



Inria



A (Biased) Introduction to Benchmarking

Anne Auger and Nikolaus Hansen
Inria and CMAP, Ecole Polytechnique, IP Paris

Full set of slides: <http://www.cmap.polytechnique.fr/~nikolaus.hansen/gecco-2022-benchmarking-tutorial.pdf>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

GECCO '22 Companion, Boston, USA

© 2022 Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM 978-1-4503-9268-6/22/07...\$15.00 <https://doi.org/10.1145/3520304.3533649>

...feel free to ask questions...

...feel free to ask questions...

Benchmarking: What Are We Talking About?

Optimization algorithm / Solver

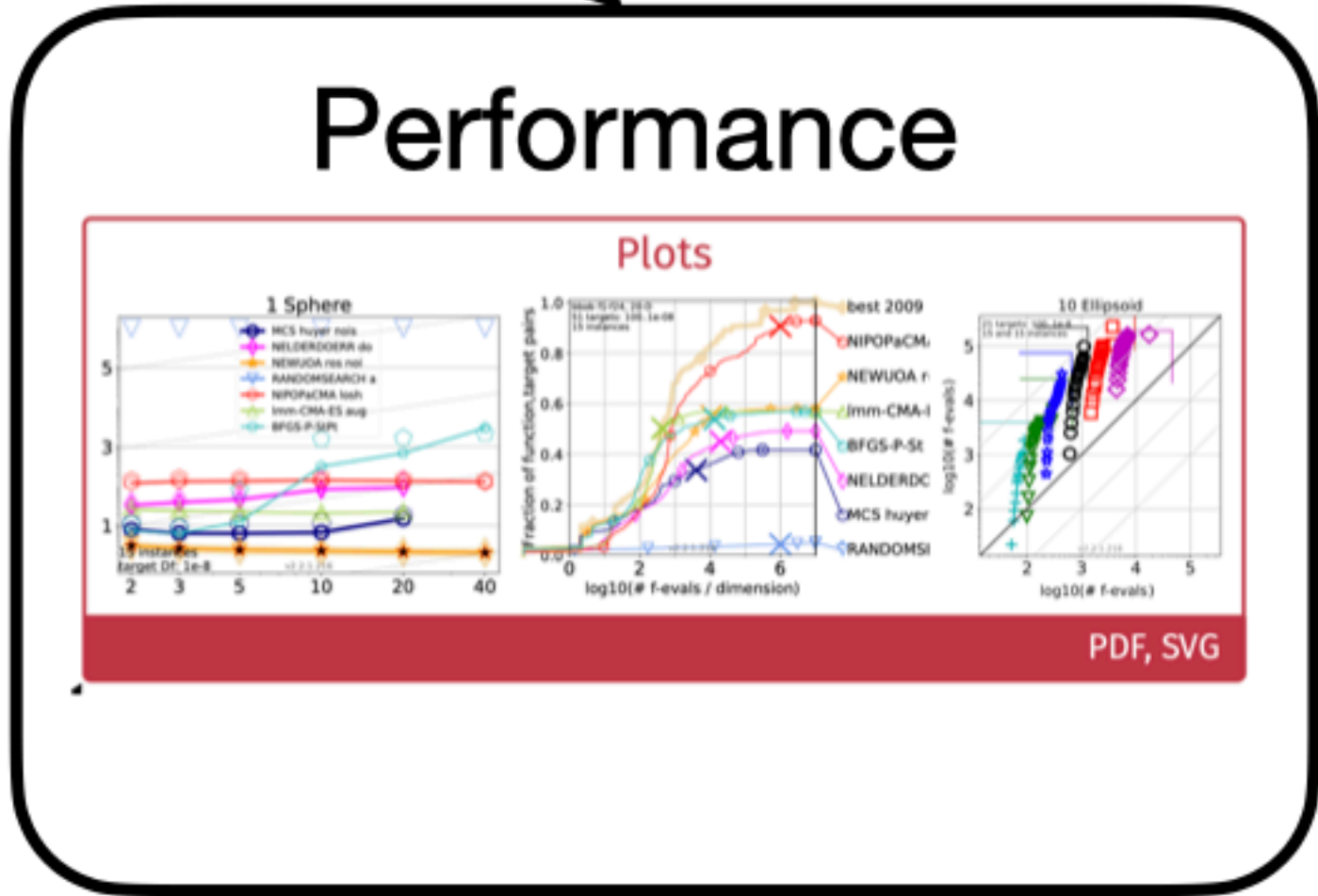
Experiments

run it on test functions

$$f(x) = \sum_{i=1}^n 10^{\frac{i-1}{n-1}} x_i^2$$
$$f(x) = 10 \left(n - \sum_{i=1}^n \cos(2\pi(x_i)) \right) + \|x\|^2$$

...

collect data



How good is it ???

Interesting! My algorithm is 10 times faster than CMA-ES on separable functions but 1000 slower on non-separable ones for dimensions between 5 and 40 !



Getting Random Things Out of the Way: **Generalization**

Does benchmarking make sense at all?

After all there is no free lunch, right? Or is there?

- A benchmark should attempt to **model observable and relevant** “real-world” optimization **problems**

*The set of all observable and relevant optimization problems is WAY smaller than most sets of **mathematically constructible** problems.*

*NFL theorems hold on sets of functions that are “closed under permutation”.
Whether all functions in such set are (equally often) observed in reality is **an empirical question**.
Practical evidence suggests: some algorithms are vastly worse than others.*

- The function or instance ID can not be input to the algorithm

*We shall not set algorithm parameters depending on each function!
AKA overfitting.*

*The benchmarking setup: an algorithm that needs to repeatedly solve “new” problems.
Crafting Effort correction for using different parameter settings on different functions¹.*

- **Invariance** of algorithms is a relevant aspect to interpret (generalizability of) benchmarking results

- Comparable data

*depends on the benchmarking setup
across publications
across functions (e.g. speedup factor)*

¹: Price KV. Differential evolution vs. the functions of the 2nd ICEO. In Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97) 1997 (pp. 153-157). IEEE.

Getting Random Things Out of the Way: **Why is Benchmarking so Important?**

“In the course of your work, you will from time to time encounter the situation where the facts and the theory do not coincide. In such circumstances [...], it is my earnest advice to respect the facts.”

— Igor Sikorsky

“If it disagrees with experiment, it's wrong. And that simple statement is the key to science. [...] That's all there is to it.”

— Richard P. Feynman
<https://youtu.be/b240PGCMwV0>

Getting Random Things Out of the Way: **What about Competitions?**

“The emphasis on competition is fundamentally anti-intellectual and does not build the sort of insight that in the long run is conducive to more effective algorithms”.

Hooker (1995) Testing Heuristics: We Have it All Wrong.

Getting Random Things Out of the Way: Trivia

- A trivial (serial) algorithm portfolio: K algorithms can solve each and every problem as fast as the fastest of these algorithms multiplied by K .
Run in parallel, they become as fast as the fastest algorithm
- What differences are we interested in?
2%, 20%, 200%, 2000%,...
- Function/problem *instances*
versus different functions
- Search domain: discrete and continuous
Examples come from the continuous domain.

(Specific) Goals of Benchmarking

We may define benchmarking as **measuring algorithm performance in a systematic and standardized way**

thereby creating a performance “profile” of an algorithm for a standardized assessment and for simplified comparison

Specific goals can be:

1. Comparing against the “state-of-the-art” or against a baseline

any comparison between two or more algorithms

2. Understanding algorithms

dedicated experimentation is often a better alternative

3. Selecting algorithms to solve a given problem

4. Regression testing after changes of an algorithm or an implementation

5. Running a competition

a competition setup needs to hide information from the competitor/experimenter

“Everybody” has to do it and it is tedious: choosing (and implementing) problems, performance measures, visualization, statistical tests, ...

which suggests to consider using tools

Benchmarking How To: The Global Picture

Two *surprisingly* (but not completely) *independent* components:

- **Which benchmark, which suite of functions/problems** do we run the algorithms on?
For example and in particular, which collection of test problems?
- **How to assess performance?**
 - experimental setup
 - data collection
 - measures used and presented

COCO/BBOB: The Global Picture

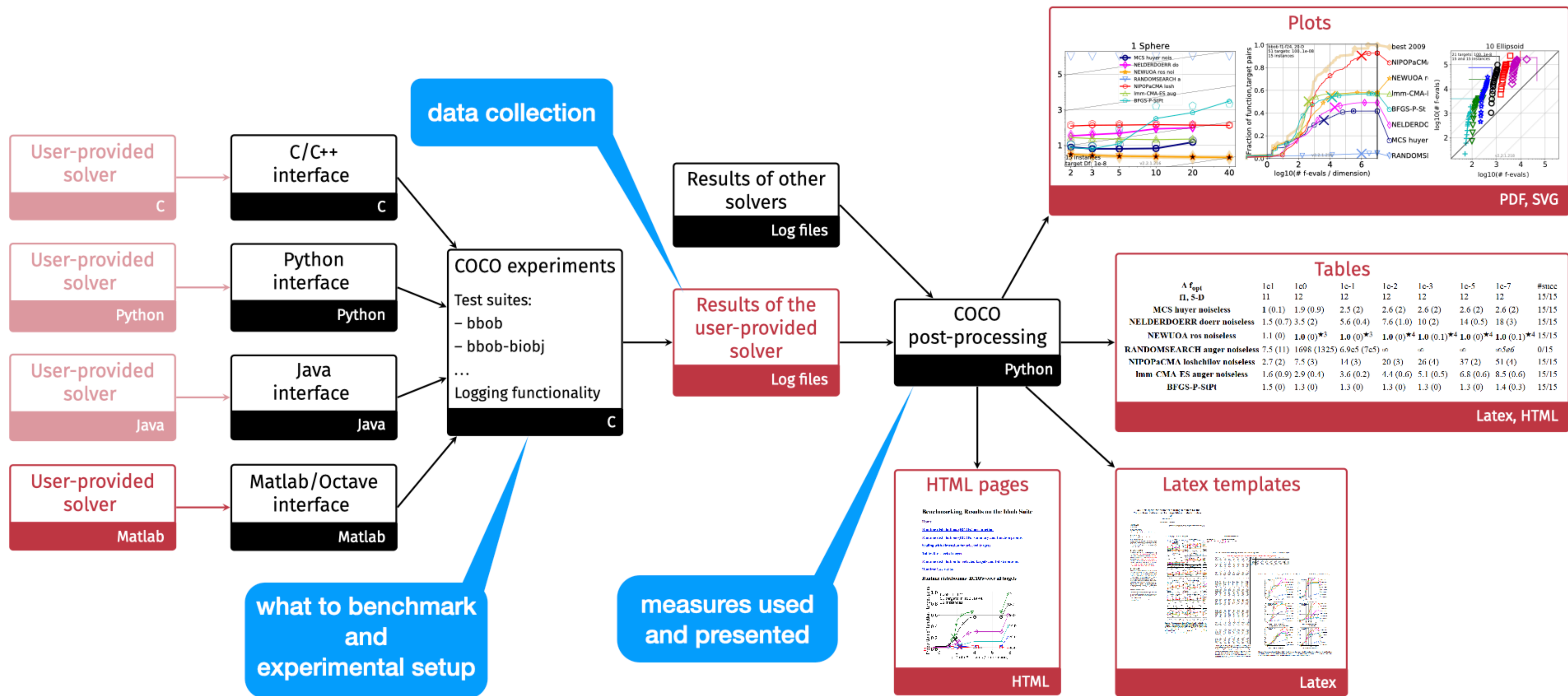


Figure by Tea Tušar in Hansen et al (2021), COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1), 114-144.

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?
- Bad idea if
 - function difficulties are **unbalanced**
too many small dimensional problems, convex problems...
 - and performance is **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-world 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?
- ...

Questions related to BBOB testbed

What is the optimal convergence rate of an algorithm?

Is separability exploited?

What is the effect of ill-conditioning?

What is the effect of asymmetry?

Can the search go outside the initial convex hull of solutions into the domain boundary?

Can the step size / population variance increase?

What is the effect of a highly asymmetric landscape?

Does the search get stuck on plateaus?

Can the search follow a long path with $D - 1$ changes in the direction?

What is the effect of rotation (non-separability)?

What is the effect of constraint-like penalization?

Can the search continuously change its search direction?

What is the effect of non-smoothness, non-differentiable ridge?

What is the effect of non-separability for a highly multimodal function?

Does ruggedness or a repetitive landscape deter the search behavior?

What is the effect of ill-conditioning?

Is the search effective without any global structure?

What is the effect of higher condition?

Can the search behavior be local on the global scale but global on a local scale?

Experimental Setup

- should allow as many **algorithm types/interfaces** as possible
bounded, unbounded, different input options, deterministic, randomized,...
- defines the **information** an algorithm is allowed to use
*search domain (and hence dimension), initial solution,
regions of “interest”, function as back-box
not: function name/ID*
- repetitions only work for randomized algorithms
- should define **what is recorded** to afterwards measure performance
- may define **a budget** (or not)
anytime vs targeted budget

Handling and Displaying Empirical Data

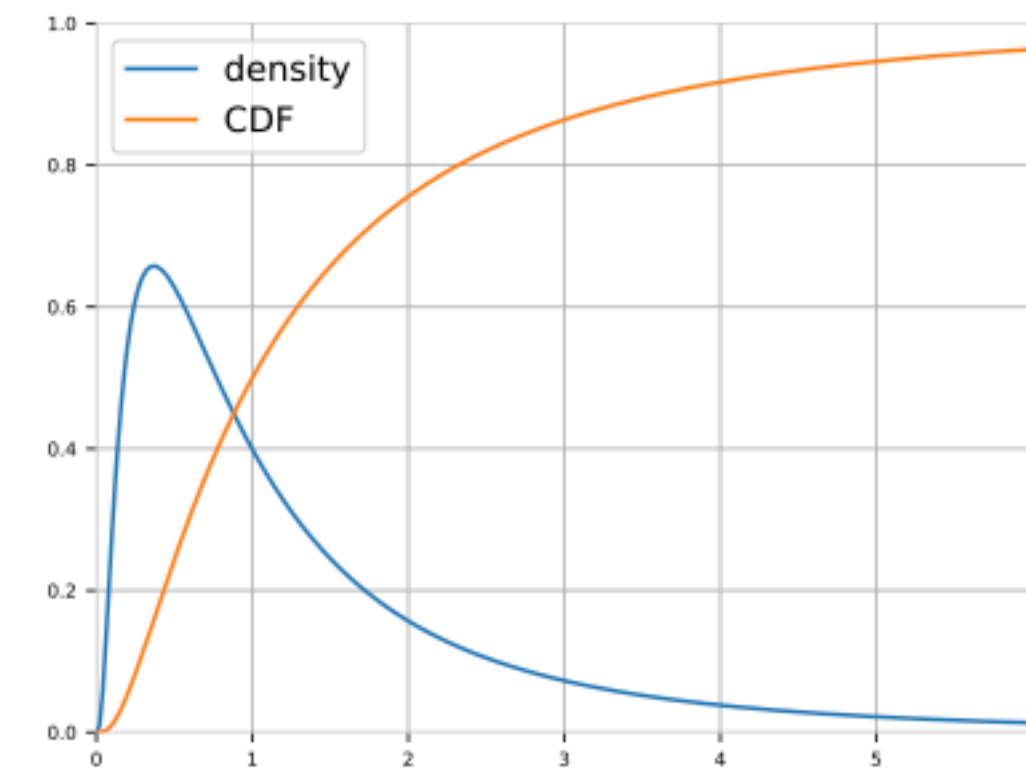
Cumulative Distribution Function (CDF)

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

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

It characterizes the probability distribution of T

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



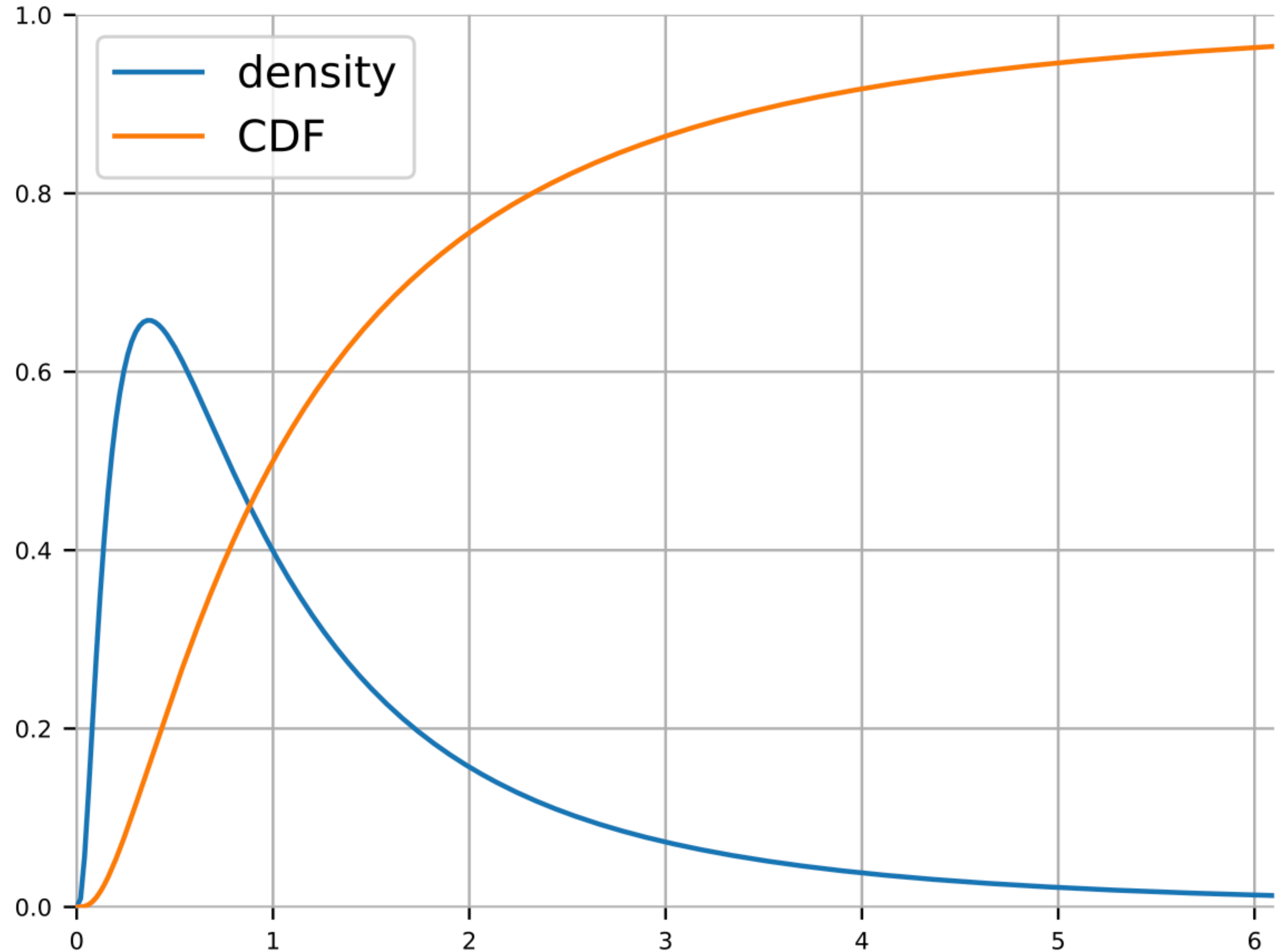
Cumulative Distribution Function (CDF)

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

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

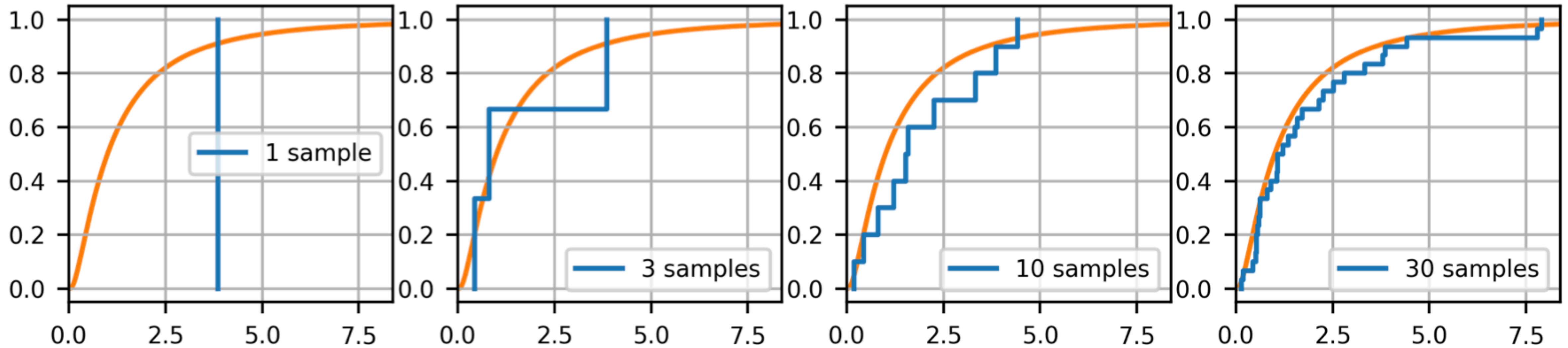
It characterizes the probability distribution of T

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



Empirical Cumulative Distribution Function

- Given a collection of data T_1, T_2, \dots, 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.



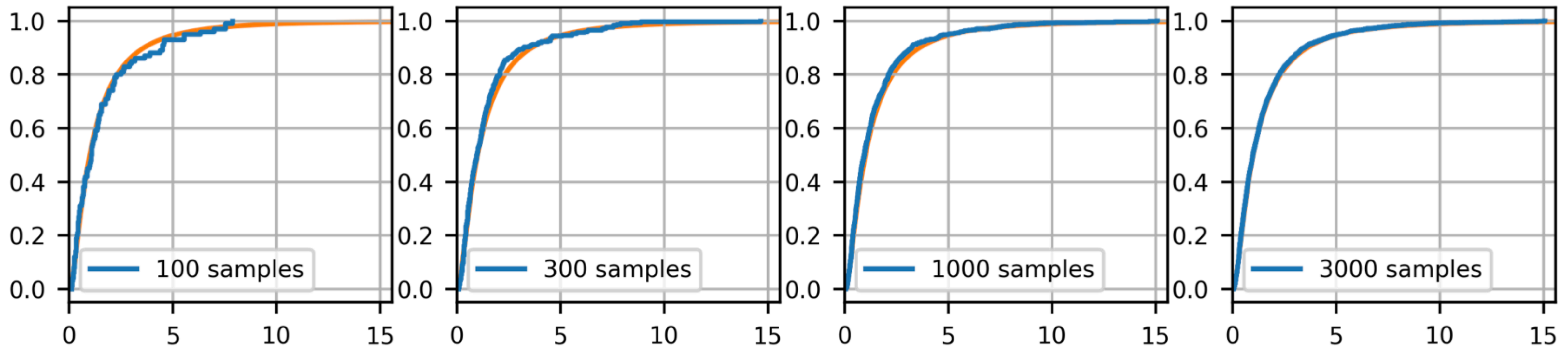
- It is an estimate of the CDF that generated the points in the sample.

Empirical Cumulative Distribution Function

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

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

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



On Performance Measure

- When comparing algorithms:

- ➔ Algorithm A is better than Algorithm B?

we want more than that

- ➔ Algorithm A is 100 times faster than Algorithm B

*We want **quantitative** statements*

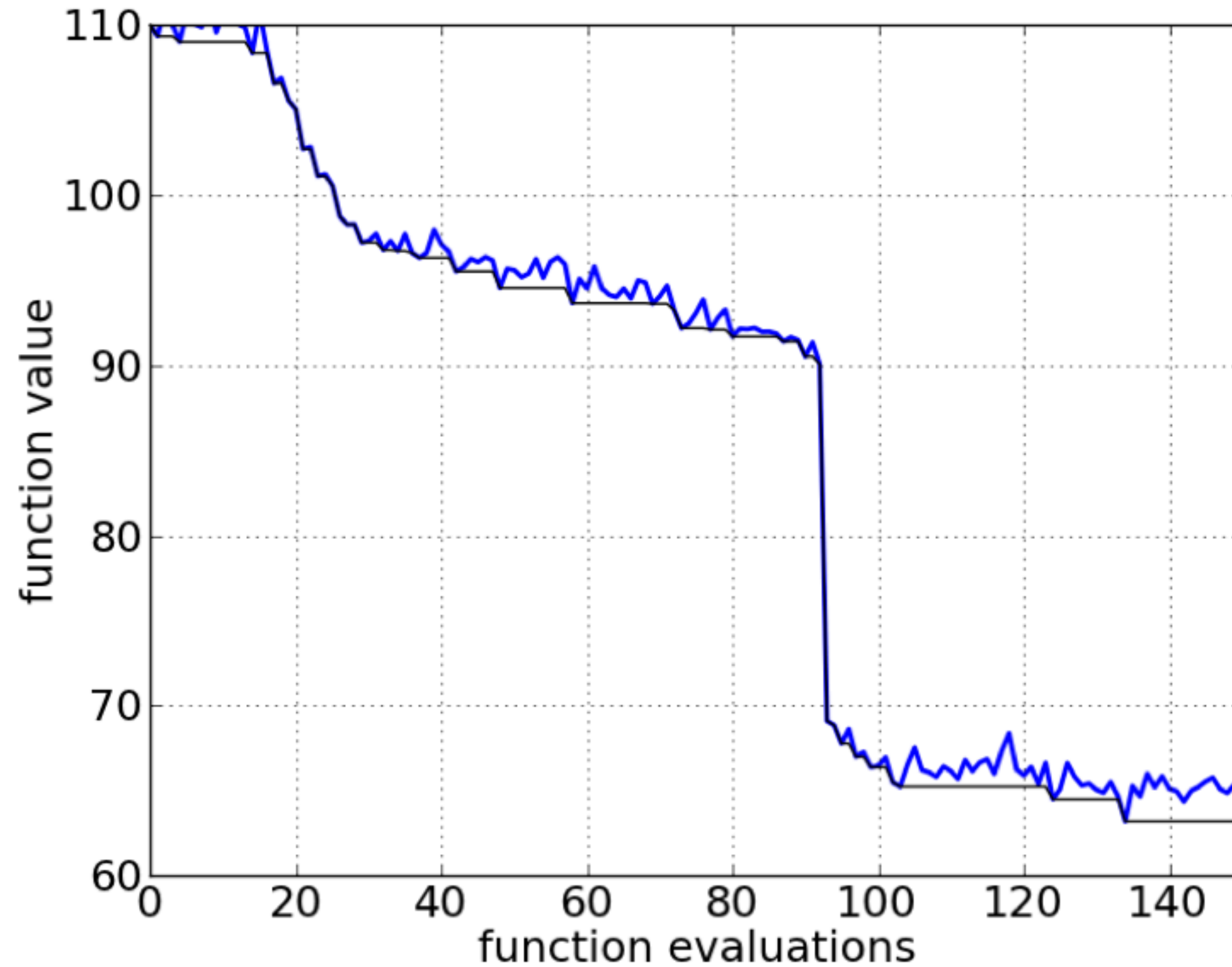
- Requires

- ➔ adequate **performance measure**

- ➔ adequate **data collection**

Collecting Empirical Data

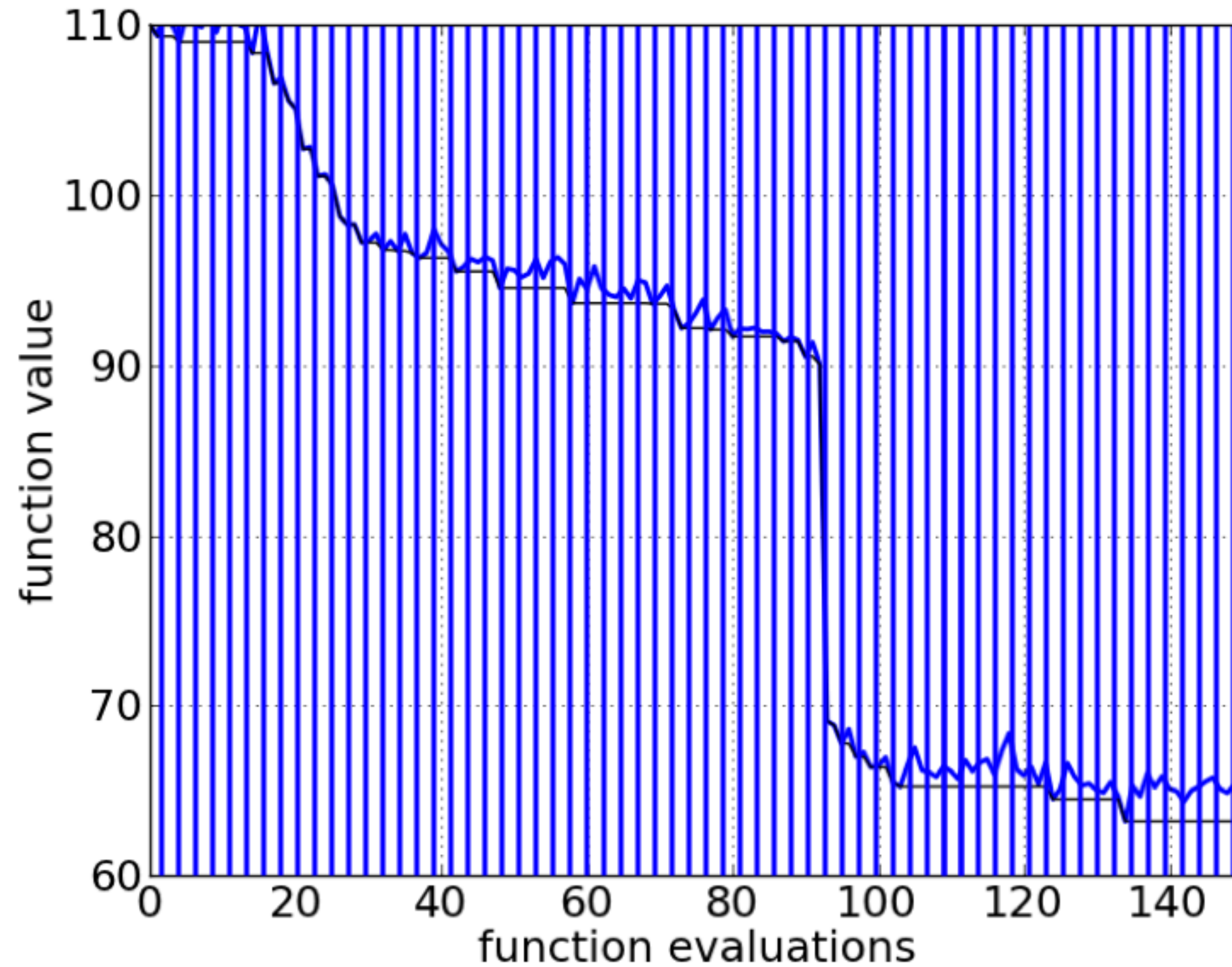
Convergence Graphs is All We Have



- a convergence graph
- lower envelope (a monotonous graph), best so-far solution

using the lower envelope is a practical choice that relates to the first hitting time

