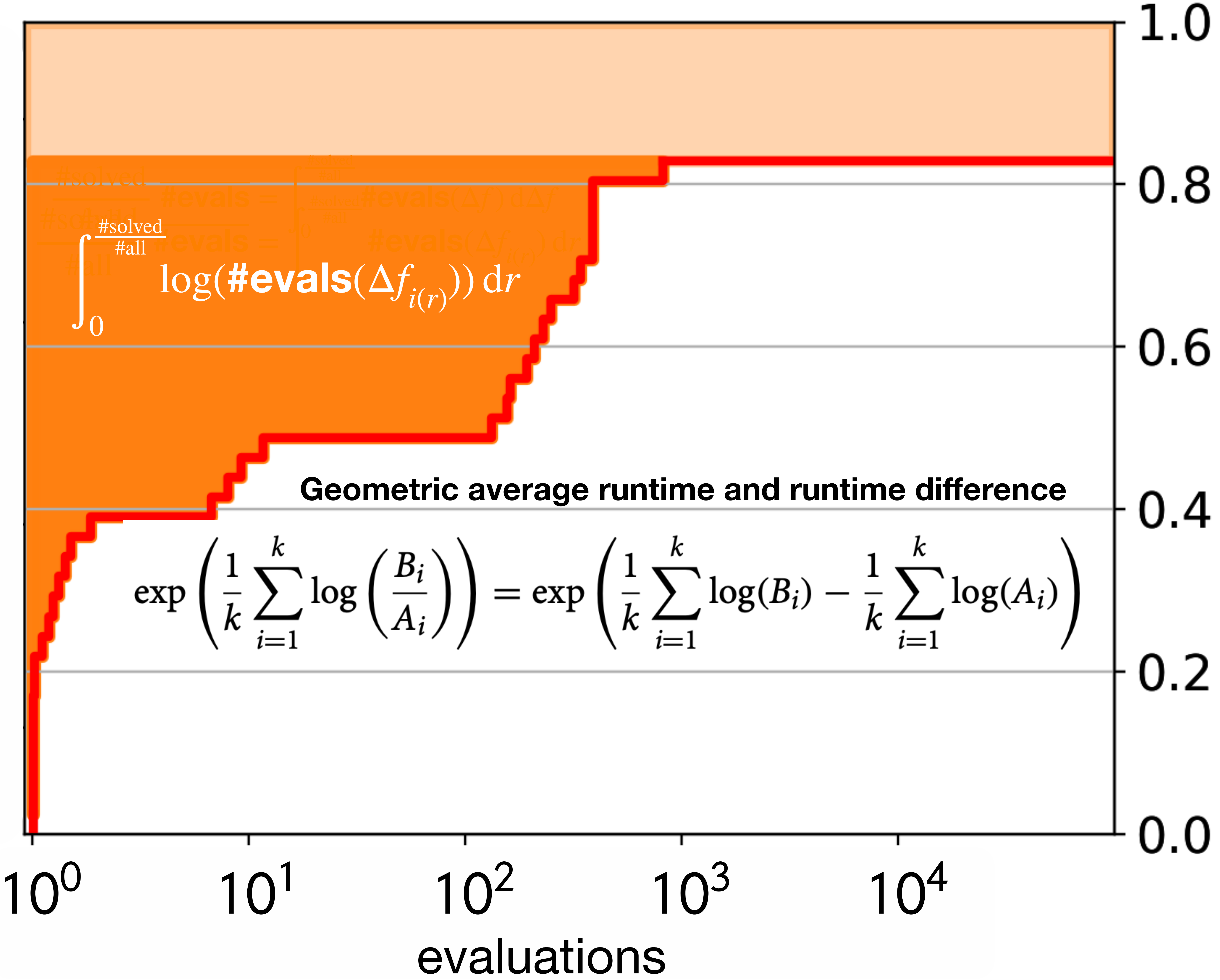
Hansen et al. 2022. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. on EC*, 26(6).

formula…

The Area Above the Empirical Runtime Distribution

the area above the runtime distribution represents a (truncated) average runtime

Specifically, the logarithm of the truncated geometric average equals the bright orange area divided by the truncation fraction



$$\int_0^{\frac{\#\text{solved}}{\#\text{all}}} \log(\textbf{\#evals}(\Delta f_{i(r)}))\,\mathrm{d}r$$

Geometric average runtime and runtime difference

$$\exp\left(\frac{1}{k}\sum_{i=1}^{k}\log\left(\frac{B_i}{A_i}\right)\right) = \exp\left(\frac{1}{k}\sum_{i=1}^{k}\log(B_i) - \frac{1}{k}\sum_{i=1}^{k}\log(A_i)\right)$$

