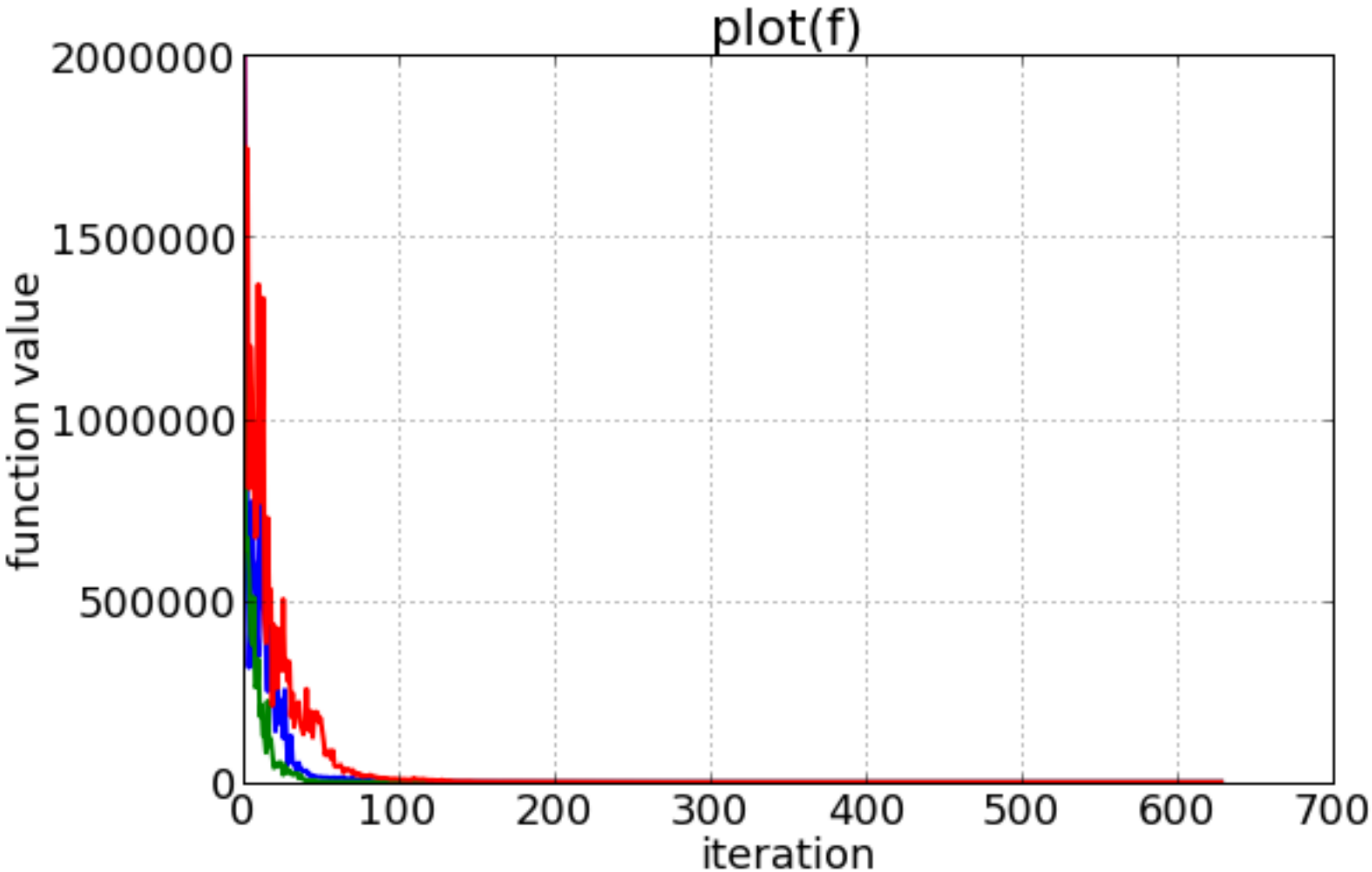
Performance Assessment in Optimization

Anne Auger, CMAP & Inria

credits to N. Hansen for some slides

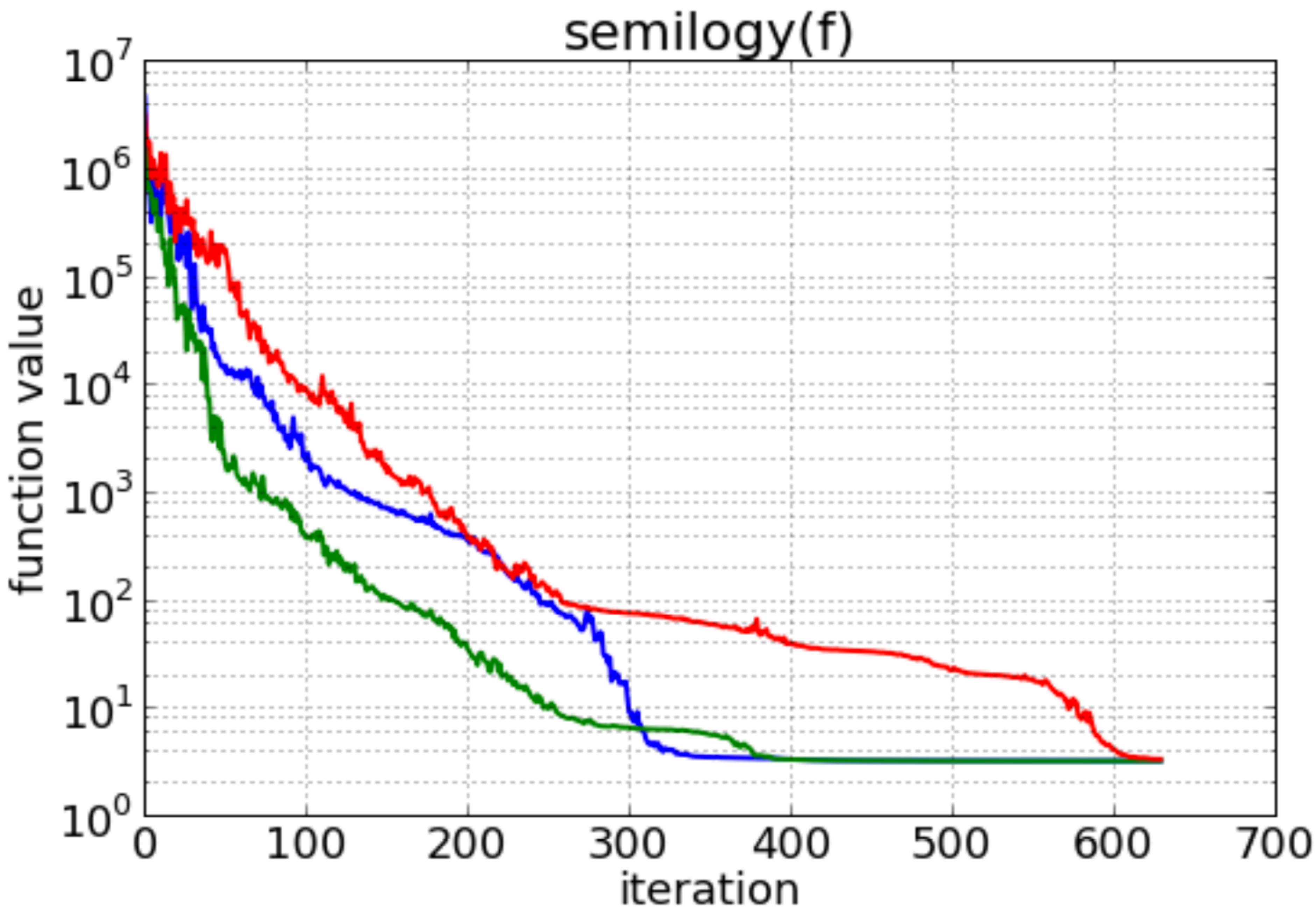
Visualization and presentation of single runs

Displaying 3 runs (three trials)



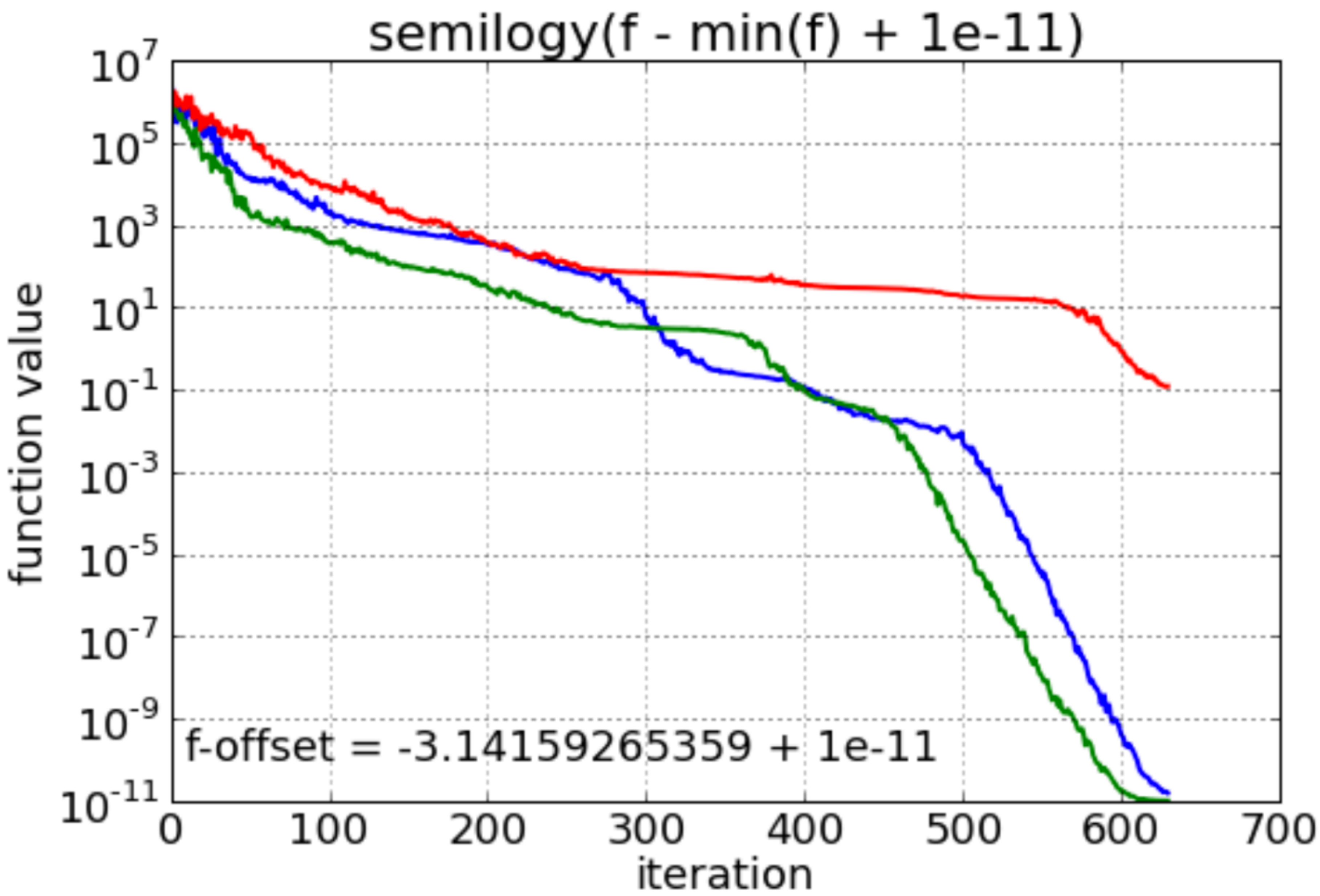
not like this (it's unfortunately a common picture)

Displaying 3 runs (three trials)



better like this (shown are the same data), caveat: fails with negative f-values

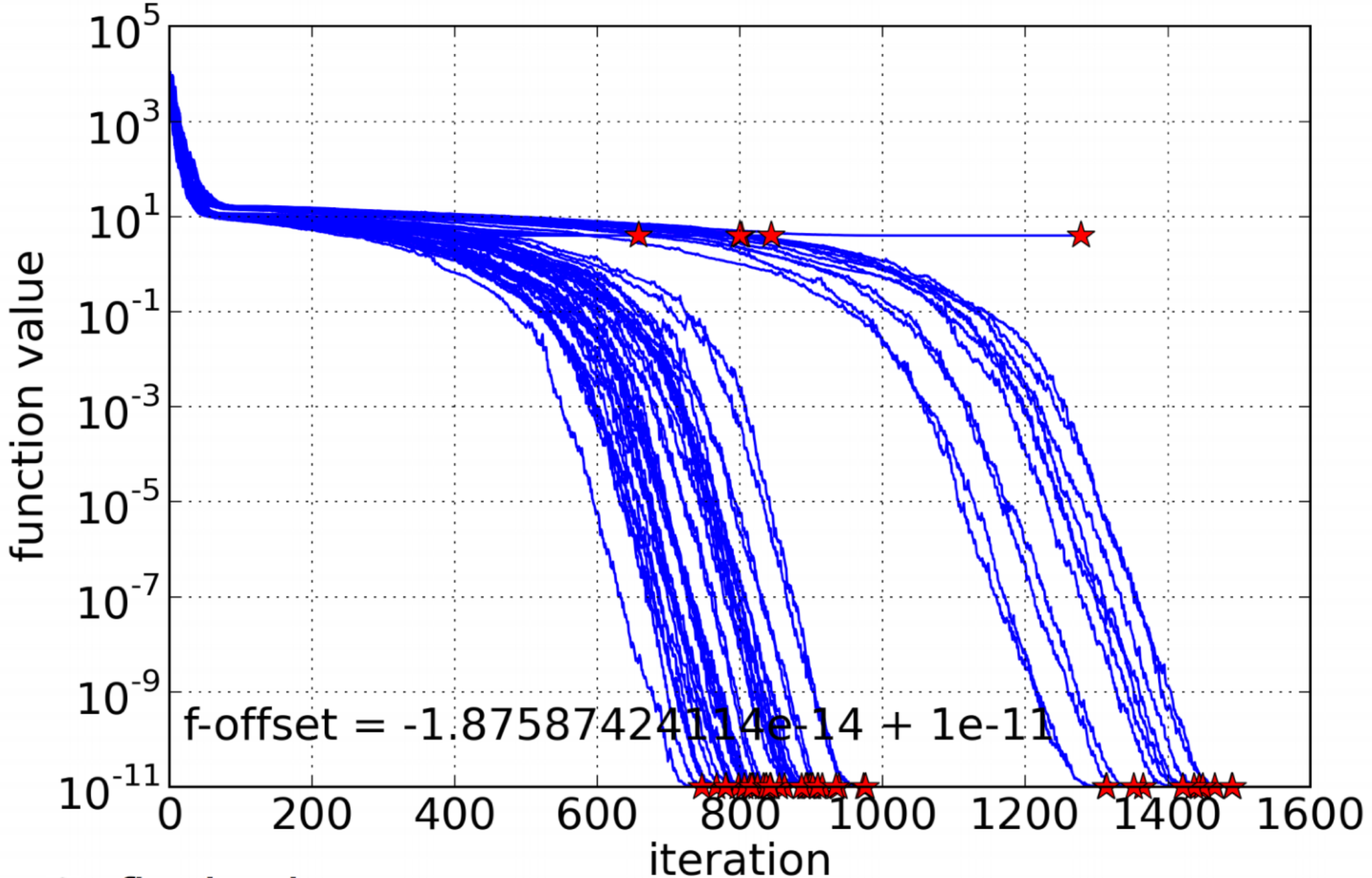
Displaying 3 runs (three trials)



even better like this: subtract minimum value over all runs

Displaying 51 runs

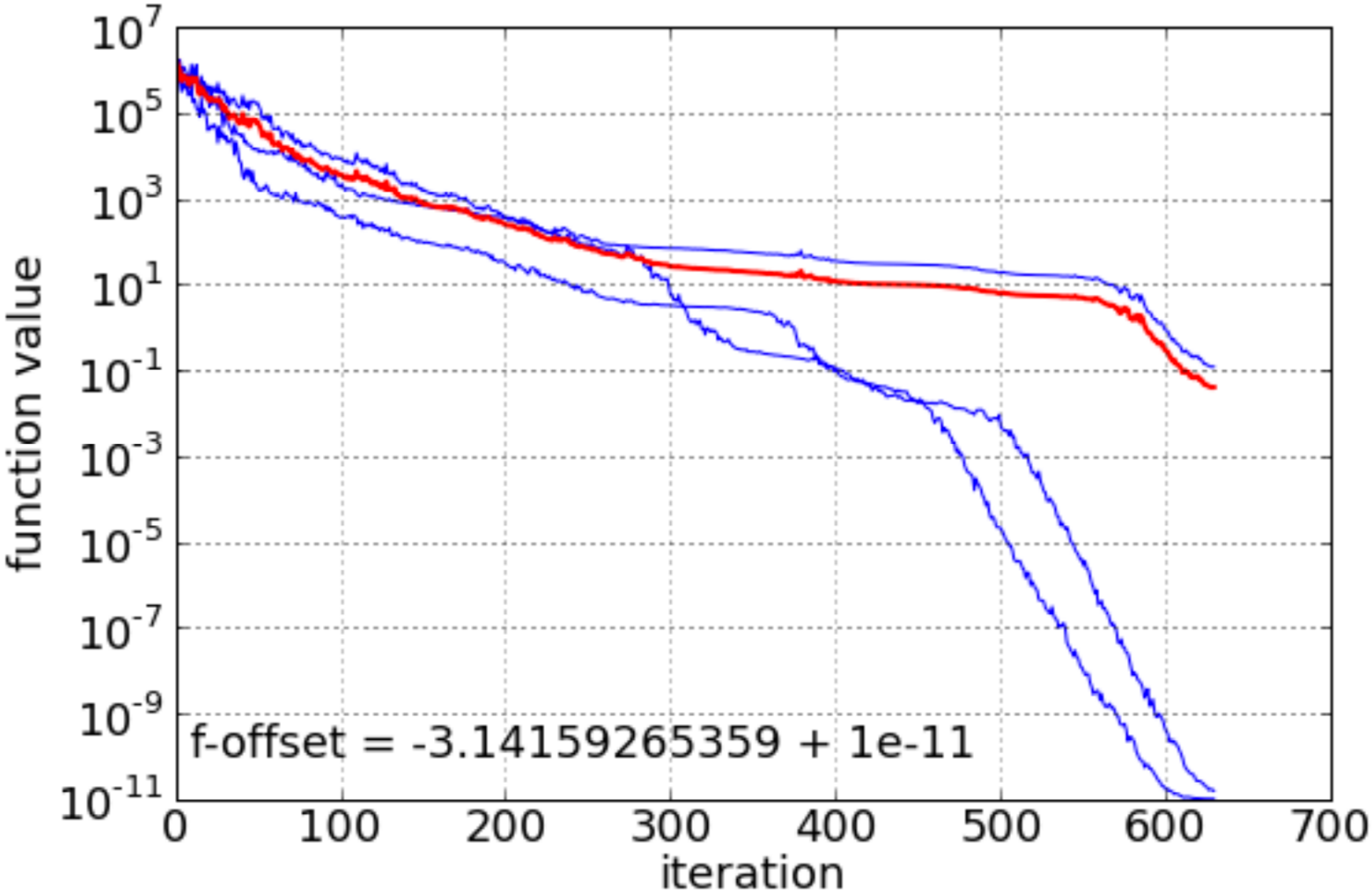
don't hesitate to display all data (the appendix is your friend)



★ : final value

observation: three different "modes", which would be difficult to represent or recover in single statistics

Which Statistics?