Discretization: Two Possibilities



- a convergence graph
- lower envelope (a monotonous graph), best so-far solution

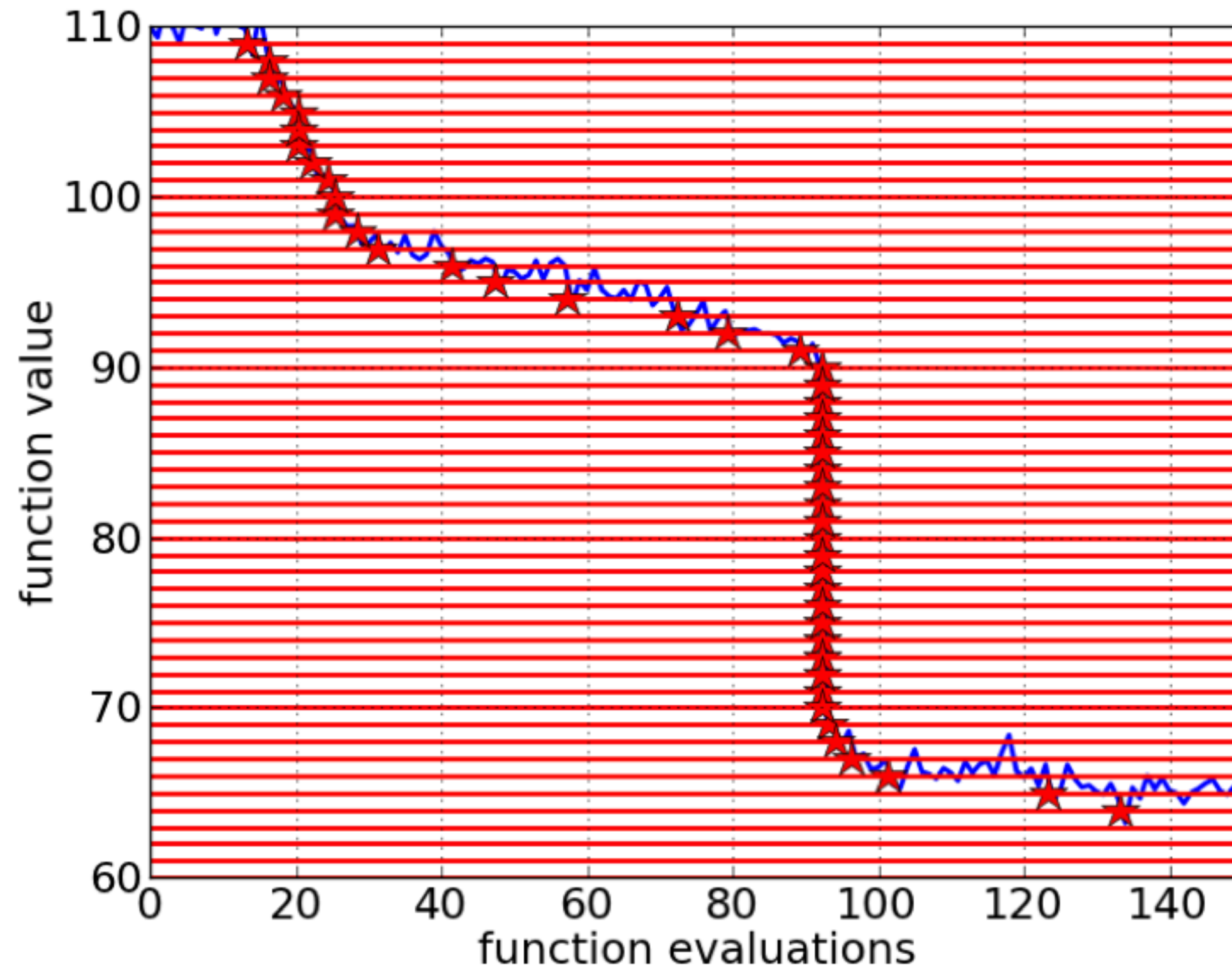
- **vertical: by evaluation is a natural discretization**

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

- **evaluations are the independent variable**

function value is the dependent variable, the measurement

Discretization: Two Possibilities



- a convergence graph
- lower envelope (a monotonous graph), best so-far solution

- **horizontal:** not a “natural” discretization

we need to determine discretization intervals

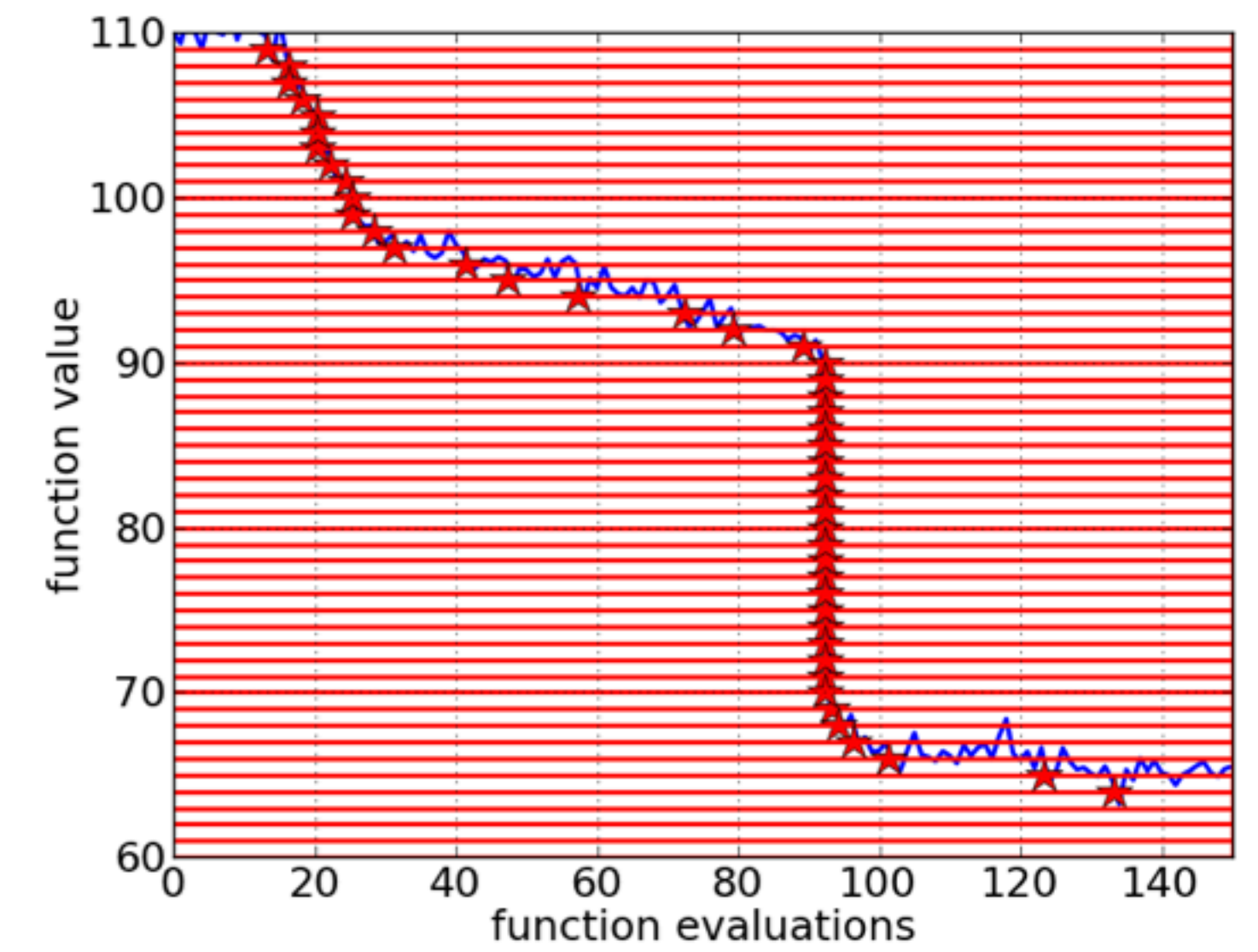
- function “target” values are the independent variable

time is the dependent variable, the measurement

- still recovers the original data

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

using the



horizontal discretization

is

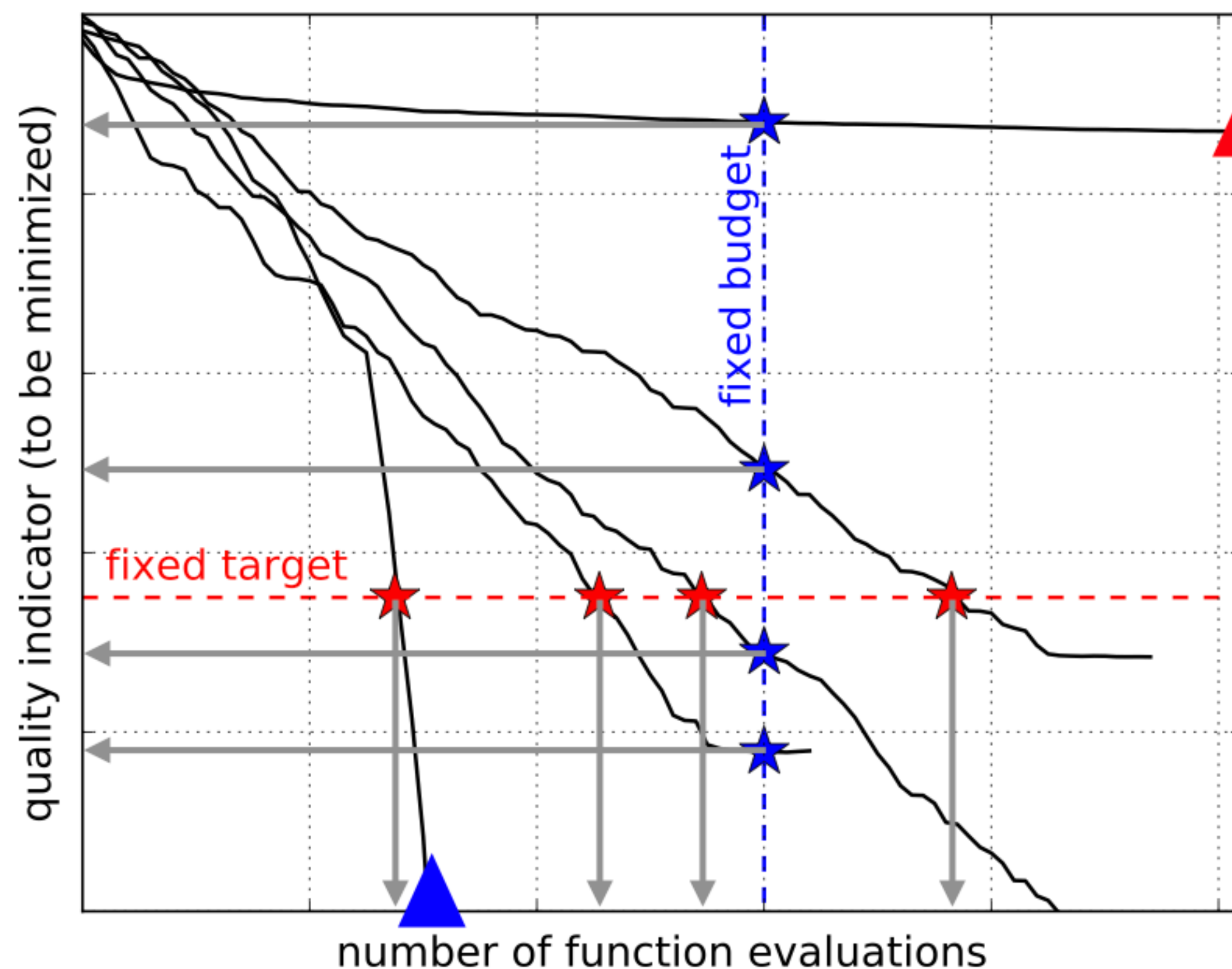
not

just

a technical subtlety

because it crucially determines the measurement we are looking at in the end

Fixed Target(s) versus Fixed Budget



- five convergence graphs
“quality indicator” versus “time”

- **Both** can lead to *imprecise data (a bound)* in some cases

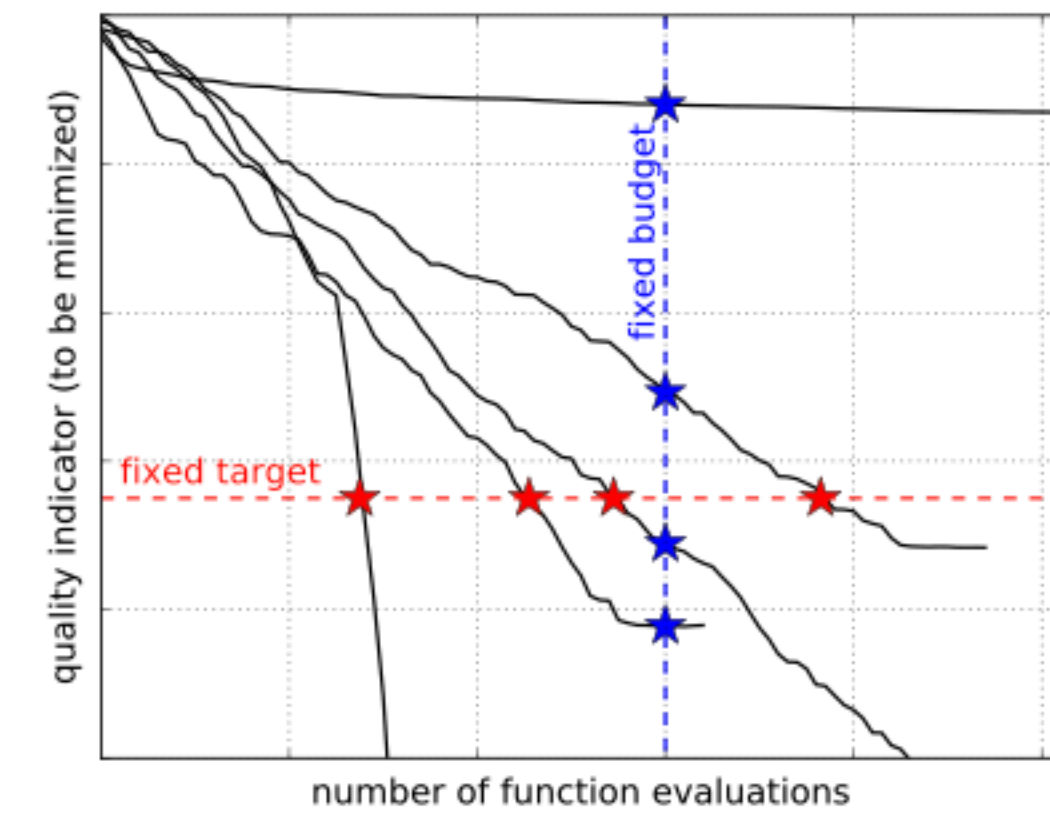
- “too” good performance

(reached global optimum up to the relevant or numerical precision before the given budget)
quick and dirty fix: assign best possible (or measured) function value

- “too” bad performance

then the data only provide a lower bound estimate for the runtime (and a fixed budget measure at the maximum budget)
quick and dirty fix: assign 10 x time_out_budget

Fixed Target(s) versus Fixed Budget



The resulting measurement

- Fixed budget (vertical, target-free) design: **function values (quality)**
- Fixed target design (budget-free) design: **evaluations (runtime)**

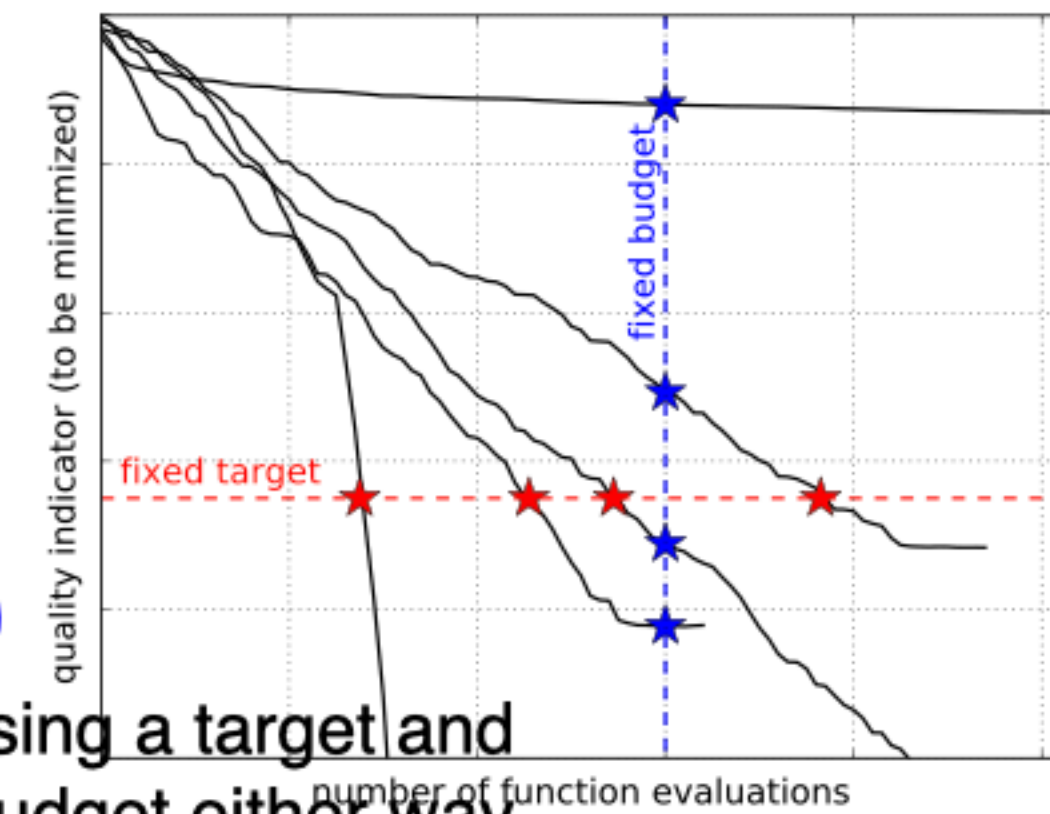
Does this make a difference?

Scales of Measurement (“Quality” of Data)

- Nominal - categorial, define a classification
- Ordinal - define an order, ranks, function *values* (fixed budget)
- Interval - differences are meaningful
- Rational - ratios are meaningful, we usually can take the logarithm, function *evaluations* (fixed target)

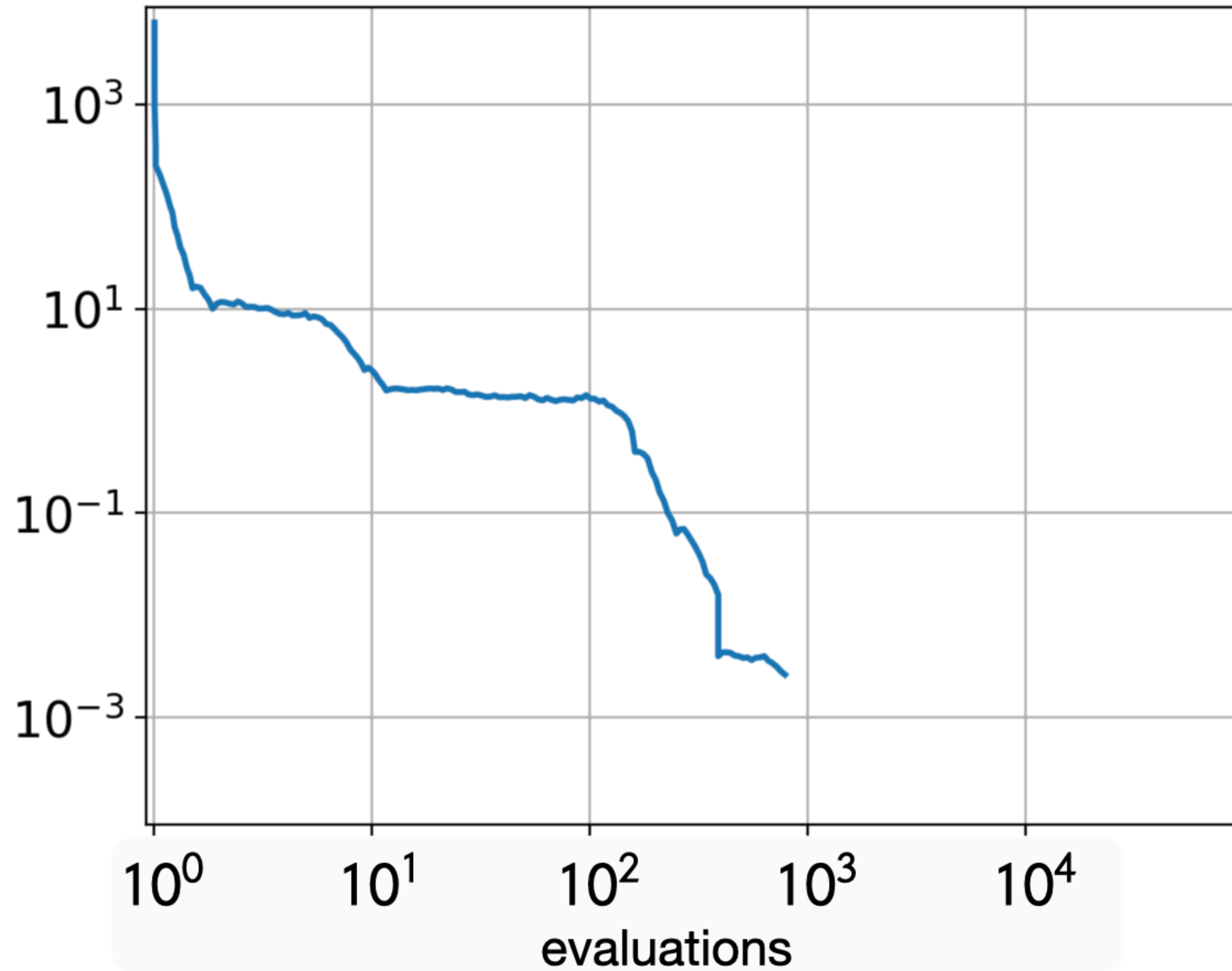
CAVEAT: mathematical and semantic treatment of data is not the same. From a classification with values $\{1, 2\}$ we can *mathematically* take differences and ratios of the values, but they have no meaningful *semantic interpretation*. Fahrenheit or Celsius versus Kelvin describe temperatures, however only Kelvin is on a rational scale of measurement.

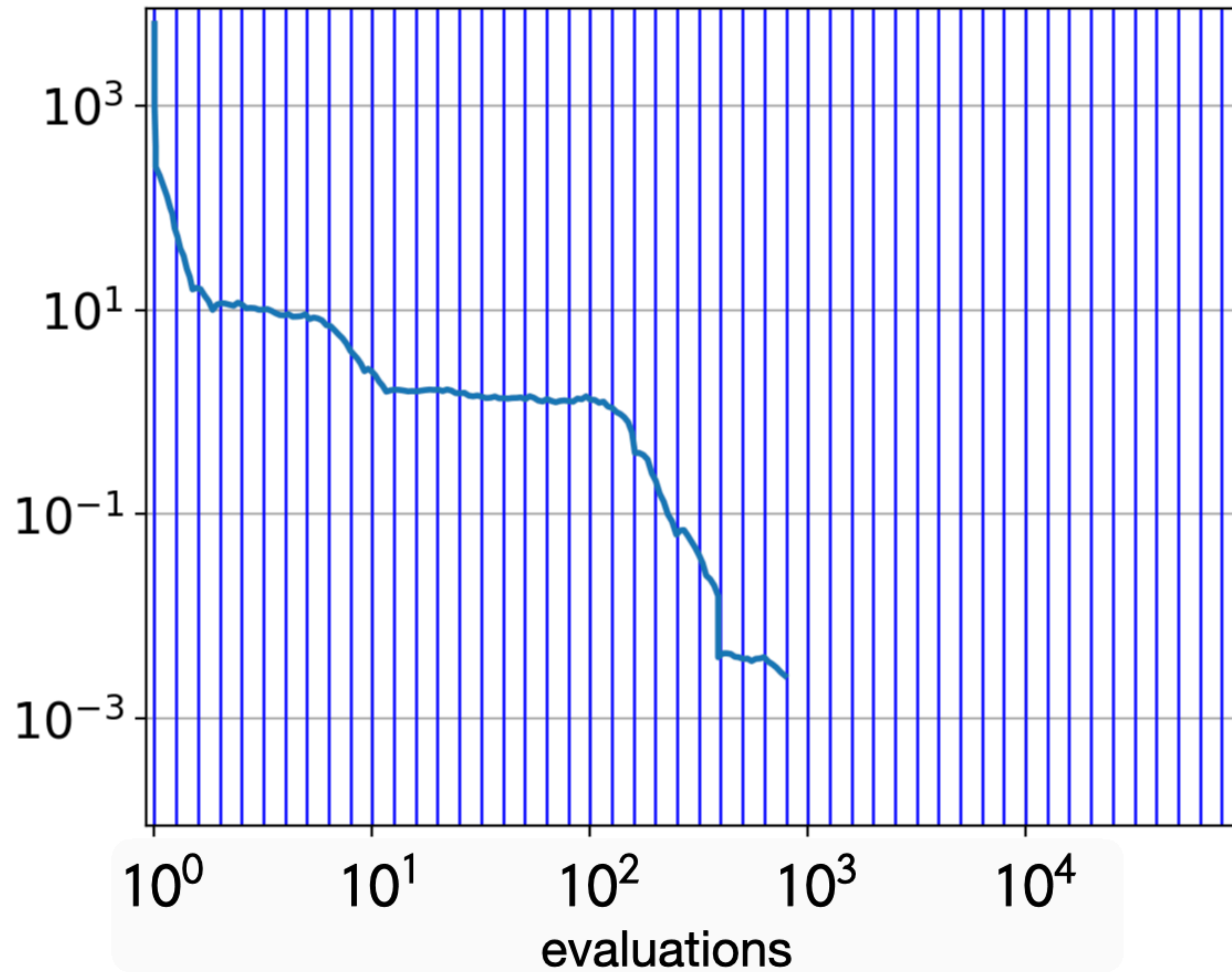
Summarizing Fixed Target(s) versus Fixed Budget