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

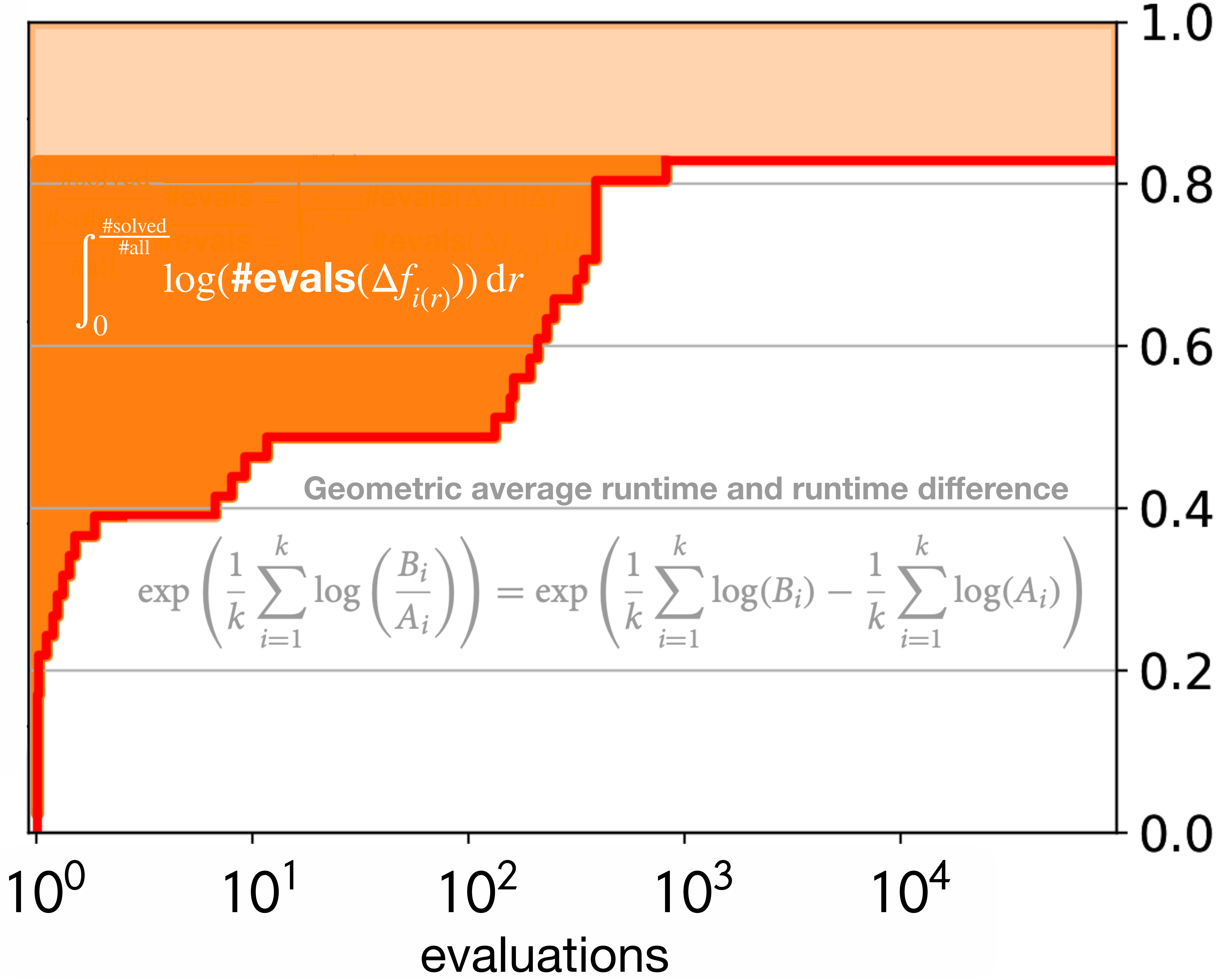
truncation…

The Area Above the Empirical Runtime Distribution

the area above the runtime distribution represents a (truncated) average runtime

Specifically, the logarithm of the truncated geometric average equals the bright orange area divided by the truncation fraction

We may choose a dynamic truncation budget for automated benchmarking



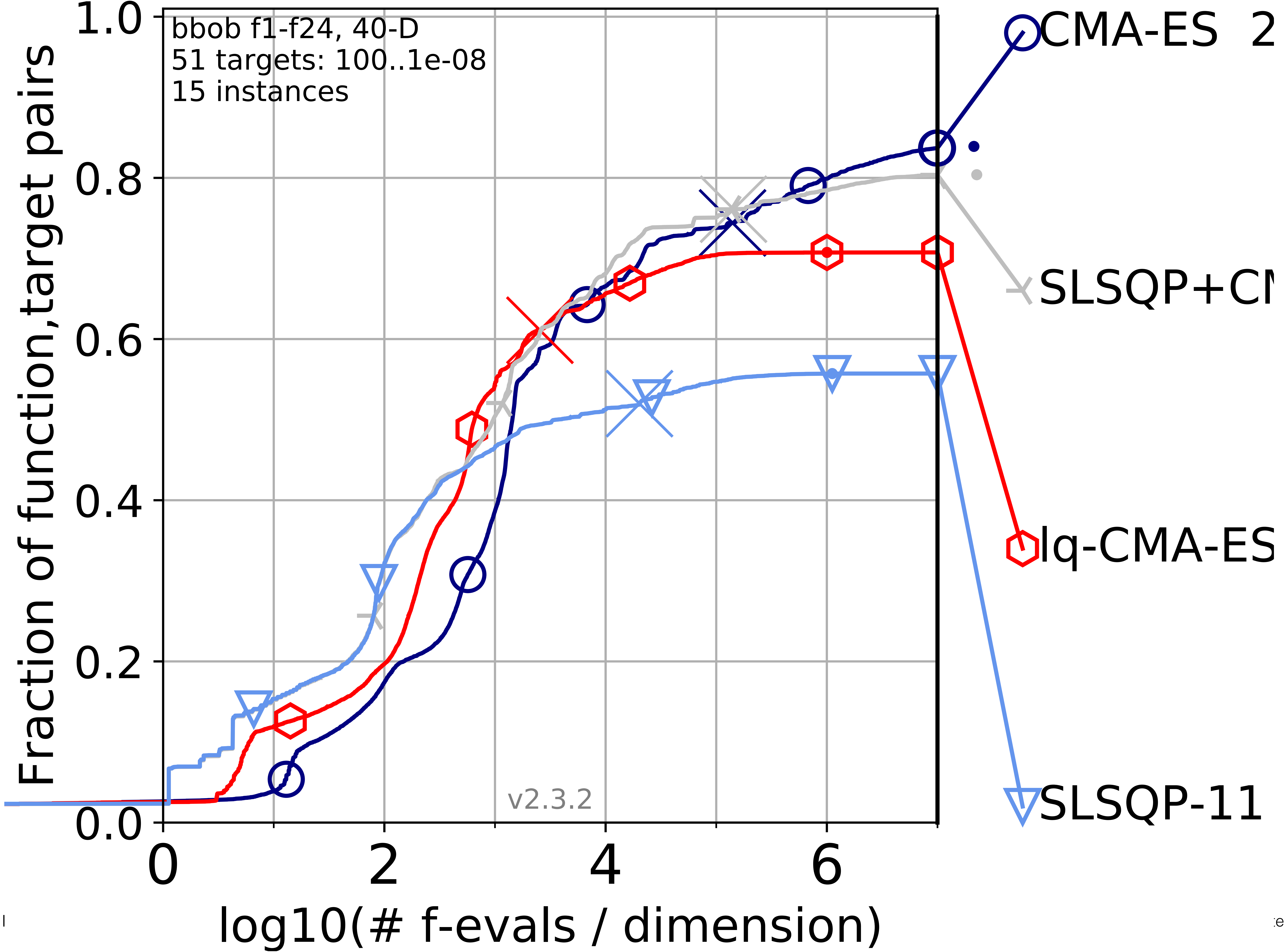
$$\int_0^{\frac{\#\text{solved}}{\#\text{all}}} \log(\textbf{\#evals}(\Delta f_{i(r)}))\,\mathrm{d}r$$