mean/average function value

- tends to emphasize large values

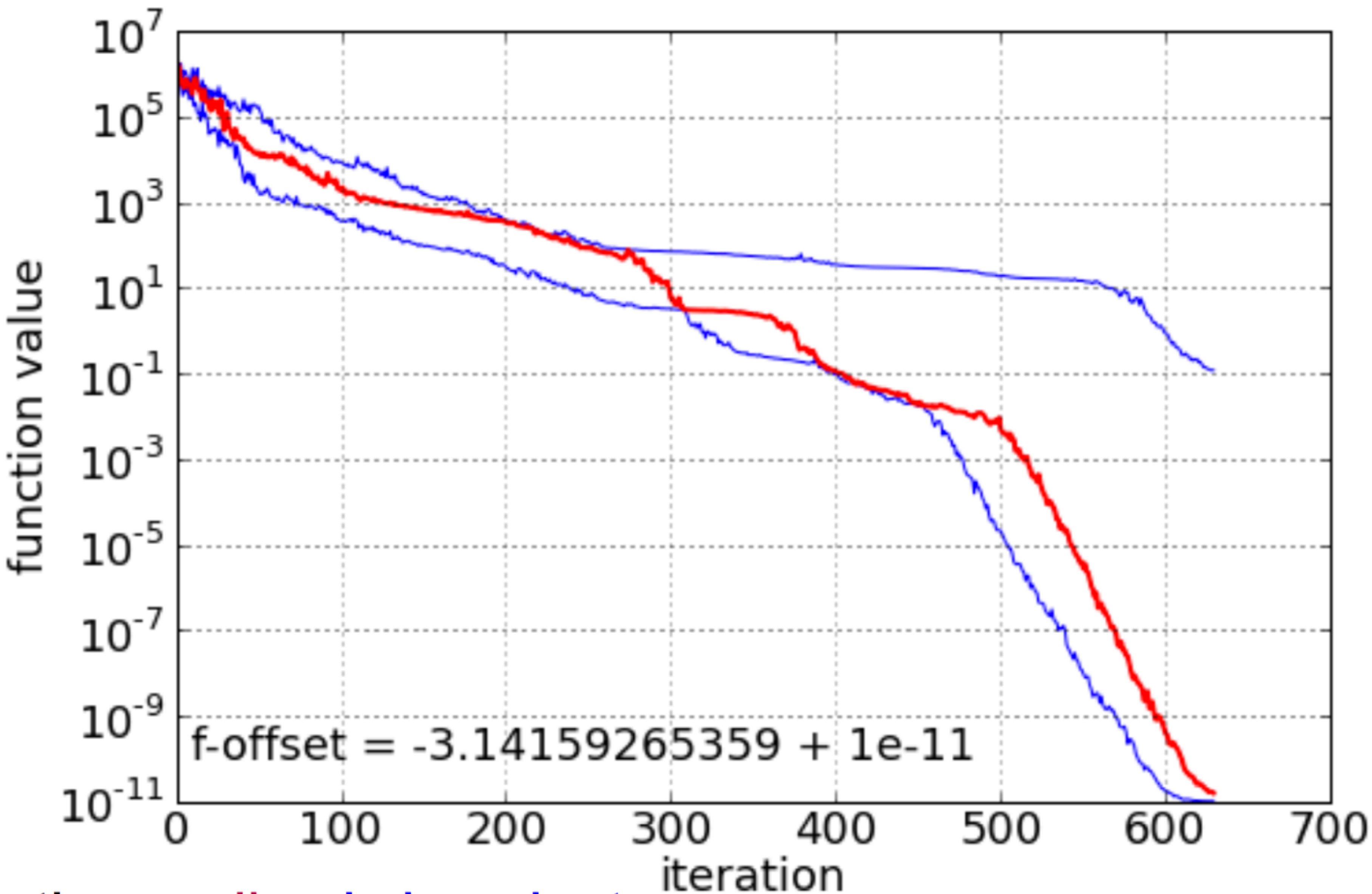
More problems with average / expectations

- to reliably estimate an expectation (from the *average*) we need to make *assumptions on the tail* of the underlying distribution
  - these can not be implied from the observed data
  - AKA: the average is well-known to be (highly) sensitive to outliers (extreme events)

- rare events can only be analyzed by collecting a *large enough number* of data

from Hansen GECCO 2019 Experimentation tutorial

Which Statistics?



the median is invariant

- unique for uneven number of data
- independent of log-scale, offset...

  median(log(data))=log(median(data))
- same when taken over x- or y-direction

Implications

- use the median as summary datum

  unless there are good reasons for a different statistics
  out of practicality: use an odd number of repetitions

- more general: use quantiles as summary data

  for example out of 15 data: 2nd, 8th, and 14th
  value represent the 10%, 50%, and 90%-tile

from Hansen GECCO 2019 Experimentation tutorial

Benchmarking Black-Box Optimizers

Benchmarking: running an algorithm on several test functions

in order to evaluate the performance of the algorithm

Why Numerical Benchmarking?

Evaluate the performance of optimization algorithms

Compare the performance of different algorithms

understand strength and weaknesses of algorithms

help in design of new algorithms

On performance measures ...

Performance measure - What to measure?

CPU time (to reach a given target)

drawbacks: depend on the implementation, on the language, on the machine

time is spent on code optimization instead of science

Testing heuristics, we have it all wrong, J.N. Hooker, 1995 Journal of Heuristics

Prefer "absolute" value: # of function evaluations to reach a given target

assumptions: internal cost of the algorithm negligible or measured independently

On performance measures - Requirements

"Algorithm A is 10/100 times faster than
Algorithm B to solve this type of problems"

On performance measures - Requirements

"Algorithm A is 10/100 times faster than Algorithm B to solve this type of problems"

quantitative measures

As opposed to

F.	EFWA *vs* EFWA-NG		
	EFWA	EFWA-NG	p-value
f_1	-1.3999E+03	-1.3999E+03	**2.316E-03**
f_2	6.8926E+05	6.5258E+05	4.256E-01
f_3	7.7586E+07	6.4974E+07	8.956E-01
f_4	-1.0989E+03	-1.0989E+03	7.858E-01
f_5	-9.9992E+02	-9.9992E+02	**4.290E-02**
f_6	-8.5073E+02	-8.4462E+02	1.654E-01
[illegible]	[illegible]	[illegible]	[illegible]

displayed: mean f-value after 3.10^5 f-evals (51 runs)

bold: statistically significant

concluded: "EFWA significantly better than EFWA-NG"

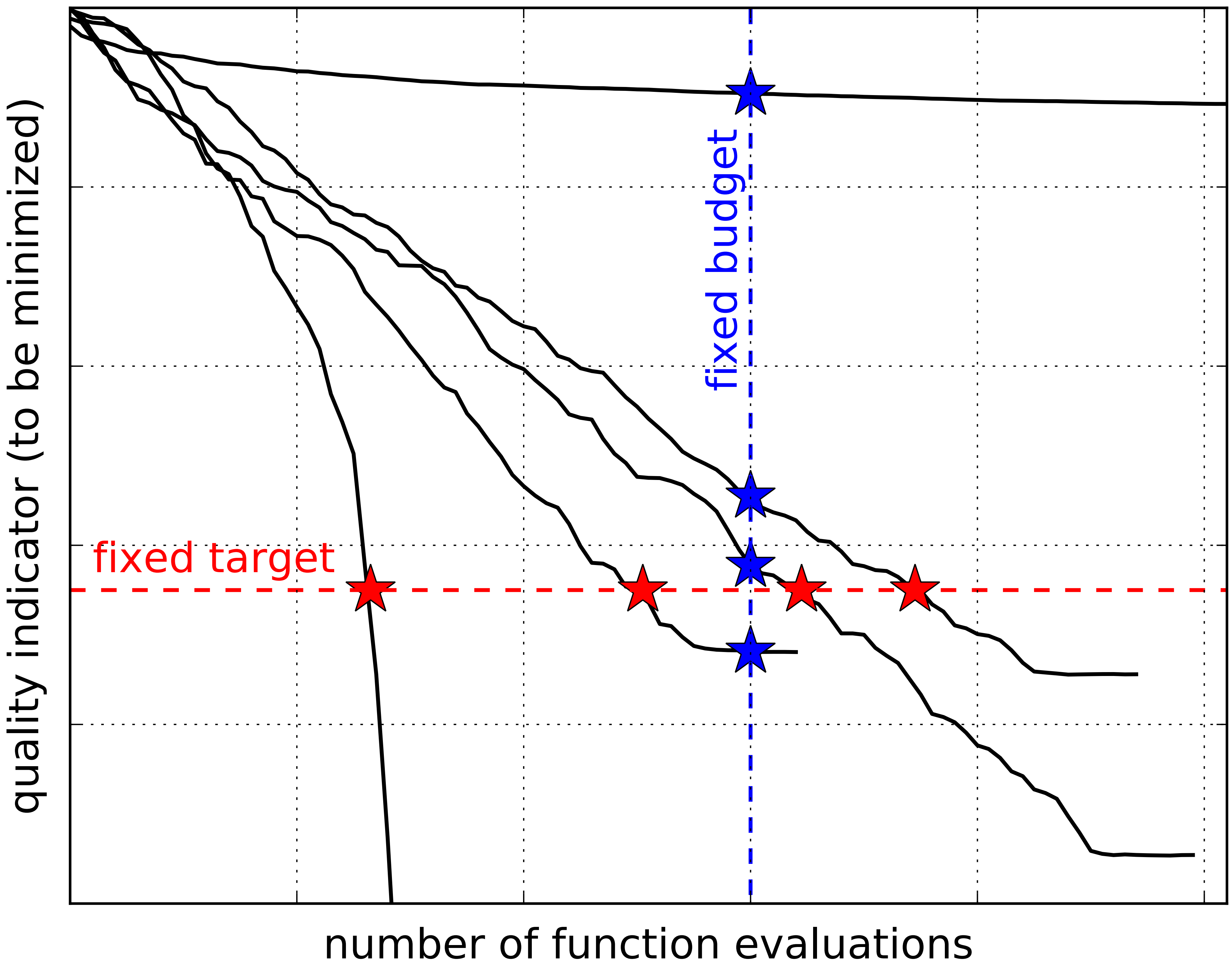
Source: Dynamic search in fireworks algorithm, Shaoqiu Zheng, Andreas Janecek, Junzhi Li and Ying Tan CEC 2014

On performance measures - Requirements

a performance measure should be

- quantitative, with a ratio scale
- well-interpretable with a meaning
- relevant in the “real world”
- simple

Fixed Cost versus Fixed Budget - Collecting Data



Fixed Cost versus Fixed Budget - Collecting Data

Collect for a given target (several target), the number of function evaluations needed to reach a target

Repeat several times:

- if algorithms are stochastic, never draw a conclusion from a single run
- if deterministic algorithm, repeat by changing (randomly) the initial conditions

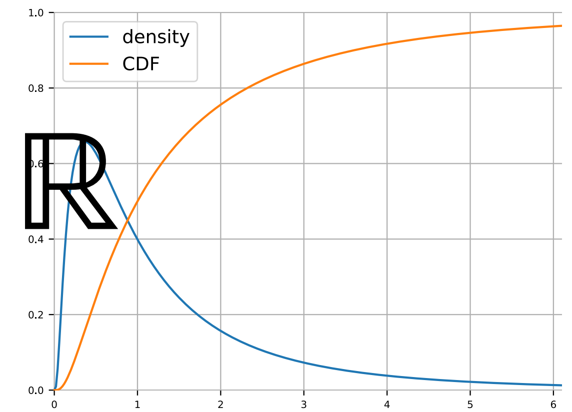
ECDF:

Empirical Cumulative Distribution Function of the Runtime

Cumulative Distribution Function (CDF)

Given a random variable T, the cumulative distribution function (CDF) is defined as

$$\mathrm{CDF}_T(t) = \Pr(T \leq t) \text{ for all } t \in \mathbb{R}$$



It characterizes the probability distribution of T

If two random variables have the same CDF, they have the same probability distribution

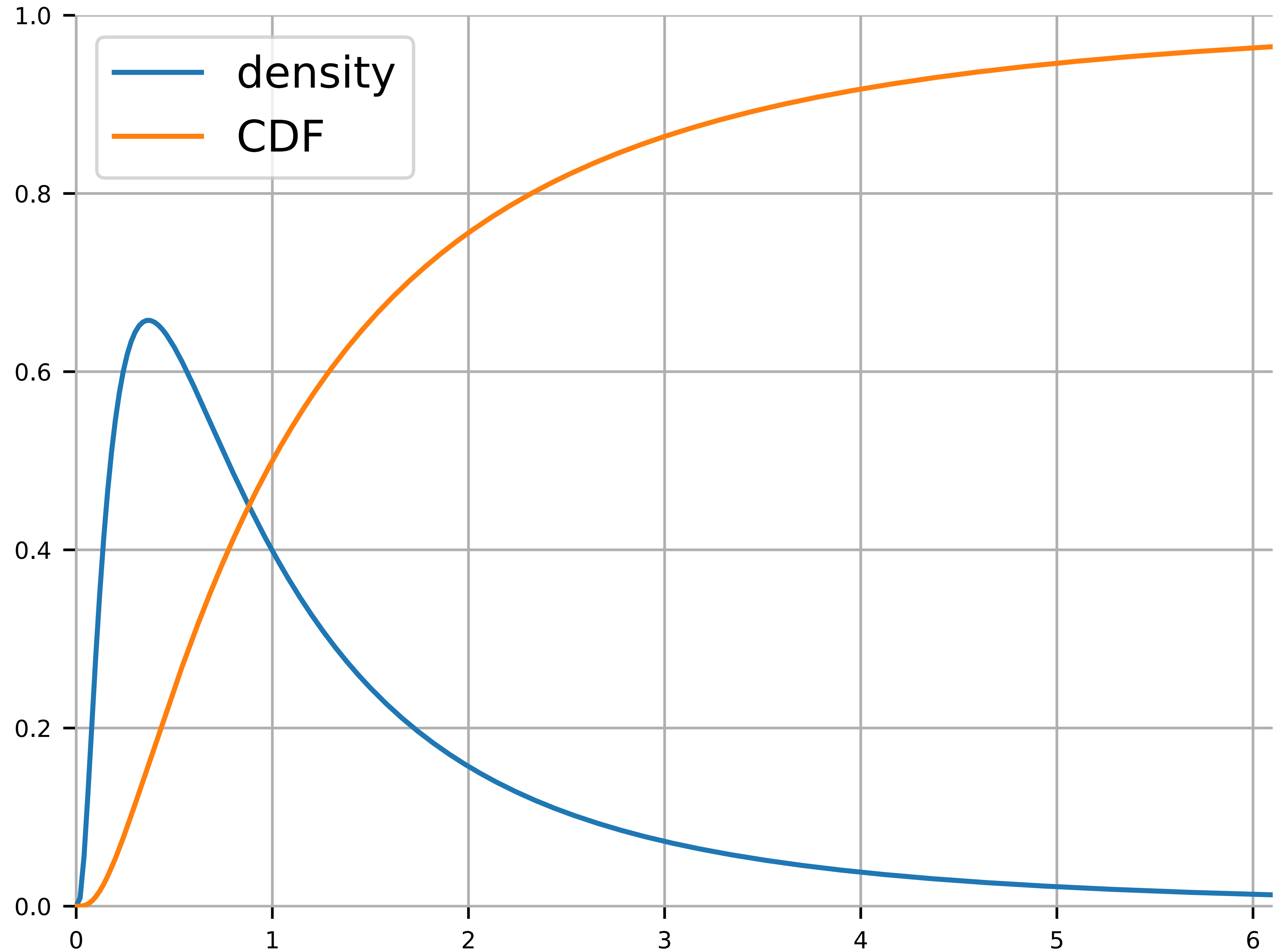
Cumulative Distribution Function (CDF)

dom variable T, the cumulat
DF) is defined as

$$F_T(t) = \Pr(T \leq t) \text{ fo}$$

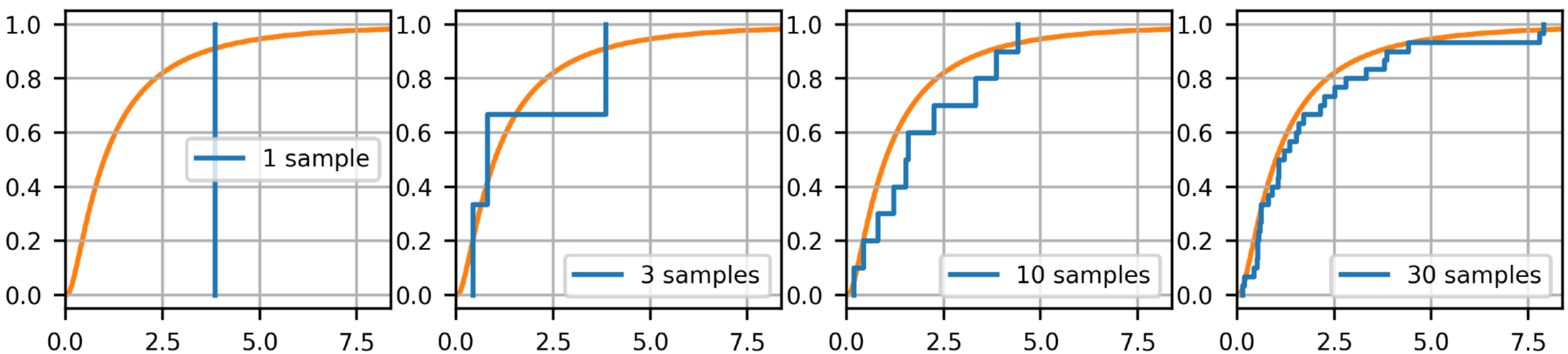
zes the probability distributio

om variables have the same (
same pro



Empirical Cumulative Distribution Function

- Given a collection of data $T_1, T_2, \ldots, T_k$ (e.g. an empirical sample of a random variable) the *empirical* cumulative distribution function (ECDF) is a step function that jumps by $1/k$ at each value in the data.



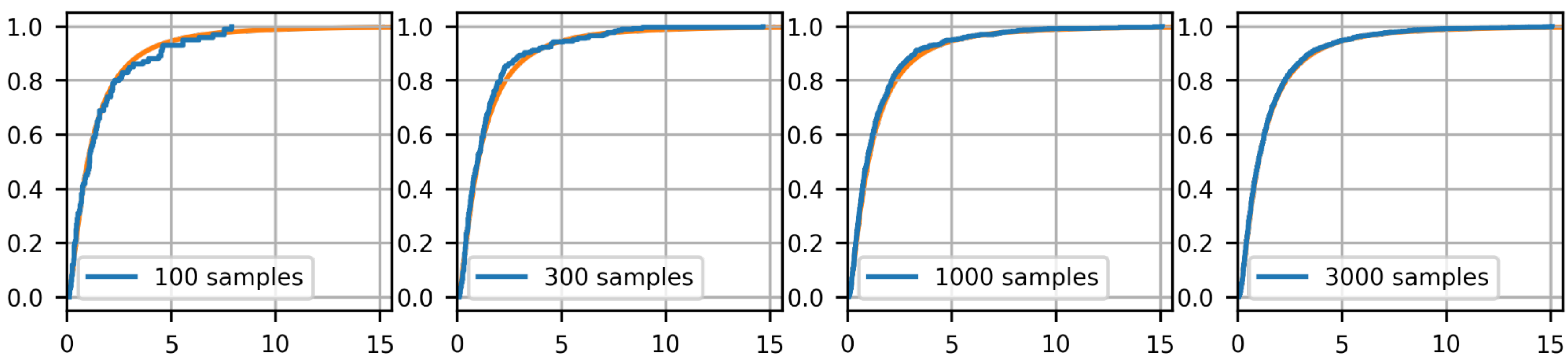
- It is an estimate of the CDF that generated the points in the sample.

