Benchmarking: state-of-the-art and beyond

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

Full set of slides: http://www.cmap.polytechnique.fr/~nikolaus.hansen/gecco-2021-benchmarking-state.pdf

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3461424

…feel free to ask questions…

…feel free to ask questions…

Getting Random Things Out of the Way

"*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: **Generalization**

Does benchmarking make sense at all?

After all there is no free lunch. Or is there?

- A benchmark must attempt to model observable and relevant "real-world" optimization problems.

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

  NFLTs prove the existence of certain mathematical constructs.
  Whether these constructs are observable 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.

  AKA overfitting.

  The benchmarking setup: an algorithm that needs to repeatedly solve "new" problems.

- Invariance of algorithms is a relevant aspect to interpret benchmarking results

Getting Random Things Out of the Way

"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

- 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

  Crafting Effort correction for using different parameter settings on different functions[1]

- 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.

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.

Goals of Benchmarking

1. Understanding algorithms.

   Dedicated experimentation is often a better alternative.

2. Selecting algorithms to solve a given problem

3. Regression testing for changes in an algorithm or implementation

4. Measuring algorithm performance in a systematic and standardized way, creating a performance "profile"
   - standardized assessment
   - simplified comparison

5. Running a competition

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

The points 2–5 require to compare algorithms

Benchmarking: The Global Picture

Two *surprisingly (but not completely) independent* questions/decisions:

- **What to benchmark**? 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

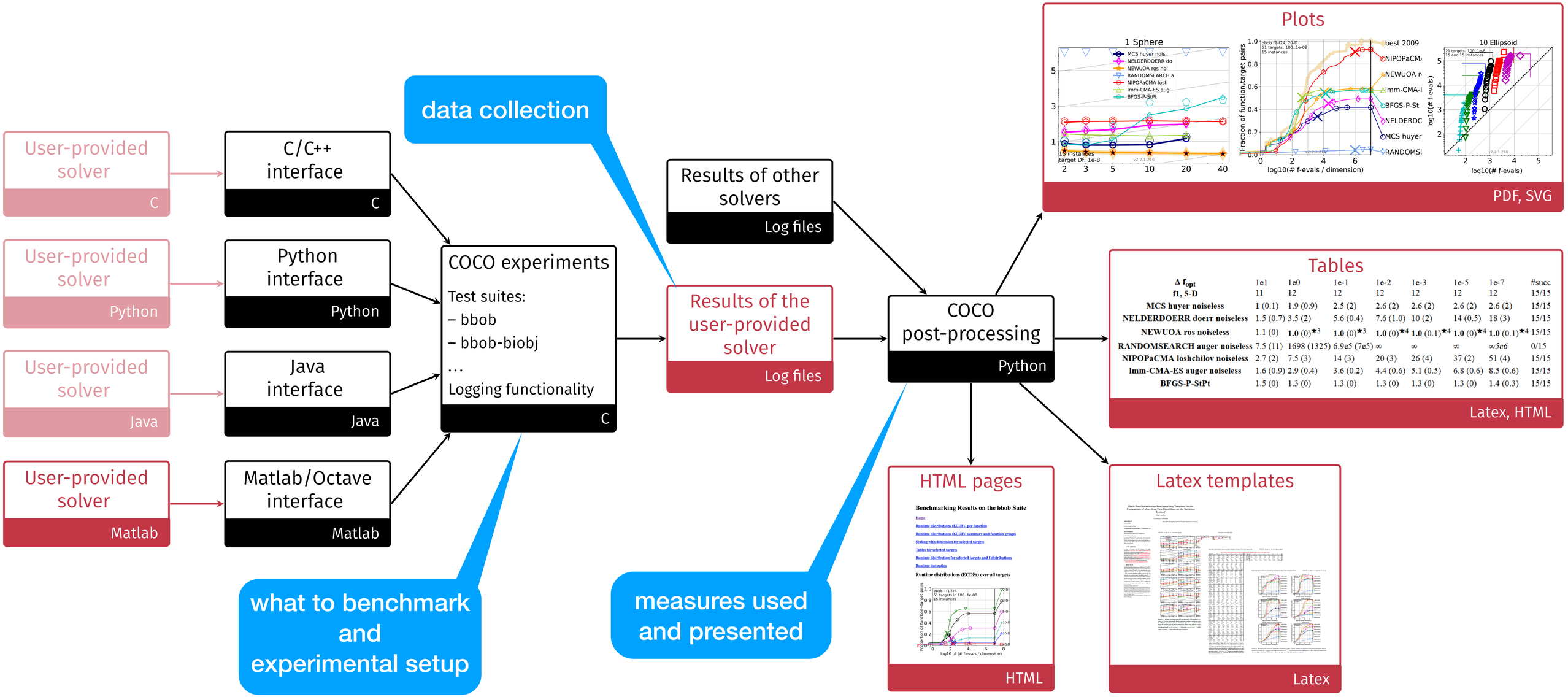


Figure by Tea Tušar, in Hansen et al (2021), COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1), 114-144.