Geometric average runtime and runtime difference

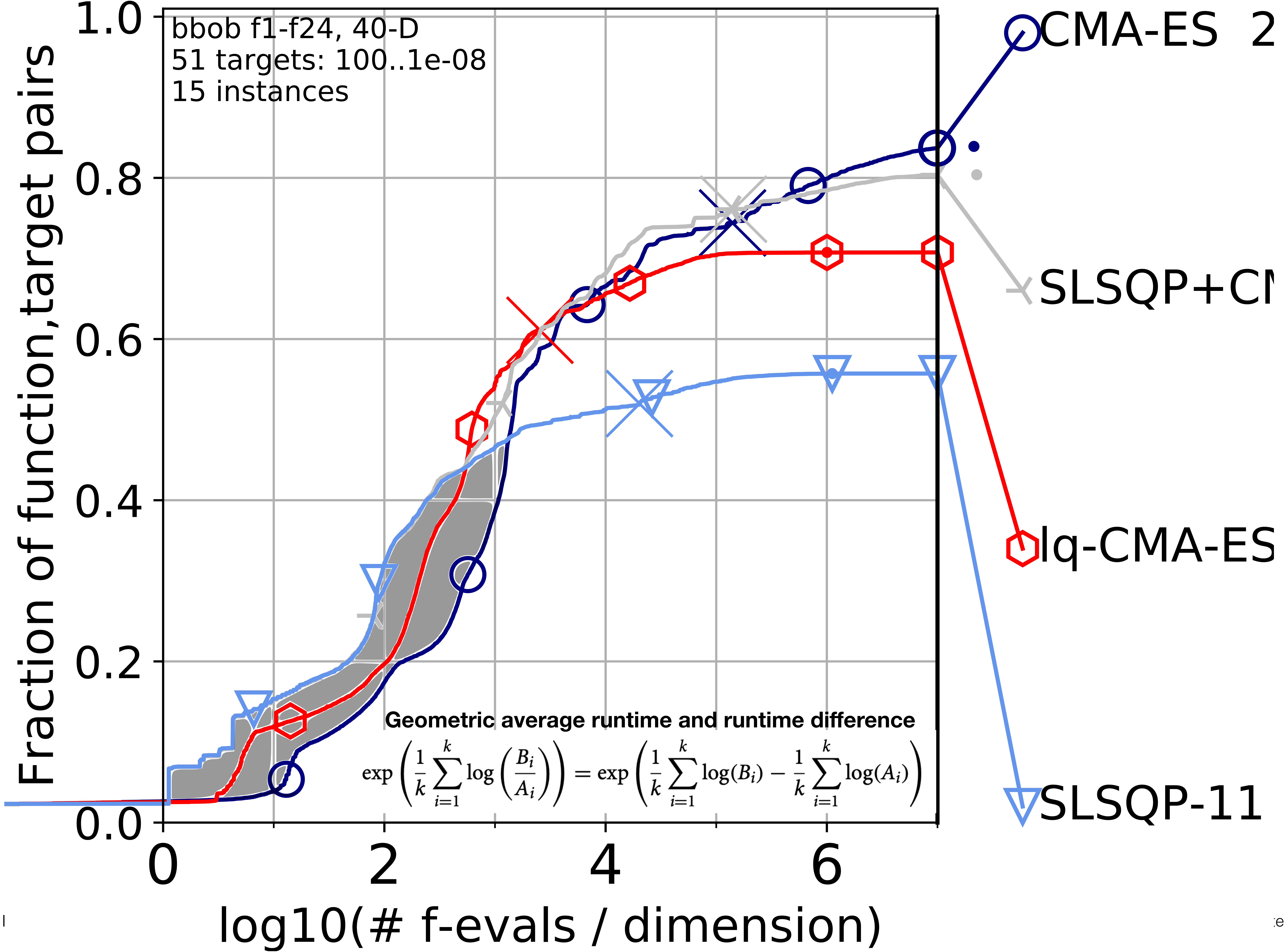
$$\exp\left(\frac{1}{k}\sum_{i=1}^{k}\log\left(\frac{B_i}{A_i}\right)\right) = \exp\left(\frac{1}{k}\sum_{i=1}^{k}\log(B_i) - \frac{1}{k}\sum_{i=1}^{k}\log(A_i)\right)$$