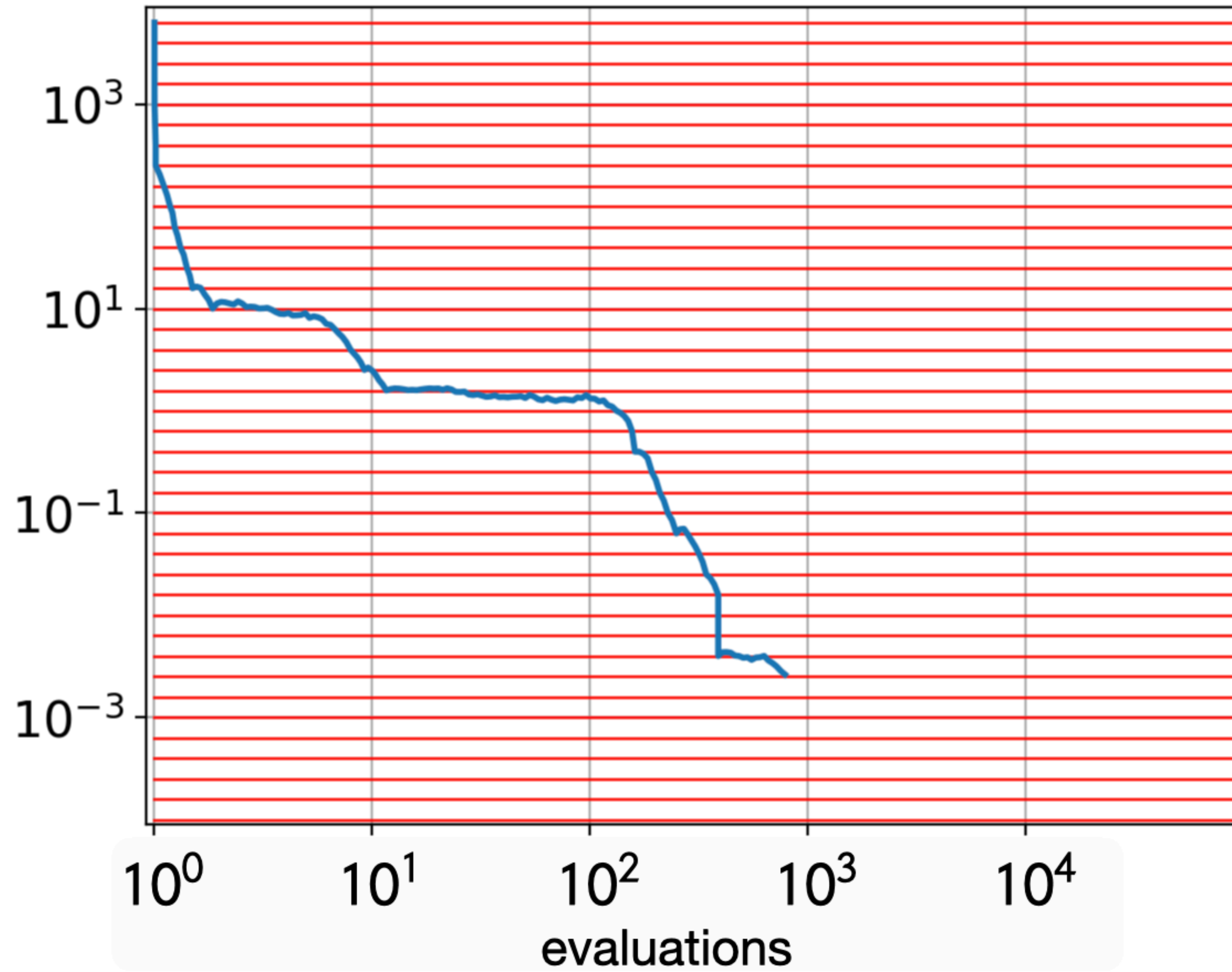


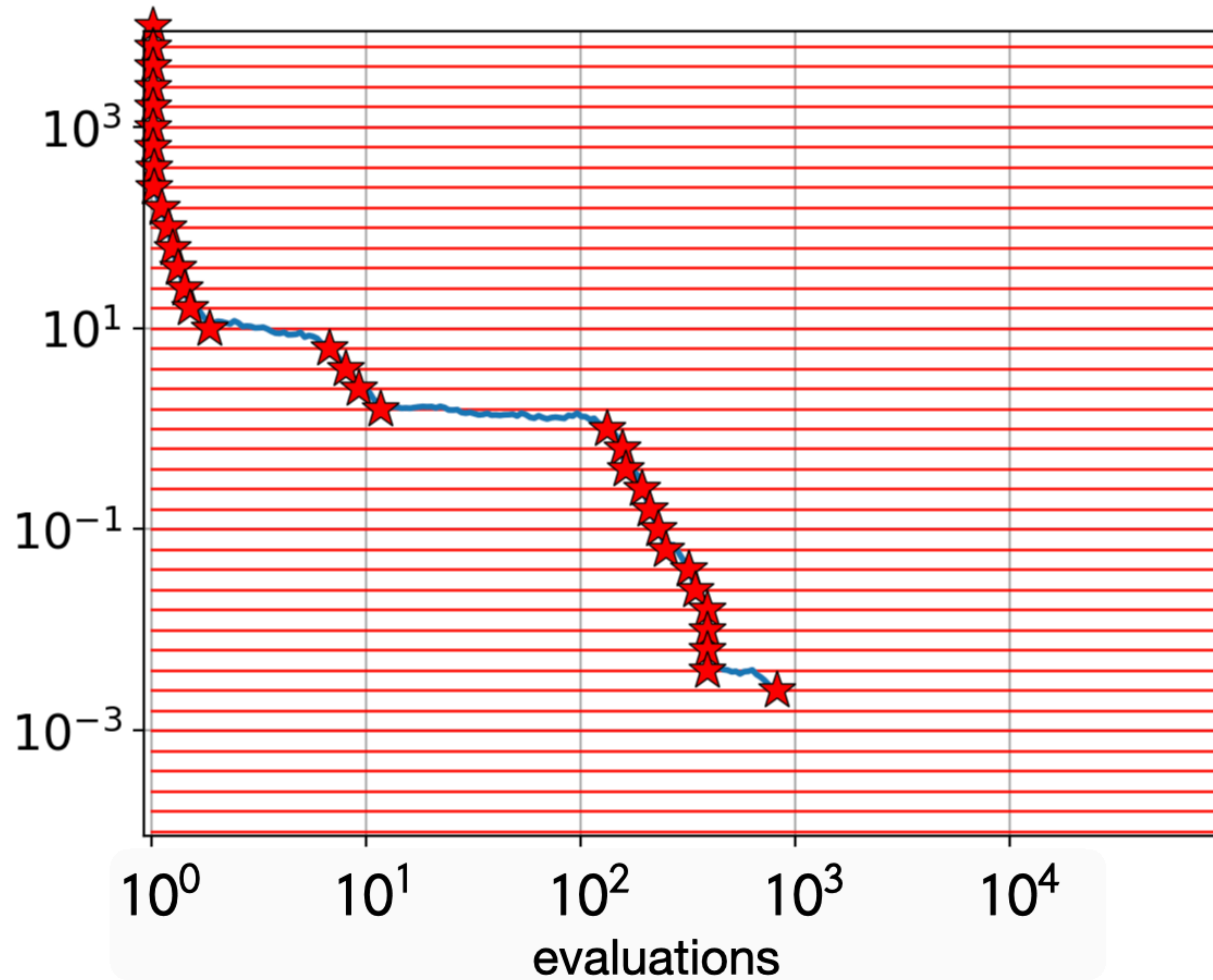
- The fixed budget (vertical) design is (much) **easier to set up**
target-free: choosing a budget is simpler than choosing a target and we need to choose a maximal “timeout” budget either way
- For the (very) same reason, results from the fixed target (horizontal) design are (much) **simpler to interpret and more conclusive**
without specific knowledge/insight, a function value is impossible to interpret beyond ordering
- Runtimes have a **quantitative interpretation**
“Algorithm A is 100 times faster than Algorithm B”
- Fixed target results can be **meaningfully aggregated** in ECDFs and geometric averages
whereas function values from different functions are in general not commensurable
- Fixed target results are **“budget-free”**
we can compare results with different maximal “timeout” budgets

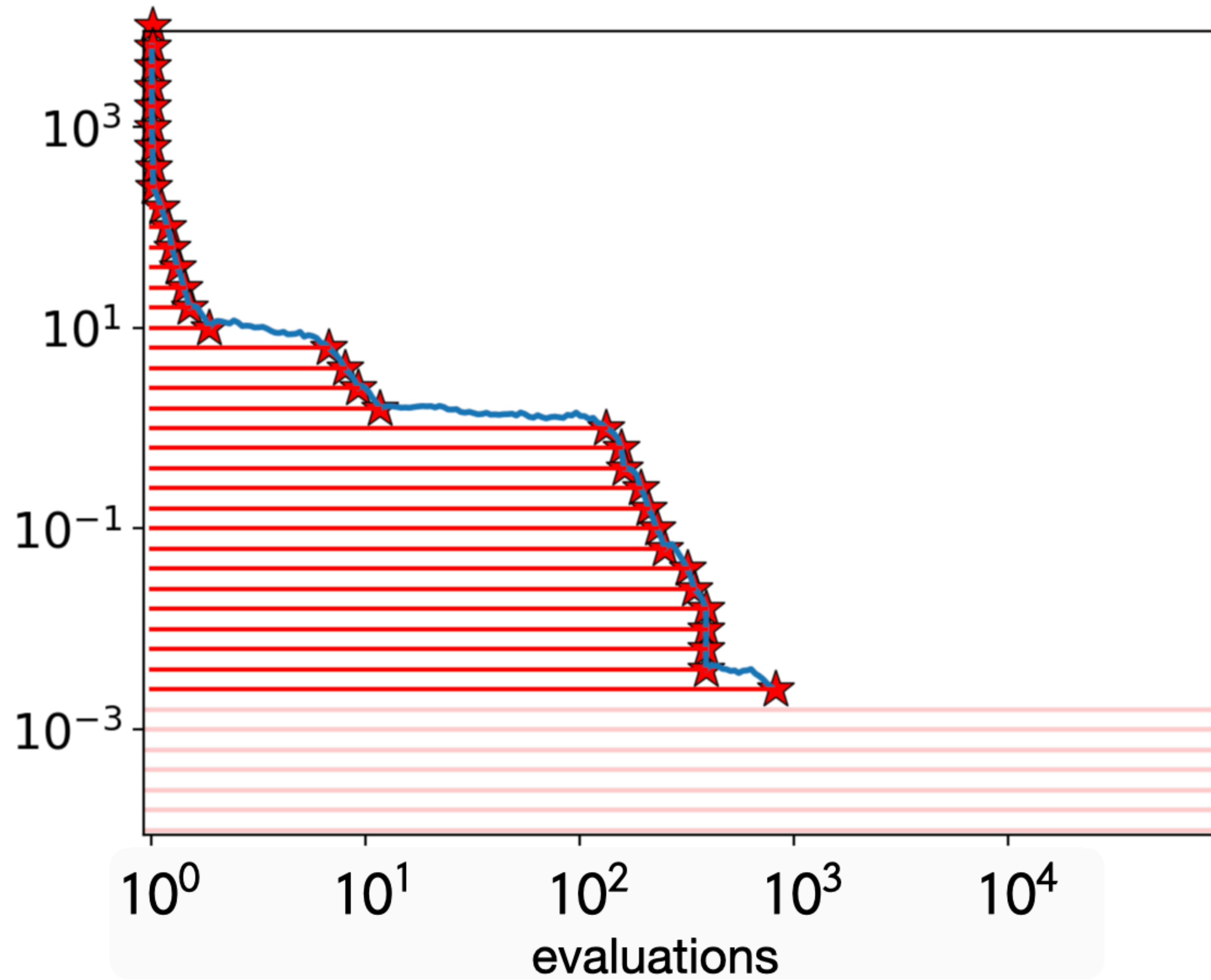
From a Convergence Graph to the Empirical Runtime Distribution