Empirical Cumulative Distribution Function

$$\mathrm{ECDF}_{(T_1,\ldots,T_k)}(t) = \frac{\text{number of } T_i \leq t}{k} = \frac{1}{k}\sum_{i=1}^{k} 1_{\{T_i \leq t\}}$$

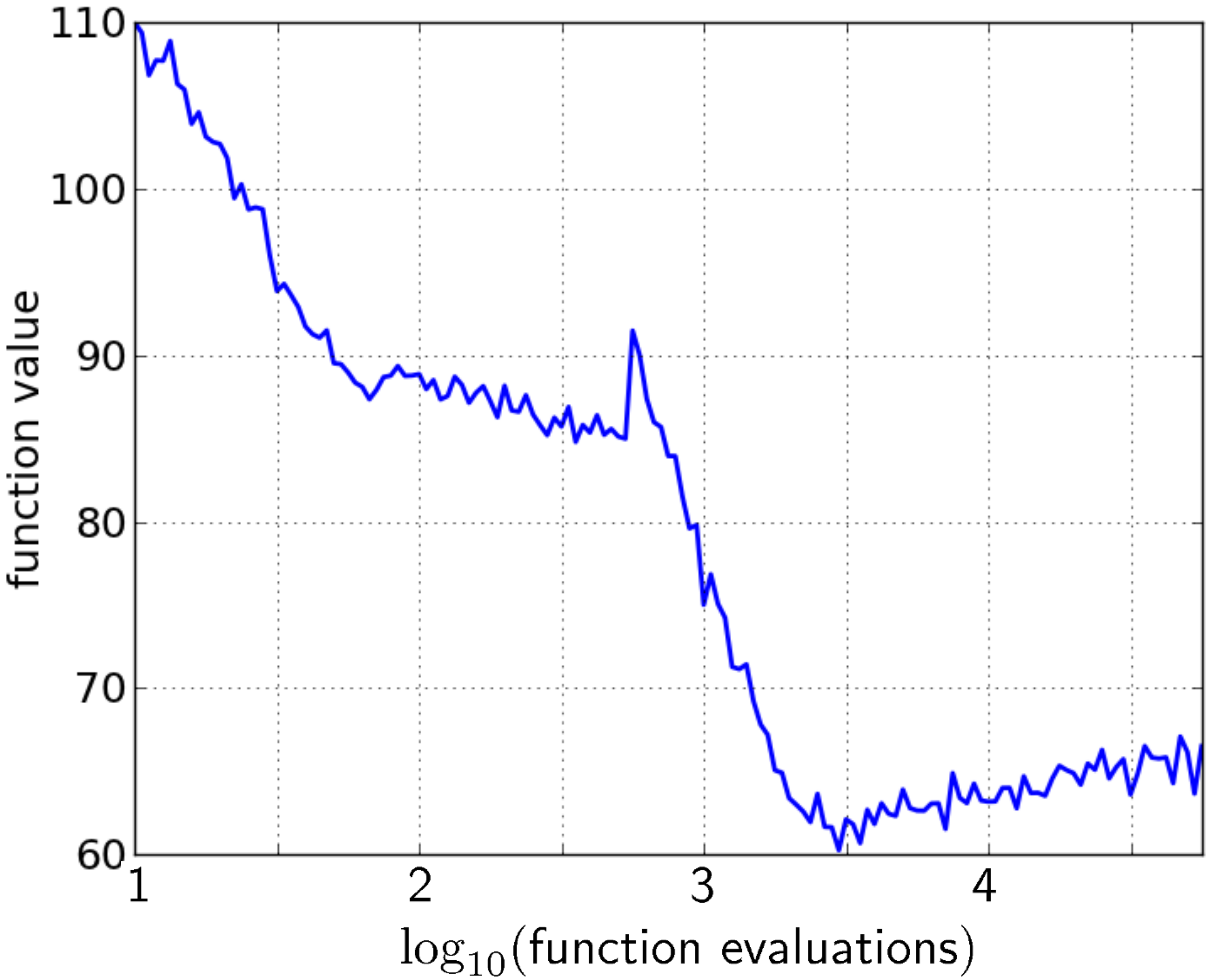
For $\{T_i : i \geq 1\}$ i.i.d. realization of a random variable T, by the LLN

$$\mathrm{ECDF}_{T_1,\ldots,T_k}(t) \xrightarrow[k\to\infty]{} \mathrm{CDF}_T(t) \text{ a.s. for all } t$$

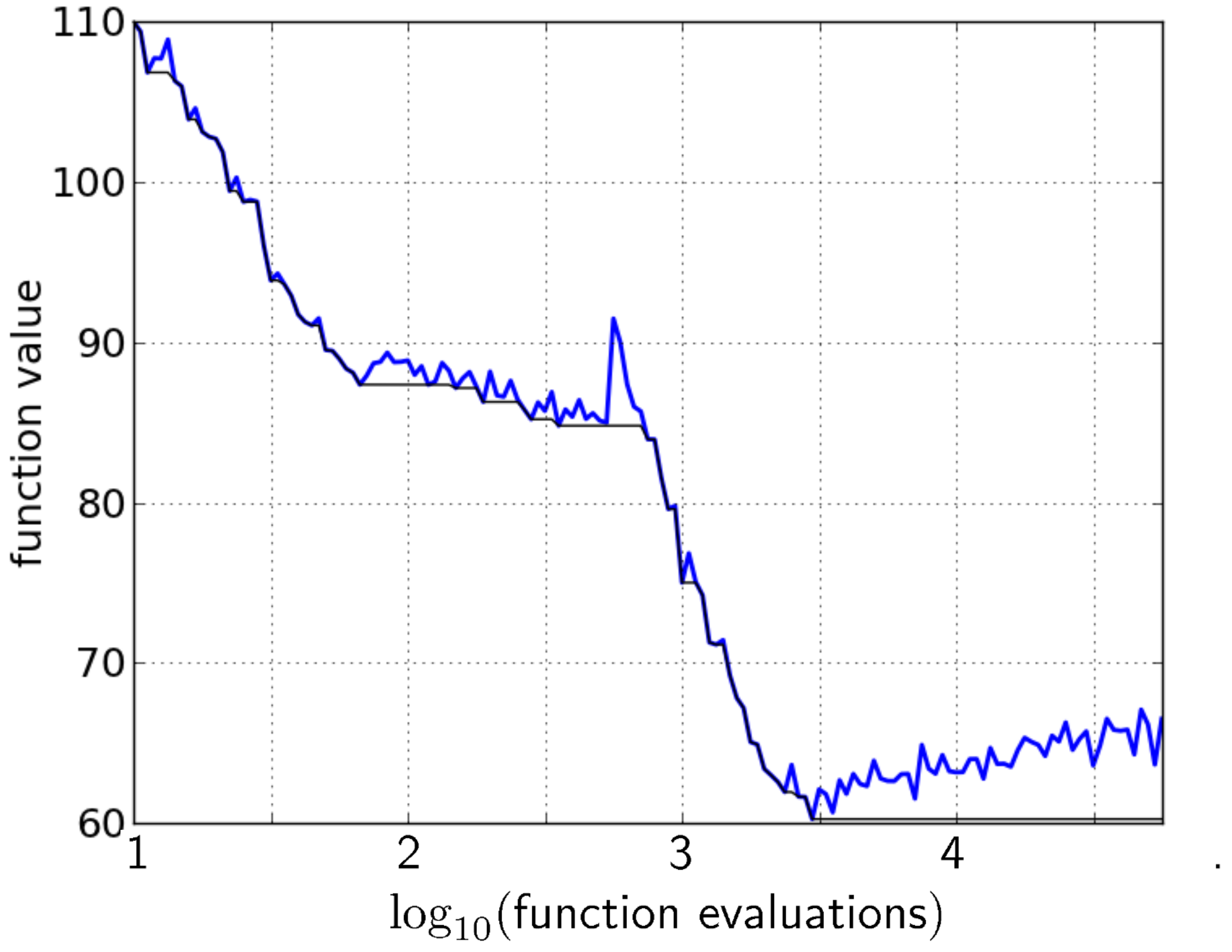


We display the ECDF of the runtime to reach target function values (see next slides for illustrations)

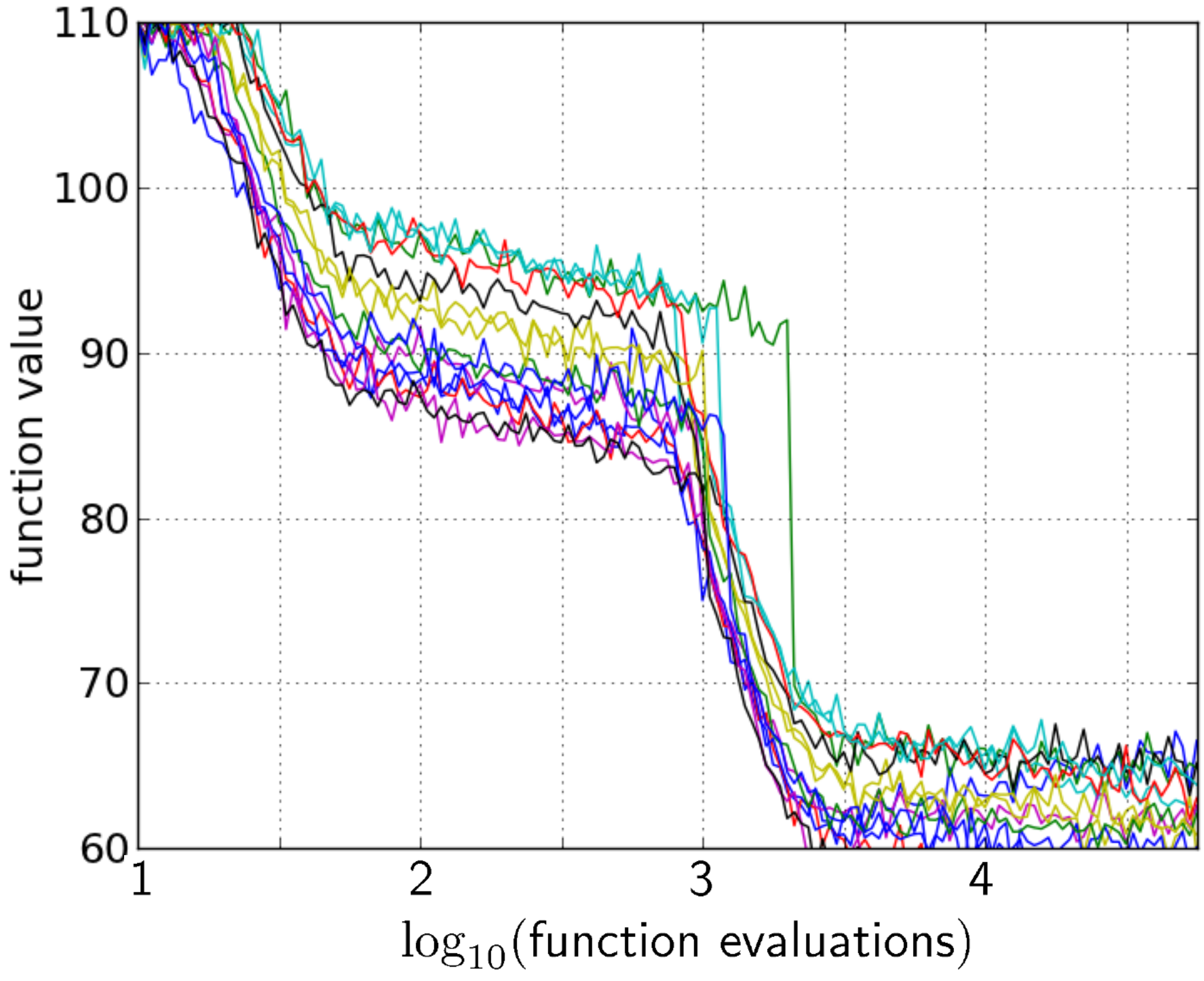
A Convergence Graph



First Hitting Time is Monotonous

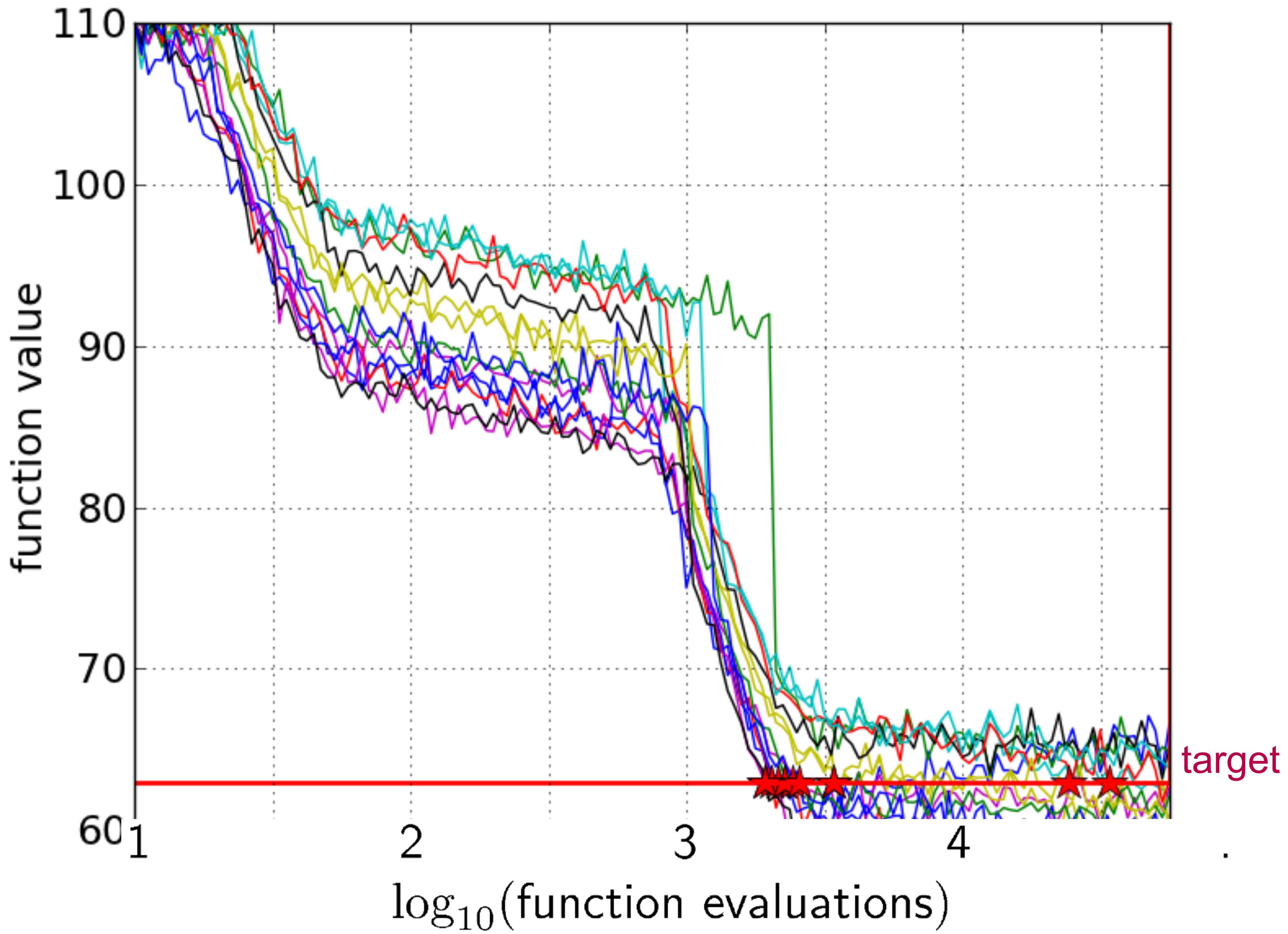


15 Runs

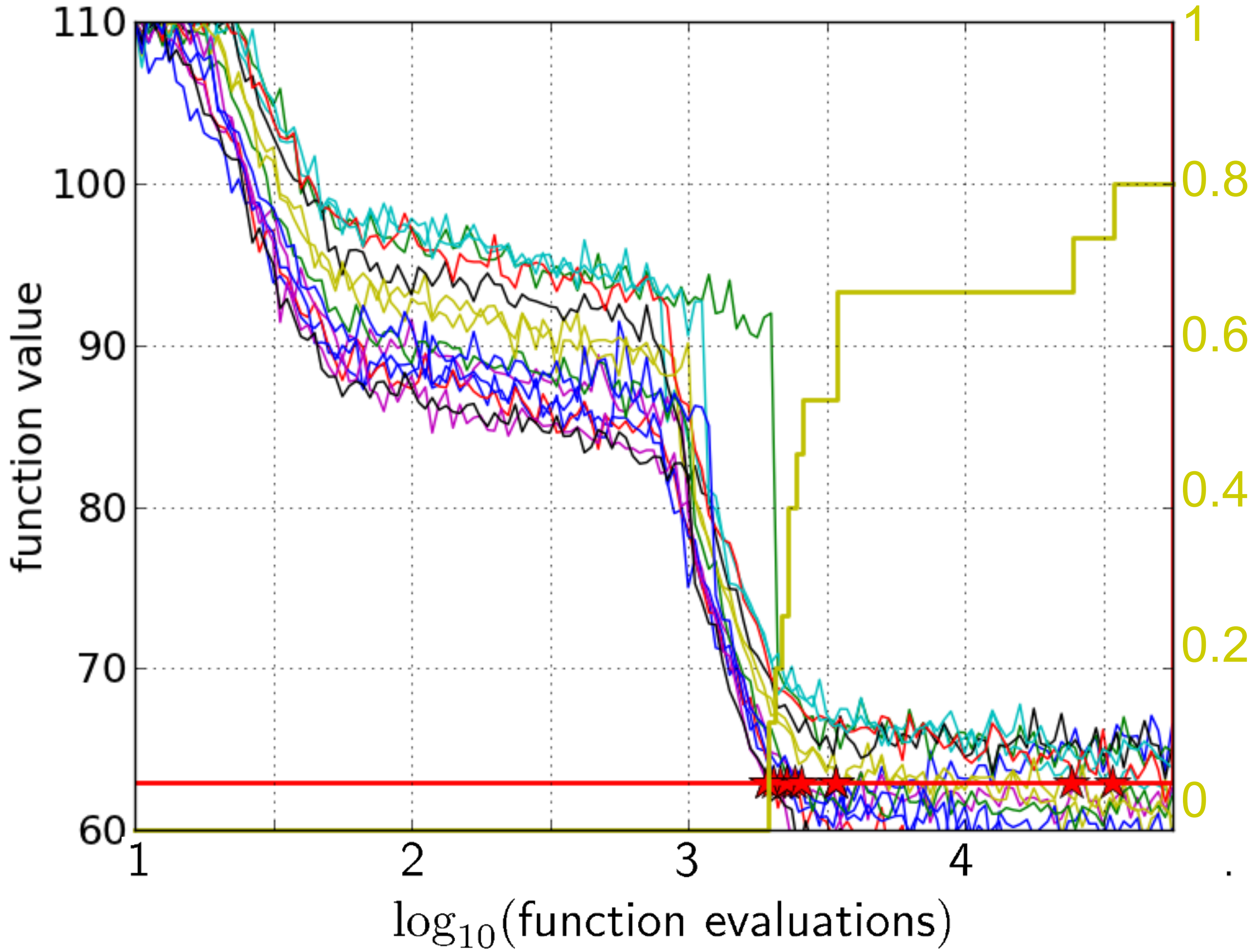


110
100
90
80
70
60
function value
1
2
3
4
$\log_{10}$(function evaluations)

15 Runs ≤ 15 Runtime Data Points



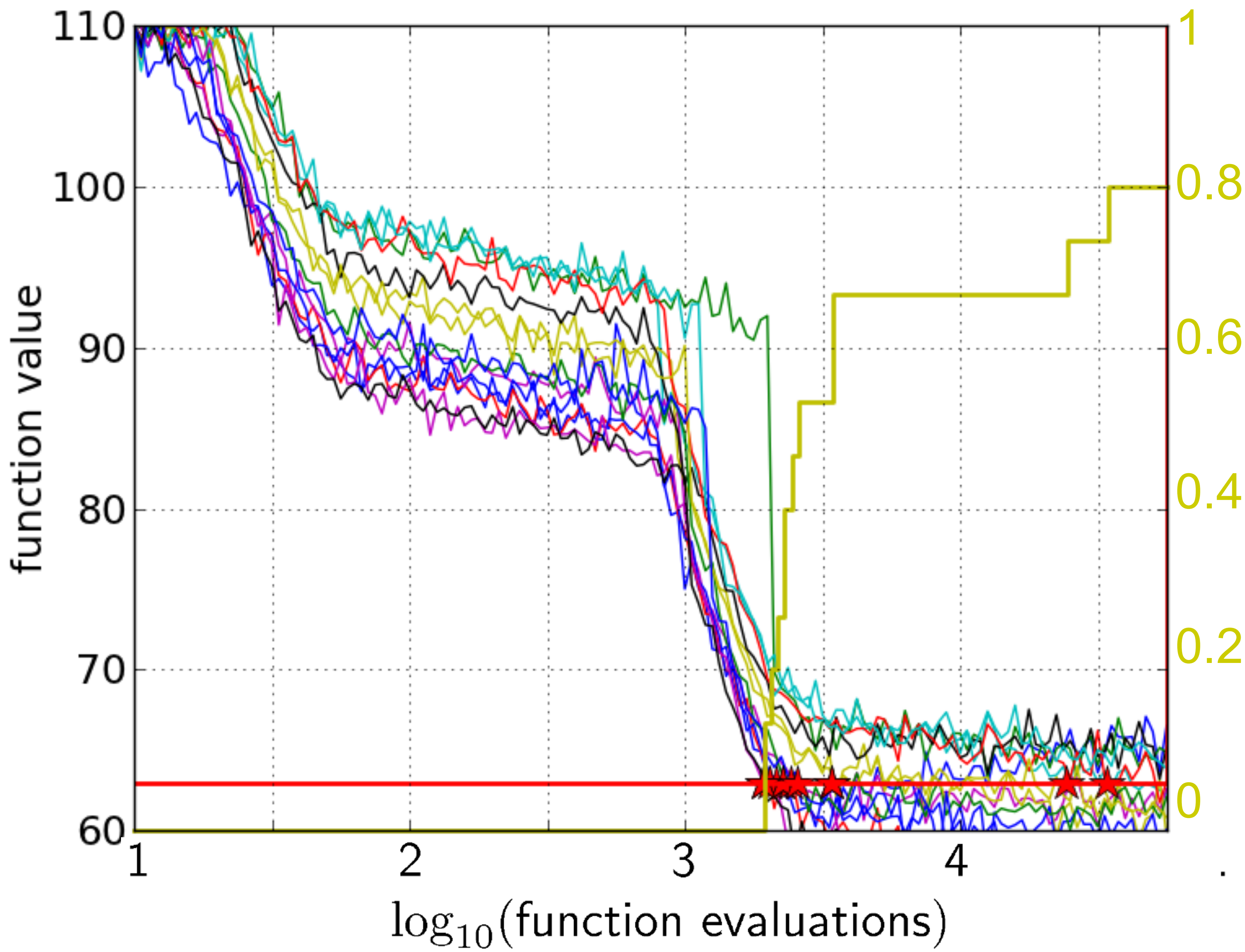
Empirical Cumulative Distribution



the ECDF of run lengths to reach the target

- has for each data point a vertical step of constant size
- displays for each x-value (budget) the count of observations to the left (first hitting times)