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

comparing graphs…

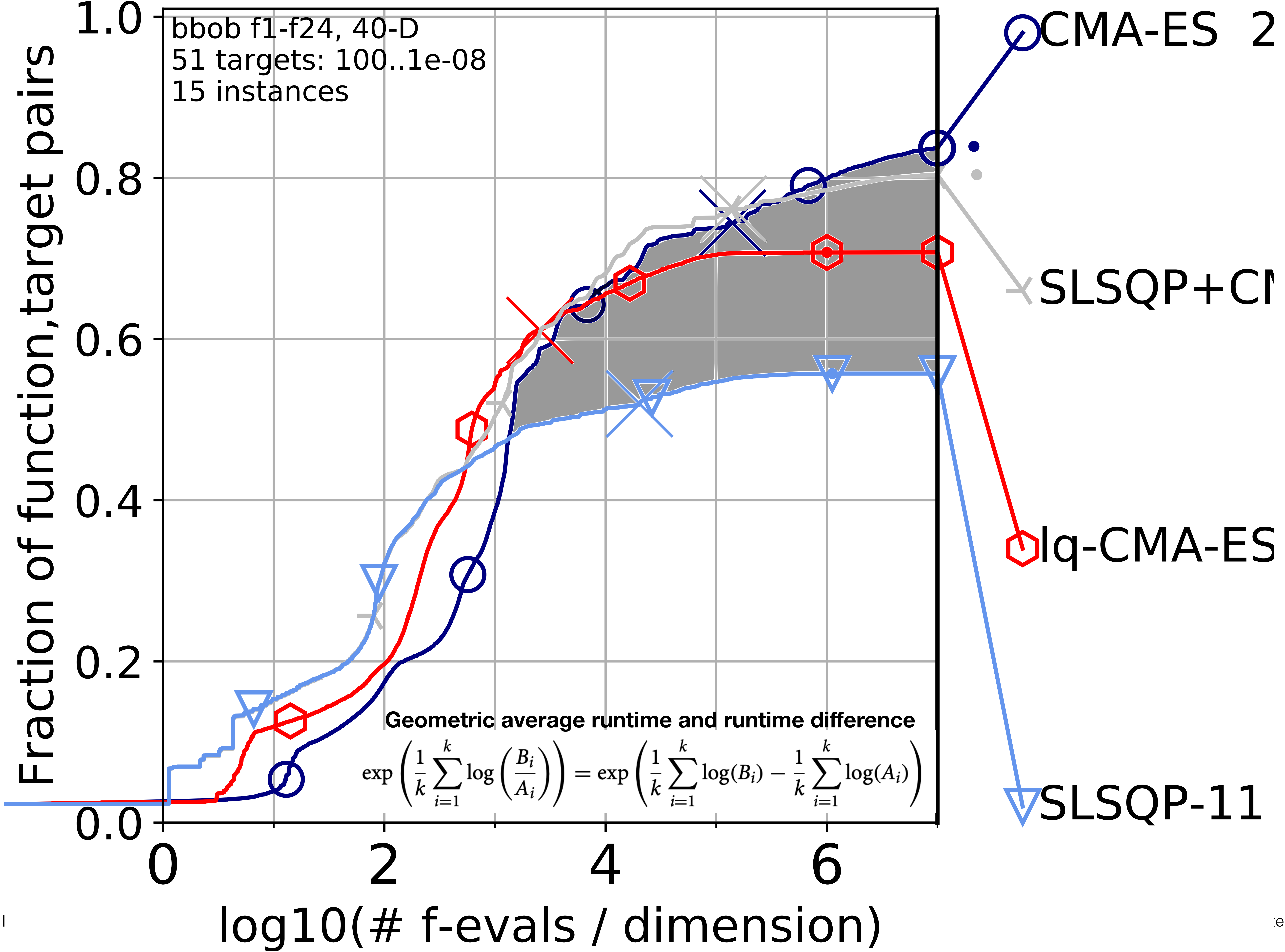


difference area…



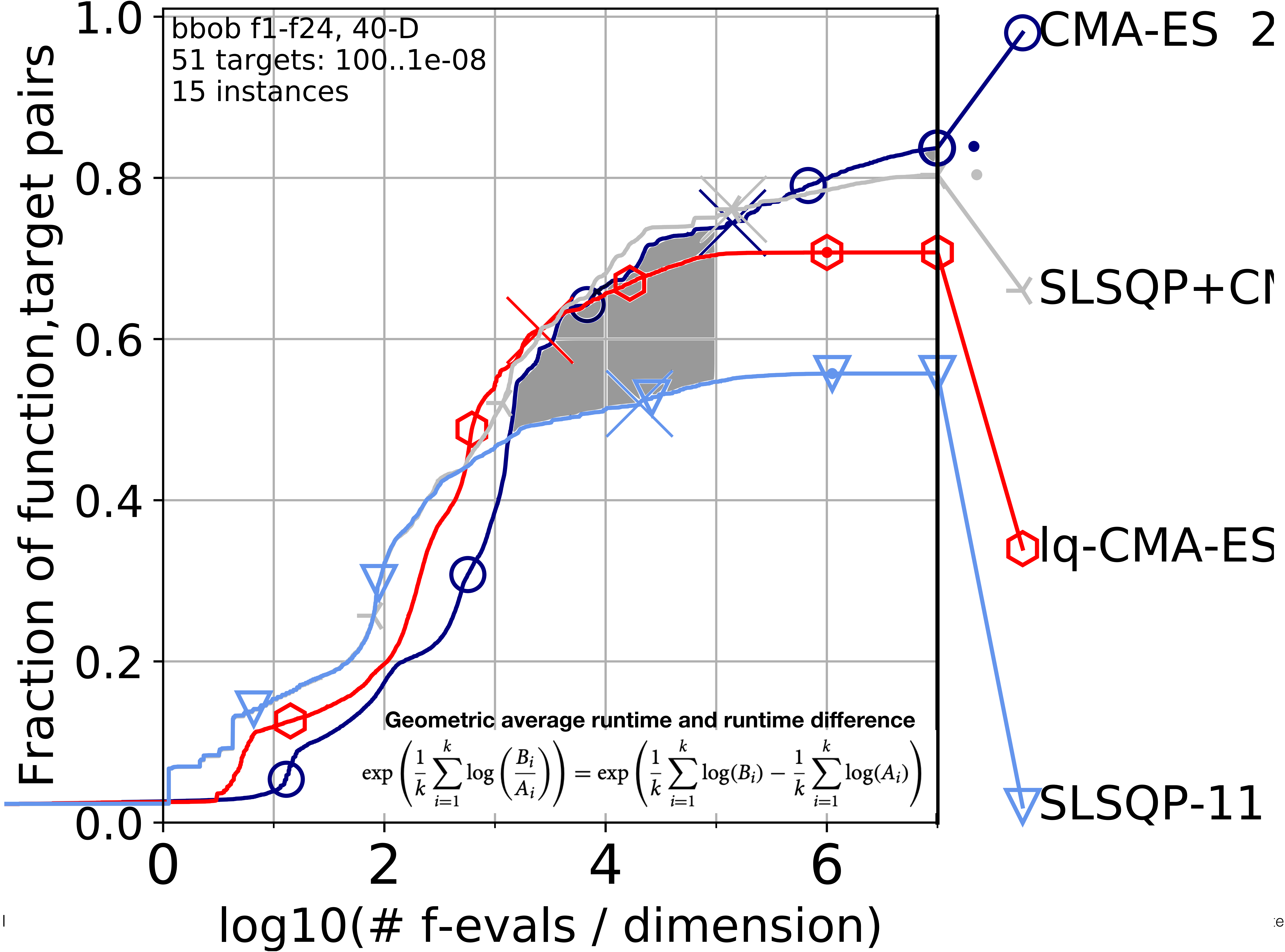
Geometric average runtime and runtime difference

$$\exp\left(\frac{1}{k}\sum_{i=1}^{k}\log\left(\frac{B_i}{A_i}\right)\right) = \exp\left(\frac{1}{k}\sum_{i=1}^{k}\log(B_i) - \frac{1}{k}\sum_{i=1}^{k}\log(A_i)\right)$$



Geometric average runtime and runtime difference

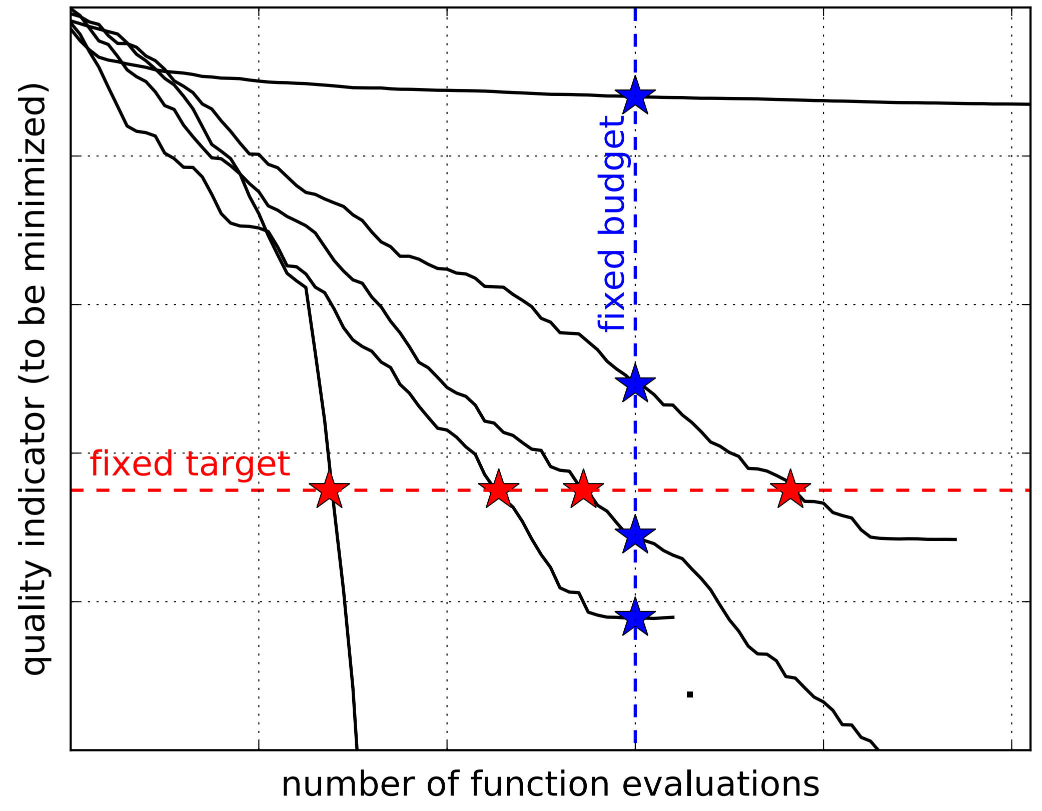
$$\exp\left(\frac{1}{k}\sum_{i=1}^{k}\log\left(\frac{B_i}{A_i}\right)\right) = \exp\left(\frac{1}{k}\sum_{i=1}^{k}\log(B_i) - \frac{1}{k}\sum_{i=1}^{k}\log(A_i)\right)$$



Geometric average runtime and runtime difference

$$\exp\left(\frac{1}{k}\sum_{i=1}^{k}\log\left(\frac{B_i}{A_i}\right)\right) = \exp\left(\frac{1}{k}\sum_{i=1}^{k}\log(B_i) - \frac{1}{k}\sum_{i=1}^{k}\log(A_i)\right)$$