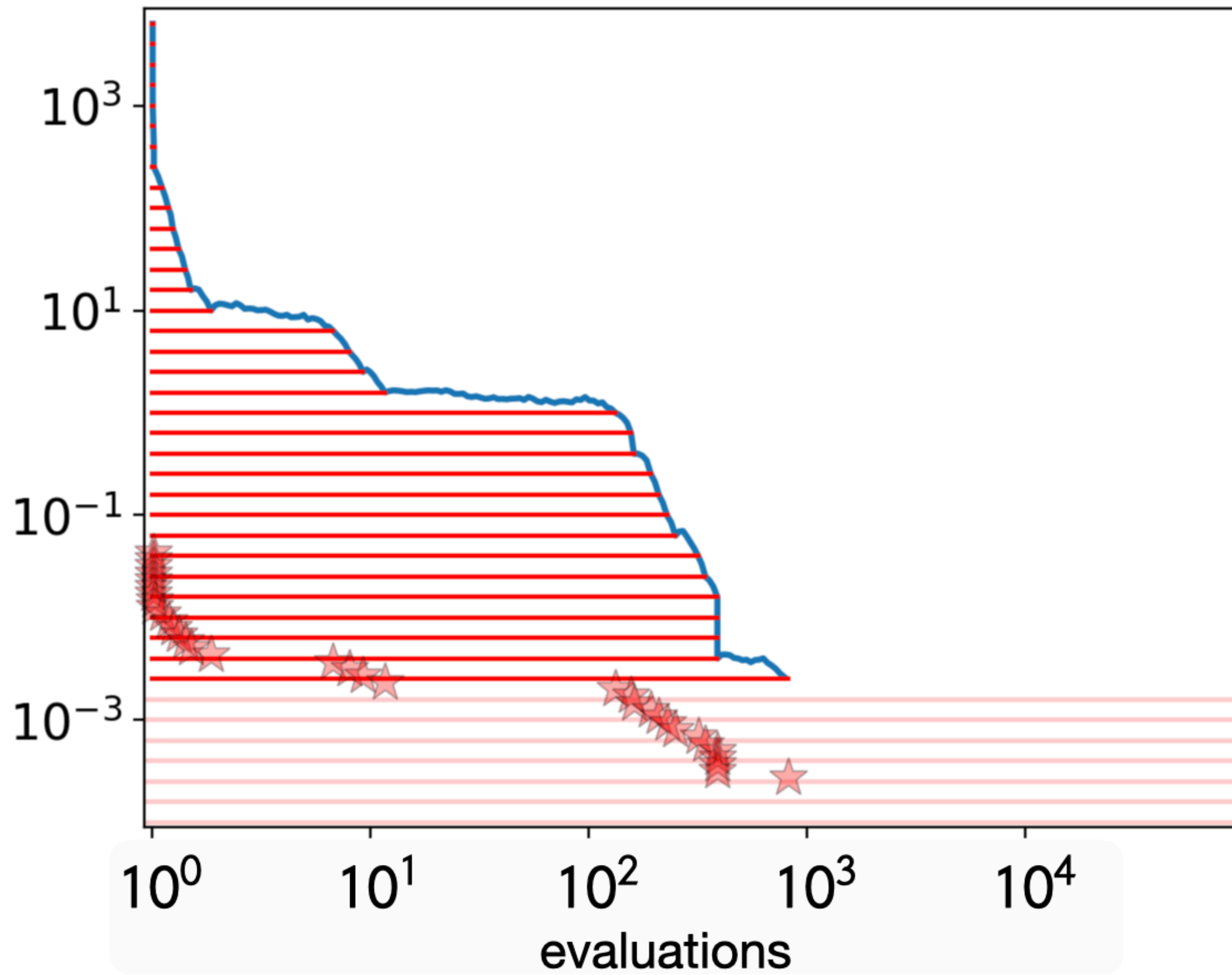


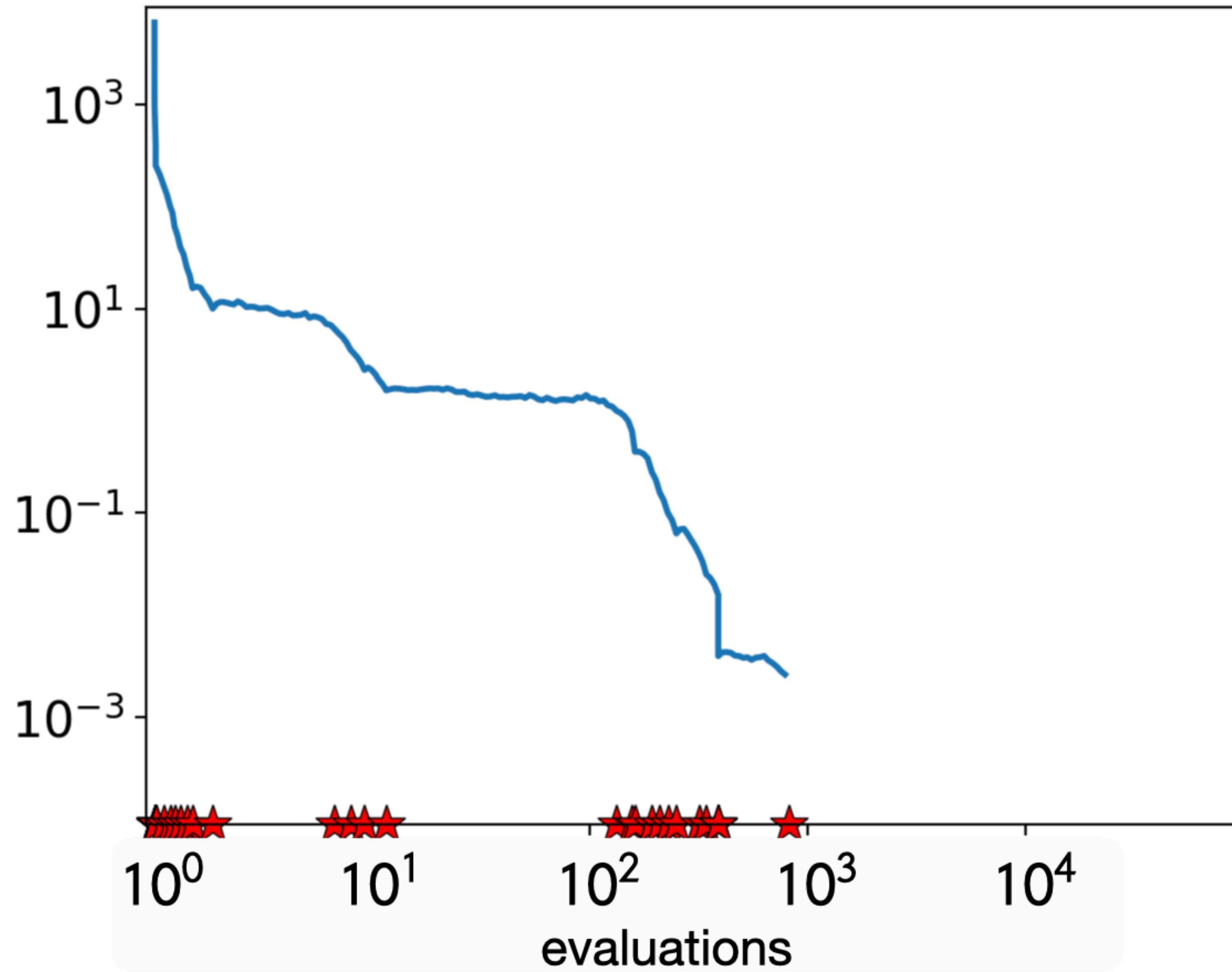


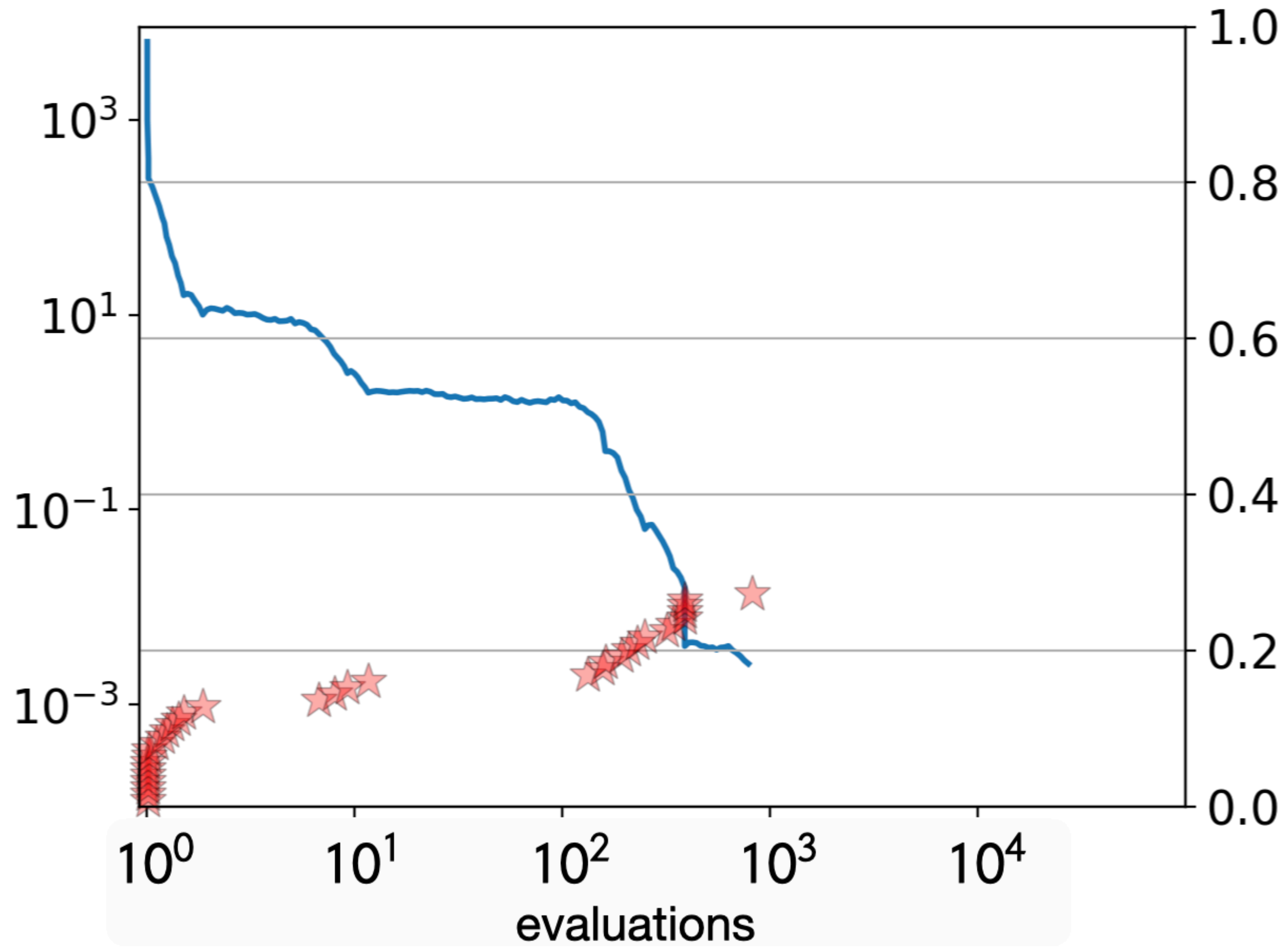


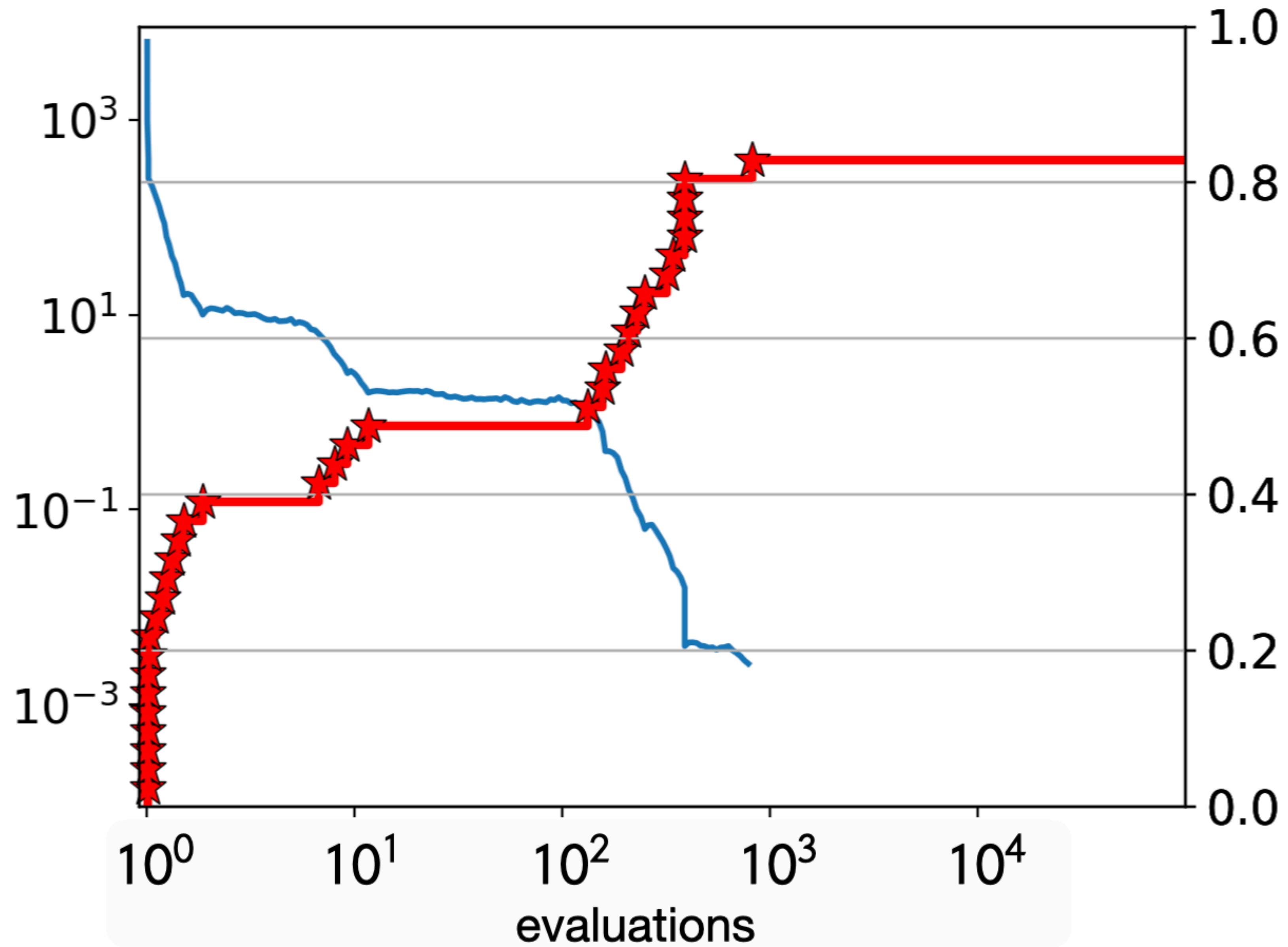


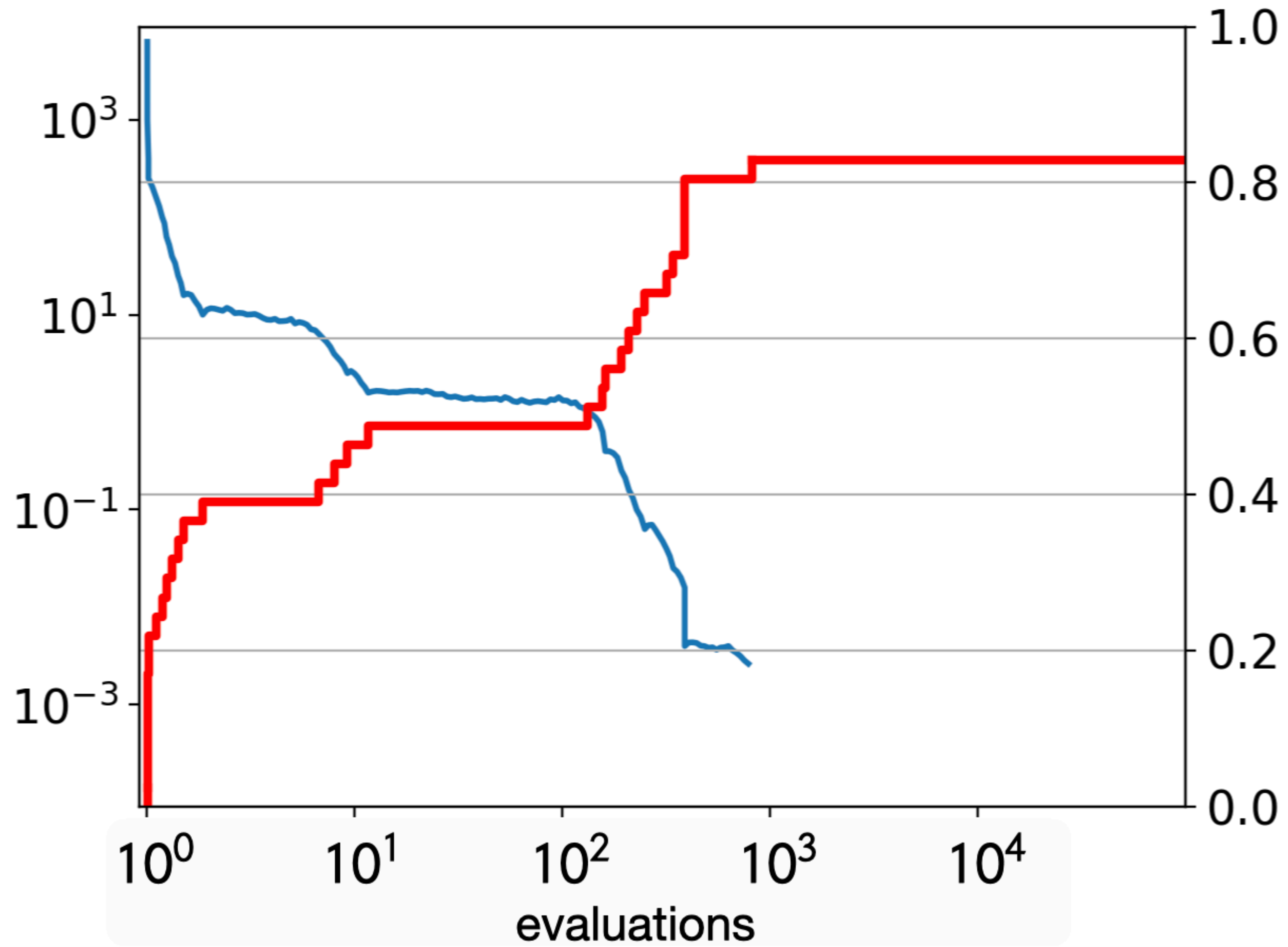


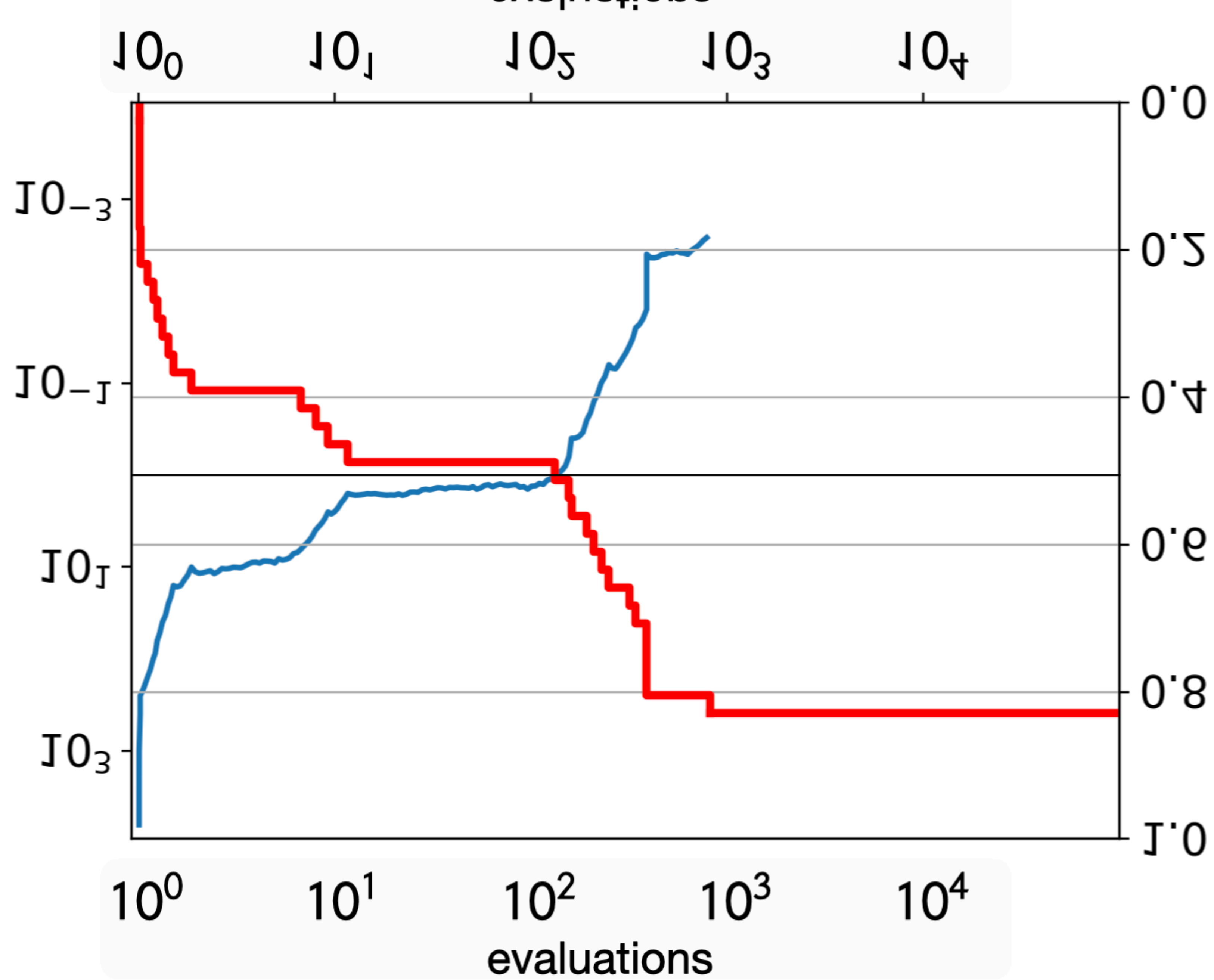


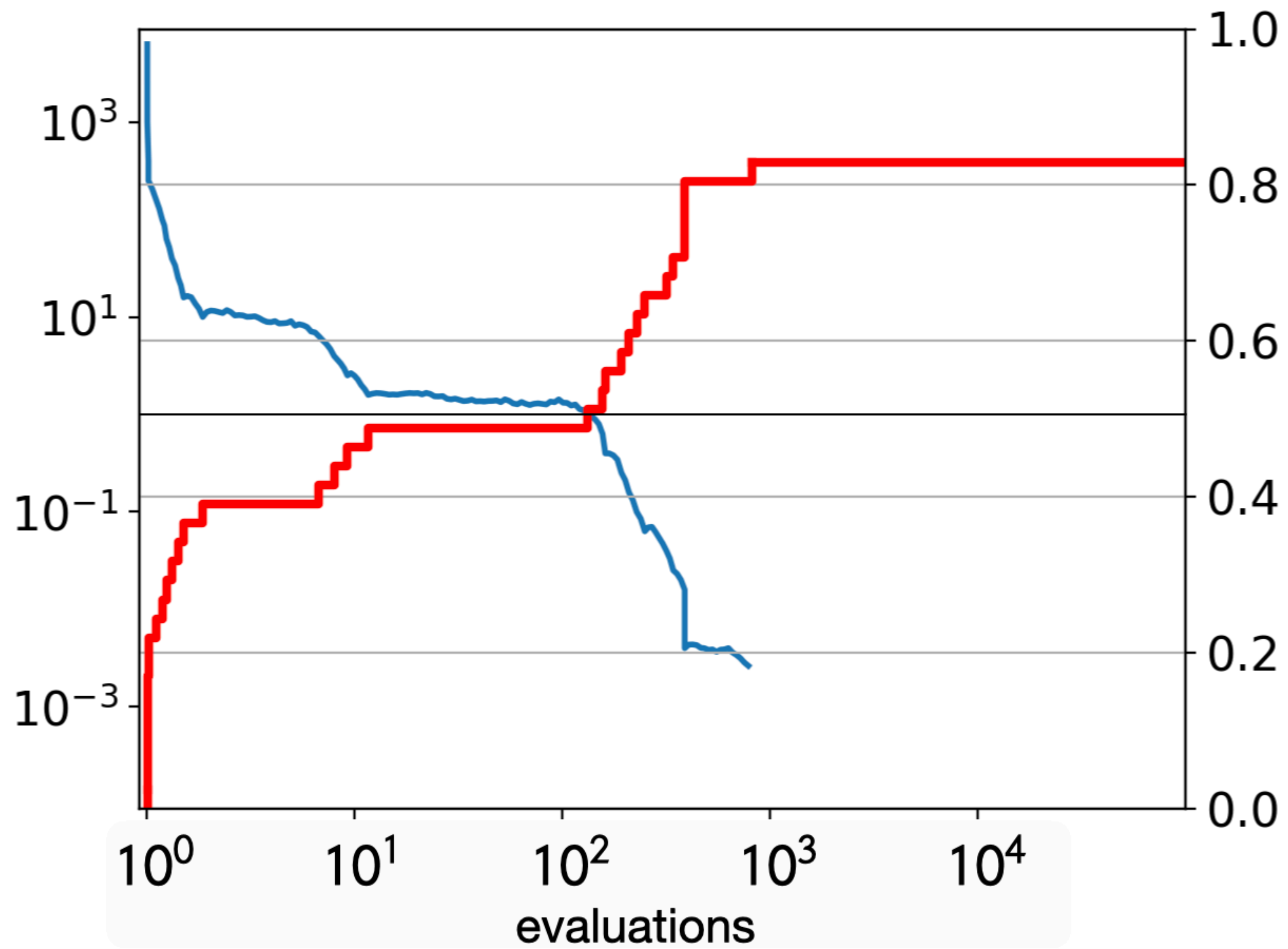


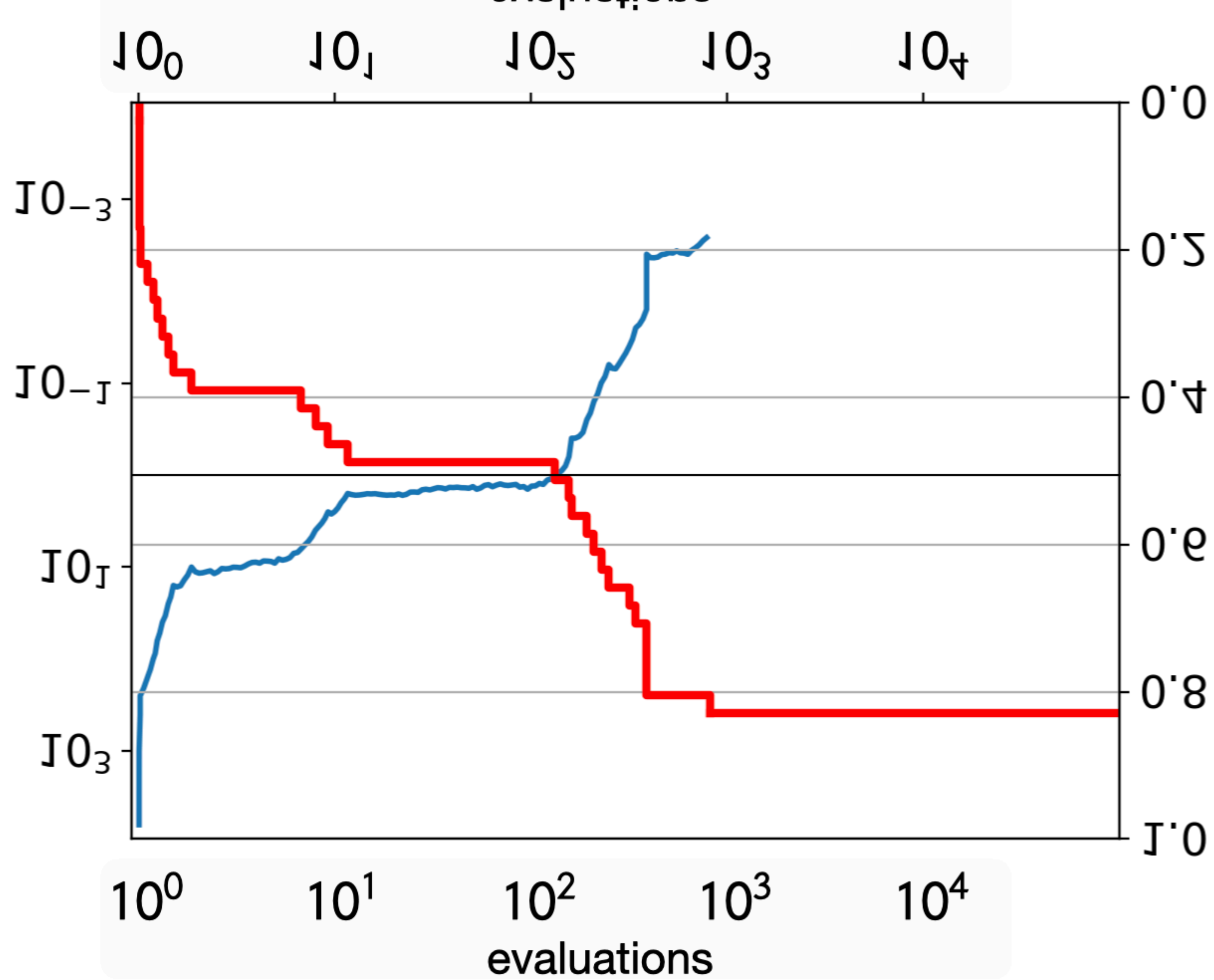




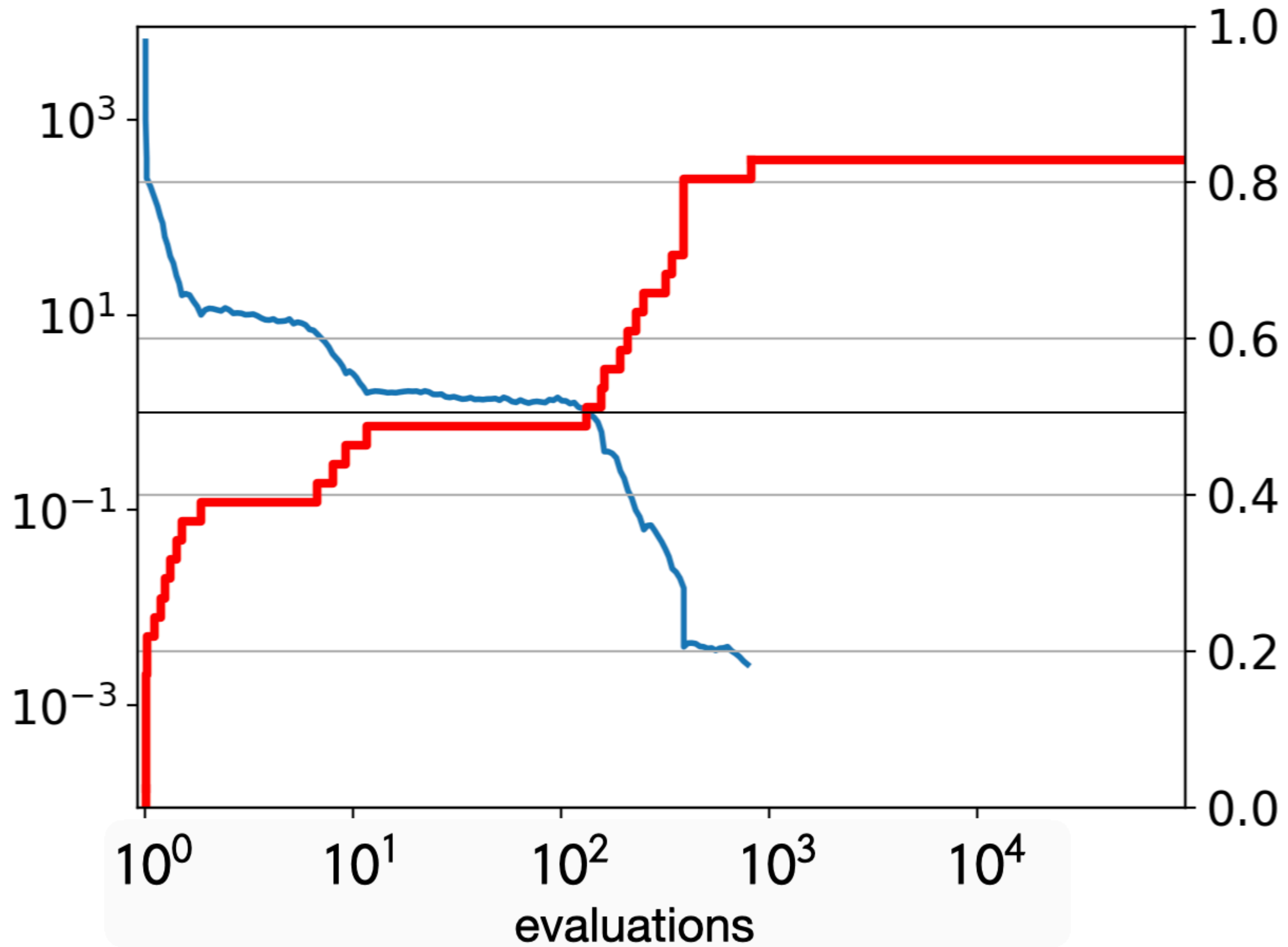


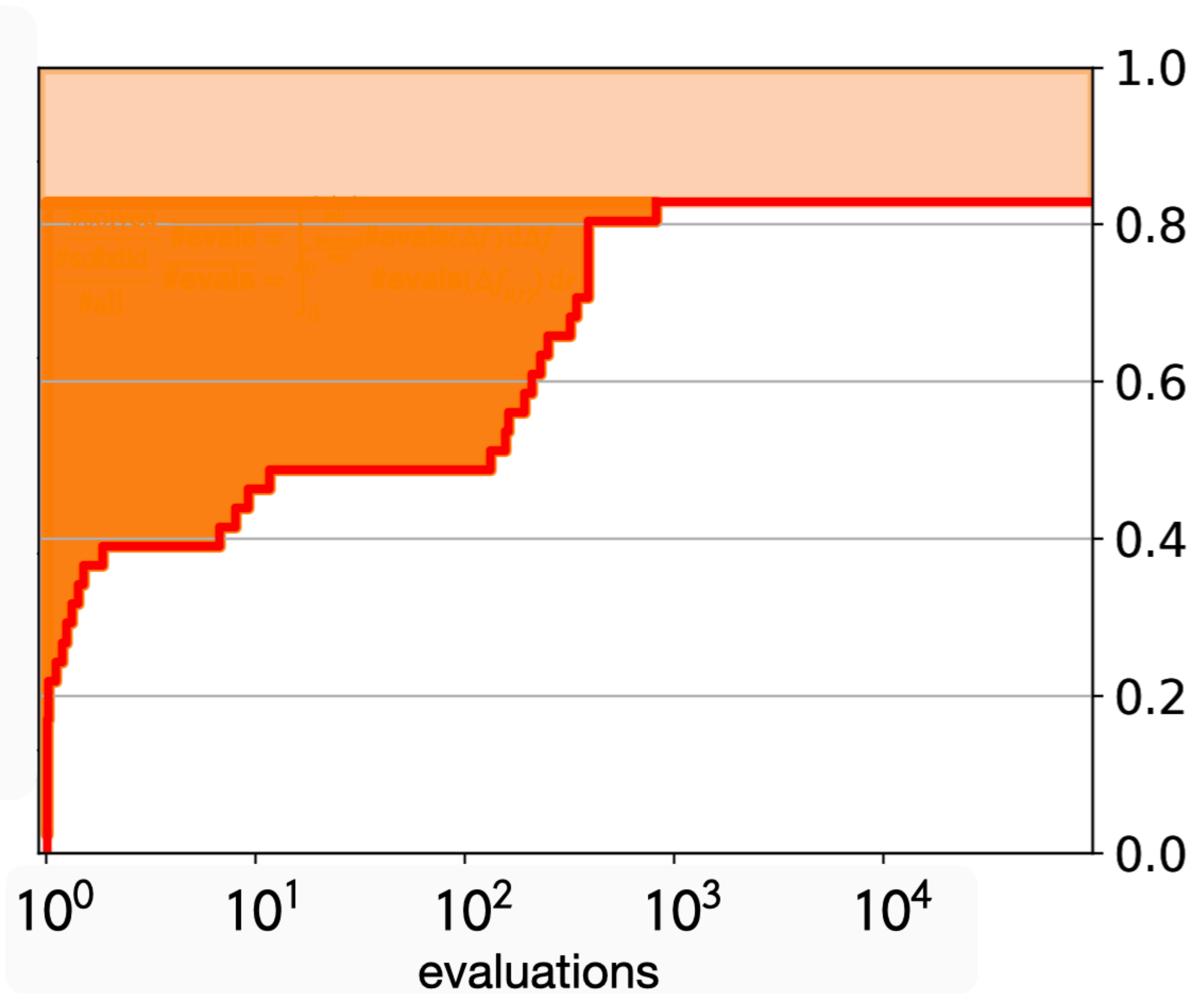






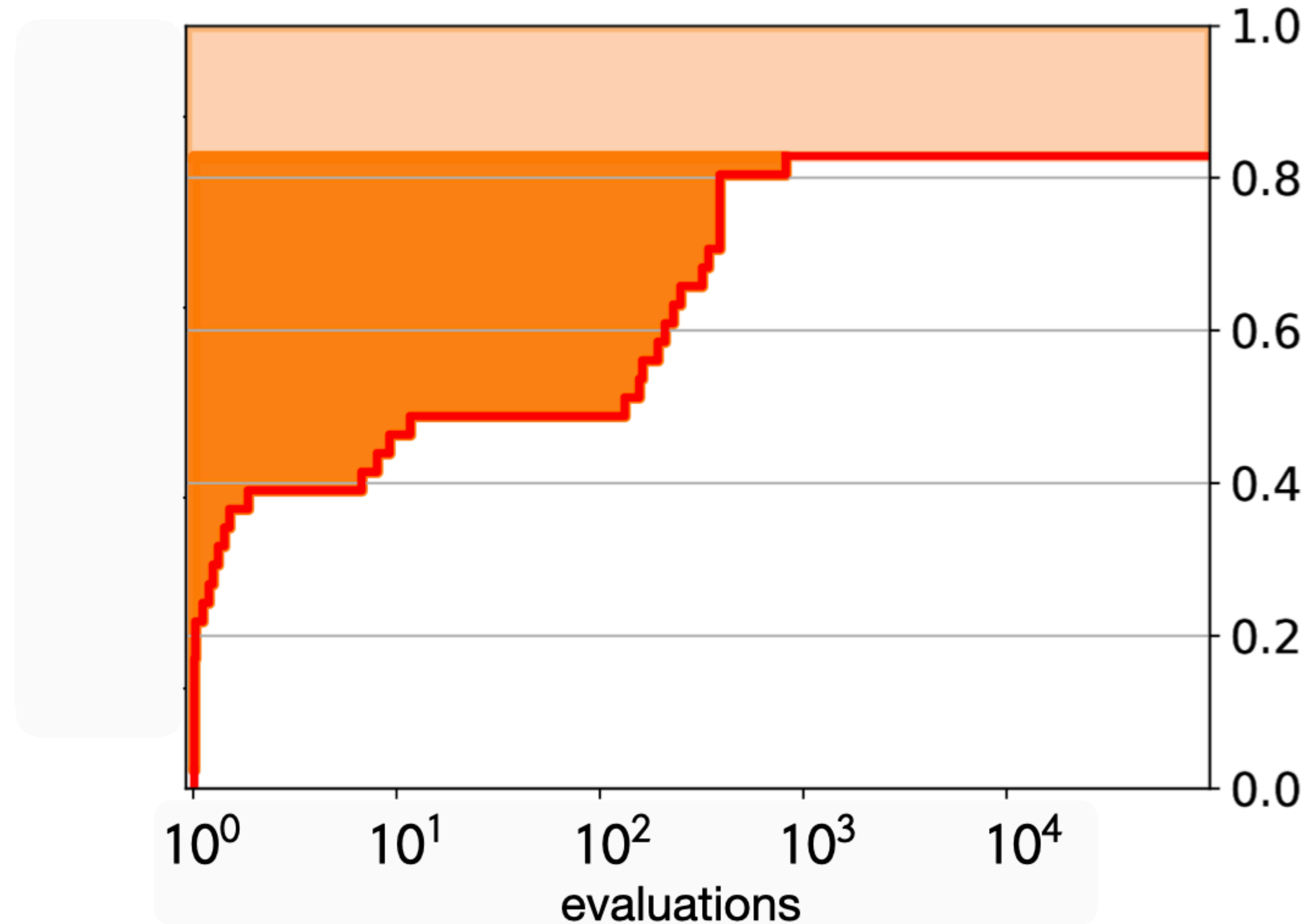
**when we maximize
(instead of minimize),
the graph can be
considered as an
empirical runtime
distribution as is**

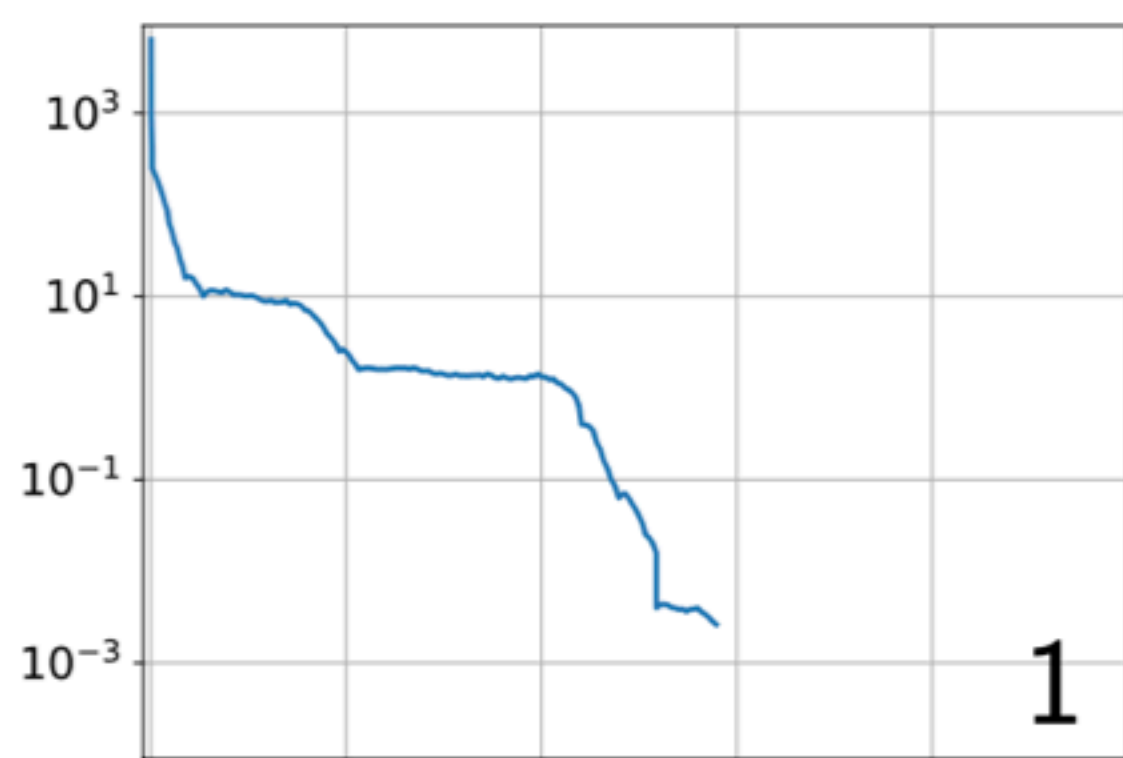




$$\overline{\#evals} = \frac{\#all}{\#solved} \int_0^{\frac{\#solved}{\#all}} \#evals(\Delta f_{i(r)}) dr$$

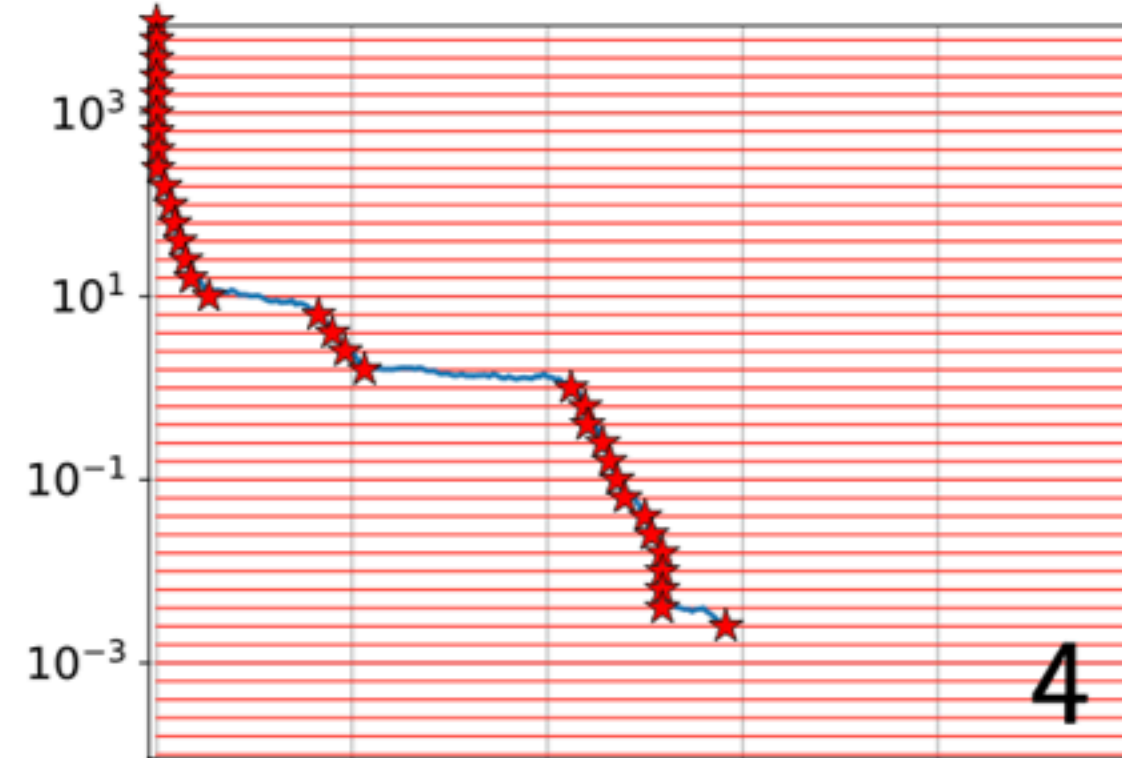
When the x-axis is in log-scale, it is the geometric average