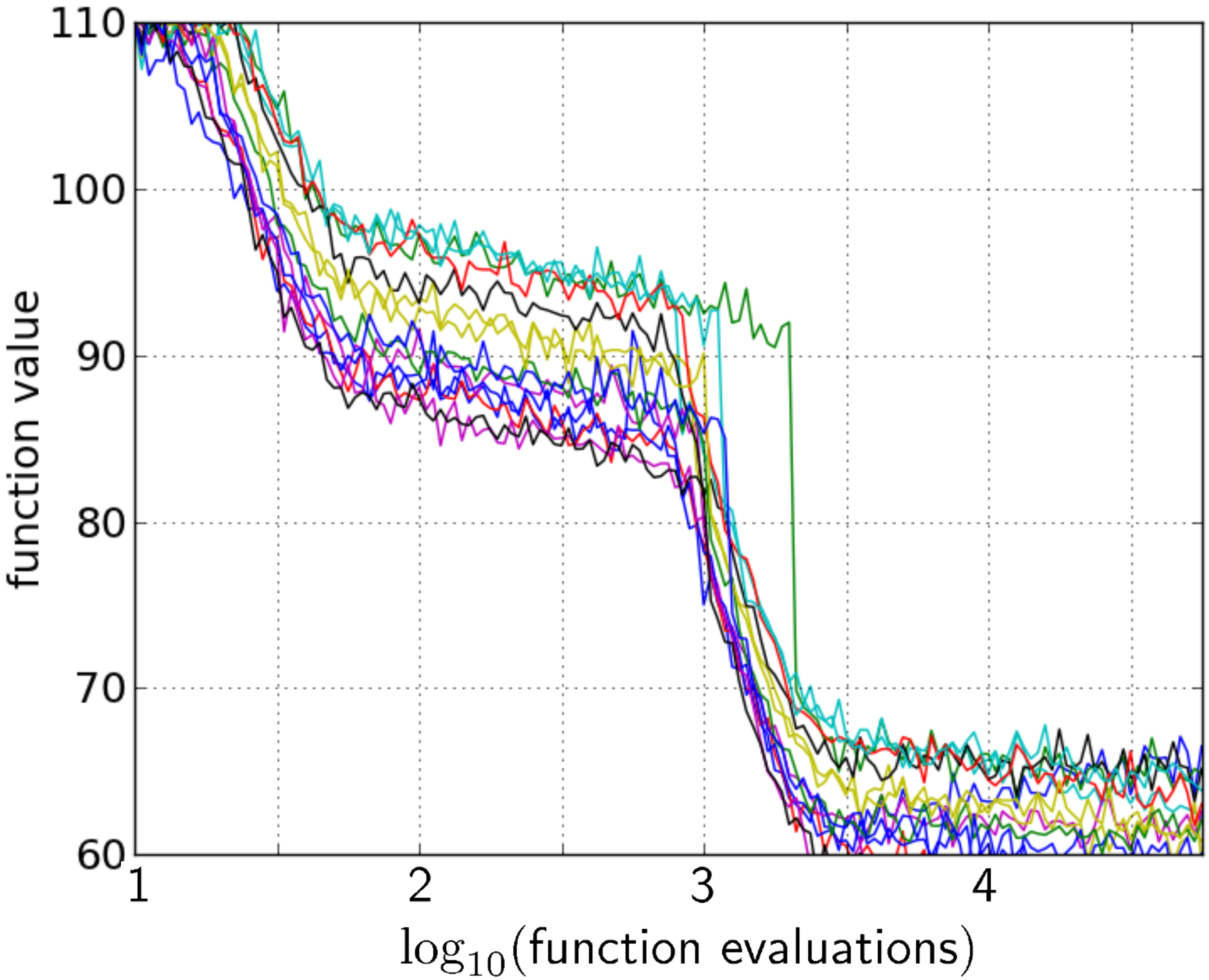
Empirical Cumulative Distribution



interpretations possible:

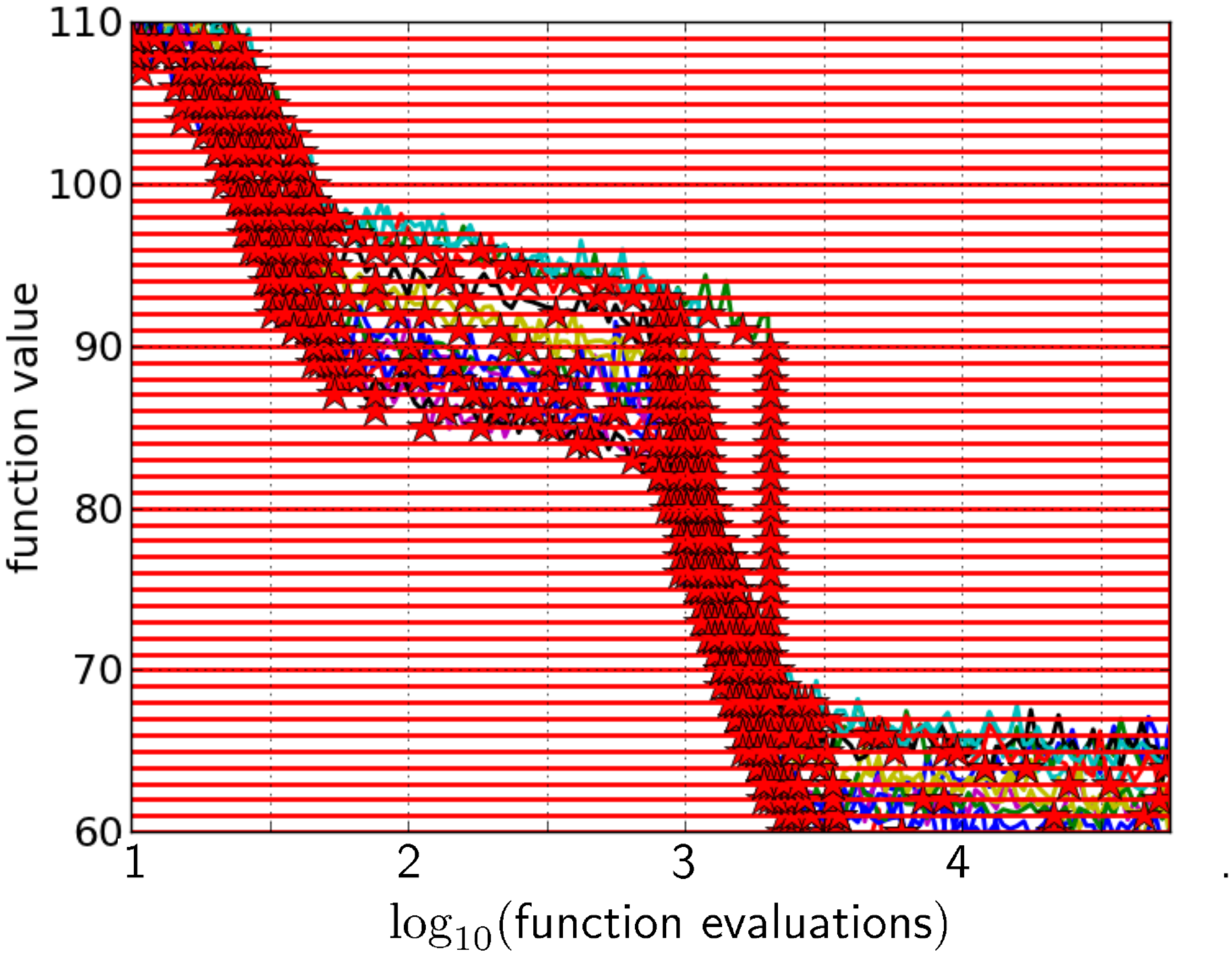
- 80% of the runs reached the target
- e.g. 60% of the runs need between 2000 and 4000 evaluations

Aggregation



15 runs

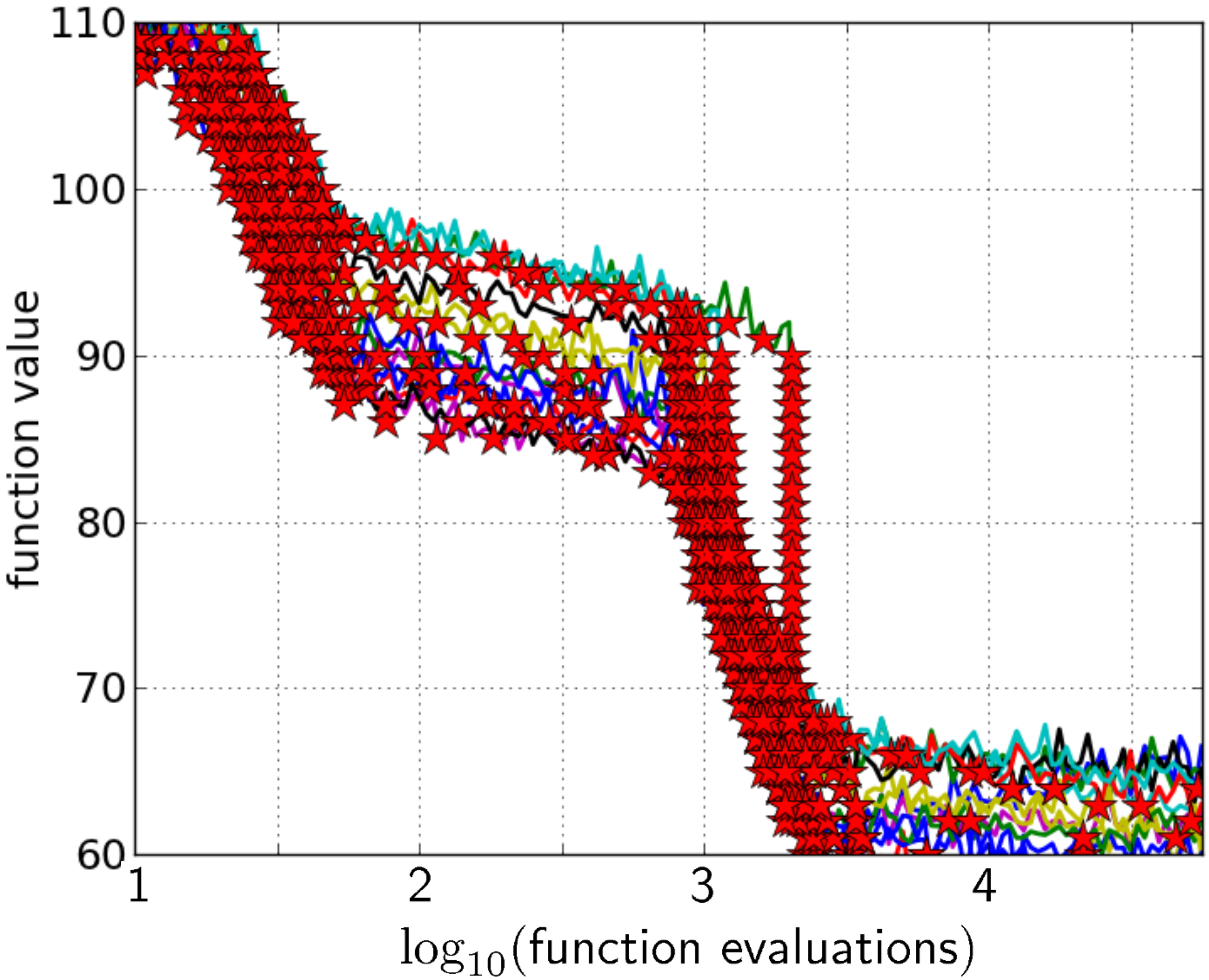
Aggregation



15 runs

50 targets

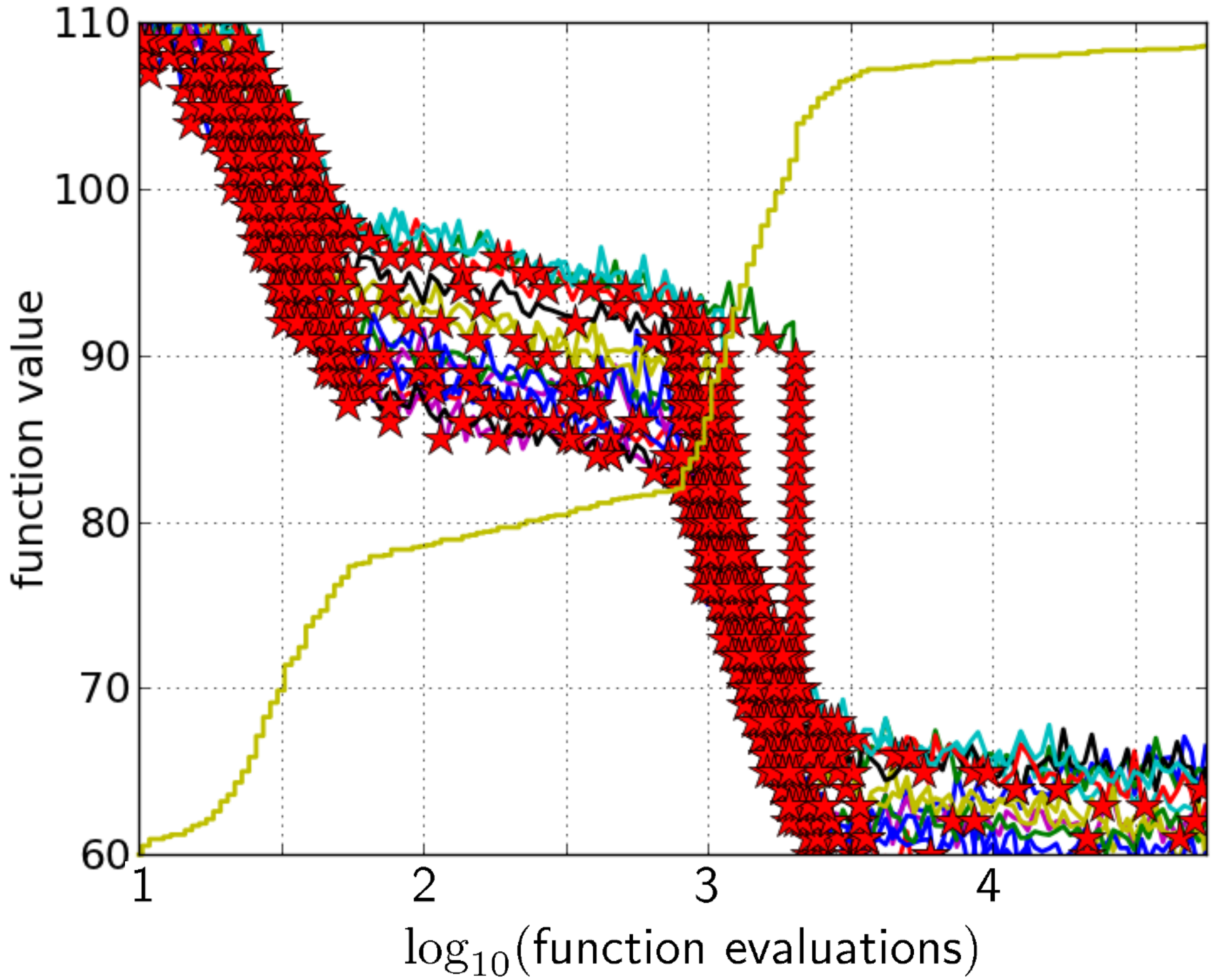
Aggregation



15 runs

50 targets

Aggregation

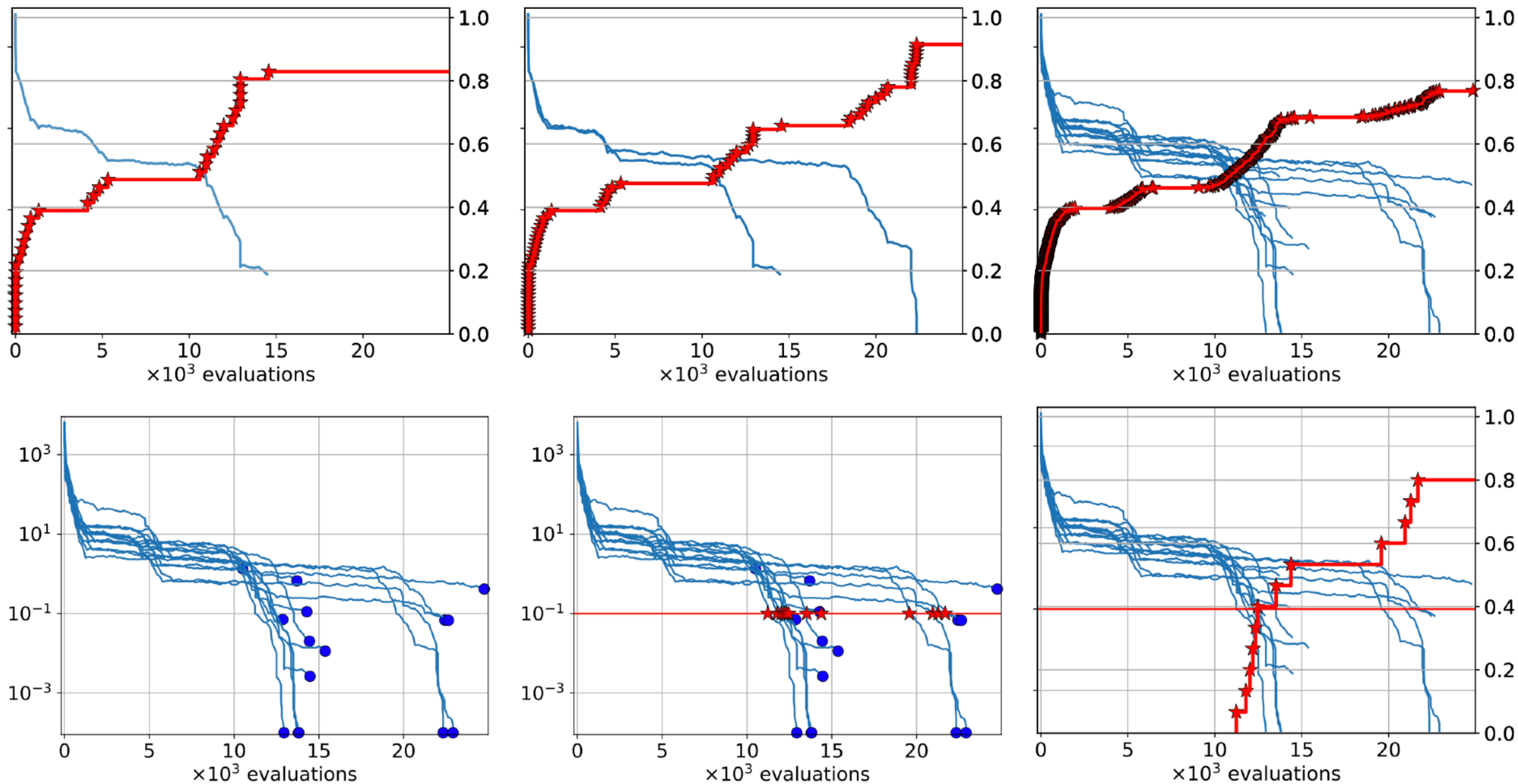


15 runs

50 targets

ECDF with 750 steps

Aggregation of Several Convergence Graphs



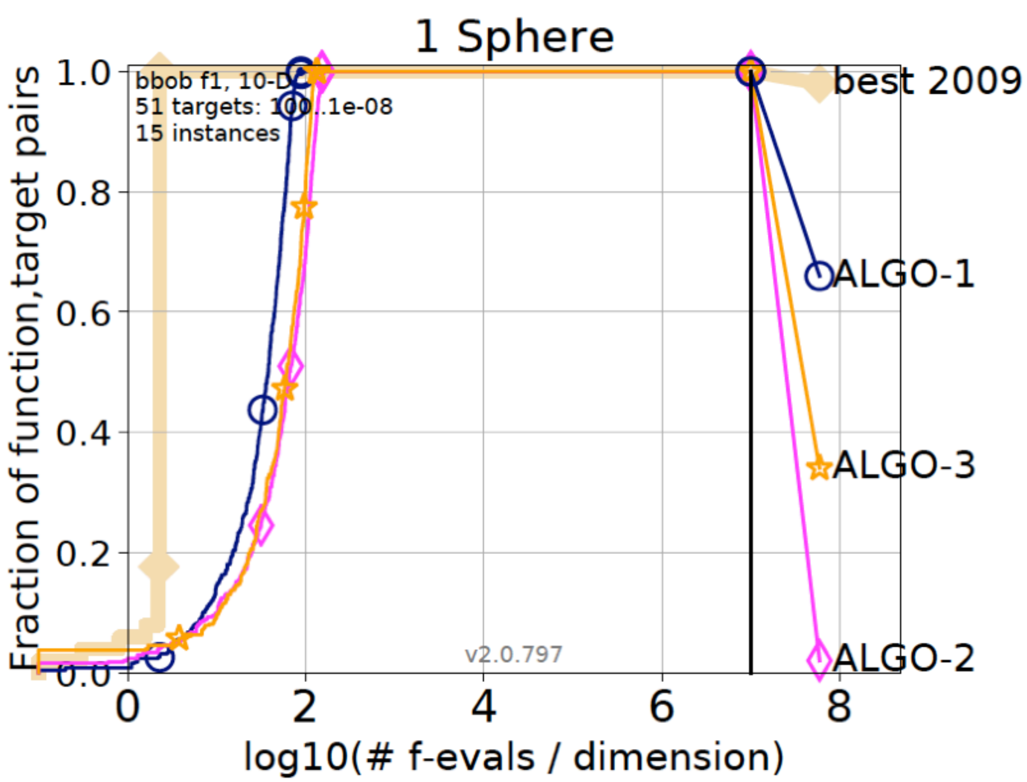
We can aggregate over:

- different targets
- different functions and targets

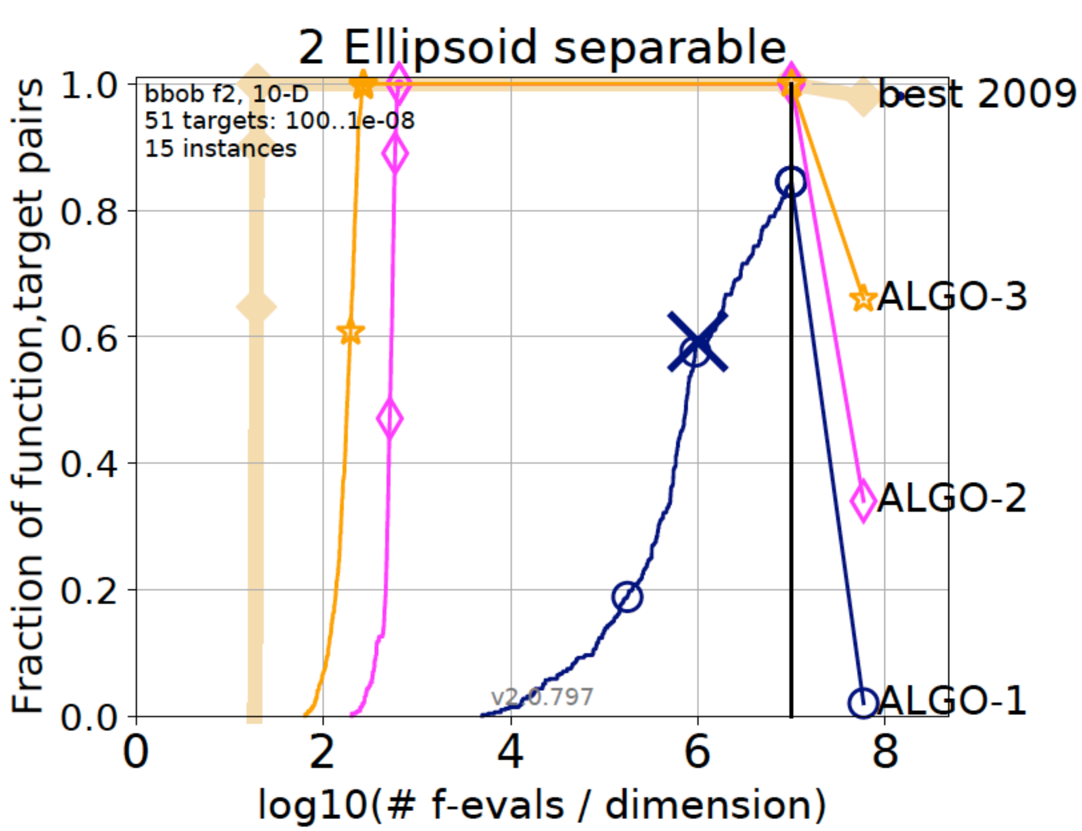
We should not aggregate over dimension

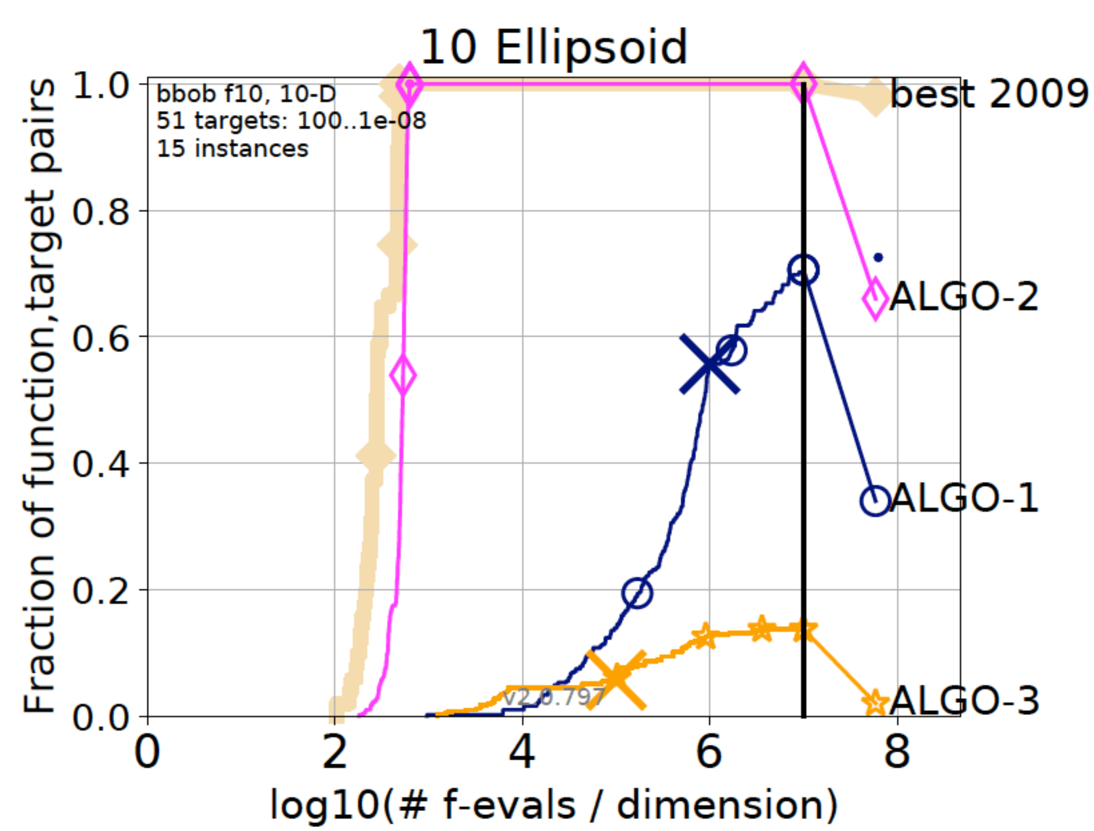
as functions of different dimensions have typically very different runtimes

ECDF aggregated over targets - single functions

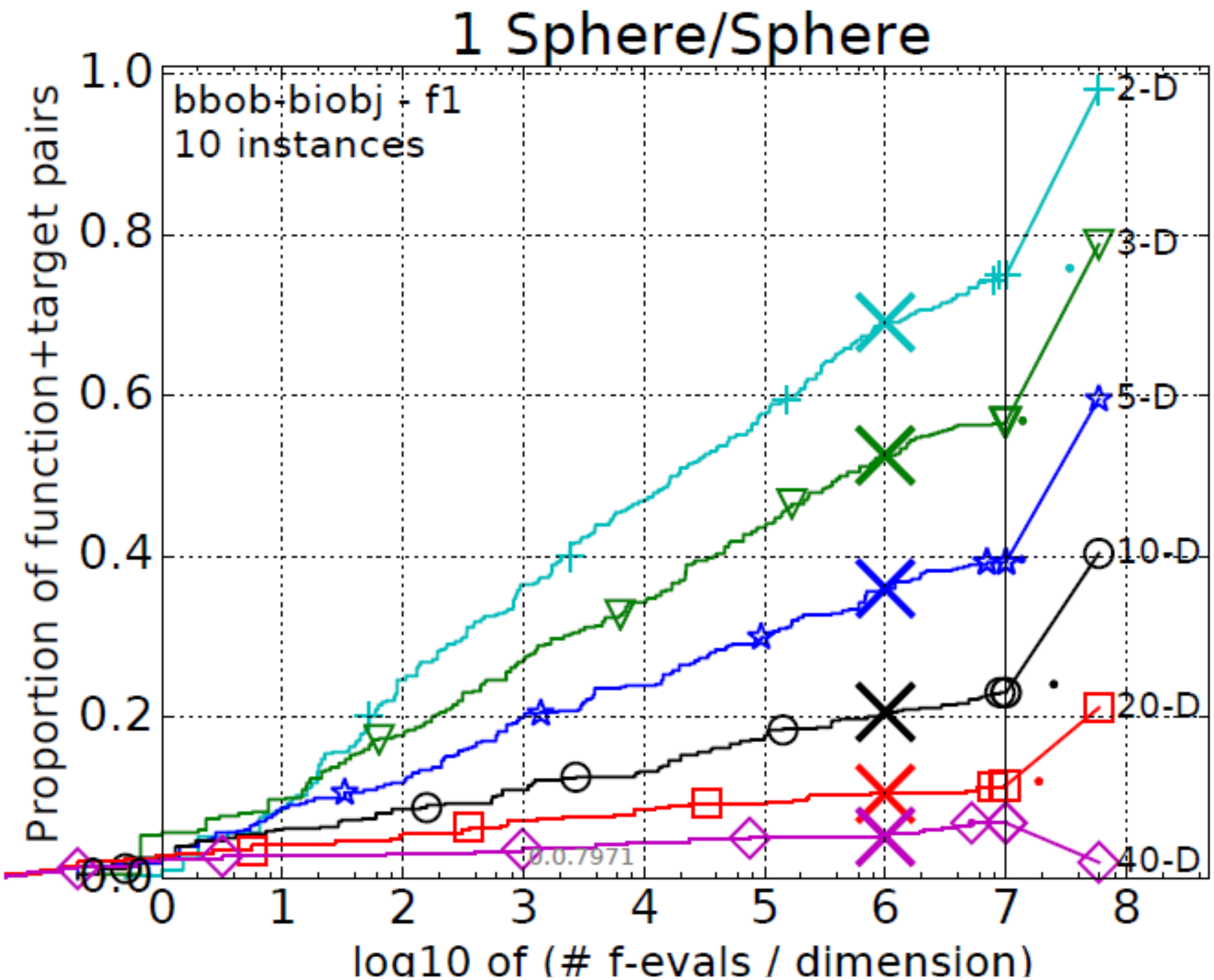


ECDF for 3 different algorithms



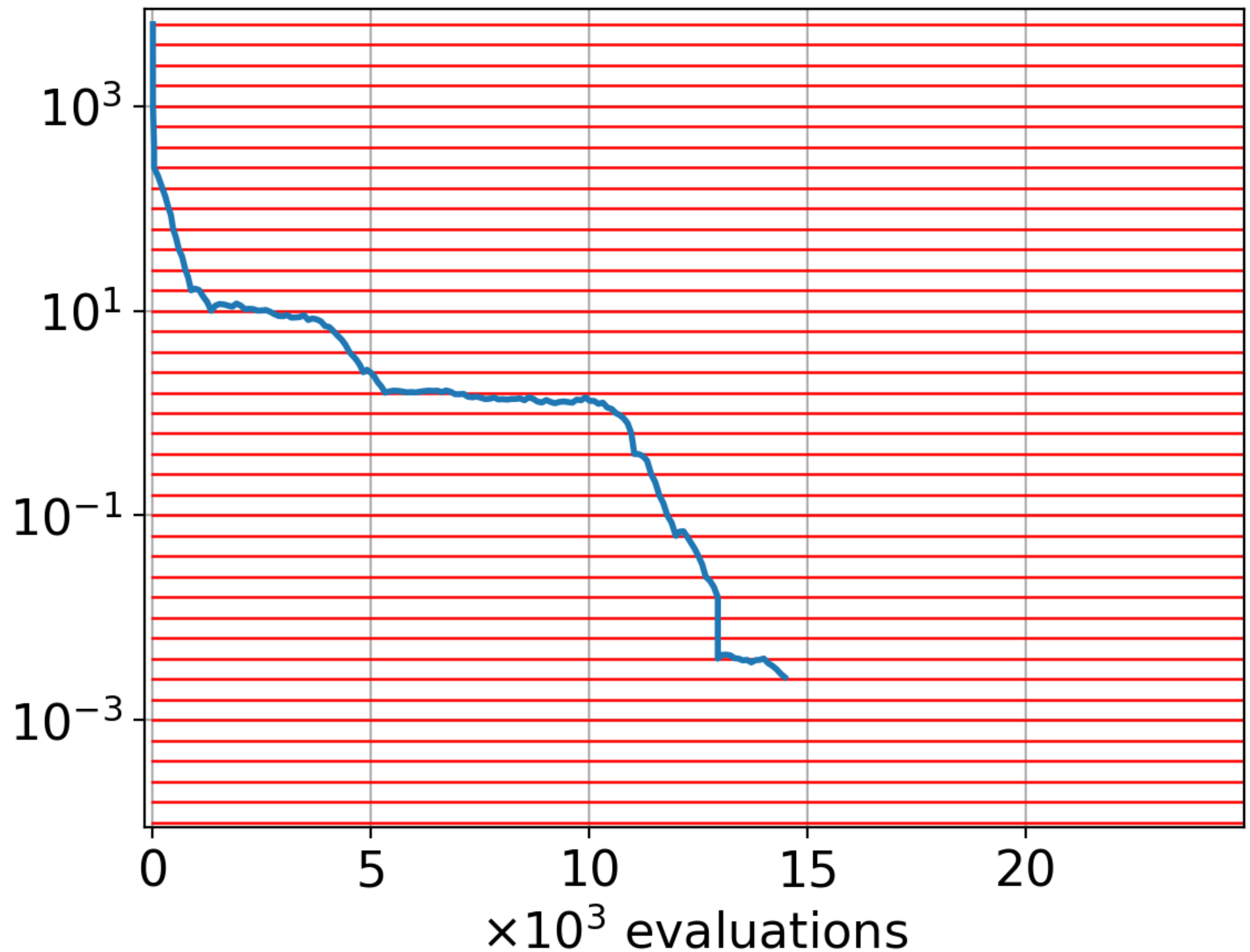


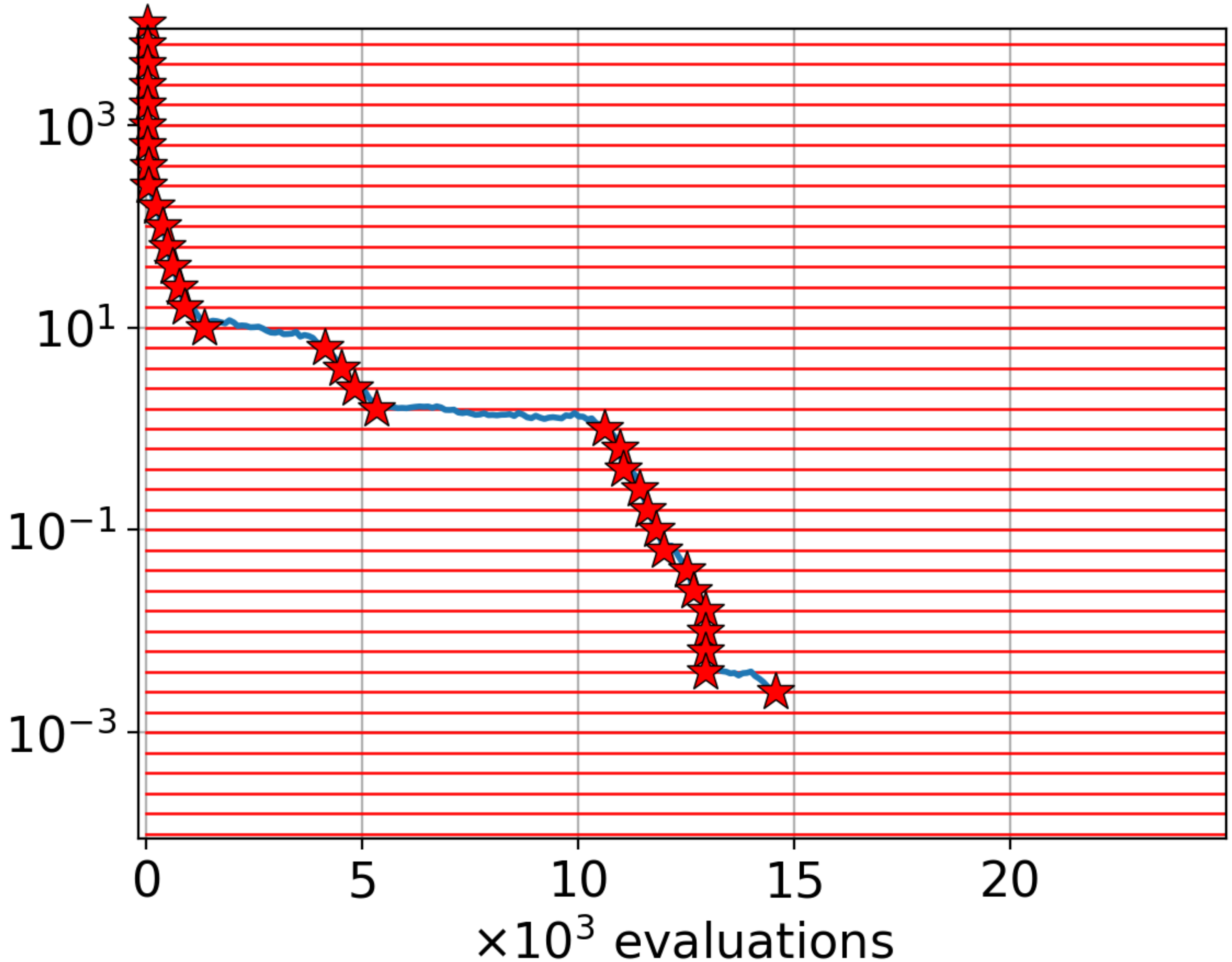
ECDF aggregated over targets - single function