COCO/BBOB: Fixed Target(s) versus Fixed Budget



- The fixed budget (vertical) design is (much) easier to set up

  choosing a budget is simpler than choosing a target and we need to chose a maximal “timeout” budget either way

- For the (very) same reason, results from the fixed target (horizontal) design are (much) simpler to interpret and more conclusive

  without specific insight, a function value is impossible to interpret beyond ordering

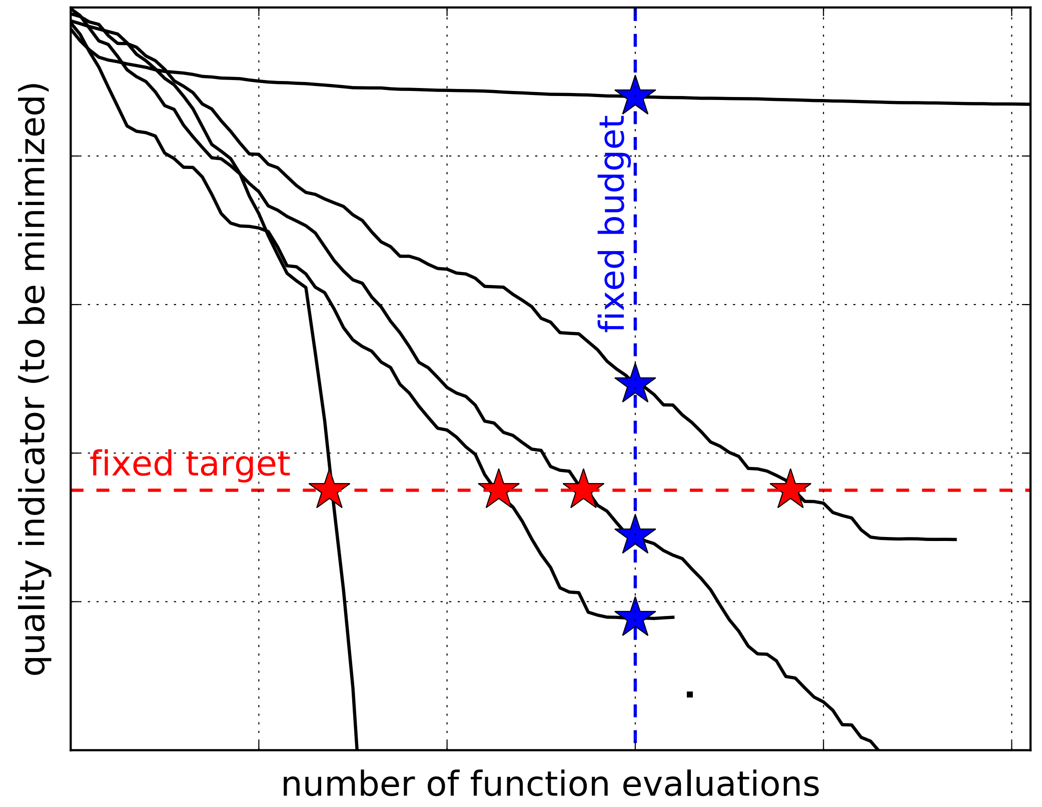
- Fixed target results
  - can’t miss a big fitness change (improvement)

    whereas with budget (vertical) discretization catching a fitness jump timely entirely relies on a fine discretization
  - are “budget-free” even for a single target

    we can compare results from runs with different maximal “timeout” budgets
  - can be meaningfully aggregated in ECDFs and geometric averages

    whereas function values from different functions are not commensurable
  - can give different weights to *different problems* to make the geometric average more representative, as in $\sum_i w_i \log(\mathrm{RT}_i)$

COCO/BBOB: Fixed Target(s) versus Fixed Budget



- The fixed budget (vertical) design is (much) easier to set up

  choosing a budget is simpler than choosing a target and we need to chose a maximal "timeout" budget either way

- For the (very) same reason, results from the fixed target (horizontal) design are (much) simpler to interpret and more conclusive

  without specific insight, a function value is impossible to interpret beyond ordering

- Fixed target results
  - can't miss a big fitness change (improvement)

    whereas with budget (vertical) discretization catching a fitness jump timely entirely relies on a fine discretization

  - are "budget-free" even for a single target

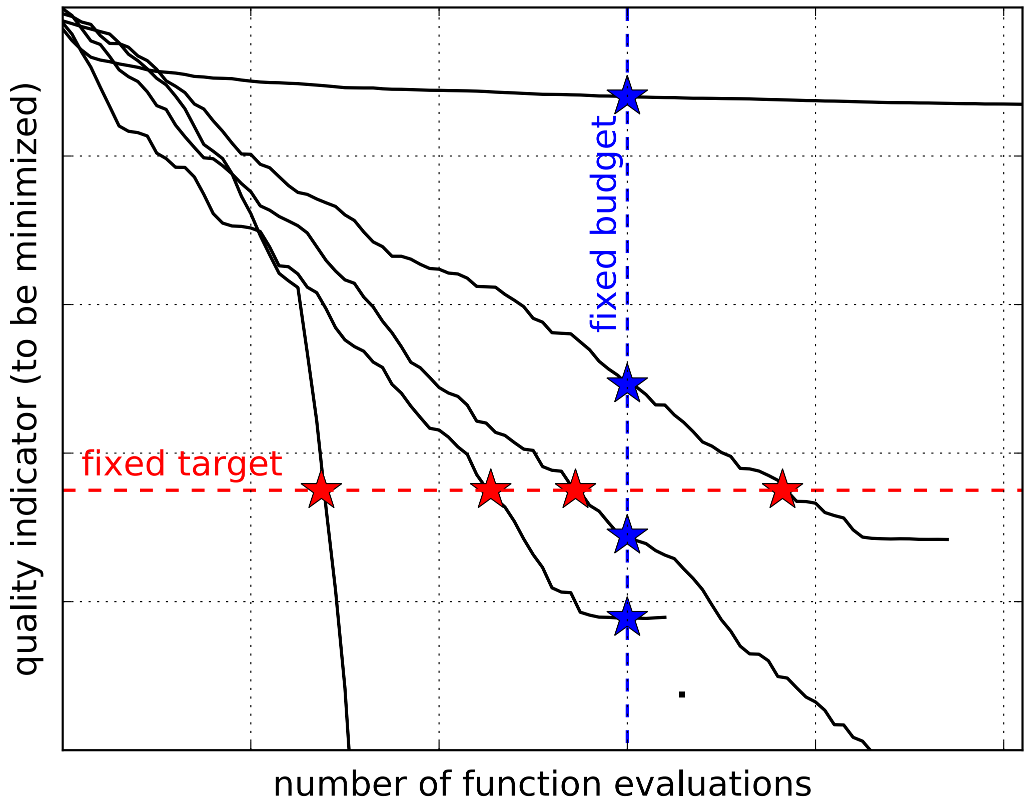
    we can compare results from runs with different maximal "timeout" budgets