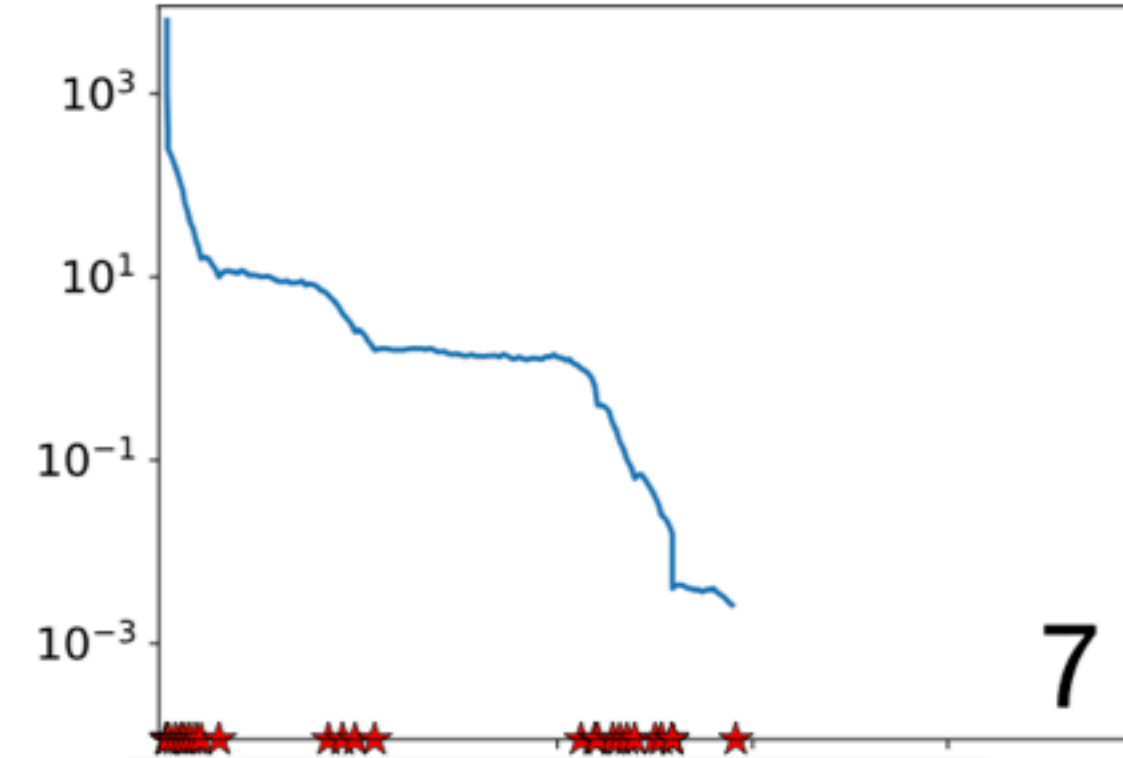
evaluations

1



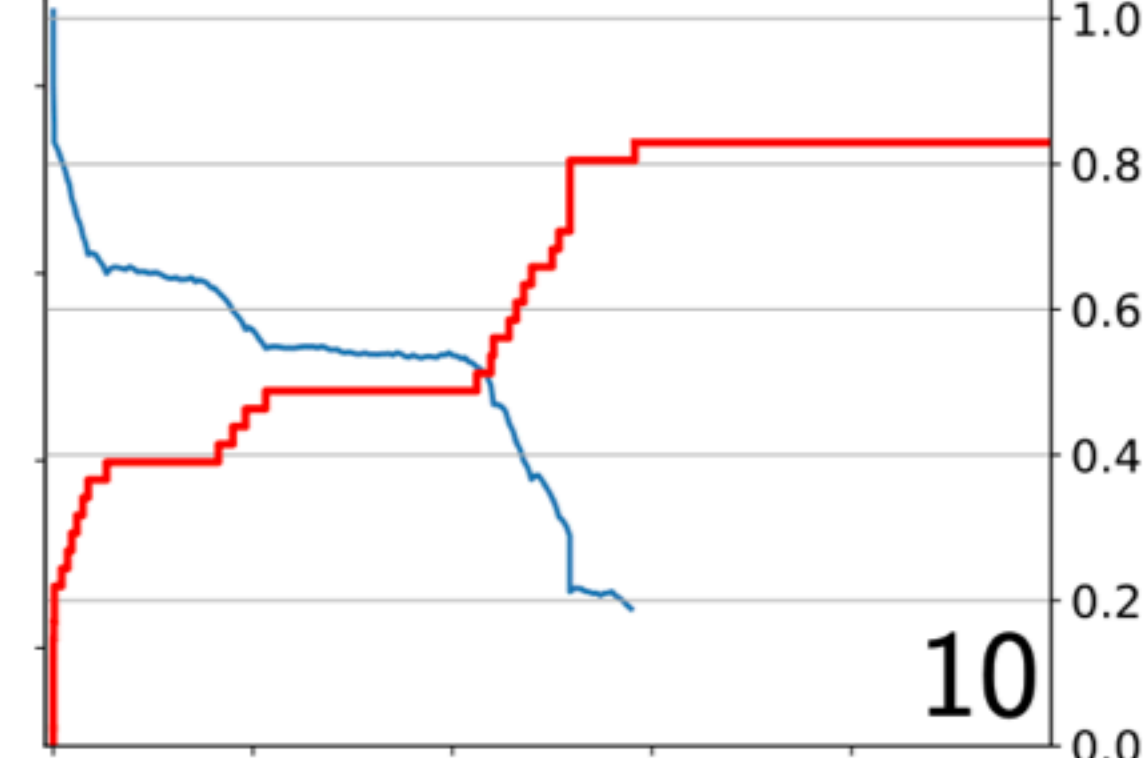
evaluations

4



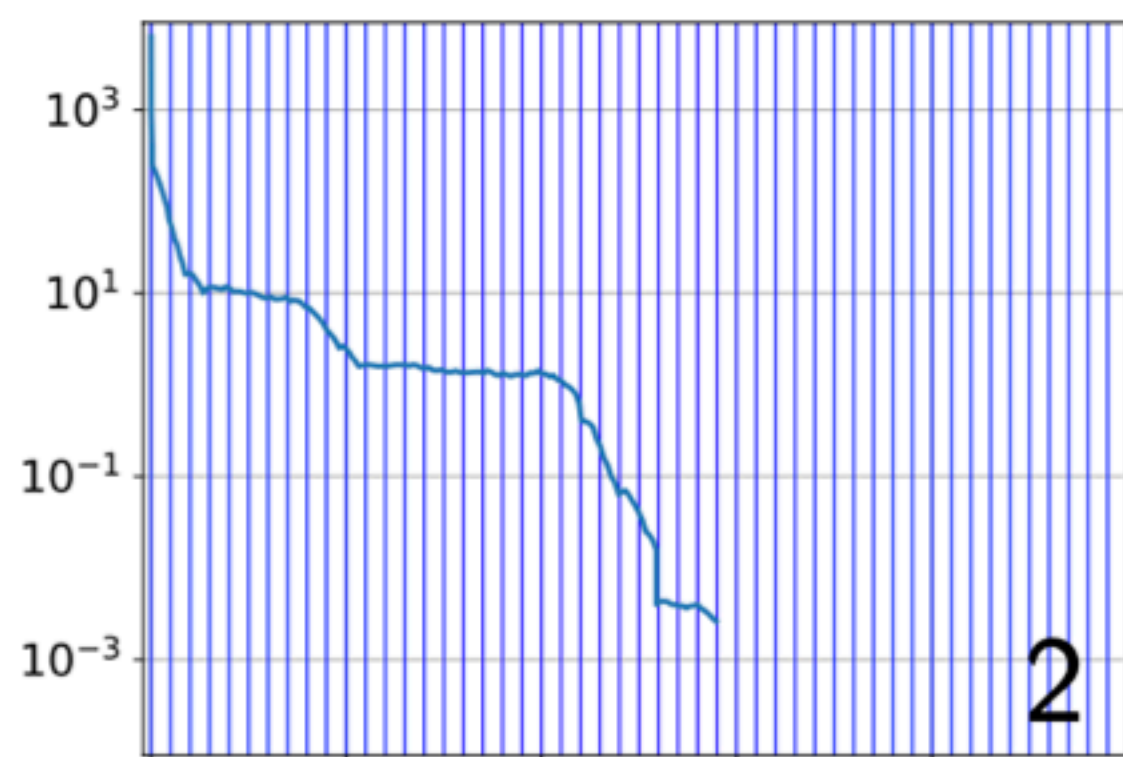
evaluations

7



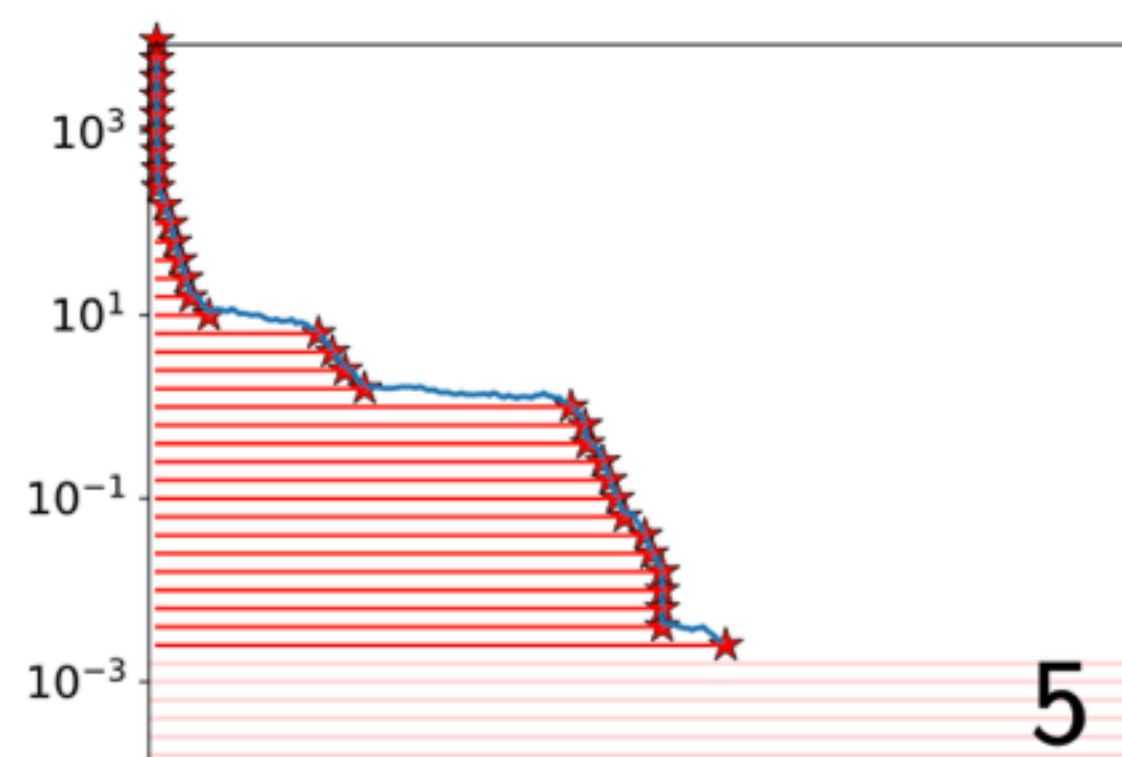
evaluations

10



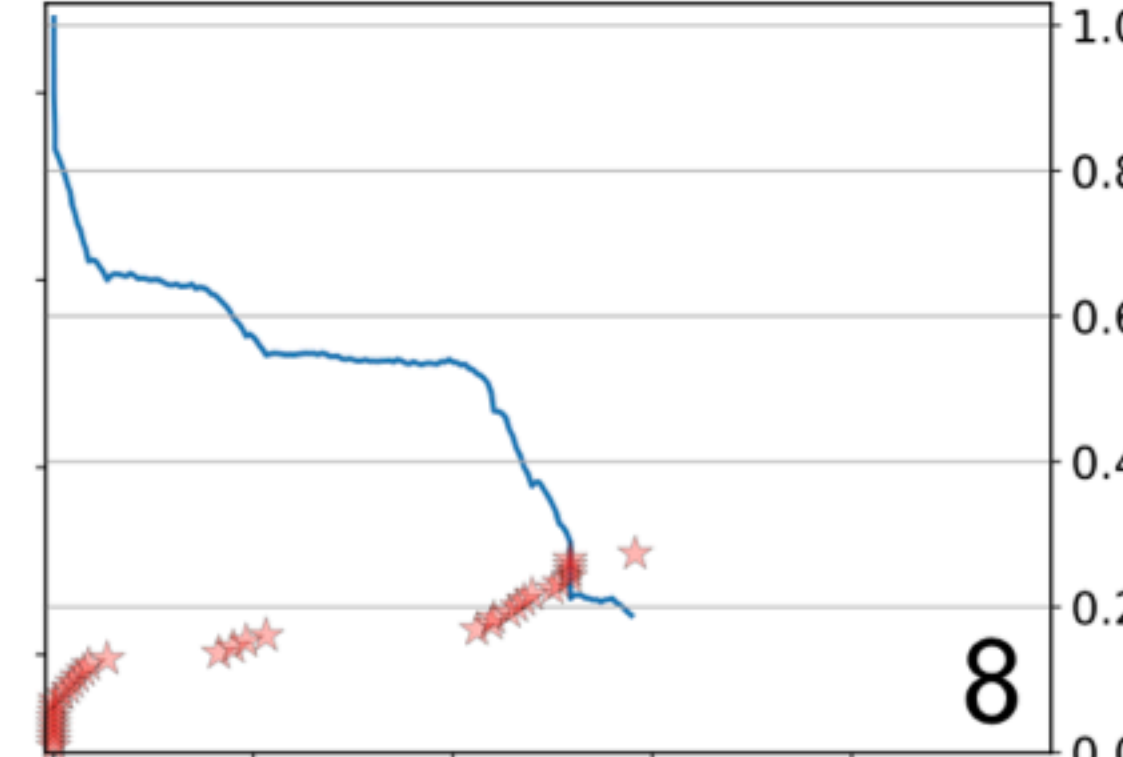
evaluations

2



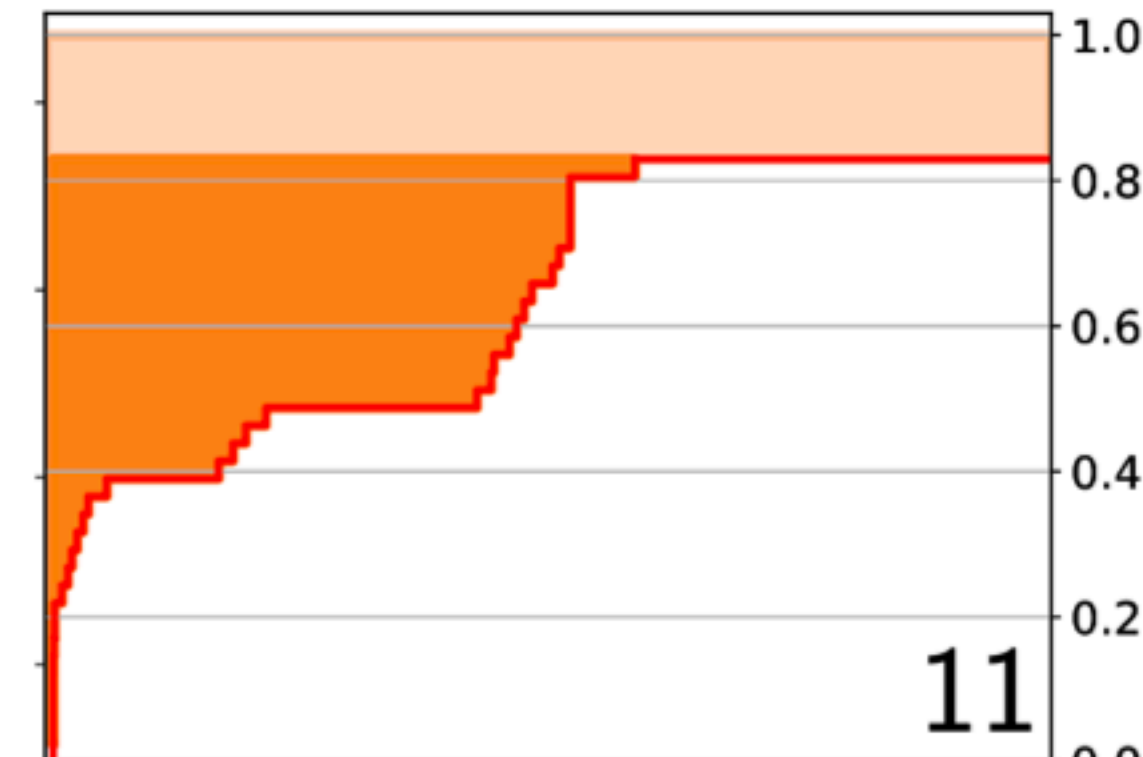
evaluations

5



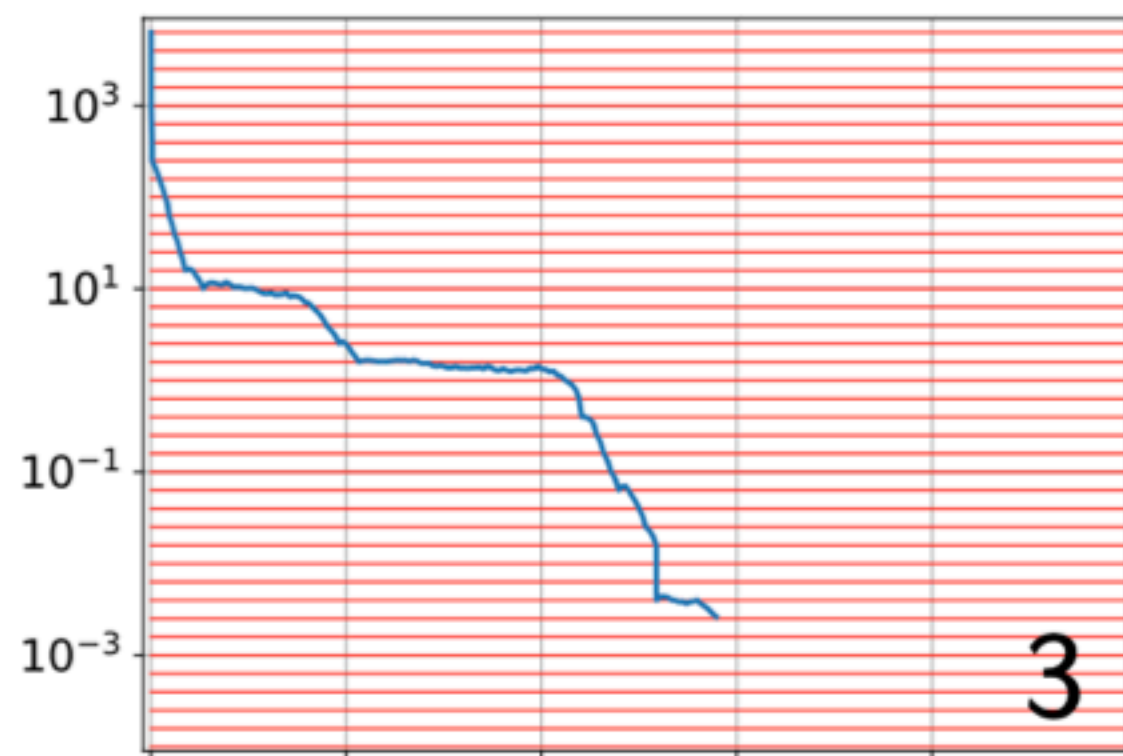
evaluations

8



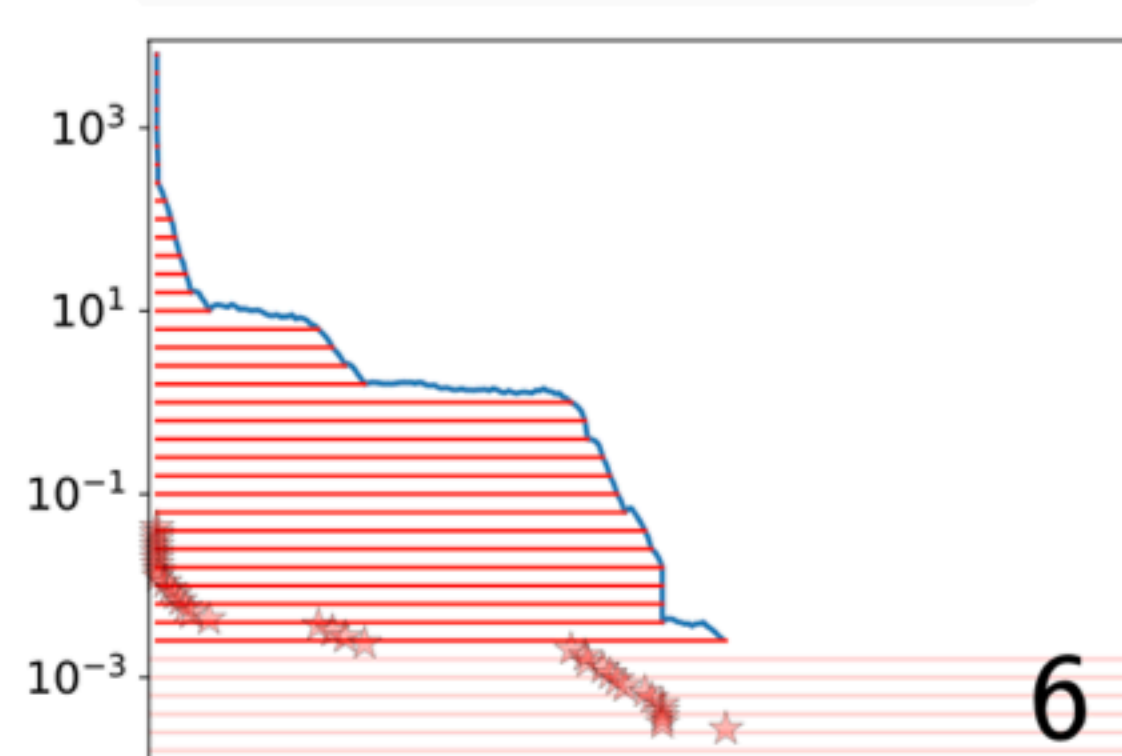
evaluations

11



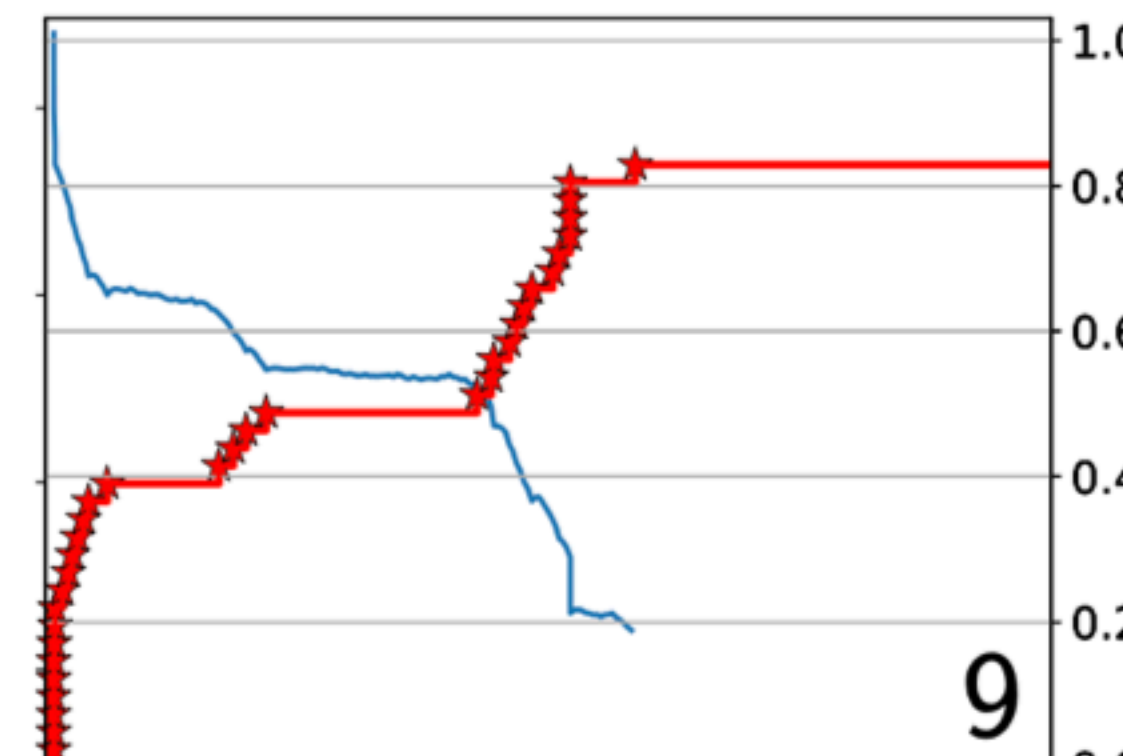
evaluations

3



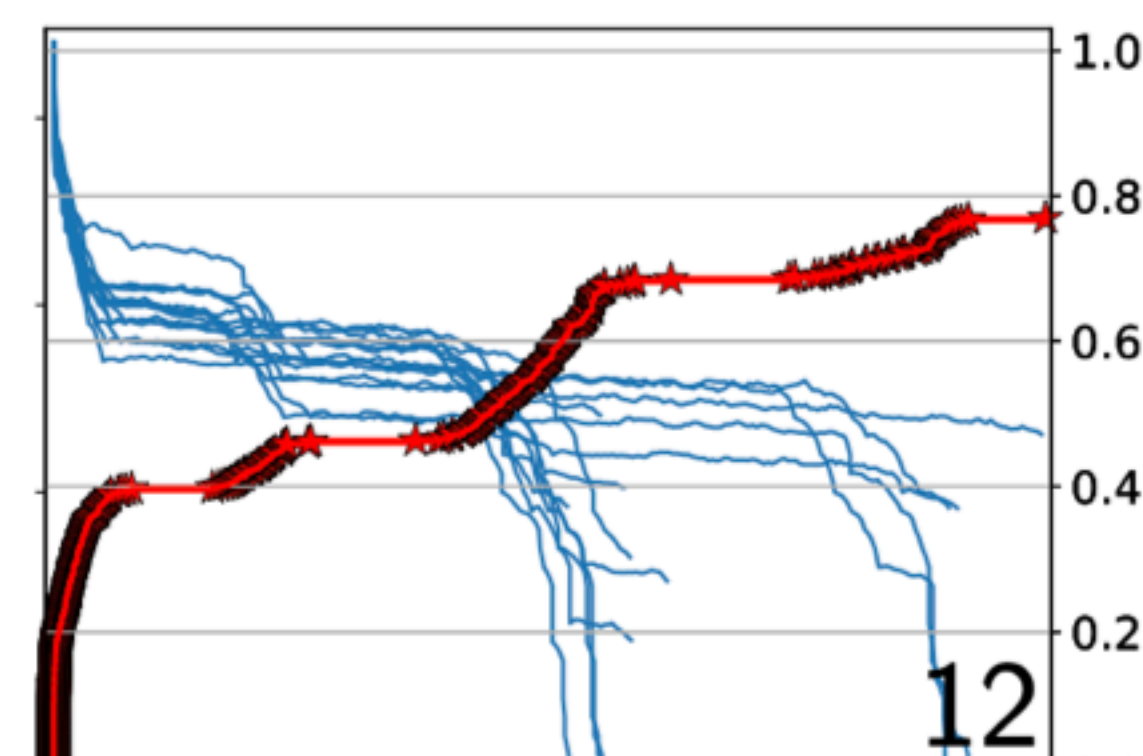
evaluations

6



evaluations

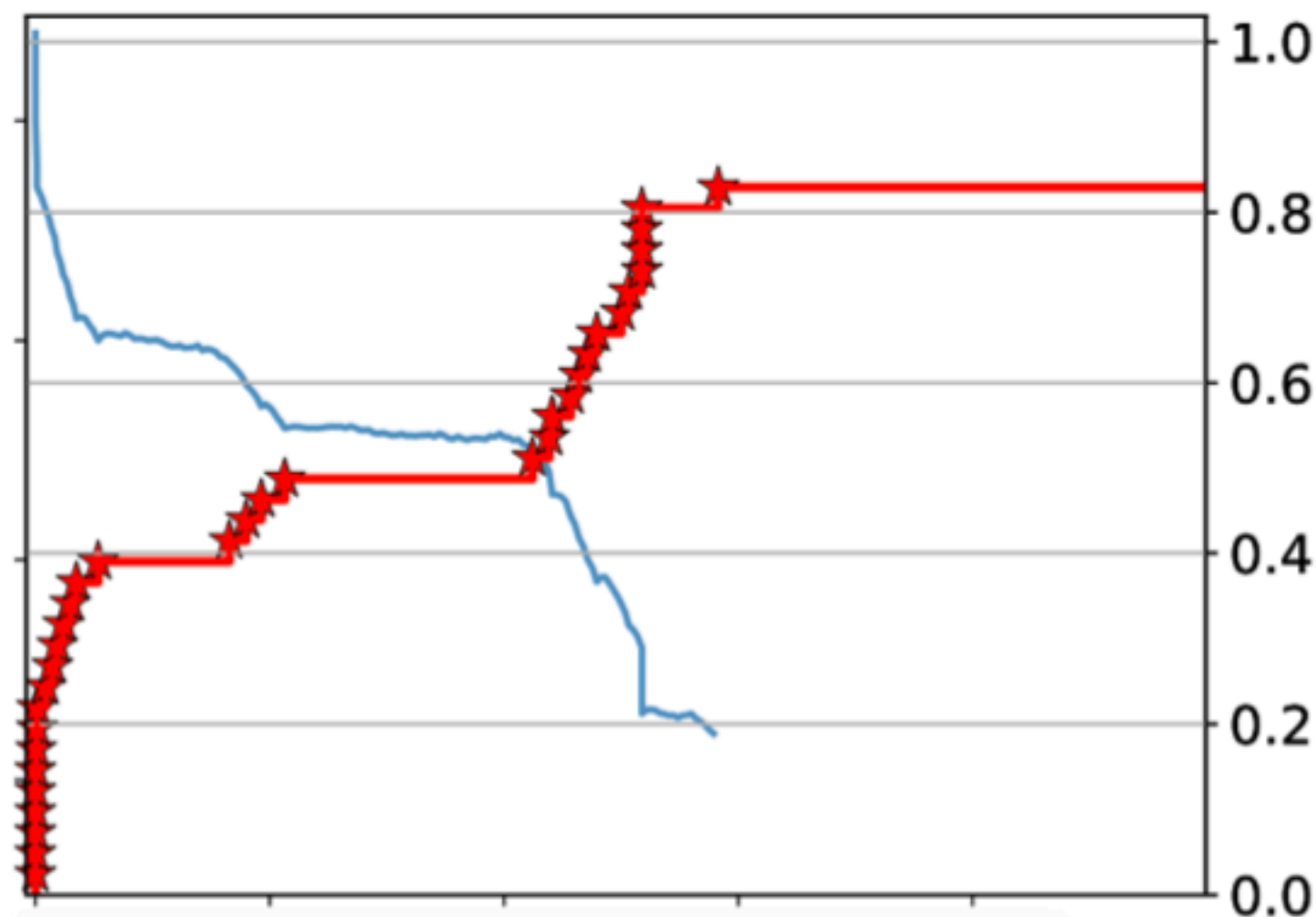
9



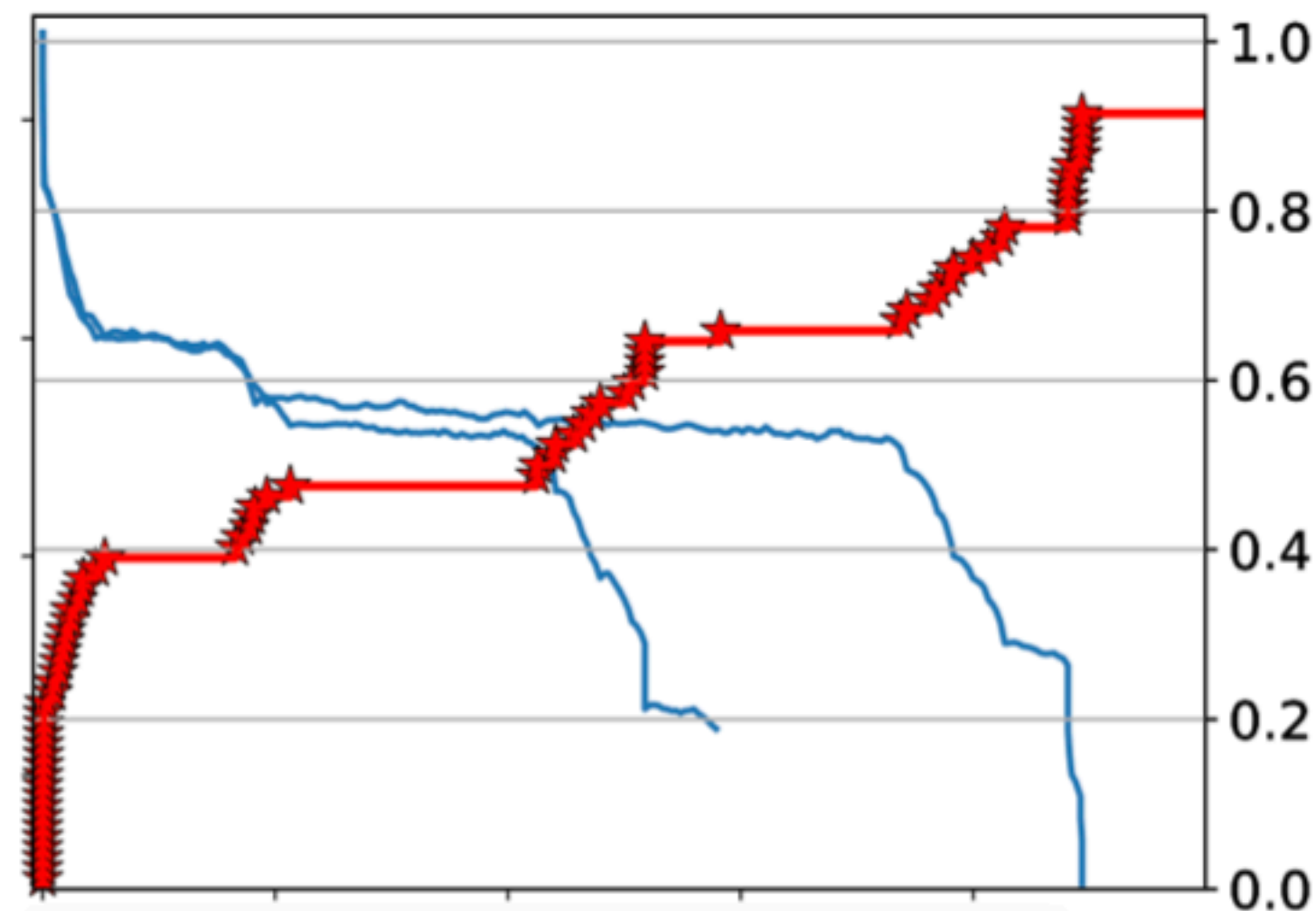
evaluations

12

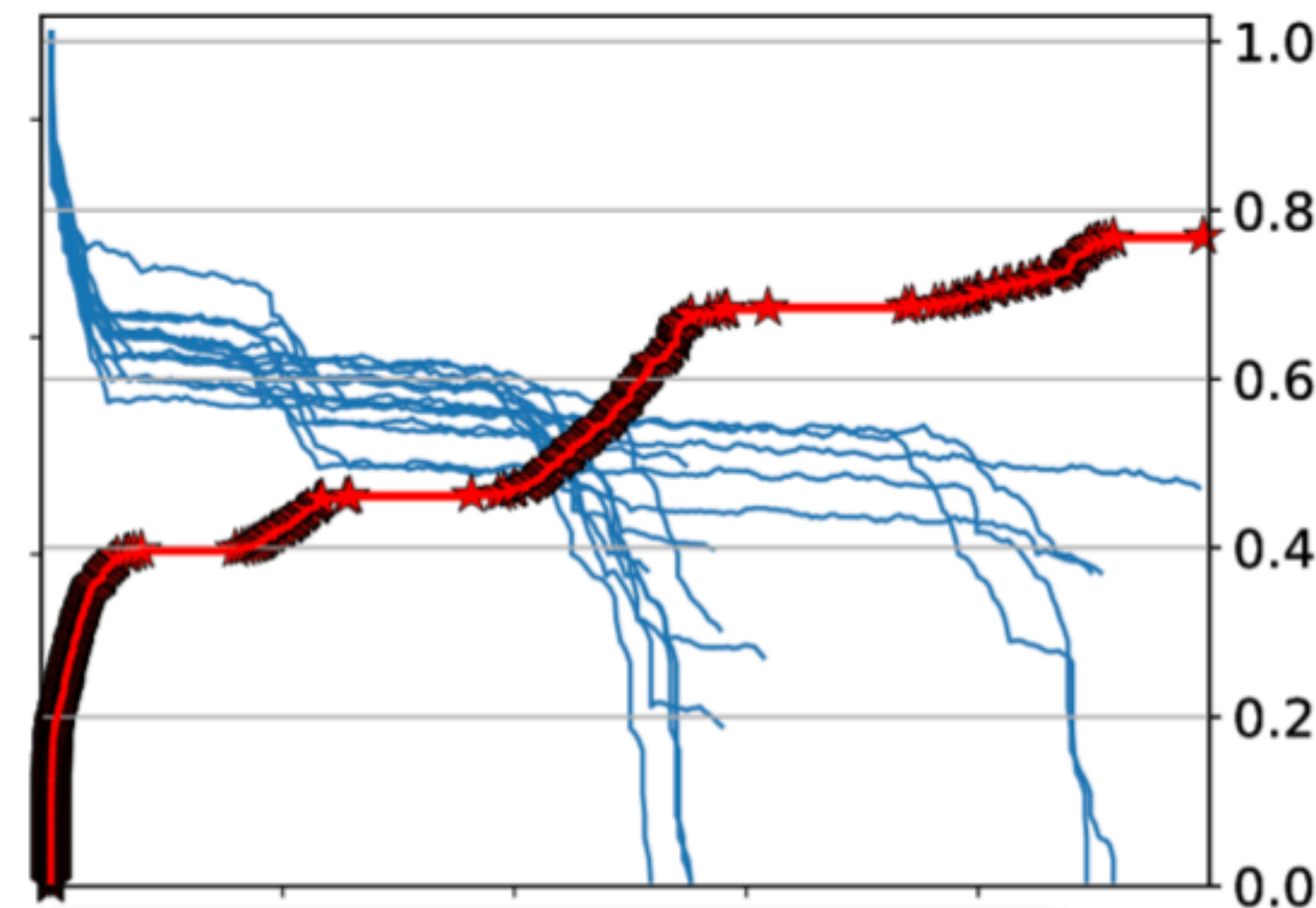
Aggregation of Several Convergence Graphs



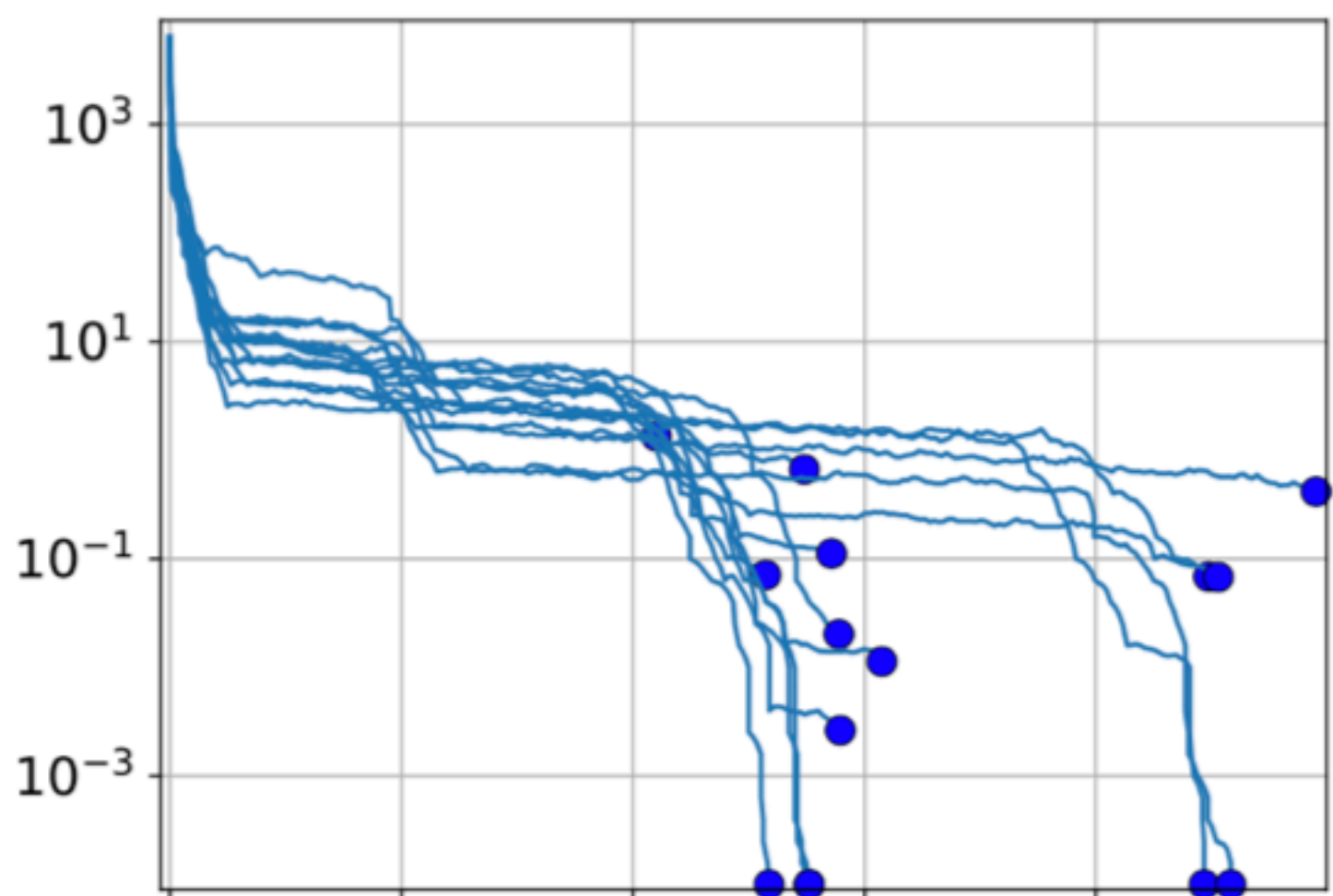
evaluations



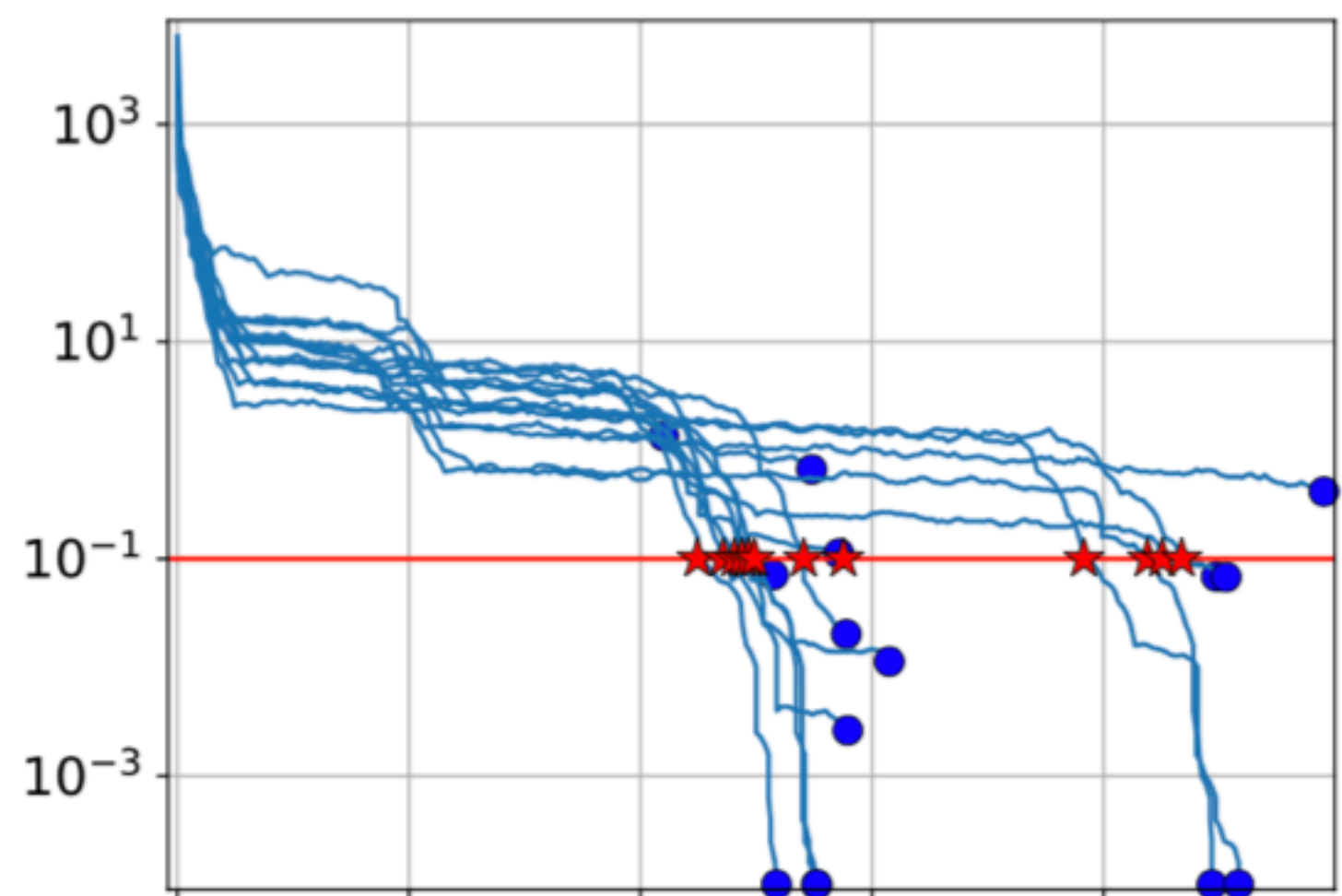
evaluations



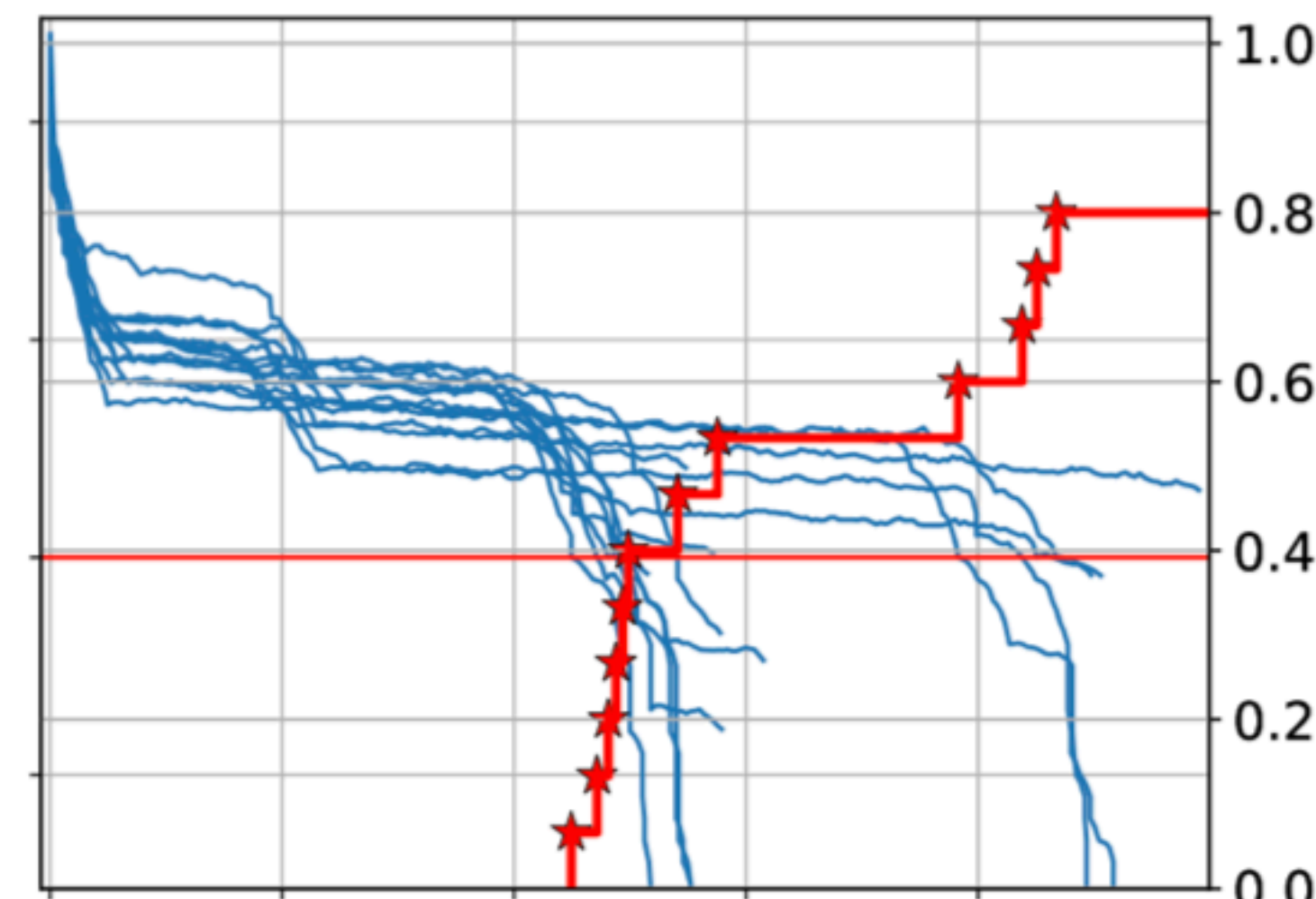
evaluations



evaluations



evaluations



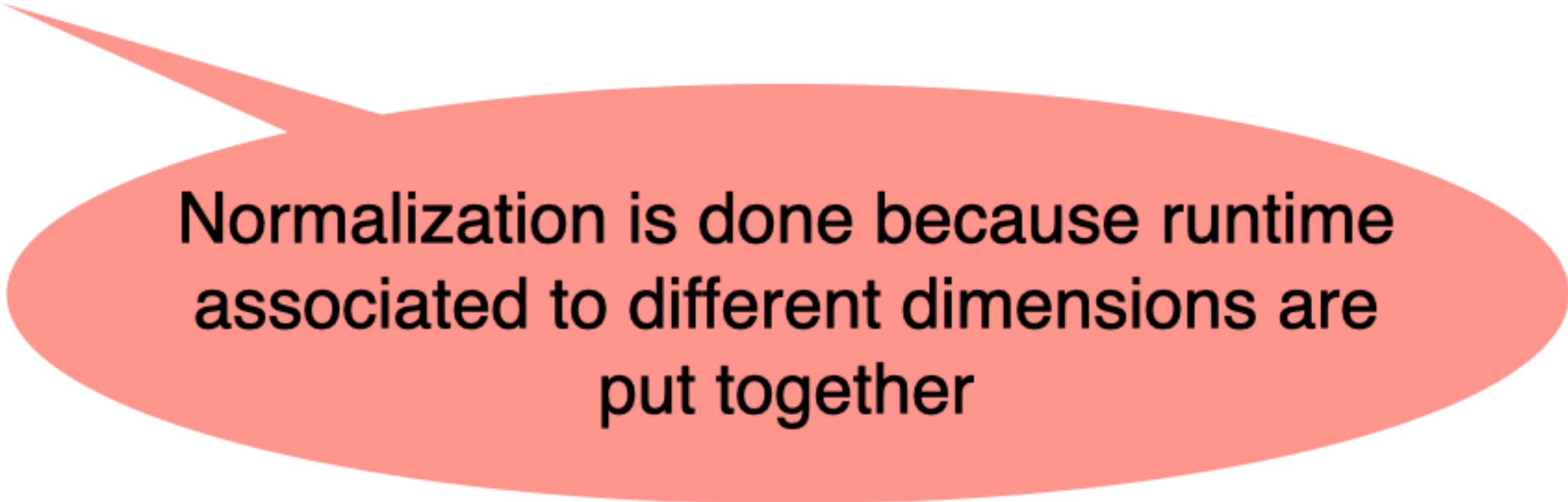
evaluations

Data and Performance Profiles

Data Profile