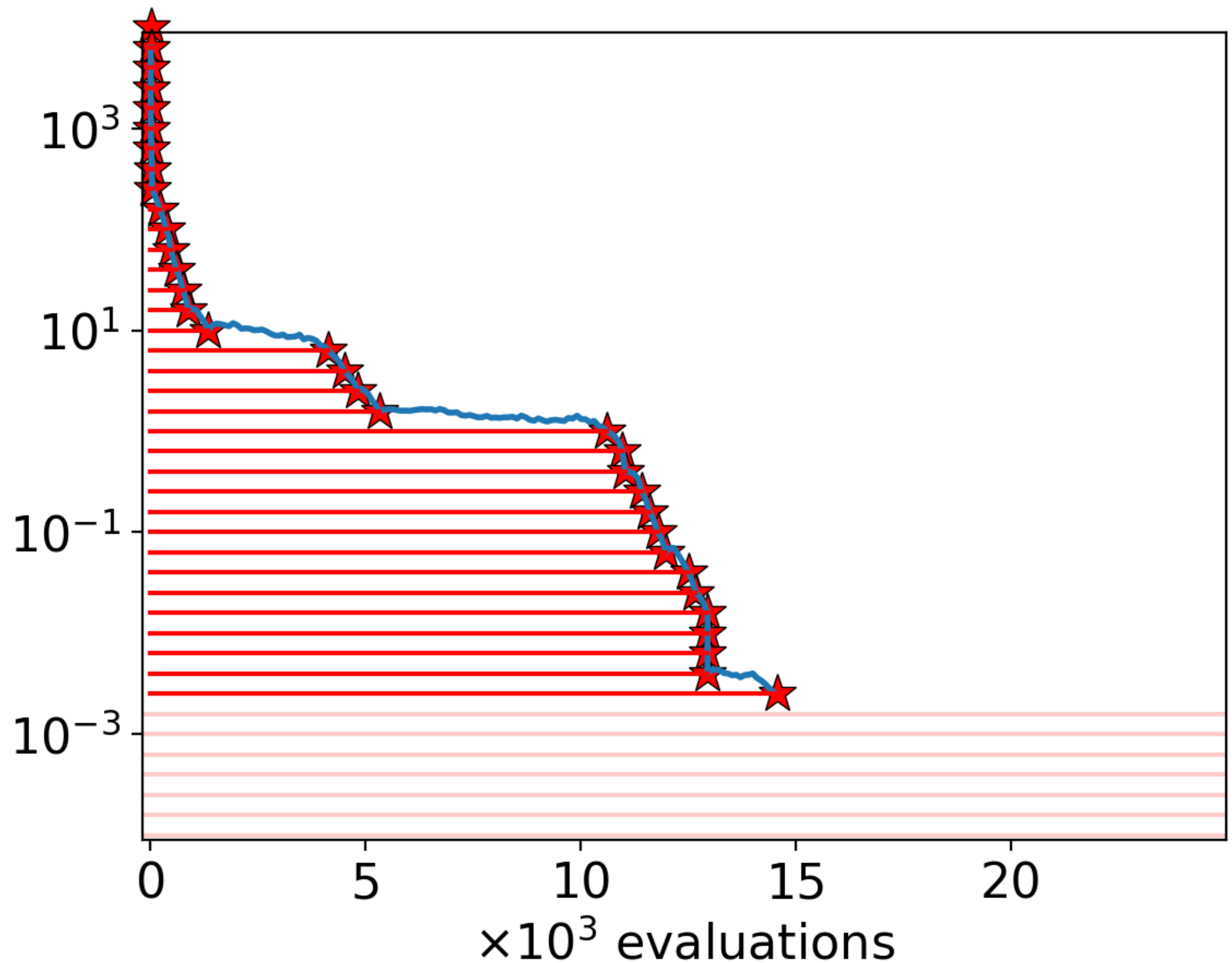
ECDF for a single algorithm different dimensions

ECDF is a generalization of a convergence graph

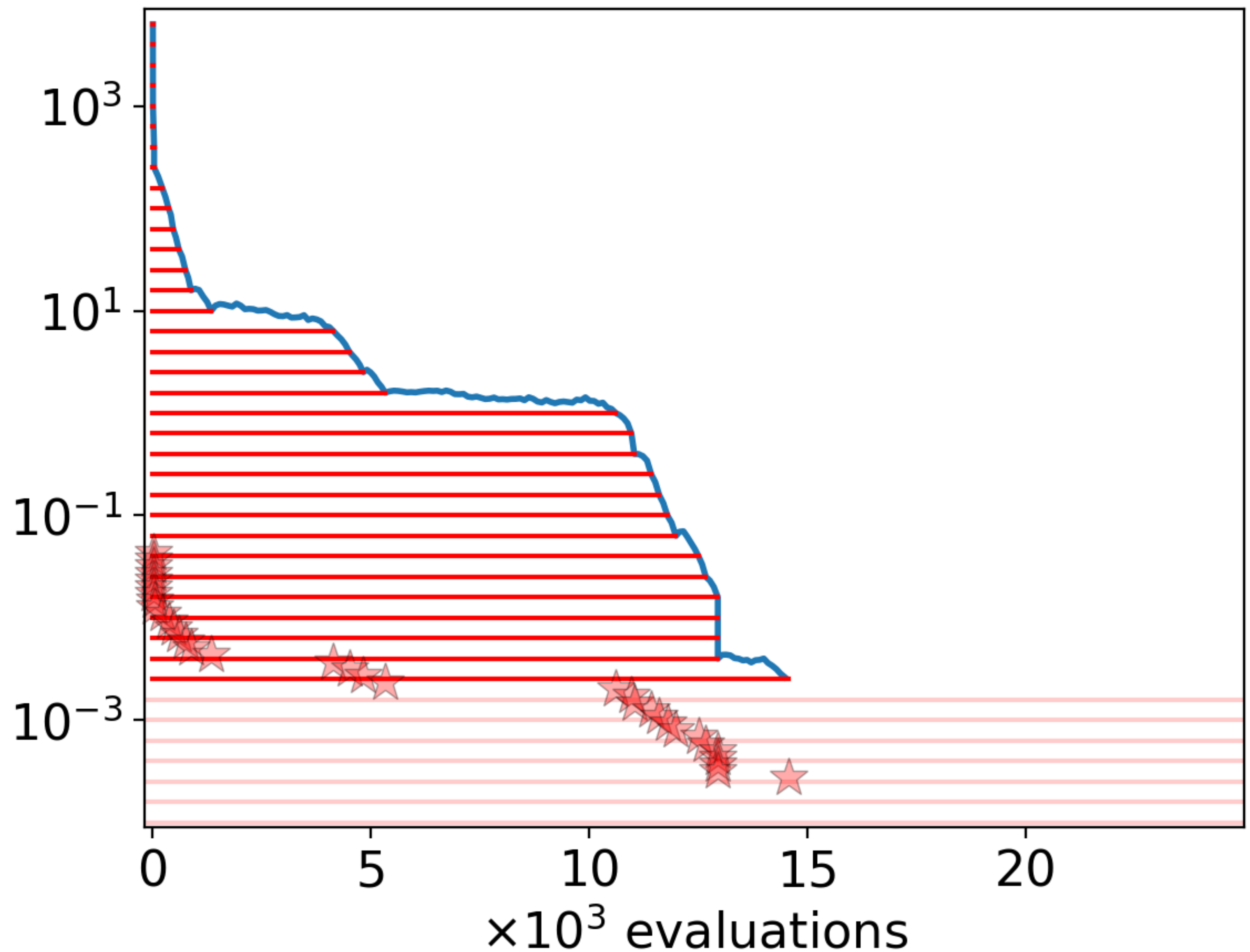

10^{3}
10^{1}
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^{3}$ evaluations



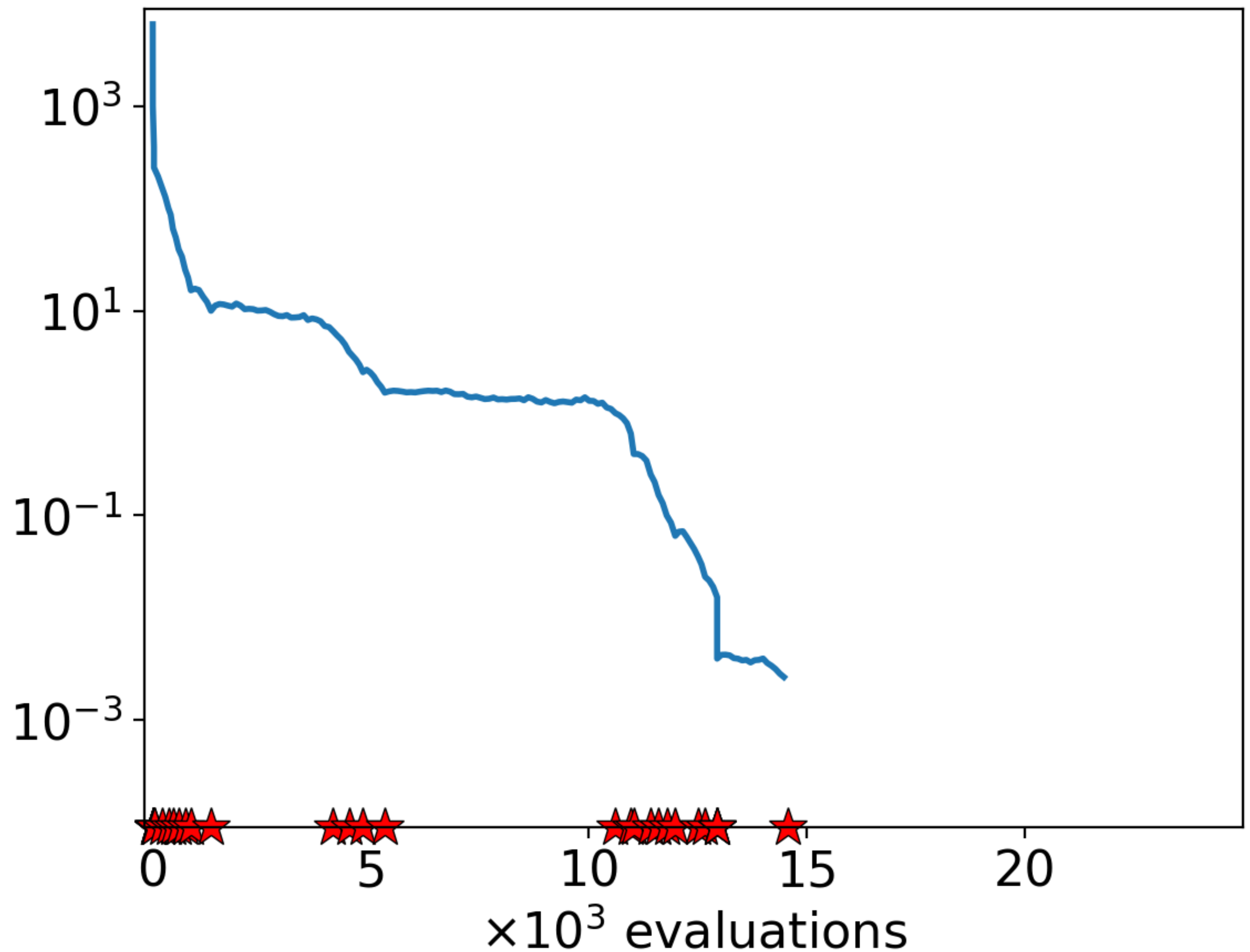
10^{3}
10^{1}
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^{3}$ evaluations



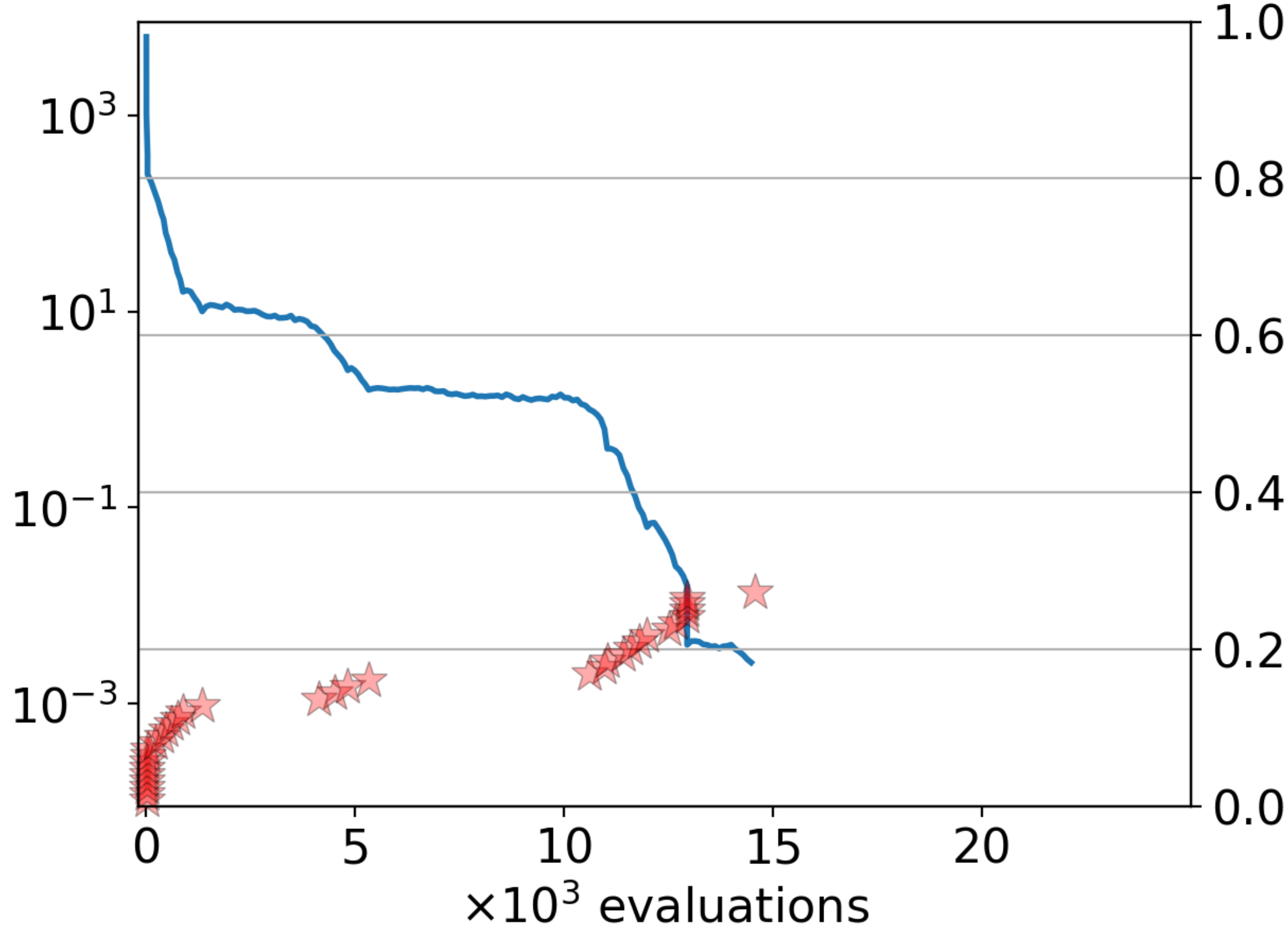
10^3
10^1
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^3$ evaluations



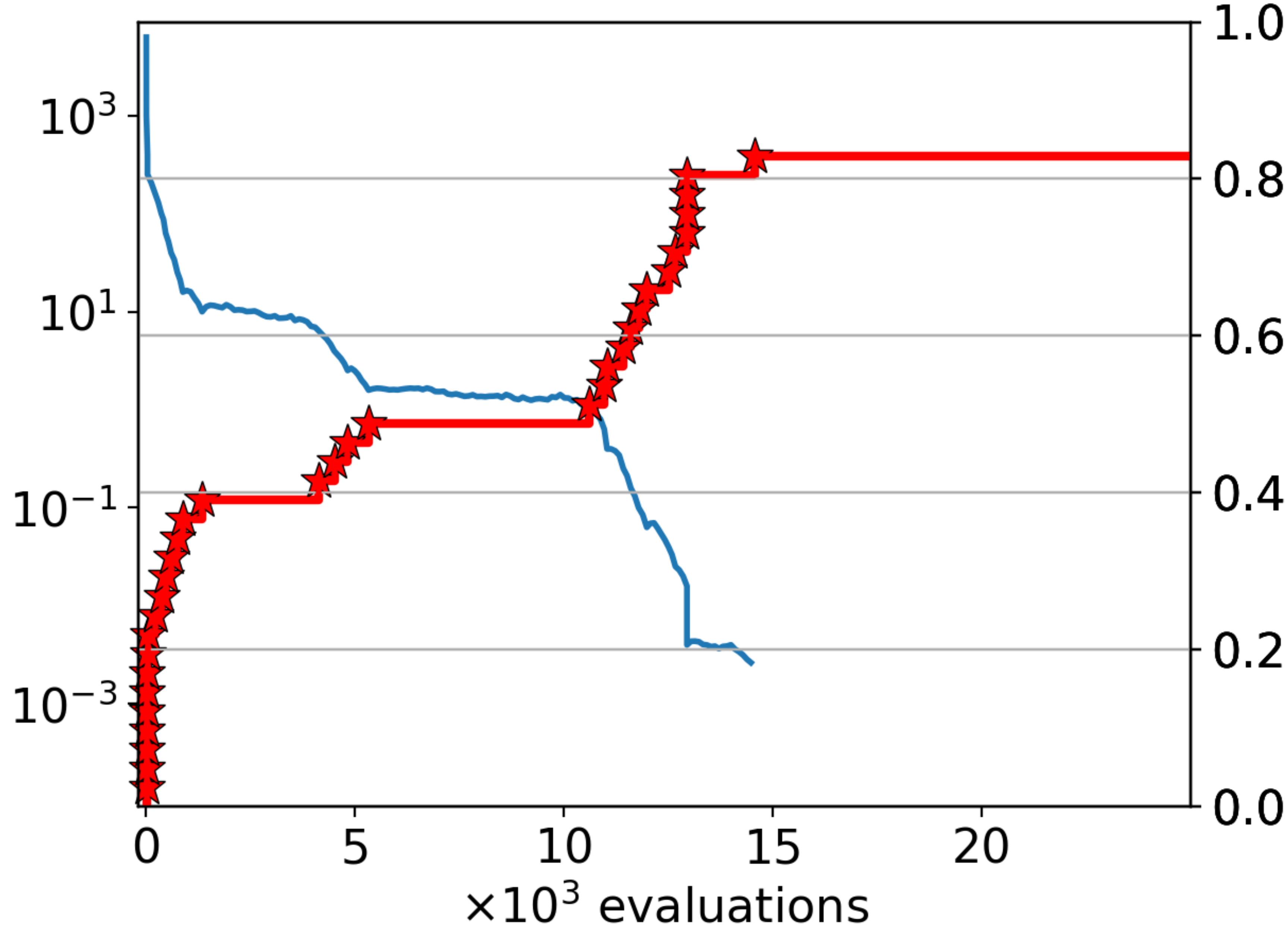
10^3
10^1
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^3$ evaluations