  - can be meaningfully aggregated in ECDFs and geometric averages

    whereas function values from different functions are not commensurable

  - can give different weights to *different problems* to make the geometric average more representative, as in $\sum_i w_i \log(\mathrm{RT}_i)$

COCO/BBOB: Fixed Target(s) versus Fixed Budget



- The fixed budget (vertical) design is (much) easier to set up

  choosing a budget is simpler than choosing a target and we need to chose a maximal "timeout" budget either way

- For the (very) same reason, results from the fixed target (horizontal) design are (much) simpler to interpret and more conclusive

  without specific insight, a function value is impossible to interpret beyond ordering

- Fixed target results
  - can't miss a big fitness change (improvement)

    whereas with budget (vertical) discretization catching a fitness jump timely entirely relies on a fine discretization

  - are "budget-free" even for a single target

    we can compare results from runs with different maximal "timeout" budgets

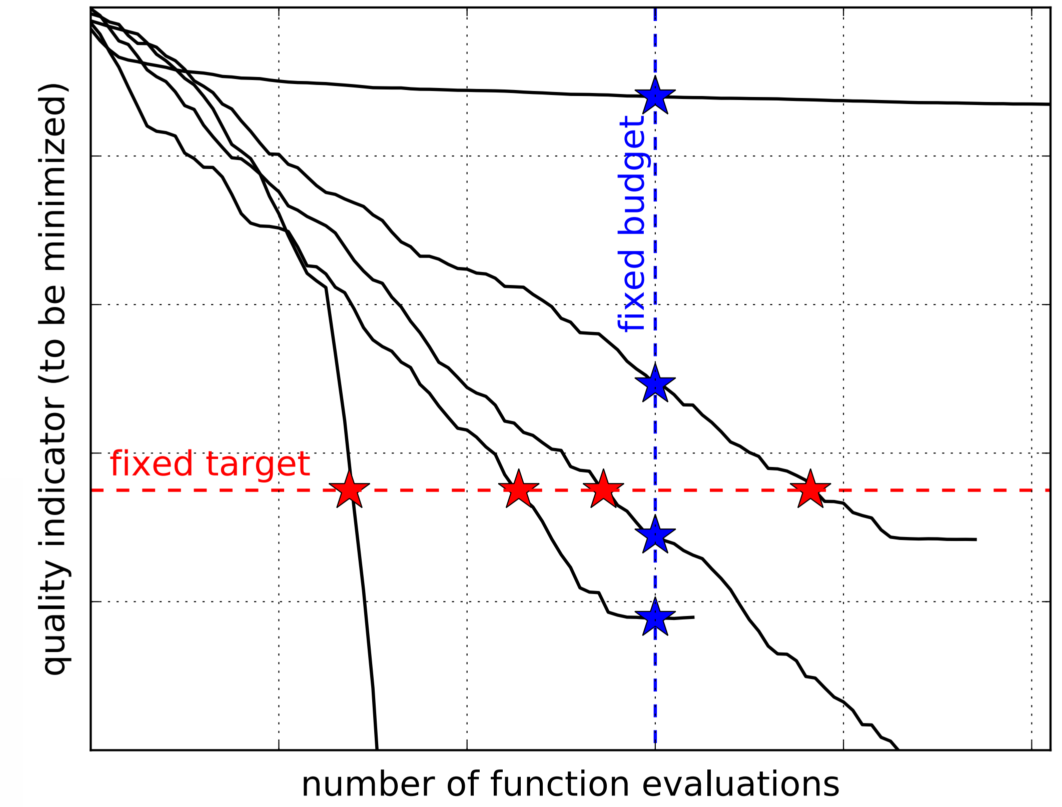
  - can be meaningfully aggregated in ECDFs and geometric averages

    whereas function values from different functions are not commensurable

  - can give different weights to *different problems* to make the geometric average more representative, as in $\sum_i w_i \log(\mathrm{RT}_i)$

automated…

Automated BBOB: Fixed Target(s) versus Fixed Budget



- The fixed budget (vertical) design is (much) easier to set up

  choosing a budget is simpler than choosing a target and we need to chose a maximal “timeout” budget either way

- For the (very) same reason, results from the fixed target (horizontal) design are (much) simpler to interpret and more conclusive

  without specific insight, a function value is impossible to interpret beyond ordering

- Fixed target results
  - can’t miss a big fitness change (improvement)

    whereas with budget (vertical) discretization catching a fitness jump timely entirely relies on a fine discretization

  - are “budget-free” even for a single target

    we can compare results from runs with different maximal “timeout” budgets

  - can be meaningfully aggregated in ECDFs and geometric averages

    whereas function values from different functions are not commensurable

  - can give different weights to *different problems* to make the geometric average more representative, as in $\sum_i w_i \log(\mathrm{RT}_i)$

simulated runtimes…

Converting Positive Success Probabilities into Runtimes

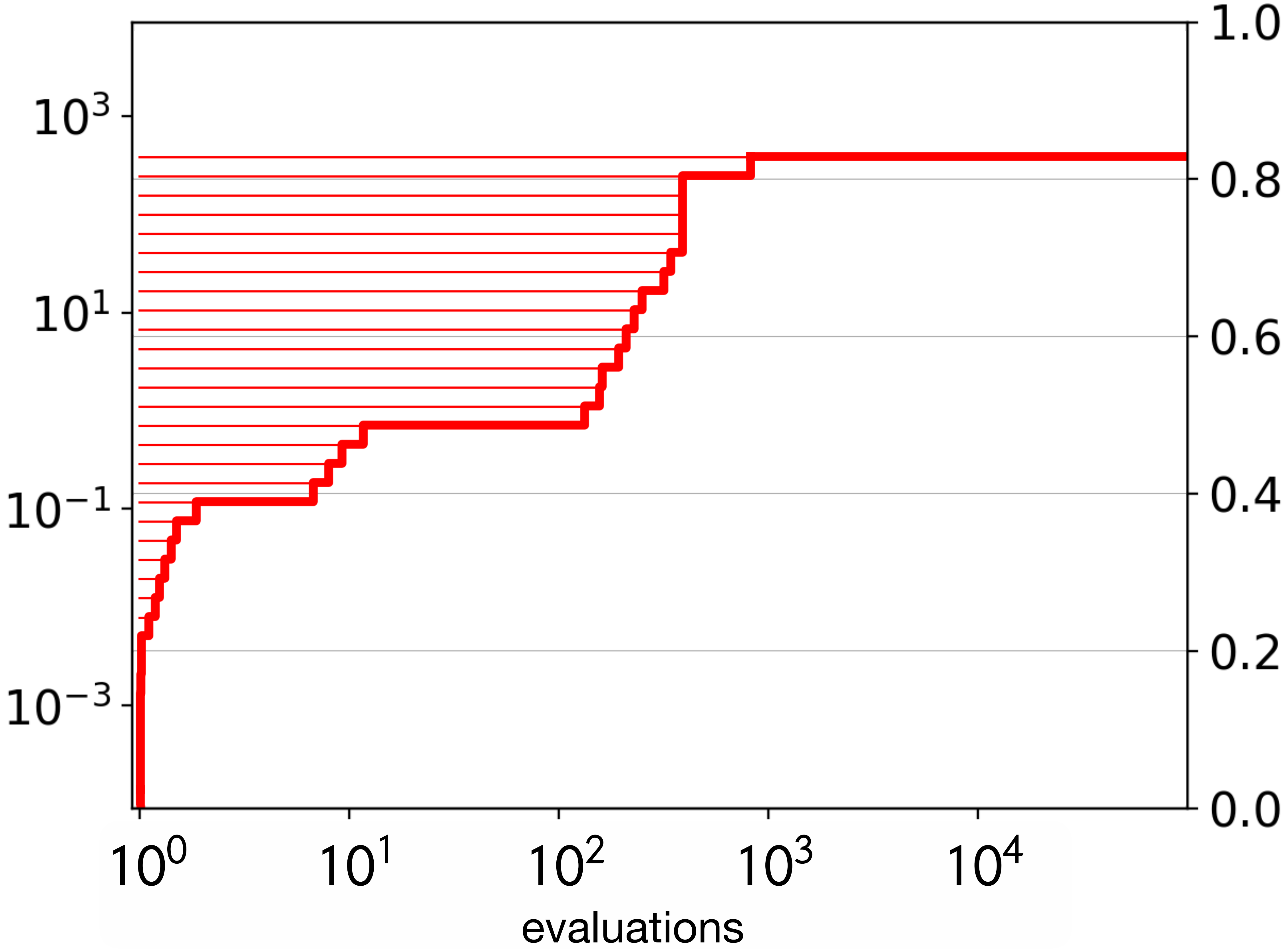
Solving the fast-versus-successful comparison dilemma