Given $T_{p,s}$ a collection of runtime (#of f-evals) for a solver s to reach **a certain target** on a problem $p \in \mathcal{P}$.

The data profile is the ECDF of $\{T_{p,s}/(n+1), p \in \mathcal{P}\}$:



Normalization is done because runtime associated to different dimensions are put together

•
Benchmarking Derivative-Free Optimization Algorithms by J. Moré and S. Wild. SIAM J. Optimization, Vol. 20 (1), pp.172-191, 2009.

Data Profile

Given $T_{p,s}$ a collection of runtime (#of f-evals) for a solver s to reach **a certain target** on a problem $p \in \mathcal{P}$.

The data profile is the ECDF of $\{T_{p,s}/(n+1), p \in \mathcal{P}\}$:

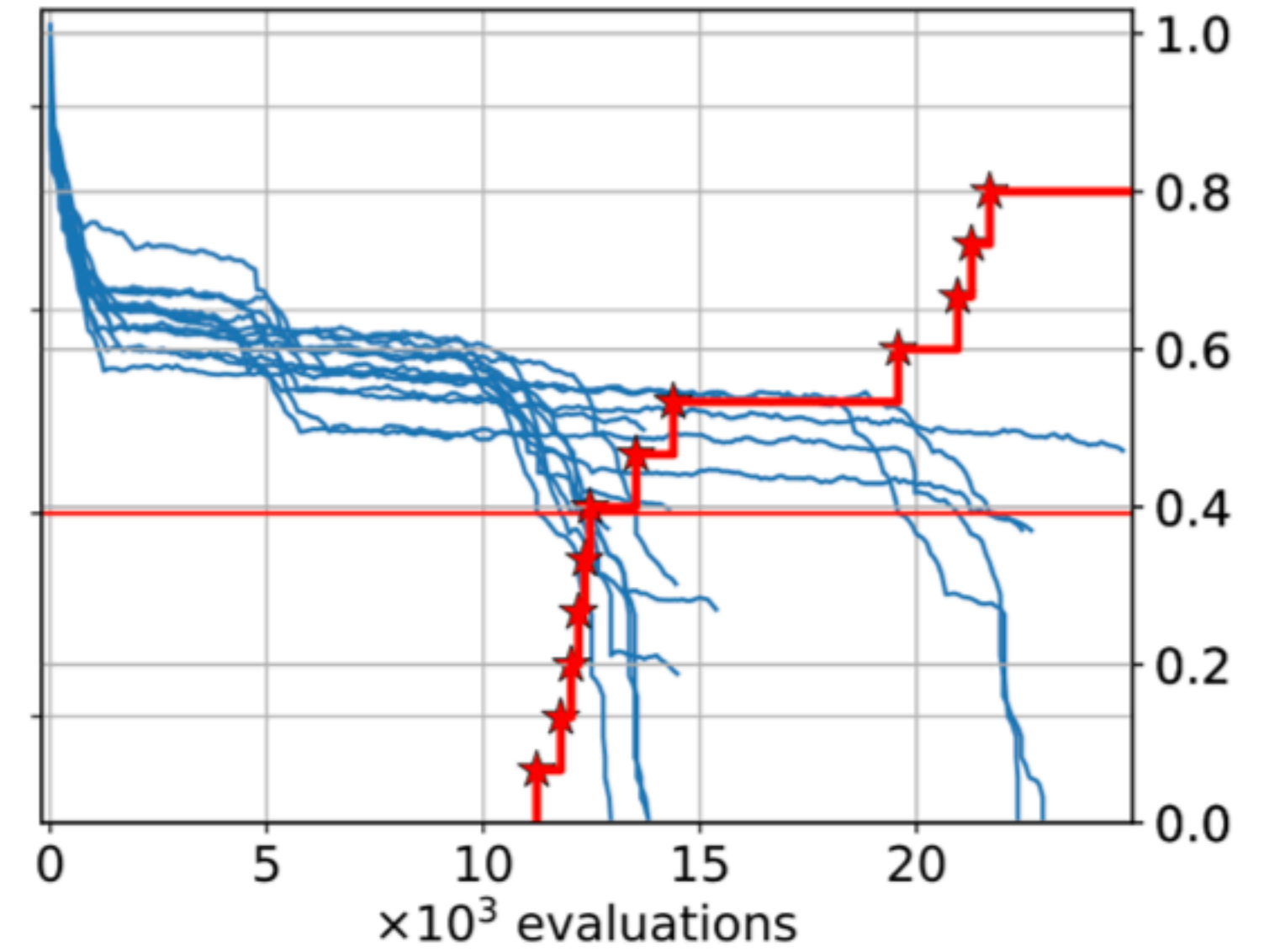
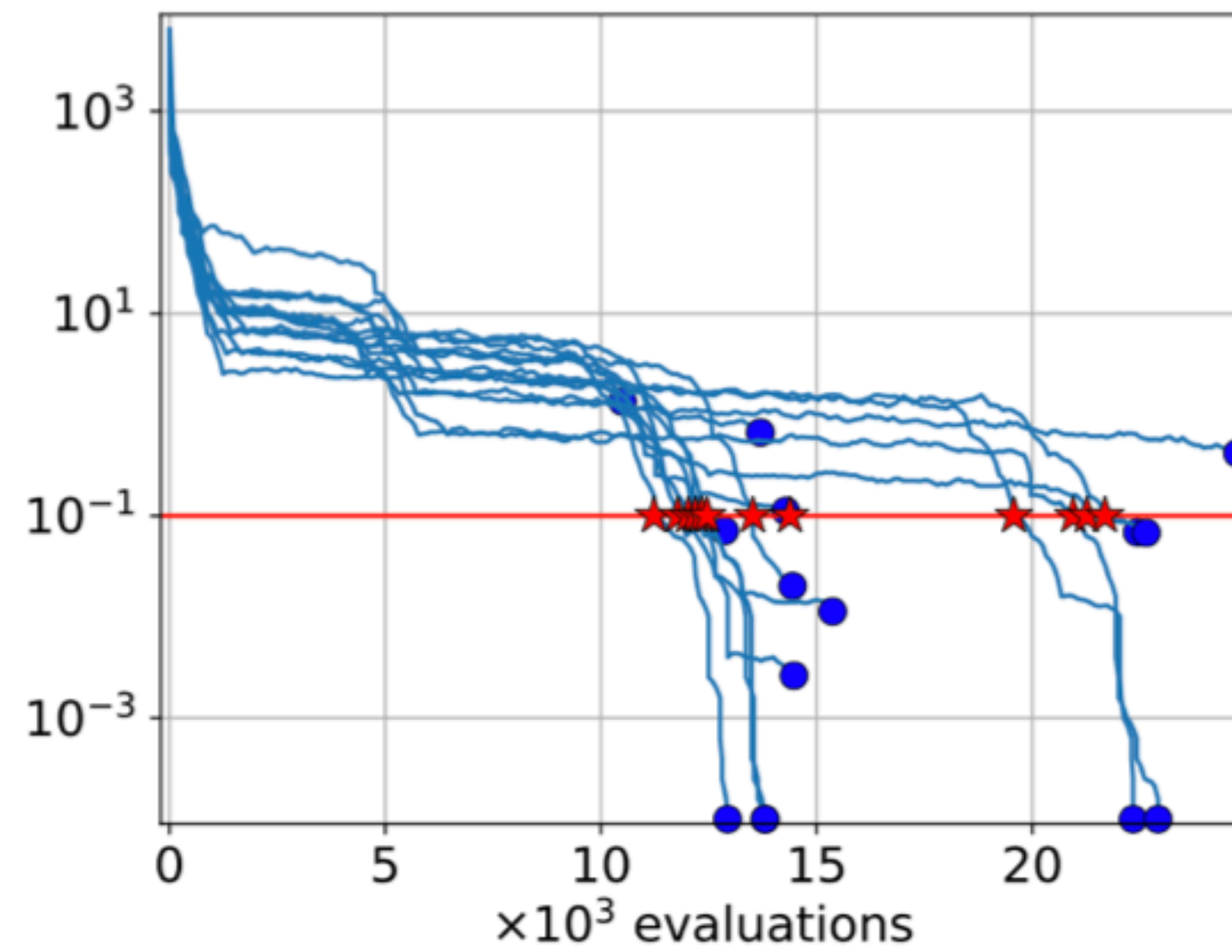
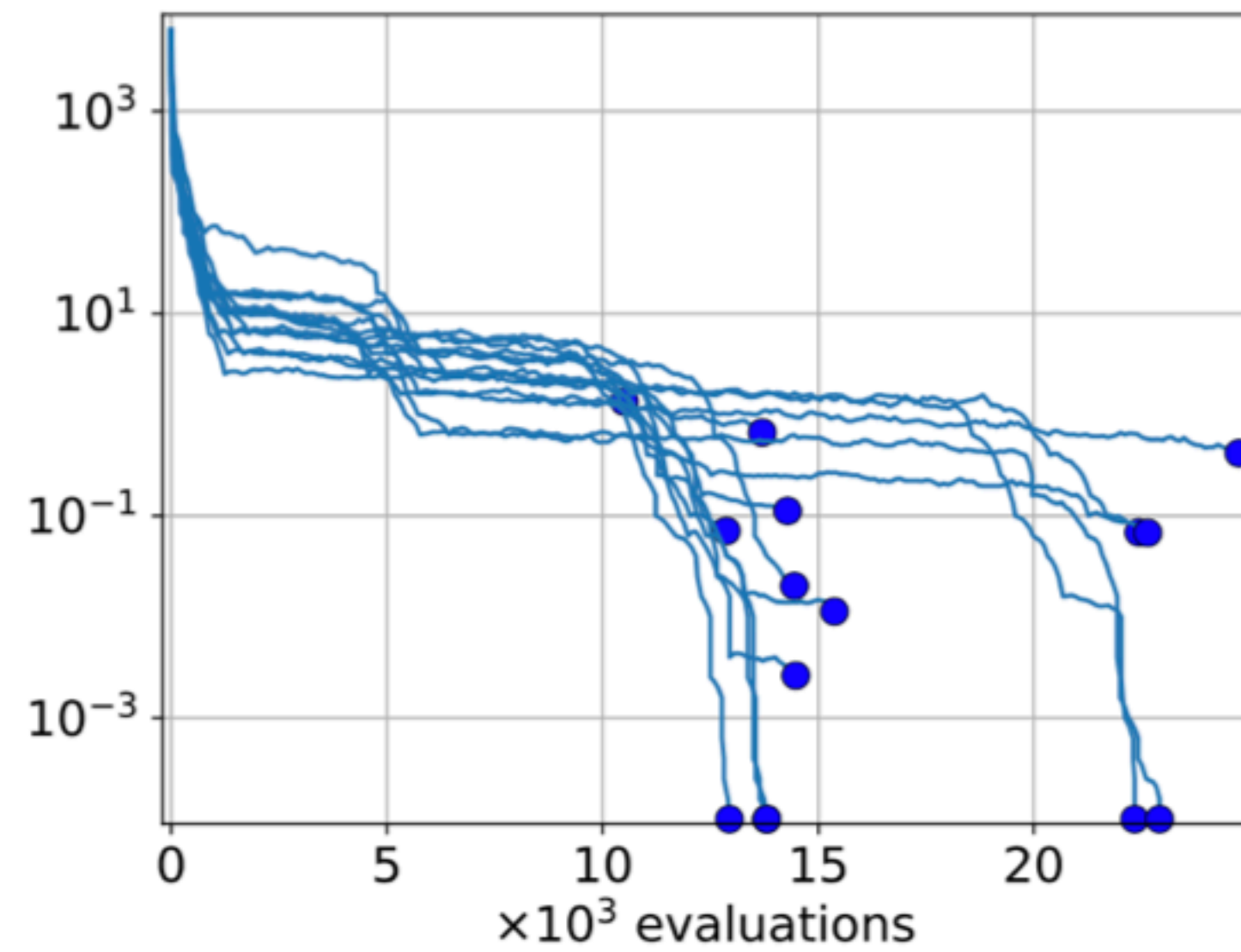
Normalization is done because runtime associated to different dimensions are put together

$$\text{ECDF}_{\{T_{p,s}/(n+1), p \in \mathcal{P}\}}(t) = \frac{1}{|\mathcal{P}|} \sum_{p=1}^{|\mathcal{P}|} \mathbf{1}_{\left\{\frac{T_{p,s}}{n+1} \leq t\right\}}$$

•

Benchmarking Derivative-Free Optimization Algorithms by J. Moré and S. Wild. SIAM J. Optimization, Vol. 20 (1), pp.172-191, 2009.