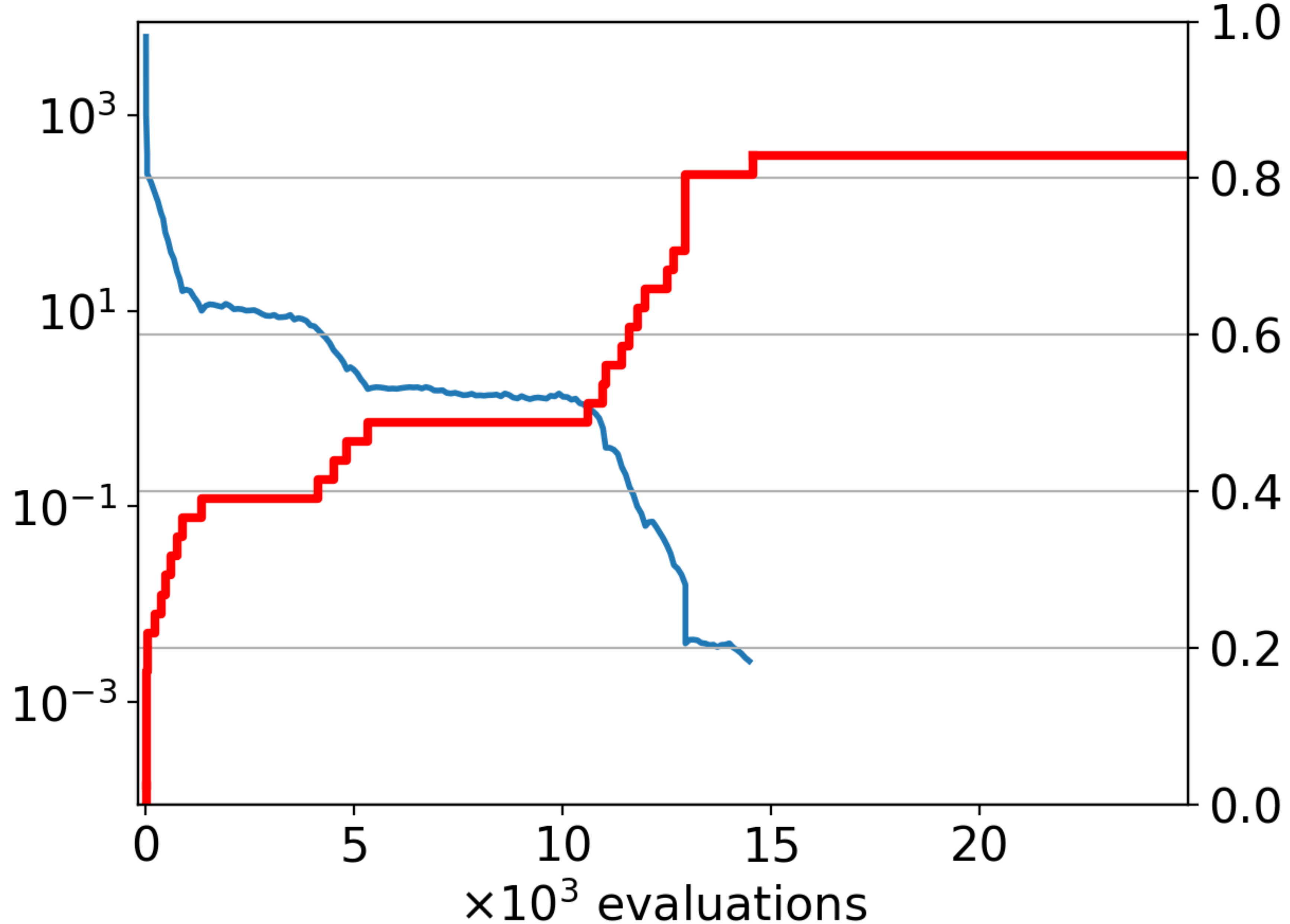


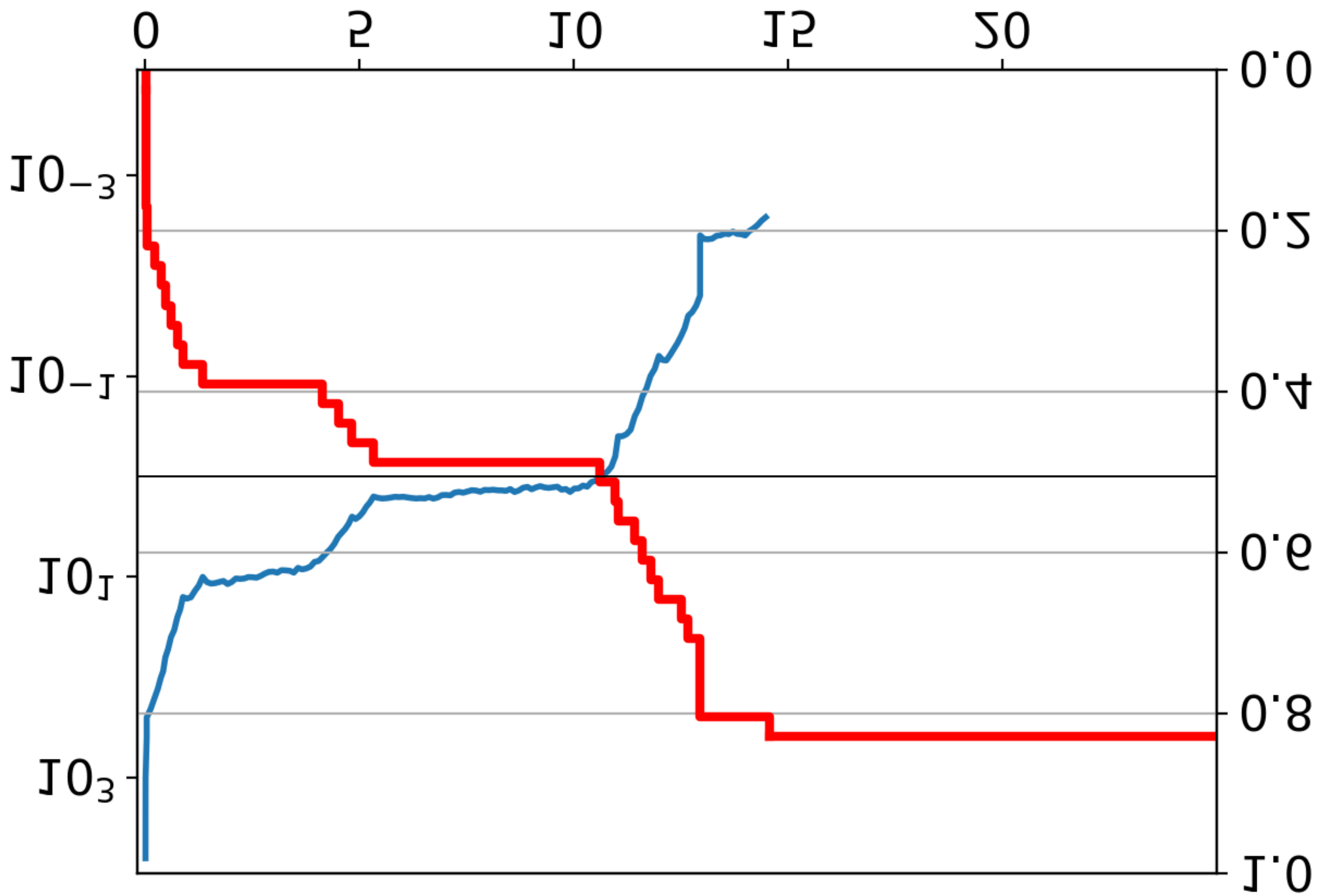
10^3
10^1
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^3$ evaluations

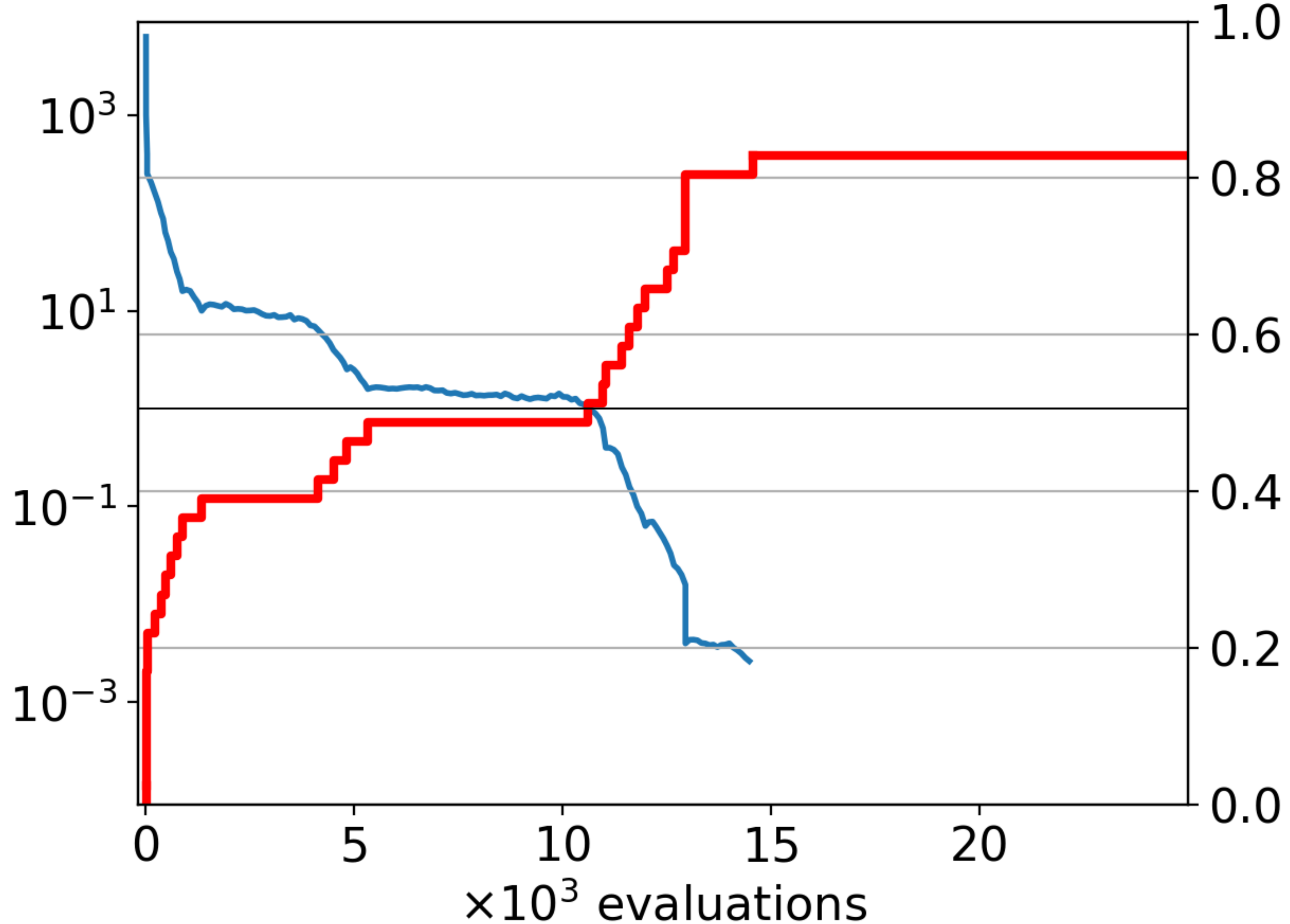


10^3
10^1
10^{-1}
10^{-3}
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
$\times 10^3$ evaluations

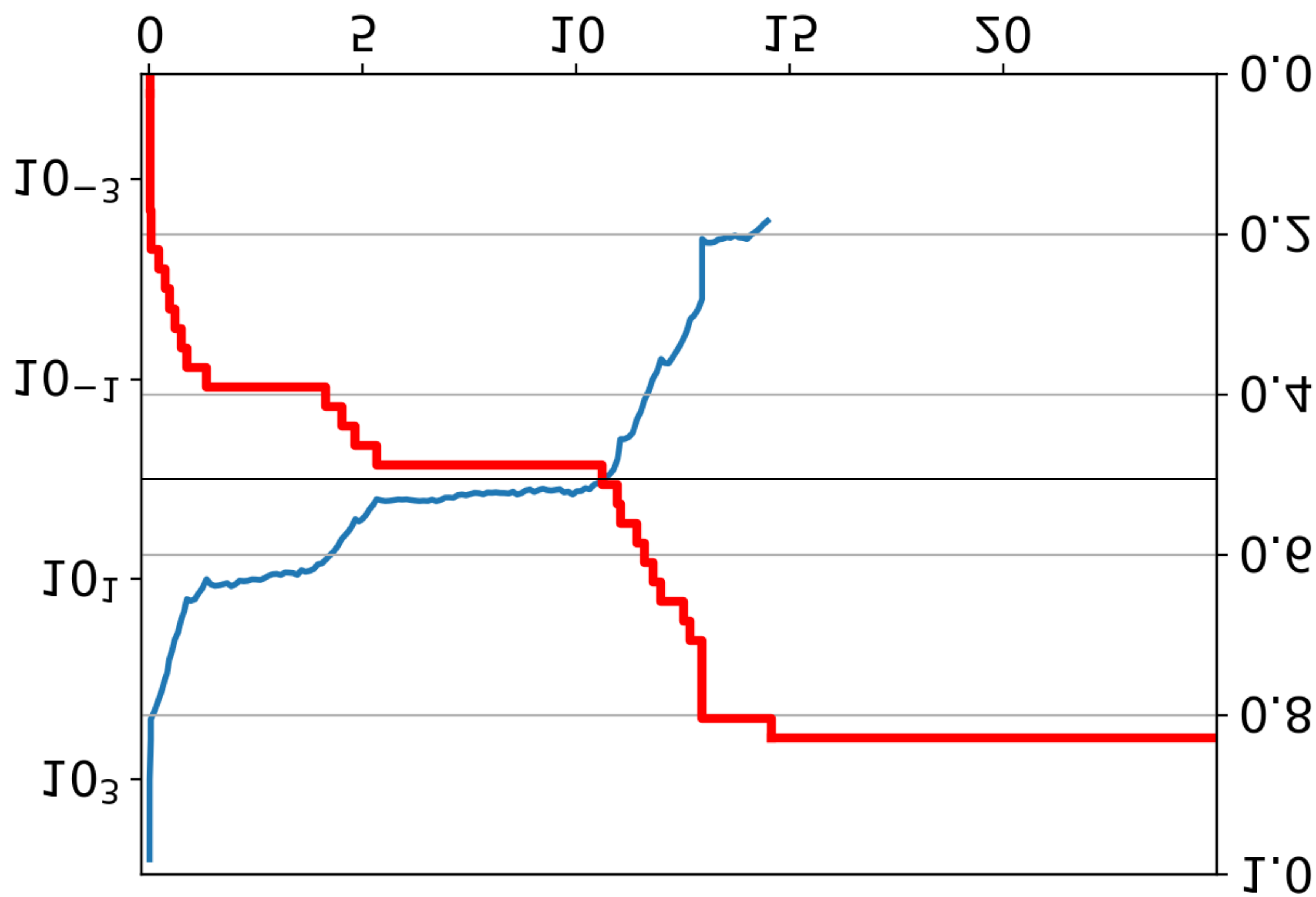

10^3
10^1
10^{-1}
10^{-3}
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
$\times 10^3$ evaluations


10^3
10^1
10^{-1}
10^{-3}
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
$\times 10^3$ evaluations

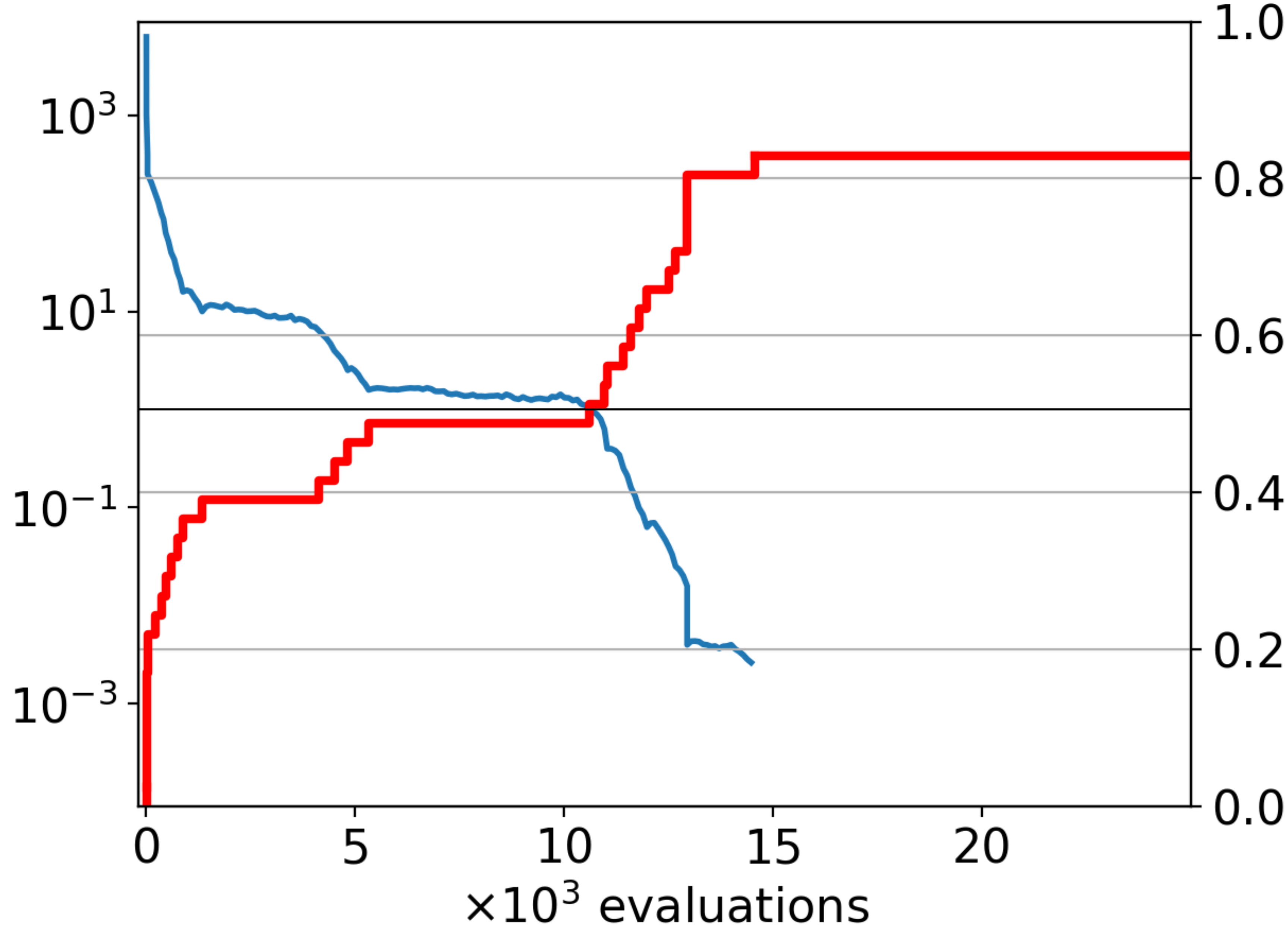




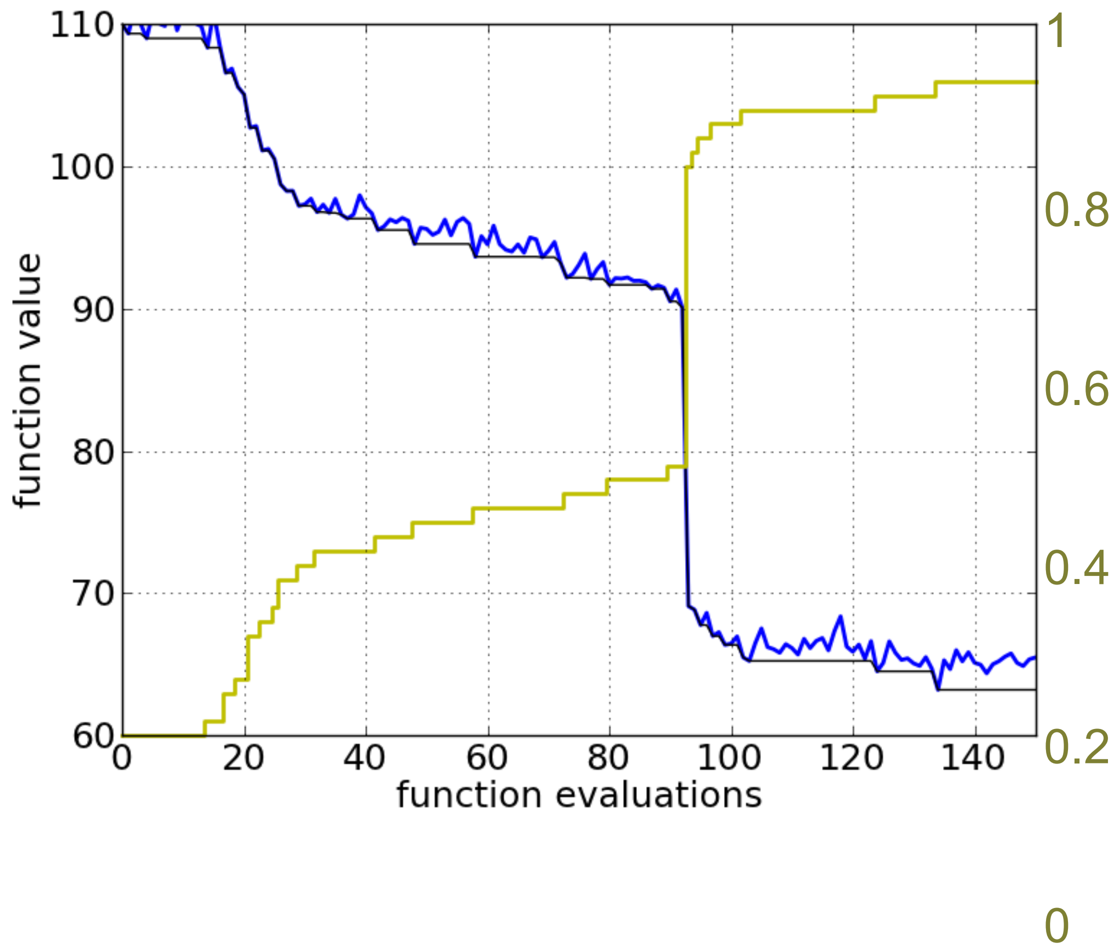
10^3
10^1
10^{-1}
10^{-3}
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
$\times 10^3$ evaluations