We construct runtimes with simulated restarts

$$\mathsf{RT}^{\mathsf{sim}} = \mathsf{RT}_j^{\mathsf{succ}} + \sum_{i=1}^{N} \mathsf{RT}_i^{\mathsf{fail}} \approx (1+N) \times \mathsf{RT}^{\mathsf{average}}$$

runtimes are drawn uniformly at random until the first success,
$N \geq 0$ is hence a random variable

Th



all runs $= \dfrac{\sum_i \mathsf{RT}_i}{\#\ \text{successes}}$

If l … runs gives an estimate of a

ru

Converting Positive Success Probabilities into Runtimes

Solving the fast-versus-successful comparison dilemma

We construct runtimes with simulated restarts

$$\mathrm{RT}^{\mathrm{sim}} = \mathrm{RT}_j^{\mathrm{succ}} + \sum_{i=1}^{N} \mathrm{RT}_i^{\mathrm{fail}} \approx (1+N) \times \mathrm{RT}^{\mathrm{average}}$$

runtimes are drawn uniformly at random until the first success,
$N \geq 0$ is hence a random variable

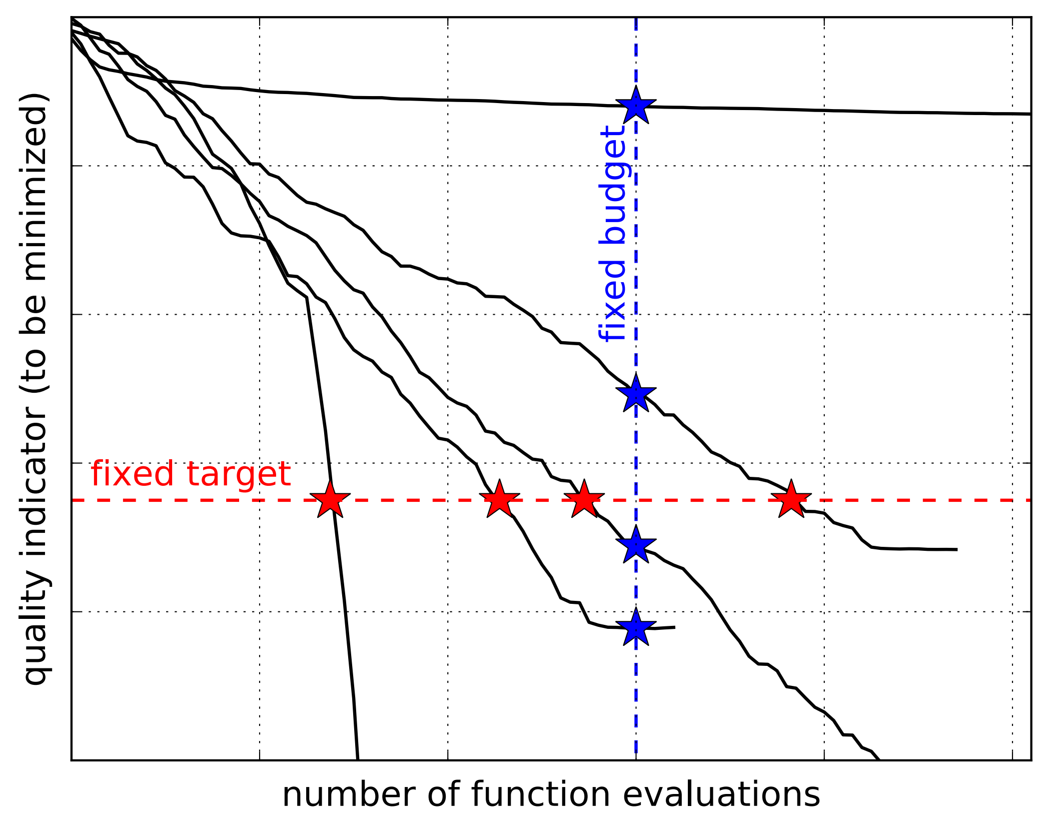
Their expected runtime is

$$\mathrm{ERT} = \frac{\text{average evaluations to target of all runs}}{p_{\mathrm{success}}} = \frac{\sum_i \mathrm{RT}_i}{\#\ \text{successes}}$$

If $p_{\mathrm{success}} = 0$, the inequality $p_{\mathrm{success}} \leq 1/N_{\mathrm{runs}}$ gives an estimate of a runtime lower bound