Data Profile



Targets may be different for each function, but choosing a different target or shifting the respective graph vertically is the same

Performance Profile

Normalize runtime by performance of best solver: Define the performance on a problem p by a solver s as the runtime divided by the runtime of best solver among a set of solvers \mathcal{S}

$$r_{p,s} = \frac{T_{p,s}}{\min\{T_{p,s} : s \in \mathcal{S}\}}$$

! It “Removes” the order of magnitude of $T_{p,s}$ and thus the information of difficulty

The **performance profile of a solver s** is the ECDF of $\{r_{p,s}, p \in \mathcal{P}\}$:

$$\text{ECDF}_{\{r_{p,s}, p \in \mathcal{P}\}}(t) = \frac{1}{|\mathcal{P}|} \sum_{p=1}^{|\mathcal{P}|} 1_{\{r_{p,s} \leq t\}}$$

E. D. Dolan and J. J. Moré, Benchmarking optimization software with performance profiles, Math. Program., 91 (2002), pp. 201–213.

Data and Performance Profile: Discussion

- Performance and Data profiles are just **ECDF of (normalized) runtime** associated to a single target per problem
- Performance profile
 - normalized by the smallest (best) runtime
 - relative to the set of solvers benchmarked
 - difficult to compare across papers*
 - we do not see the problem difficulty anymore: normalization removes absolute value

Aggregation of Data

- is necessary

e.g., BBOB takes about $24 \times 15 \times 30 \approx 10,000$ single measurements for each algorithm in each dimension

- **implicit assumption**: uniform distribution over all problems we aggregate over shall somewhat reflect the problem distribution in reality

- properties that can be **inexpensively probed** should not (never) be aggregated over different values

For example: dimensionality. Why?

- any **runtimes** can be meaningfully aggregated

Assuming they come in the same unit of measurement (here evaluations).

However: not all ways to aggregate runtimes are meaningful.

We should use a log scale when they come from different distributions.

- **successful and unsuccessful runs** can be meaningfully aggregated,

solving the fast vs successful comparison “dilemma” once and for all.

Using simulated restarts or Enes/ERT/SP2, see “Treating success probabilities”.

Aggregation of Data

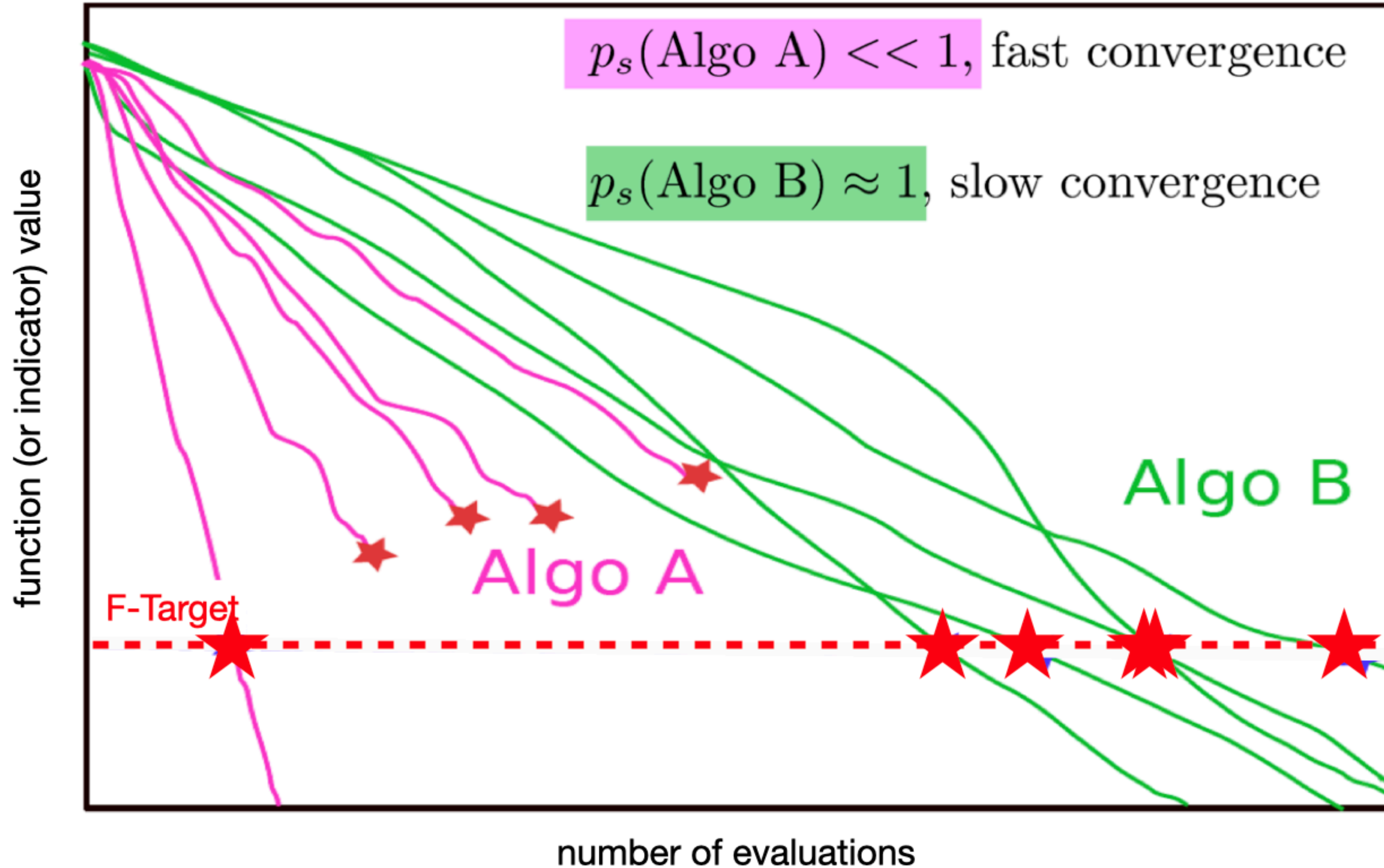
1. aggregating **repetitions** over the same (or very similar) problem(s)
in particular with unsuccessful trials
2. aggregating data from **different problems**

Expected RunTime (ERT)

Aggregated measurement of repetitions

Treating Success Probabilities

Solving the fast-versus-successful comparison dilemma



Treating Success Probabilities

Solving the fast-versus-successful comparison dilemma

We can **simulate a runtime distribution** by simulated (artificial) restarts using the given independent runs

Algo Restart A:



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

Algo Restart B:



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

Caveat: the performance of algorithm A critically depends on termination methods (before to hit the target)

which reflects the situation on a practical problem unless many runs can be done in parallel

Expected Runtime of Restart Algorithm

Algo Restart A:



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

Algo Restart B:



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

comparable runtimes

Expected Runtime of Restart Algorithm:

$$\mathbb{E}[RT^r] = \left(\frac{1}{p_s} - 1 \right) \mathbb{E}[RT_{\text{unsucc}}] + \mathbb{E}[RT_{\text{success}}]$$

Expected time to see the first success

Expected RunTime - ERT

Expected runtime (ERT, aka Enes, SP2, aRT) estimates $\mathbb{E}[RT^r]$

$$\text{ERT} = \frac{\text{\#evaluations(until to hit target or stop)}}{\text{\#successes}}$$

unsuccessful runs count
(only) in the nominator

defined (only) for $\text{\#successes} > 0$

$$\mathbb{E}[RT^r] = \left(\frac{1}{p_s} - 1 \right) \mathbb{E}[RT_{\text{unsucc}}] + \mathbb{E}[RT_{\text{succ}}]$$

$$\text{ERT} = \left(\frac{N_{\text{success}} + N_{\text{unsucc}}}{N_{\text{success}}} - 1 \right) \text{avg}(\text{eval}_{\text{unsucc}}) + \text{avg}(\text{eval}_{\text{succ}})$$

$$= \left(\frac{N_{\text{unsucc}}}{N_{\text{success}}} \right) \text{avg}(\text{eval}_{\text{unsucc}}) + \text{avg}(\text{eval}_{\text{succ}})$$

odds ratio

ERT Related Performance Measures

$$\text{ERT} = \left(\frac{N_{\text{unsuccess}}}{N_{\text{success}}} \right) \text{avg}(\text{eval}_{\text{unsucc}}) + \text{avg}(\text{eval}_{\text{succ}})$$

$$\approx \left(\frac{N_{\text{unsuccess}}}{N_{\text{success}}} \right) \text{avg}(\text{eval}_{\text{succ}}) + \text{avg}(\text{eval}_{\text{succ}})$$

$$= \left(\frac{N_{\text{unsuccess}} + N_{\text{success}}}{N_{\text{success}}} \right) \text{avg}(\text{eval}_{\text{succ}})$$

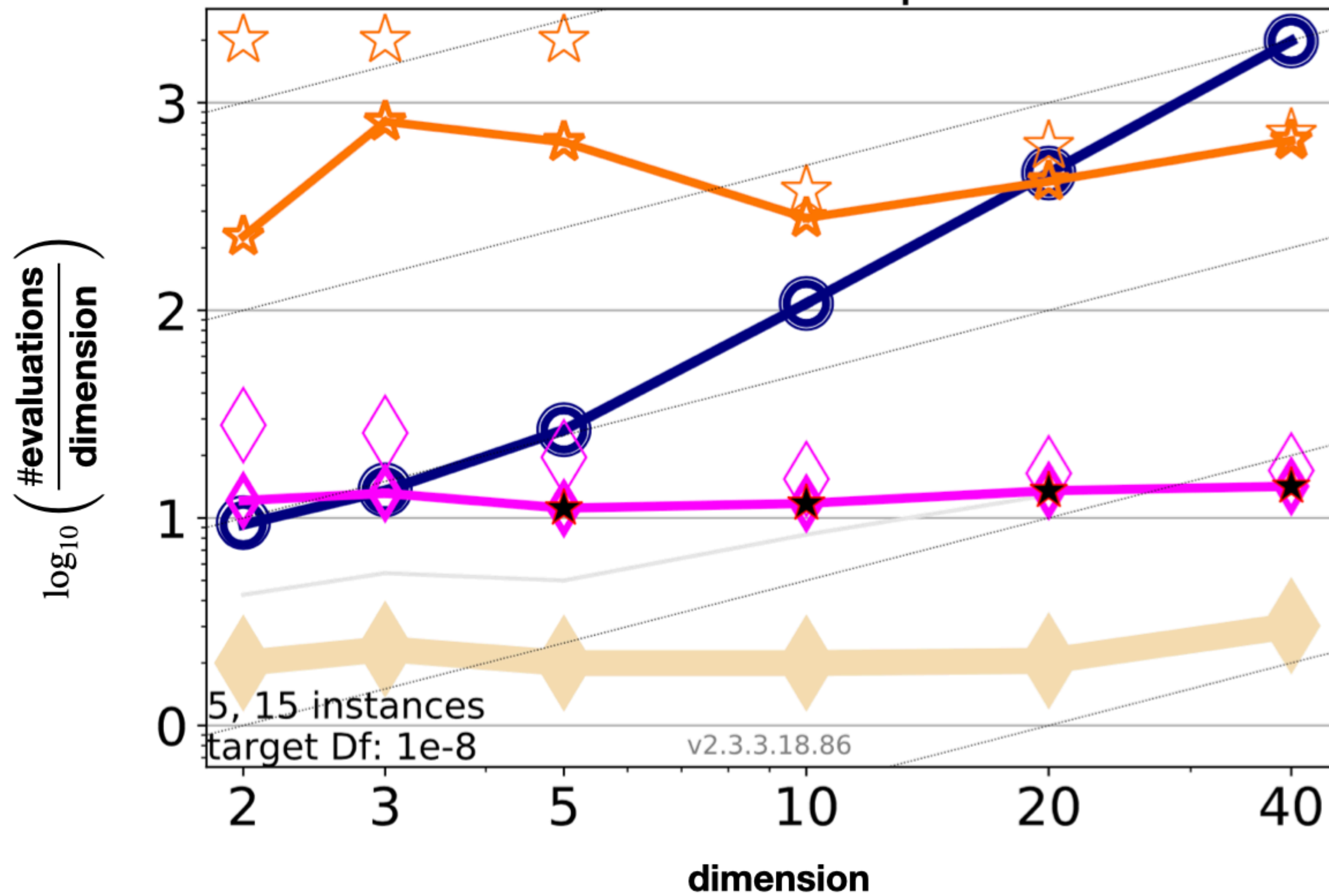
$$= \left(\frac{1}{\text{success rate}} \right) \text{avg}(\text{eval}_{\text{succ}})$$

may or may not
be the case

The last three lines are AKA Q-measure or SP1 (success performance).

See [Price 1997] and [Auger&Hansen 2005]

On Scaling

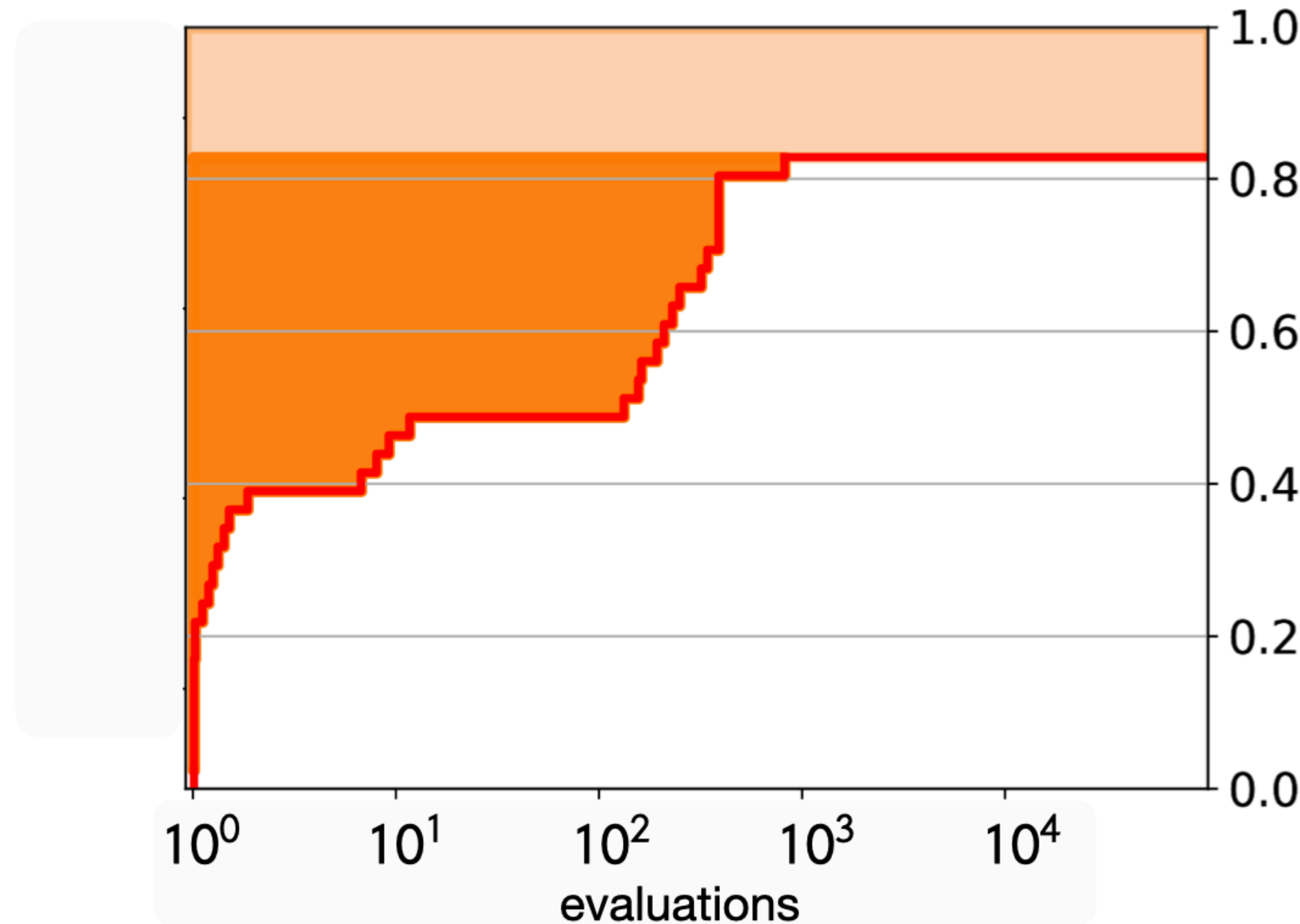


Aggregation of Data

1. aggregating **repetitions** over the same (or very similar) problem(s)
in particular with unsuccessful trials
2. aggregating data from **different problems**
already a single convergence graph contains different problems

$$\overline{\#evals} = \frac{\#all}{\#solved} \int_0^{\frac{\#solved}{\#all}} \#evals(\Delta f_{i(r)}) dr$$

When the x-axis is in log-scale, it is the geometric average



Aggregation of Data: ECDFs With Different Problems

- ECDFs (re-)order the data (sort the data)

*hence we **lose the problem label**
single convergence graph ECDFs are not affected*

- The **average runtime ratio**

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

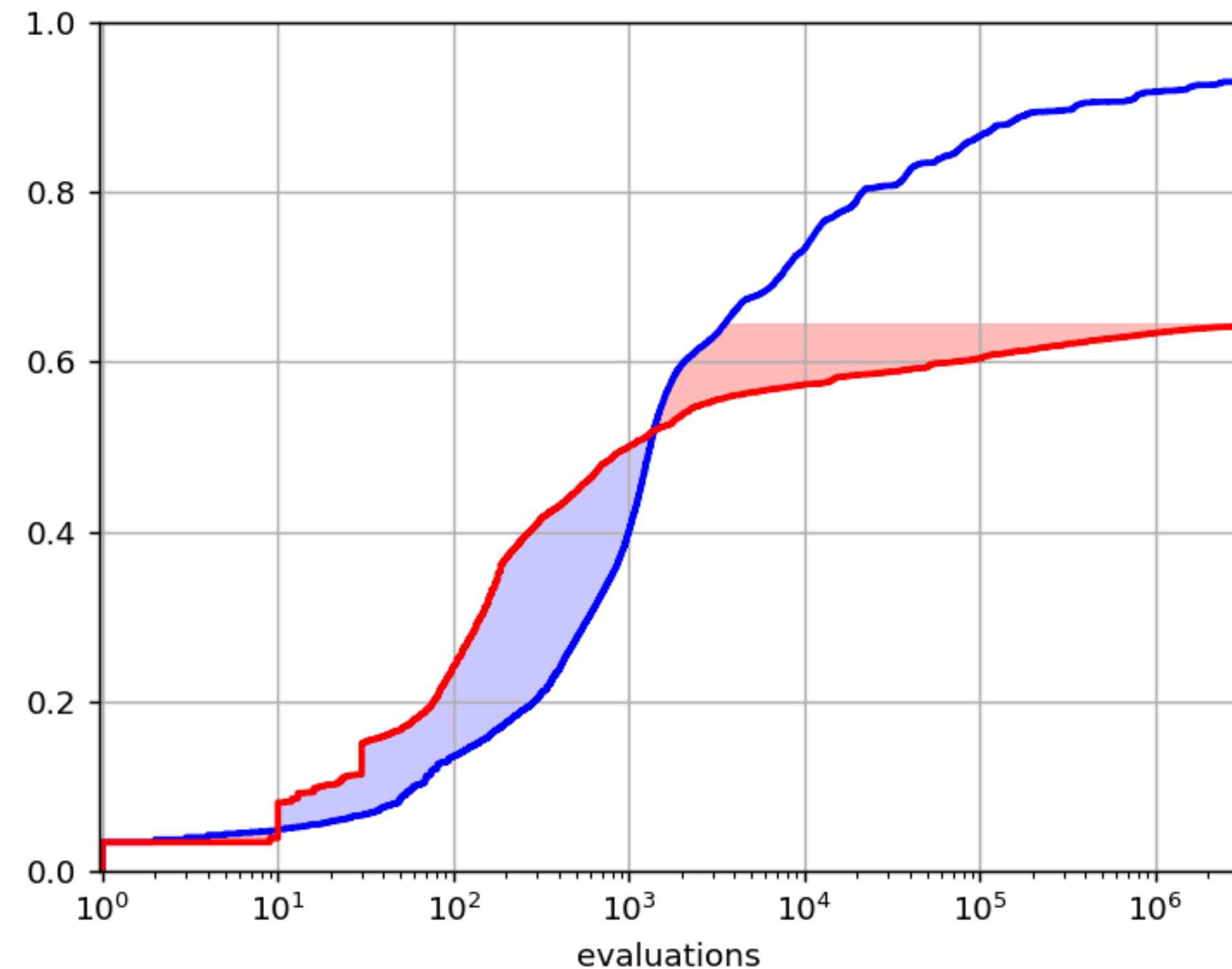
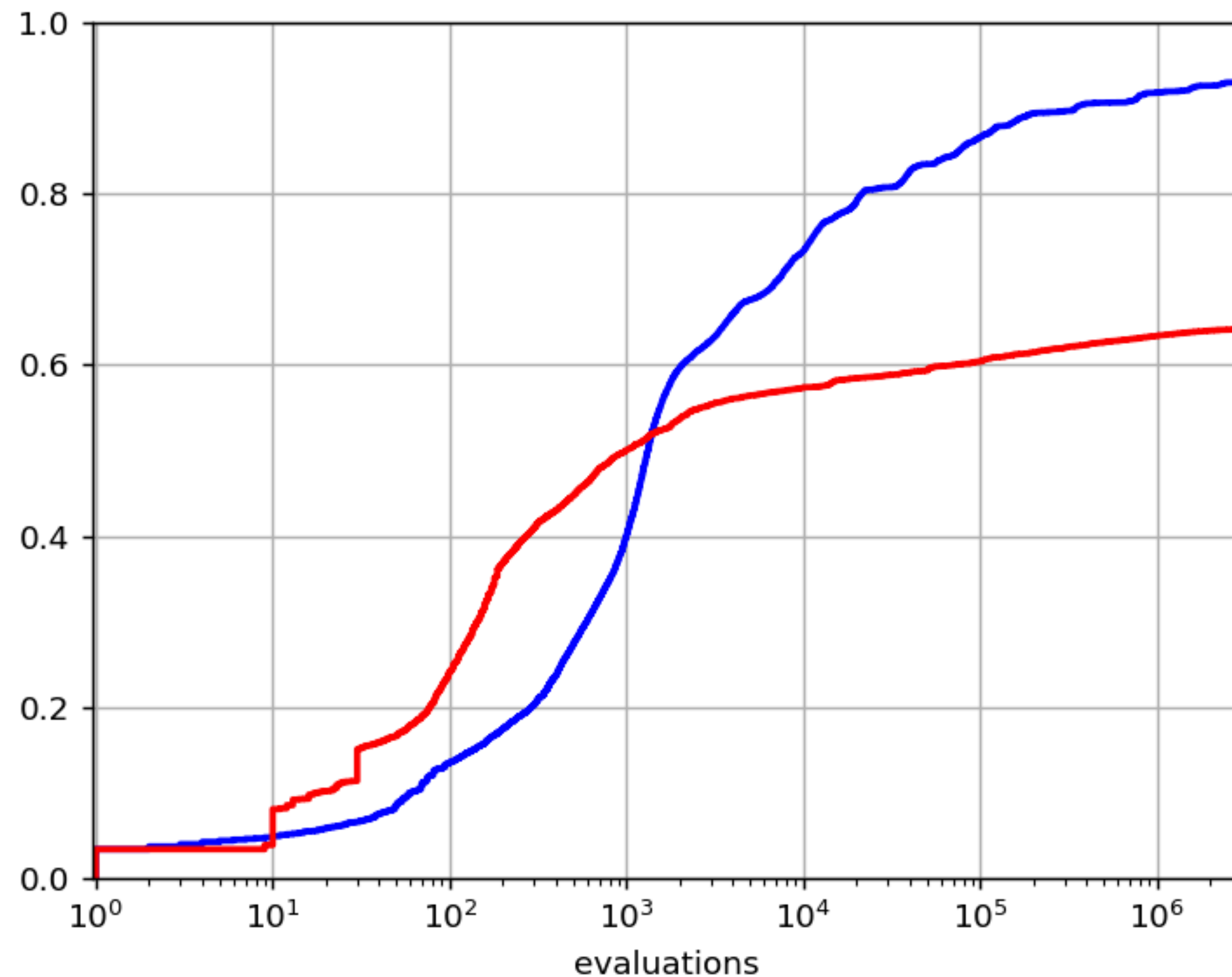
is the area between the runtime distribution graphs of two algorithms A,B

*is the geometric average when the x-axis is in log-scale
with the geometric average it is invariant under the exchange of operators:
the ratio of the averages equals the average of the ratios*

- The **average runtime ratio**

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

is the area between the runtime distribution graphs of two algorithms A,B
is the geometric average when the x-axis is in log-scale
with the geometric average it is invariant under the exchange of operators:
the ratio of the averages equals the average of the ratios



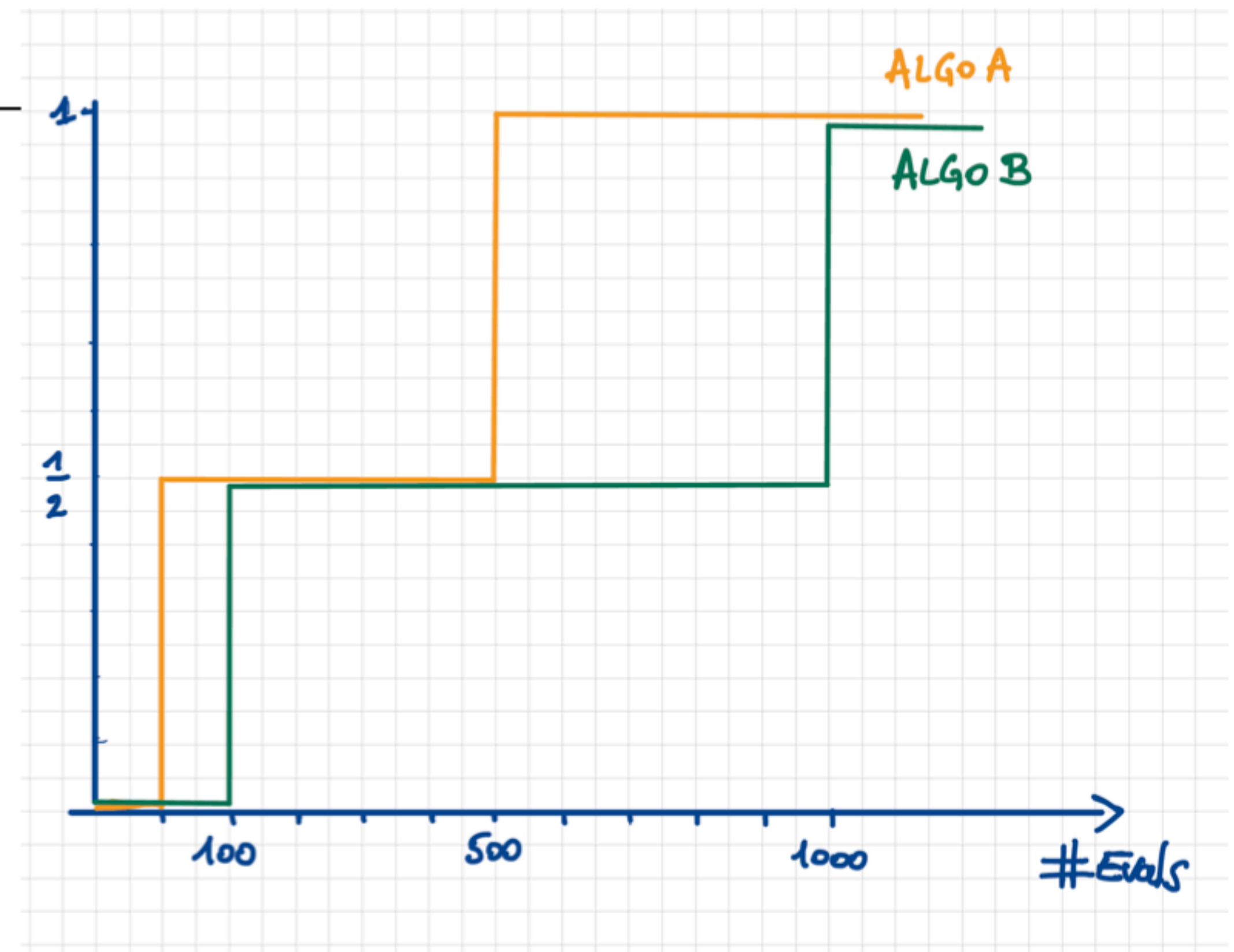
Discussion of Aggregation (Caveat)

	Problem 1	Problem 2
Algorithm A	50	500
Algorithm B	1000	100

Algorithm B = 5 x faster Algorithm A

Algorithm A = 20 x faster Algorithm B

Domination in each point of an empirical runtime distribution does not imply equal or better performance on each problem!