0
5
10
15
20
10^{-3}
10^{-1}
10^{1}
10^{3}
0.0
0.2
0.4
0.6
0.8
1.0

the ECDF recovers the monotonous graph, discretised and flipped



Runtime distribution from a single graph



the ECDF recovers the monotonous graph, discretised and flipped

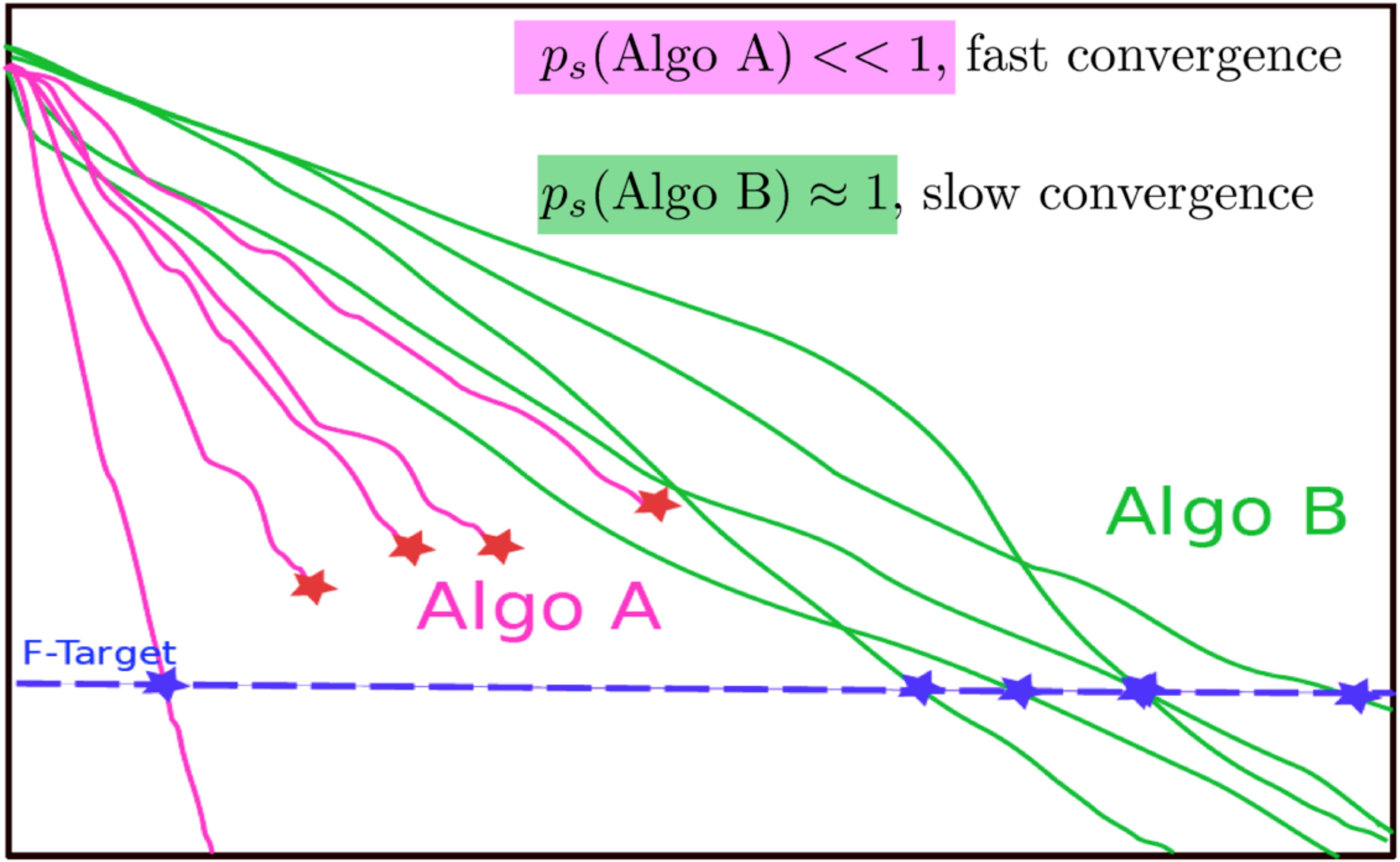
AKA runtime distribution

ERT/ART:

Average Runtime

Which performance measure ?

to compare the two following scenario?



Which performance measure ?

Algo Restart A:

$$p_s(\text{Algo Restart A}) = 1$$

Algo Restart B:

$$p_s(\text{Algo Restart B}) = 1$$

Expected Running Time (restart algo)

$$\text{ERT} = E[RT^r] = \frac{1-p_s}{p_s} E[RT_{\text{unsuccessful}} + E[RT_{\text{successful}}]$$

Estimator for ERT

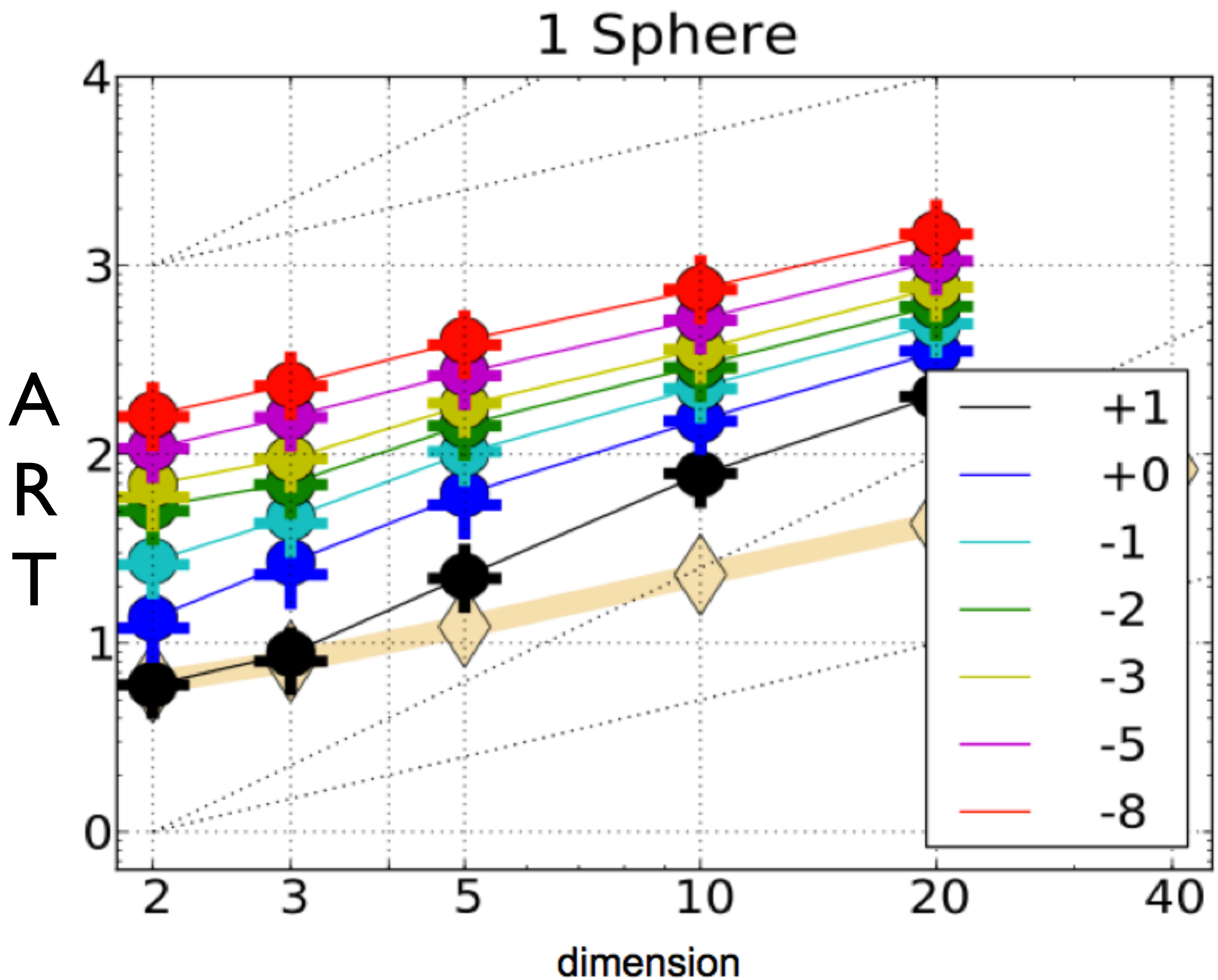
$$\widehat{p_s} = \frac{\#\text{succ}}{\#Runs}$$

$$\widehat{RT_{\text{unsucc}}} = \text{Average Evals of unsuccessful runs}$$

$$\widehat{RT_{\text{succ}}} = \text{Average Evals of successful runs}$$

$$\text{ART} = \frac{\#\text{Evals}}{\#\text{success}}$$

Example: scaling behavior



ART on f1 of a variant of CMA-ES – linear scaling

On Test functions

What is the Benchmark?

Choice of Test Problems

What to Benchmark?

Furious activity is no substitute for understanding
(H.H. Williams)

- Taking all possible functions from a repository?

What to Benchmark?

Furious activity is no substitute for understanding (H.H. Williams)

- Taking all possible functions from a repository?
- Bad idea if
    - function difficulties are unbalanced
      too many small dimensional problems, convex problems…
    - and performance are aggregated
- Leads to bias in the performance assessment

What to Benchmark?

- test functions should be representative of difficulties we want to test

  therefore NFL has no relevance as assumption of being closed under permutation has no relevance wrt real world problems

- related to real-word difficulties

  for performance to be generalizable to RW

- scalable

  dimension plays a big role in performance
  curse of dimensionality

- comprehensible but not too easy

  BB optimization does not mean BB benchmarking

- we should still hide properties from the solver (hide optimum, …)

  solvers should not be able to exploit the benchmark intentionally or not

Example: COCO/BBOB Test Suite(s)

Functions are

- based on known analytical functions, modeling a "known" difficulty

  related to real-world problems

- comprehensible
- scalable
- difficult (also non-separable)

  compared to typical standards (at that time)

- quasi-randomized as instances

  with arbitrary shifts and smallish irregularities

  to avoid artificial exploits and mitigate overfitting, emulates repetition of experiments

Example: COCO/BBOB Test Suite(s)

1 Separable Functions	
f1	Sphere Function
f2	Ellipsoidal Function
f3	Rastrigin Function
f4	Büche-Rastrigin Function
f5	Linear Slope
2 Functions with low or moderate conditioning	
f6	Attractive Sector Function
f7	Step Ellipsoidal Function
f8	Rosenbrock Function, original
f9	Rosenbrock Function, rotated
3 Functions with high conditioning and unimodal	
f10	Ellipsoidal Function
f11	Discus Function
f12	Bent Cigar Function
f13	Sharp Ridge Function
f14	Different Powers Function

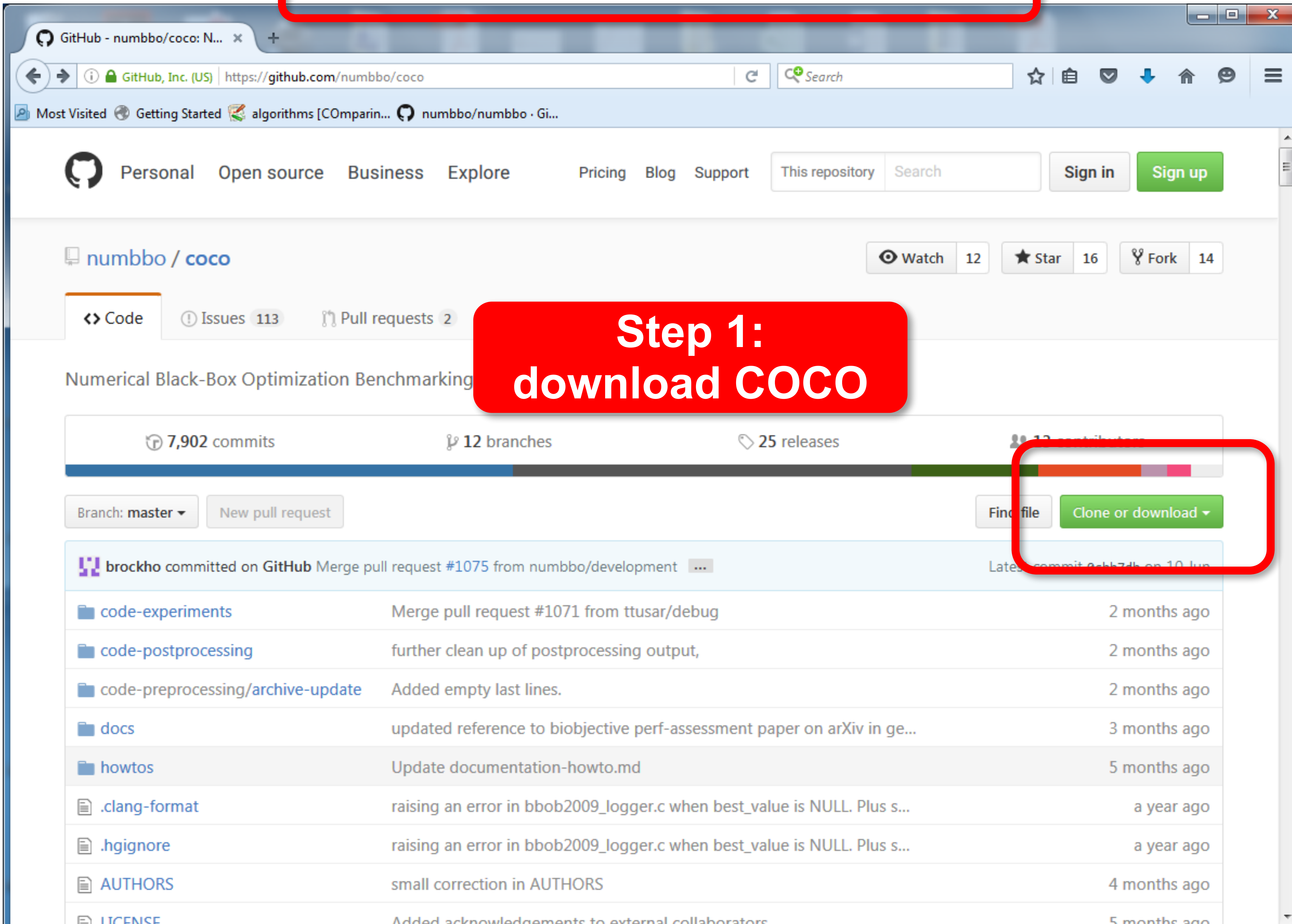
4 Multi-modal functions with adequate global structure	
f15	Rastrigin Function
f16	Weierstrass Function
f17	Schaffers F7 Function
f18	Schaffers F7 Functions, moderately ill-conditioned
f19	Composite Griewank-Rosenbrock Function F8F2
5 Multi-modal functions with weak global structure	
f20	Schwefel Function
f21	Gallagher's Gaussian 101-me Peaks Function
f22	Gallagher's Gaussian 21-hi Peaks Function
f23	Katsuura Function
f24	Lunacek bi-Rastrigin Function

Consider Questions to be Answered

- what is the performance on a specific (class of) problem(s)?
- how does the algorithm scale with dimension?
- how does the algorithm perform on
    - ill-conditioned problems
    - multimodal problems
- does the algorithm exploit separability?
- …

COCO platform: automatizing the benchmarking process

https://github.com/numbbo/coco



https://github.com/numbbo/coco

GitHub - numbbo/coco: N...

GitHub, Inc. (US) https://github.com/numbbo/coco

Most Visited · Getting Started · algorithms [COmparin... · numbbo/numbbo · Gi...

corresponds to the master branch as linked above.

3. In a system shell, **`cd` into** the `coco` or `coco-<version>` folder (framework root), where the file `do.py` can be found. Type, i.e. **execute**, one of the following commands once

```
python do.py run-c
python do.py run-java
python do.py run-matlab
python do.py run-octave
python do.py run-python
```

depending on which language shall be used to run the experiments. `run-*` will build the respective code and run the example experiment once. The build result and the example experiment code can be found under `code-experiments/build/<language>` (`<language>=matlab` for Octave). `python do.py` lists all available commands.

4. On the computer where experiment data shall be post-processed, run

```
python do.py install-postprocessing
```

Step 2: installation of post-processing

to (user-locally) install the post-processing. From ... the builds to a new release.

5. **Copy** the folder `code-experiments/build/YOUR-FAVORITE-LANGUAGE` and its content to another location. In Python it is sufficient to copy the file `example_experiment.py`. Run the example experiment (it already is compiled, in case). As the details vary, see the respective read-me's and/or example experiment files:

   - `C` read me and example experiment
   - `Java` read me and example experiment
   - `Matlab/Octave` read me and example experiment

http://coco.gforge.inria.fr/doku.php?id=algorithms

Step 3:
downloading data

[[algorithms]]

COMPARING CONTINUOUS OPTIMISERS: COCO

Show pagesource | Old revisions | Recent changes | Sitemap | Login

Search

The following table lists all algorithms related to the BBOB workshops and special sessions in the years 2009 till 2015 together with links to their data. In order to sort the table according to some columns, please click on the corresponding table header. If available, the source codes of the algorithms can be downloaded by clicking on the link with the corresponding algorithm name in the second column.

No	Algorithm	Year	Author(s)	Data Noiseless (Raw)	Data Noisy (Raw)	related PDFs and Remarks
1	ALPS	2009	Hornby	noiselessData	noisyData	PDF
2	AMALGAM	2009	Bosman et al.	noiselessData	noisyData	PDFnoiseless PDFnoisy
3	BAYEDA	2009	Gallagher	noiselessData	noisyData	PDFnoiseless PDFnoisy
4	BFGS	2009	Ros	noiselessData	noisyData	PDFnoiseless PDF
5	BIPOP-CMA-ES	2009	Hansen	noiselessData	noisyData	PDFnoiseless PDF
6	Cauchy-EDA	2009	Pošík	noiselessData	n/a	PDF
7	CMA-ESPLUSSEL	2009	Auger and Hansen	noiselessData	noisyData	PDFnoiseless PDF
8	DASA	2009	Korošec and Šilc	noiselessData	noisyData	PDFnoiseless PDFnoisy
9	DE-PSO	2009	García-Nieto et al.	noiselessData	noisyData	PDFnoiseless PDFnoisy
10	DIRECT	2009	Pošík	noiselessData	n/a	PDF algorithm is deterministic and thus, only run on each instance once
11	EDA-PSO	2009	El-Abd and Kamel	noiselessData	noisyData	PDF

Navigation

- Home
- Documentation
- download latest old code
- new code homepage
- download new code directly
- BBOB 2016
- BBOB 2015 @ GECCO
  - Algorithms
  - Downloads
- BBOB 2013
  - Algorithms
  - Results
  - Schedule
  - Downloads
- BBOB 2012
  - Algorithms
  - Results
  - Downloads
- BBOB 2010

for the moment:
IPOP-CMA-ES

https://github.com/numbbo/coco

6. Now you can **run** your favorite algorithm on the `bbob-biobj` (for multi-objective algorithms) or on the `bbob` suite (for single-objective algorithms). Output is automatically generated in the specified data `result_folder`.

7. **Postprocess** the data from the results folder by typing

```
python -m bbob_pproc [-o OUTPUT_FOLDERNAME] YOURDATAFOLDER [MORE_DATAFOLDERS]
```

postprocess

The name `bbob_pproc` will become `cocopp` in future. Any subfolder in the folder argu data. That is, experiments from different batches can be in different folders collected u `YOURDATAFOLDER` folder. We can also compare more than one algorithm by specifying several data result folders generated by different algorithms.

A f

pp

us

python -m bbob_pproc IPOP-CMA-ES

Fo

form

`bbob-biobj` suite will follow in a later release. A basic html output is also available in the result folder of the postprocessing (file `templateBBOBarticle.html`).

8. Once your algorithm runs well, **increase the budget** in your experiment script, if necessary implement randomized independent restarts, and follow the above steps successively until you are happy.

If you detect bugs or other issues, please let us know by opening an issue in our issue tracker at https://github.com/numbbo/coco/issues.

Description by Folder