Thank you

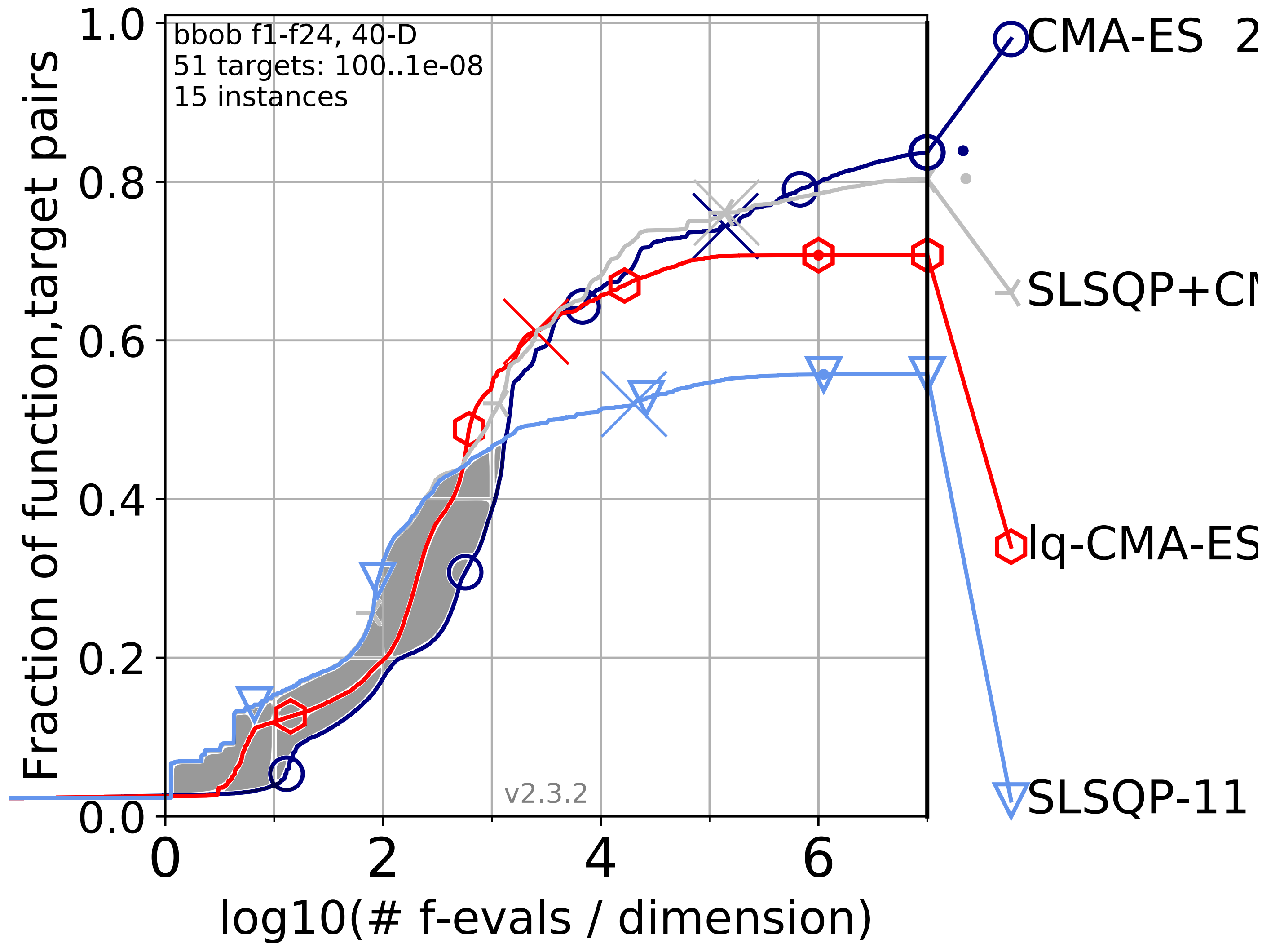
We construct runtimes with simulated restarts

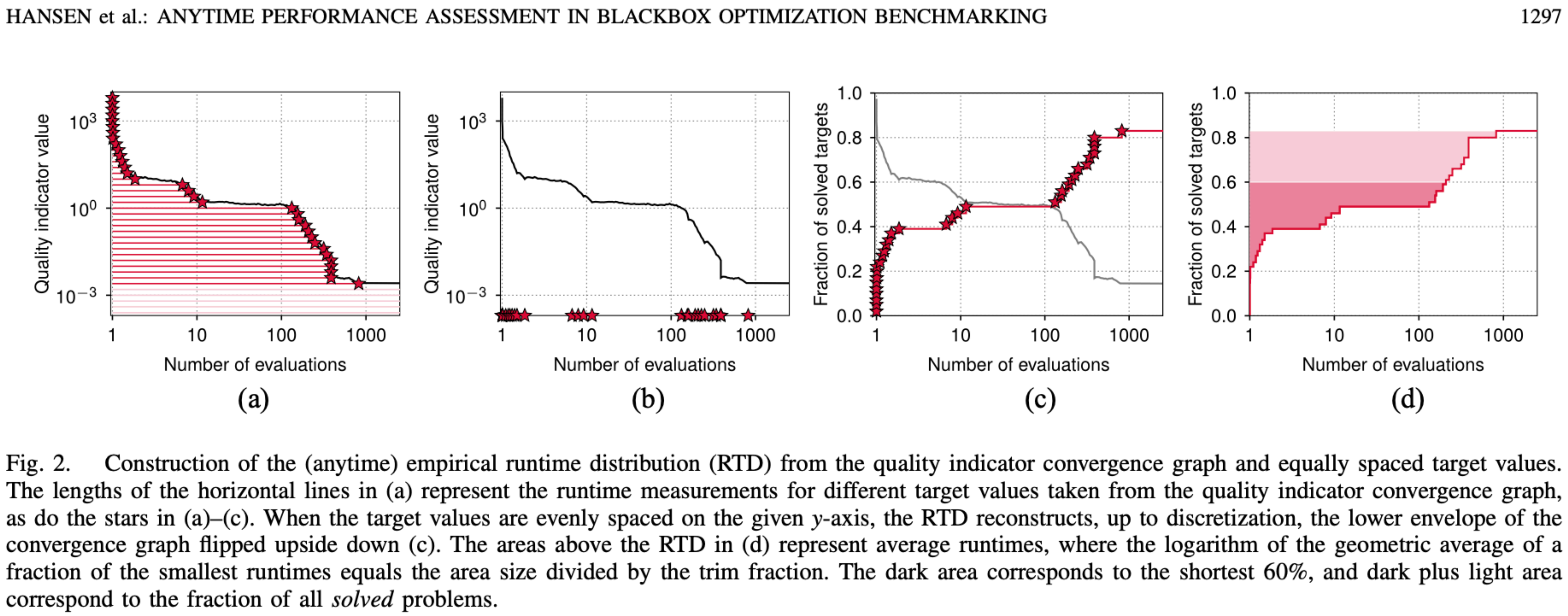


$$\mathrm{RT}^{\mathrm{sim}} = \mathrm{RT}_j^{\mathrm{succ}} + \sum_{i=1}^{N} \mathrm{RT}_i^{\mathrm{fail}} \approx (1+N)\times \mathrm{RT}^{\mathrm{average}}$$

runtimes are drawn uniformly at random until the first success,
$N \geq 0$ is hence a random variable

Time for discussion





HANSEN et al.: ANYTIME PERFORMANCE ASSESSMENT IN BLACKBOX OPTIMIZATION BENCHMARKING 1297

Quality indicator value
Fraction of solved targets
Number of evaluations
(a)
(b)
(c)
(d)

Fig. 2. Construction of the (anytime) empirical runtime distribution (RTD) from the quality indicator convergence graph and equally spaced target values. The lengths of the horizontal lines in (a) represent the runtime measurements for different target values taken from the quality indicator convergence graph, as do the stars in (a)–(c). When the target values are evenly spaced on the given *y*-axis, the RTD reconstructs, up to discretization, the lower envelope of the convergence graph flipped upside down (c). The areas above the RTD in (d) represent average runtimes, where the logarithm of the geometric average of a fraction of the smallest runtimes equals the area size divided by the trim fraction. The dark area corresponds to the shortest 60%, and dark plus light area correspond to the fraction of all *solved* problems.