Take Home Messages

- Select a **balanced testbed**

*furious activity is no substitute for understanding
using “all functions” is likely to introduce a bias
(too many simple or low dimensional problems)*

- Use **quantitative measurements**

*which should preferably be comparable across publications
empirical CDFs are a very useful tool*

- Don't aggregate over attributes that are simple to determine

like dimension

- Benchmarking is **tedious but necessary**

use a provided platform?

Using COCO

Running an experiment

```
$ ### get and install the code
$ git clone https://github.com/numbbo/coco.git # get coco using git
$ cd coco
$ python do.py run-python # install Python experimental module cocoex
$ python do.py install-postprocessing install-user # install postprocessing :-)
```

```
$ ### (optional) run an example from the shell
$ mkdir my-first-experiment
$ cd my-first-experiment
$ cp ../code-experiments/build/python/example_experiment2.py .
$ python example_experiment2.py # run the current "default" experiment
$ # and the post-processing
$ # and open browser when finished
```

```
#!/usr/bin/env python
"""Python script to benchmark fmin of scipy.optimize"""
from __future__ import division # not needed in Python 3
import cocoex, cocopp # experimentation and post-processing modules
import scipy.optimize # to define the solver to be benchmarked

### input
suite_name = "bbob"
output_folder = "scipy-optimize-fmin"
```

```
#!/usr/bin/env python
"""Python script to benchmark fmin of scipy.optimize"""
from __future__ import division # not needed in Python 3
import cocoex, cocopp # experimentation and post-processing modules
import scipy.optimize # to define the solver to be benchmarked

### input
suite_name = "bbob"
output_folder = "scipy-optimize-fmin"
fmin = scipy.optimize.fmin
budget_multiplier = 2 # increase to 10, 100, ...

### prepare
suite = cocoex.Suite(suite_name, "", "")
observer = cocoex.Observer(suite_name, "result_folder: " + output_folder)

### go
for problem in suite: # this loop will take several minutes or longer
    problem.observe_with(observer) # will generate the data for cocopp
    # restart until the problem is solved or the budget is exhausted
    while (not problem.final_target_hit and
           problem.evaluations < problem.dimension * budget_multiplier):
        fmin(problem, problem.initial_solution_proposal())
    # we assume that 'fmin' evaluates the final/returned solution
```

```
import cocoex, cocopp # experimentation and post-processing modules
import scipy.optimize # to define the solver to be benchmarked

### input
suite_name = "bbob"
output_folder = "scipy-optimize-fmin"
fmin = scipy.optimize.fmin
budget_multiplier = 2 # increase to 10, 100, ...

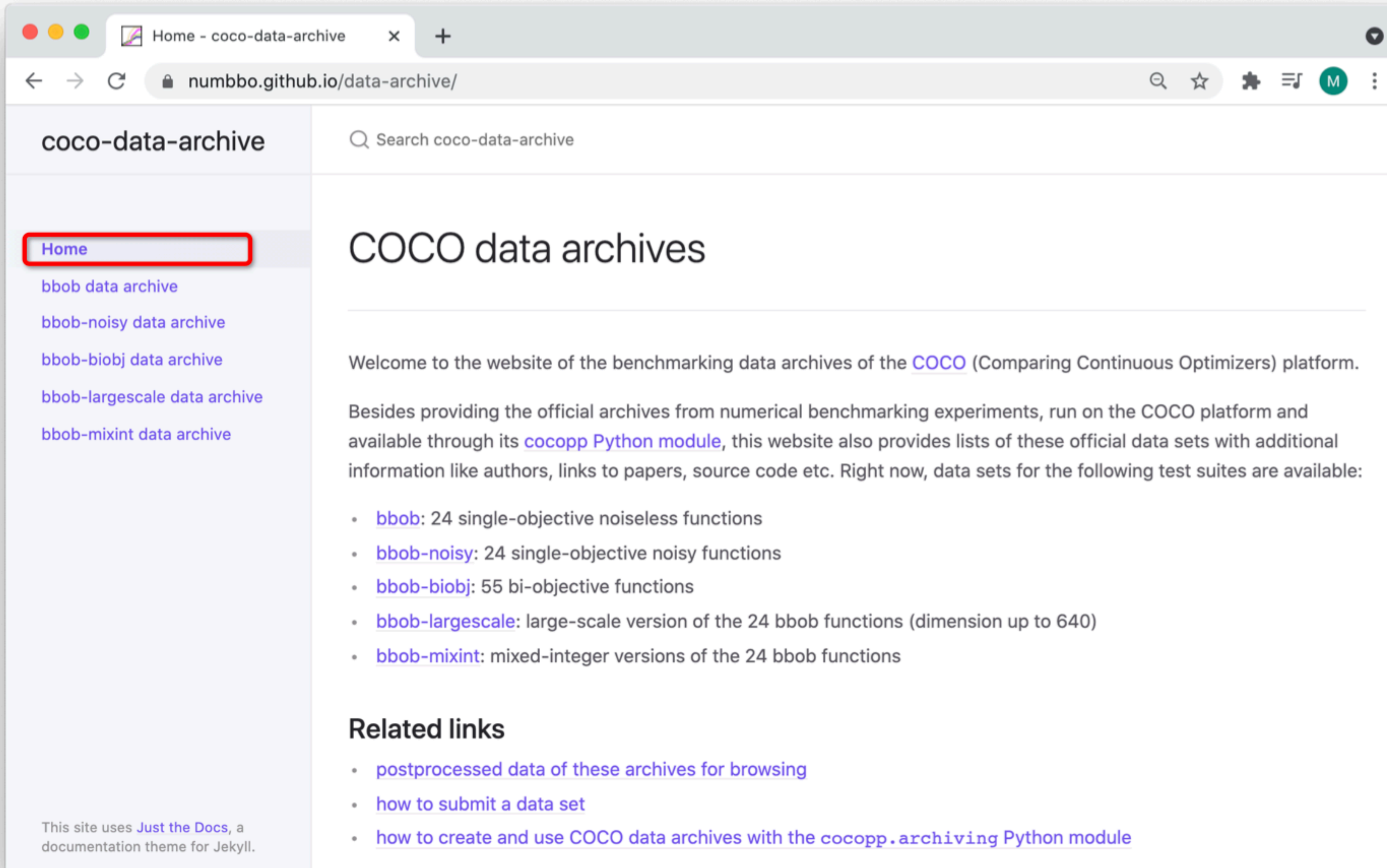
### prepare
suite = cocoex.Suite(suite_name, "", "")
observer = cocoex.Observer(suite_name, "result_folder: " + output_folder)

### go
for problem in suite: # this loop will take several minutes or longer
    problem.observe_with(observer) # will generate the data for cocopp
    # restart until the problem is solved or the budget is exhausted
    while (not problem.final_target_hit and
           problem.evaluations < problem.dimension * budget_multiplier):
        fmin(problem, problem.initial_solution_proposal())
        # we assume that 'fmin' evaluates the final/returned solution

### post-process data
cocopp.main(observer.result_folder) # re-run folders look like "...-001" etc
```

Selecting algorithms for comparison

Using COCO



The screenshot shows a web browser window with the URL `numbbo.github.io/data-archive/`. The page title is "coco-data-archive" and the main heading is "COCO data archives". The left sidebar contains a navigation menu with "Home" highlighted in a red box, and other links for "bbob data archive", "bbob-noisy data archive", "bbob-biobj data archive", "bbob-largescale data archive", and "bbob-mixint data archive". The main content area includes a search bar, a welcome message, a list of available test suites, and related links.

Home - coco-data-archive

numbbo.github.io/data-archive/

coco-data-archive

Search coco-data-archive

COCO data archives

Welcome to the website of the benchmarking data archives of the [COCO](#) (Comparing Continuous Optimizers) platform.

Besides providing the official archives from numerical benchmarking experiments, run on the COCO platform and available through its [cocopp Python module](#), this website also provides lists of these official data sets with additional information like authors, links to papers, source code etc. Right now, data sets for the following test suites are available:

- [bbob](#): 24 single-objective noiseless functions
- [bbob-noisy](#): 24 single-objective noisy functions
- [bbob-biobj](#): 55 bi-objective functions
- [bbob-largescale](#): large-scale version of the 24 bbob functions (dimension up to 640)
- [bbob-mixint](#): mixed-integer versions of the 24 bbob functions

Related links

- [postprocessed data of these archives for browsing](#)
- [how to submit a data set](#)
- [how to create and use COCO data archives with the `cocopp.archiving` Python module](#)

This site uses [Just the Docs](#), a documentation theme for Jekyll.

Using COCO

The screenshot shows a web browser window with the address bar containing `numbbio.github.io/data-archive/bbob/`. The page title is "coco-data-archive" and it features a search bar. The main heading is "Algorithm data sets for the bbob test suite". Below the heading, there is explanatory text and a table of data sets. The table has columns for Number, Algorithm Name, Year, Author(s), link to data, and related PDFs, source code, etc. The table lists five data sets: ALPS, AMALGAM, BAYEDA, BFGS, and BIPOP-CMA-ES.

Home
bbob data archive
bbob-noisy data archive
bbob-biobj data archive
bbob-largescale data archive
bbob-mixint data archive

Algorithm data sets for the bbob test suite

In the first table below, you will find all official algorithm data sets on the bbob test suite, together with their year of publication, the authors, and related PDFs for each data set. Links to the source code to run the corresponding experiments/algorithms are provided whenever available.

A second table mentions data sets that have been collected on the bbob suite, but which are not complete in the sense that they miss at least one of the requested dimensions 2, 3, 5, 10, 20.

To sort the tables, simply click on the table header of the corresponding column.

Number	Algorithm Name	Year	Author(s)	link to data	related PDFs, source code, etc.
000	ALPS	2009	Hornby	data	pdf
001	AMALGAM	2009	Bosman et al.	data	pdf noiseless - pdf noisy
002	BAYEDA	2009	Gallagher	data	pdf noiseless - pdf noisy
003	BFGS	2009	Ros	data	pdf noiseless - pdf noisy
004	BIPOP-CMA-ES	2009	Hansen	data	pdf noiseless - pfd noisy

This site uses [Just the Docs](#), a documentation theme for Jekyll.

Visit <https://numbbio.github.io/data-archive/>

https://numbbo.github.io/ppdata x +

numbbo.github.io/ppdata-archive/

COCO ppdata-archive

This archive contains *postprocessed* data displaying benchmarking experiments of various numerical optimization algorithms on the various test suites provided by the [Comparing Continuous Optimizers platform](#). Experiments are conducted in a blackbox setting and data are collected by year.

bbob	bbob-noisy	bbob-biobj	bbob-largescale	bbob-mixint
24 functions single-objective continous domain 200+ algorithm data sets	30 functions noisy evaluations single-objective 45 algorithm data sets	55 functions bi-objective noiseless 32 algorithm data sets	24 bbob functions single-objective dimensions 20 to 640 11 algorithm data sets	24 functions 80% discrete variables single-objective no official data yet
2009 2010 2012 2013 2014 2015-CEC 2015-GECCO 2016 2017 2018 2019	2009 2010 2012 2016	2016 2017 2019	2019	

Visit <https://numbbo.github.io/ppdata-archive/>

[Home](#)

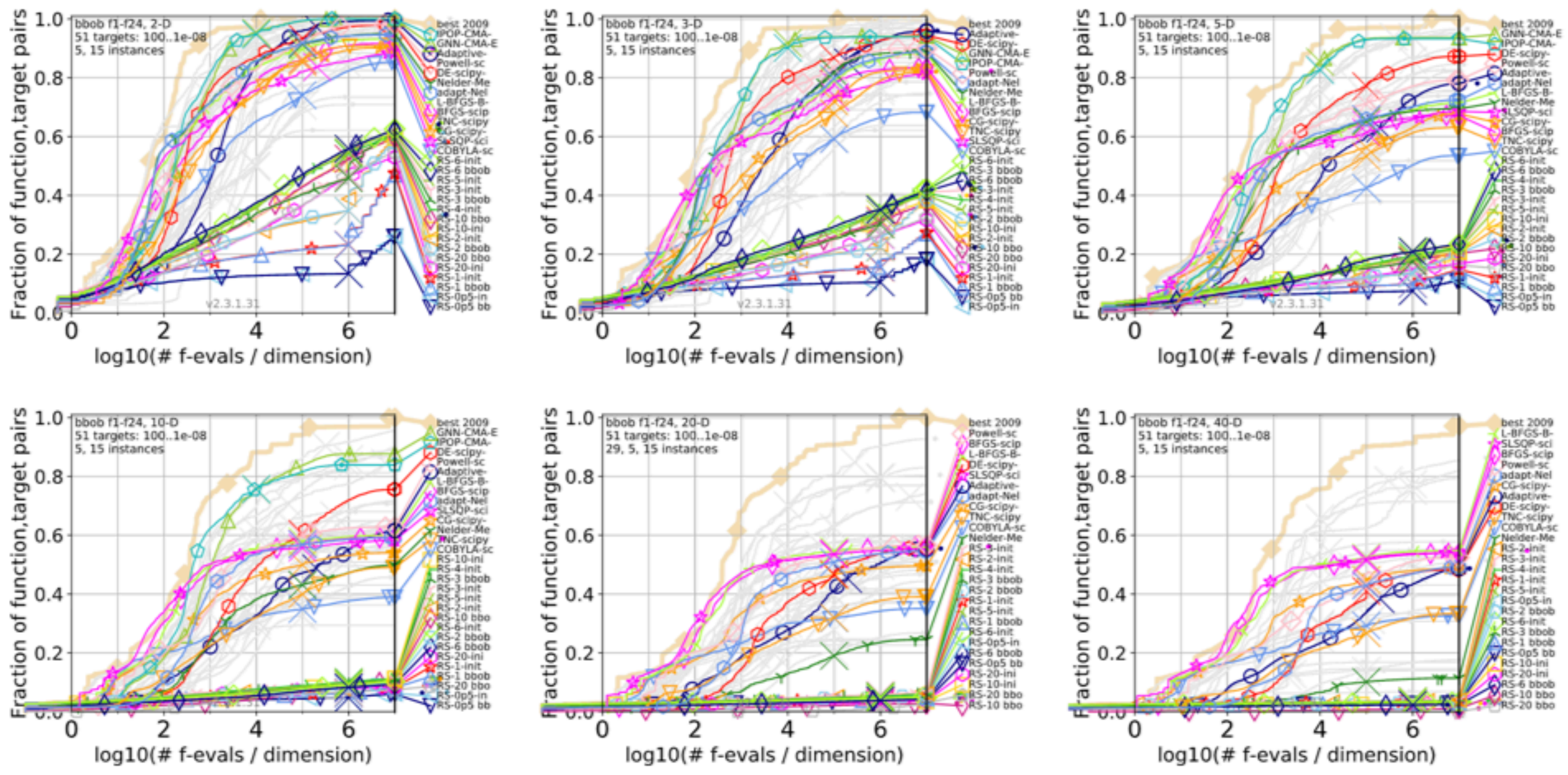
[Runtime distributions \(ECDFs\) per function](#)

[Runtime distributions \(ECDFs\) summary and function groups](#)

[Scaling with dimension](#)

[Tables for selected targets](#)

Runtime distributions (ECDFs) over all targets



Running the postprocessing

```
benchmarking-tutorial-20 x benchmarking-tutorial-20 convergence-to-ecdf-plot x Python 3
[1]: 1 %pylab
Using matplotlib backend: TkAgg
Populating the interactive namespace from numpy and matplotlib

[2]: 1 try:
2     import cocopp
3 except ImportError:
4     !pip install cocopp
5     import cocopp

[3]: 1 cocopp.archives.bbob('bfgs')

[3]: ['2009/BFGS_ros_noiseless.tgz',
'2012/DE-BFGS_voglis_noiseless.tgz',
'2012/PSO-BFGS_voglis_noiseless.tgz',
'2014-others/BFGS-scipy_Baudis.tgz',
'2014-others/L-BFGS-B-scipy_Baudis.tgz',
'2018/BFGS-M-17_Blelly.tgz',
'2018/BFGS-P-09_Blelly.tgz',
'2018/BFGS-P-Instances_Blelly.tgz',
'2018/BFGS-P-StPt_Blelly.tgz',
'2018/BFGS-P-range_Blelly.tgz',
'2019/BFGS-scipy-2019_Varelas.tgz',
'2019/L-BFGS-B-scipy-2019_Varelas.tgz']

[4]: 1 cocopp.archives.bbob('slsq')

[4]: ['2014-others/SLSQP-scipy_Baudis.tgz',
'2019/SLSQP-scipy-2019_Varelas.tgz',
'2020/SLSQP+lq-CMA-ES_Hansen.tgz',
'2020/SLSQP-11-scipy_Hansen.tgz']

[5]: 1 cocopp.archives.bbob('nelder')

[5]: ['2009/NELDERDOERR_doerr_noiseless.tgz',
'2009/NELDER_hansen_noiseless.tgz',
'2014-others/Nelder-Mead-scipy_Baudis.tgz',
'2019/Nelder-Mead-scipy-2019_Varelas.tgz',
'2019/adapt-Nelder-Mead-scipy-2019_Varelas.tgz']

[6]: 1 cocopp.archiving.ArchivesKnown()

[6]: ['http://coco.gforge.inria.fr/data-archive',
'http://coco.gforge.inria.fr/data-archive/bbob',
'http://lq-cma.gforge.inria.fr/data-archives/lq-gecco2019']

[7]: 1 # any URL containing a "valid" COCO archive is eligible
2 lqarch = cocopp.archiving.get('http://lq-cma.gforge.inria.fr/data-archives/lq-gecco2019')
3 lqarch

[7]: ['CMA-ES_2019-gecco-surr.tgz',
'SLSQP+CMA_2019-gecco-surr.tgz',
'SLSQP-11_2019-gecco-surr.tgz',
'lq-CMA-ES_2019-gecco-surr.tgz']

[*]: 1 cocopp.main('BFGS_ros! BFGS-P-St /NEWU0! nelderdoerr! SLSQP-11- 19/SLSQP-sci mcs! ' + lqarch.get('lq-cma'));

Post-processing (2+)
Using 8 data sets:
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archive/bbob/2009/BFGS_ros_noiseless.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archive/bbob/2018/BFGS-P-StPt_Blelly.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2009/NELDERDOERR_doerr_noiseless.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2009/NELDER_hansen_noiseless.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2014-others/Nelder-Mead-scipy_Baudis.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2019/Nelder-Mead-scipy-2019_Varelas.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2019/adapt-Nelder-Mead-scipy-2019_Varelas.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2012/DE-BFGS_voglis_noiseless.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2012/PSO-BFGS_voglis_noiseless.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2014-others/BFGS-scipy_Baudis.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2014-others/L-BFGS-B-scipy_Baudis.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2018/BFGS-M-17_Blelly.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2018/BFGS-P-09_Blelly.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2018/BFGS-P-Instances_Blelly.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2018/BFGS-P-StPt_Blelly.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2018/BFGS-P-range_Blelly.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2019/BFGS-scipy-2019_Varelas.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/bbob/2019/L-BFGS-B-scipy-2019_Varelas.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/lq-gecco2019/lq-CMA-ES_2019-gecco-surr.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/lq-gecco2019/SLSQP+CMA_2019-gecco-surr.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/lq-gecco2019/SLSQP-11_2019-gecco-surr.tgz
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archives/lq-gecco2019/lq-CMA-ES_2019-gecco-surr.tgz
```

```
[1]: 1 %pylab
```

```
Using matplotlib backend: TkAgg  
Populating the interactive namespace from numpy and matplotlib
```

```
[2]: 1 try:  
2     import cocopp  
3 except ImportError:  
4     !pip install cocopp  
5     import cocopp
```

```
[3]: 1 cocopp.archives.bbob('bfgs')
```

```
[3]: ['2009/BFGS_ros_noiseless.tgz',  
      '2012/DE-BFGS_voglis_noiseless.tgz',  
      '2012/PSO-BFGS_voglis_noiseless.tgz',  
      '2014-others/BFGS-scipy_Baudis.tgz',  
      '2014-others/L-BFGS-B-scipy_Baudis.tgz',  
      '2018/BFGS-M-17_Blelly.tgz',  
      '2018/BFGS-P-09_Blelly.tgz',  
      '2018/BFGS-P-Instances_Blelly.tgz',  
      '2018/BFGS-P-StPt_Blelly.tgz',  
      '2018/BFGS-P-range_Blelly.tgz',  
      '2019/BFGS-scipy-2019_Varelas.tgz',  
      '2019/L-BFGS-B-scipy-2019_Varelas.tgz']
```

```
[4]: 1 cocopp.archives.bbob('slsq')
```

```
[4]: ['2014-others/SLSQP-scipy_Baudis.tgz',  
      '2019/SLSQP-scipy-2019_Varelas.tgz',  
      '2020/SLSQP+lq-CMA-ES_Hansen.tgz',  
      '2020/SLSQP-11-scipy_Hansen.tgz']
```

```
[4]: 1 cocopp.archives.bbop('slsq')
```

```
[4]: ['2014-others/SLSQP-scipy_Baudis.tgz',  
      '2019/SLSQP-scipy-2019_Varelas.tgz',  
      '2020/SLSQP+lq-CMA-ES_Hansen.tgz',  
      '2020/SLSQP-11-scipy_Hansen.tgz']
```

```
[5]: 1 cocopp.archives.bbop('nelder')
```

```
[5]: ['2009/NELDERDOERR_doerr_noiseless.tgz',  
      '2009/NELDER_hansen_noiseless.tgz',  
      '2014-others/Nelder-Mead-scipy_Baudis.tgz',  
      '2019/Nelder-Mead-scipy-2019_Varelas.tgz',  
      '2019/adapt-Nelder-Mead-scipy-2019_Varelas.tgz']
```

```
[6]: 1 cocopp.archiving.ArchivesKnown()
```

```
[6]: ['http://coco.gforge.inria.fr/data-archive',  
      'http://coco.gforge.inria.fr/data-archive/bbob',  
      'http://lq-cma.gforge.inria.fr/data-archives/lq-gecco2019']
```

```
[7]: 1 # any URL containing a "valid" COCO archive is eligible  
     2 lqarch = cocopp.archiving.get('https://cma-es.github.io/lq-cma/data-archives/lq-gecco2019')  
     3 lqarch
```

```
[7]: ['CMA-ES__2019-gecco-surr.tgz',  
      'SLSQP+CMA_2019-gecco-surr.tgz',  
      'SLSQP-11_2019-gecco-surr.tgz',  
      'lq-CMA-ES_2019-gecco-surr.tgz']
```

```
[*]: 1 cocopp.main('BFGS_ros! BFGS-P-St /NEWUO! nelderdoerr! SLSQP-11- 19/SLSQP-sci mcs! ' + lqarch.get('lq-cma'));
```

Post-processing (2+)

Using 8 data sets:

```
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archive/bbob/2009/BFGS_ros_noiseless.tgz  
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archive/bbob/2018/BFGS-P-StPt_Blelly.tgz  
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archive/bbob/2009/NEWUOA ros noiseless.tgz
```



```
[4]: 1 cocopp.archives.bbop('slsq')
```

```
[4]: ['2014-others/SLSQP-scipy_Baudis.tgz',  
      '2019/SLSQP-scipy-2019_Varelas.tgz',  
      '2020/SLSQP+lq-CMA-ES_Hansen.tgz',  
      '2020/SLSQP-11-scipy_Hansen.tgz']
```

```
[5]: 1 cocopp.archives.bbop('nelder')
```

```
[5]: ['2009/NELDERDOERR_doerr_noiseless.tgz',  
      '2009/NELDER_hansen_noiseless.tgz',  
      '2014-others/Nelder-Mead-scipy_Baudis.tgz',  
      '2019/Nelder-Mead-scipy-2019_Varelas.tgz',  
      '2019/adapt-Nelder-Mead-scipy-2019_Varelas.tgz']
```

```
[6]: 1 cocopp.archiving.ArchivesKnown()
```

```
[6]: ['http://coco.gforge.inria.fr/data-archive',  
      'http://coco.gforge.inria.fr/data-archive/bbob',  
      'http://lq-cma.gforge.inria.fr/data-archives/lq-gecco2019']
```

```
[7]: 1 # any URL containing a "valid" COCO archive is eligible  
2 lqarch = cocopp.archiving.get('https://cma-es.github.io/lq-cma/data-archives/lq-gecco2019')  
3 lqarch
```

```
[7]: ['CMA-ES__2019-gecco-surr.tgz',  
      'SLSQP+CMA_2019-gecco-surr.tgz',  
      'SLSQP-11_2019-gecco-surr.tgz',  
      'lq-CMA-ES_2019-gecco-surr.tgz']
```

```
[*]: 1 cocopp.main('BFGS_ros! BFGS-P-St /NEWUO! nelderdoerr! SLSQP-11- 19/SLSQP-sci mcs! ' + lqarch.get('lq-cma'));
```

Post-processing (2+)

Using 8 data sets:

```
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archive/bbob/2009/BFGS_ros_noiseless.tgz  
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archive/bbob/2018/BFGS-P-StPt_Blelly.tgz  
/Users/hansen/.cocopp/data-archives/coco.gforge.inria.fr/data-archive/bbob/2009/NEWUOA ros noiseless.tgz
```

Home

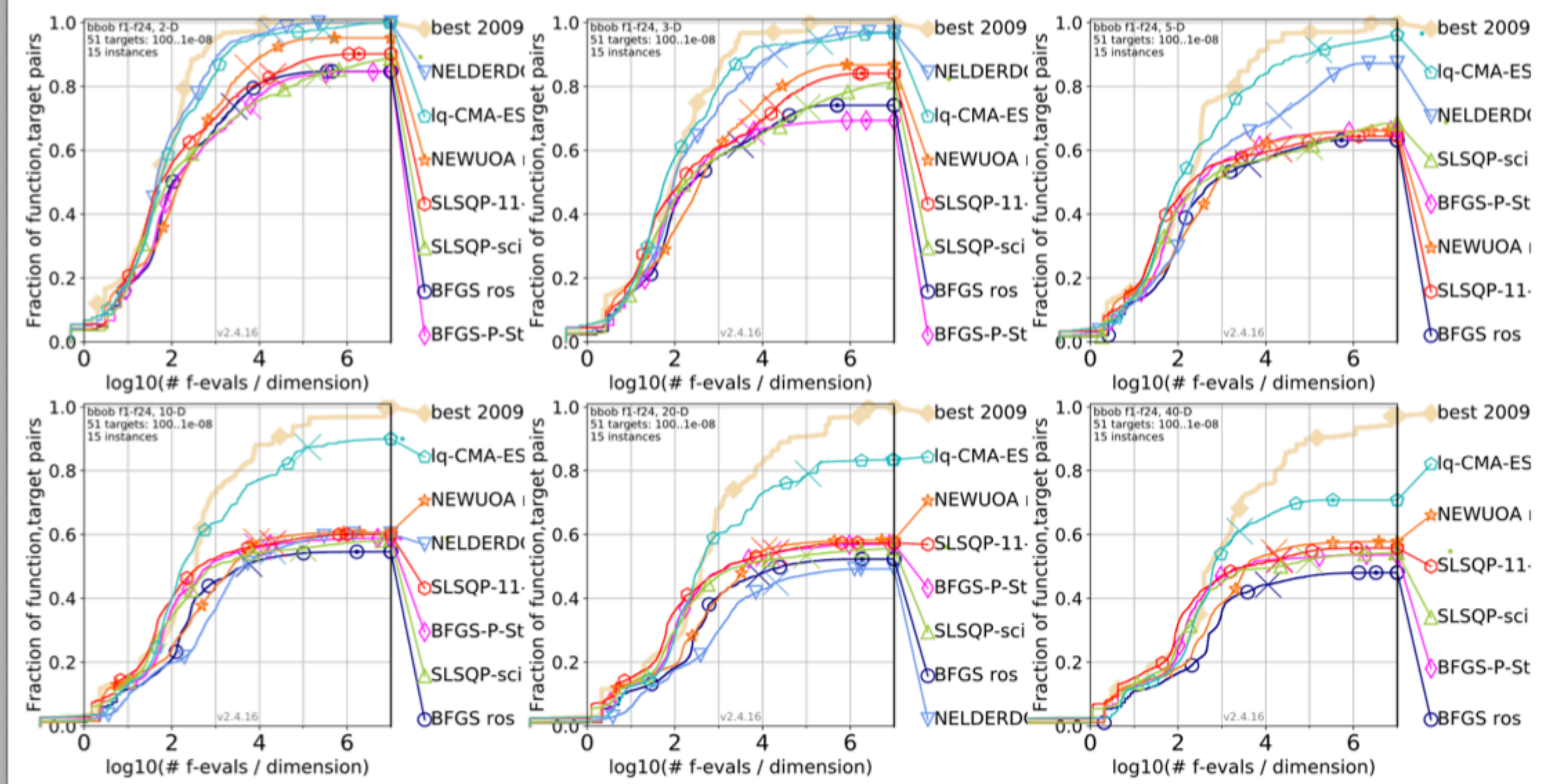
Runtime distributions (ECDFs) per function

Runtime distributions (ECDFs) summary and function groups

Scaling with dimension

Tables for selected targets

Runtime distributions (ECDFs) over all targets



Open "file:///Users/hansen/gitlab/nhansen/convergence-graph-to-ecdf/ppdata/B...ELDE_SLSQP_SLSQP_lq-CM_et_al/pprldmany_03D_noiselessall.svg" in a new tab

Data Sets and Usage Statistics

Table 1. Visibility of COCO. All citations as of November 19, 2019, in Google Scholar.

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 “ <i>comparing continuous optimizers</i> ” OR “ <i>black-box optimization benchmarking (BBOB)</i> ”		559
Citations to the COCO documentation including		1,455

Any `cocopp.archiving.create(folder)`-ed data sets provided under an URL can be loaded with `av = cocopp.archiving.get(URL)` and used in the data processing. See [Hansen et al 2020].

Take Home Messages

- Select a **balanced testbed**

*furious activity is no substitute for understanding
using “all functions” is likely to introduce a bias
(too many simple or low dimensional problems)*

- Use **quantitative measurements**

*which should preferably be comparable across publications
empirical CDFs are a very useful tool*

- Don't aggregate over attributes that are simple to determine

like dimension

- Benchmarking is **tedious but necessary**

use a provided platform?

Your questions?