What to 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 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?
- …

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, function as back-box

  not: function name/ID

- repetitions only work for randomized algorithms

- may define a budget (or not)

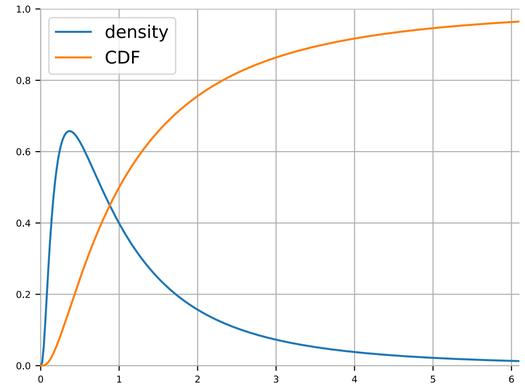
  anytime vs targeted budget

Handling and Displaying Empirical Data

Cumulative Distribution Function (CDF)

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

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



It characterizes the probability distribution of T

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

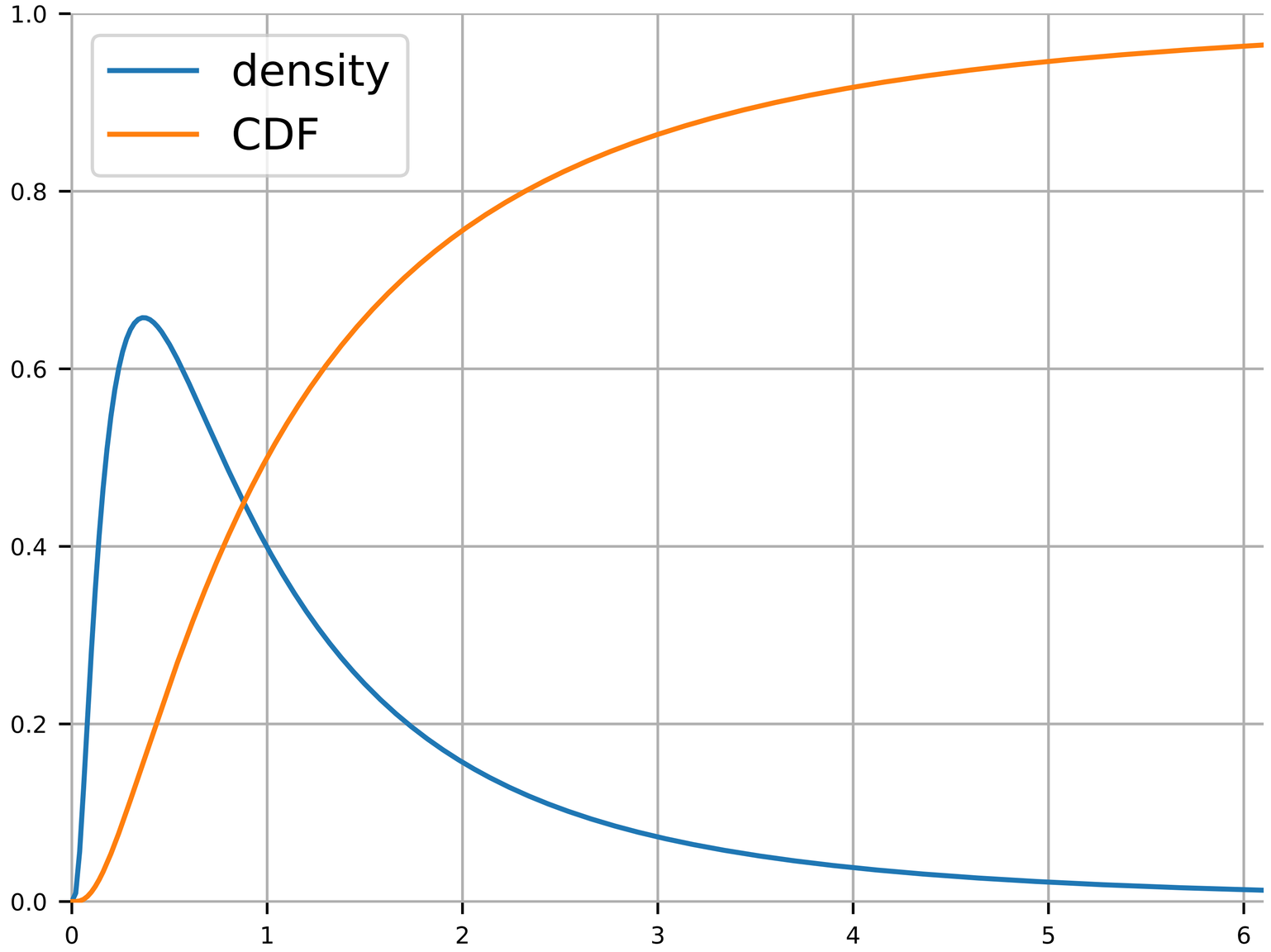
Cumulative Distribution Function (CDF)

om variable T, the cumulative dis
ned as

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

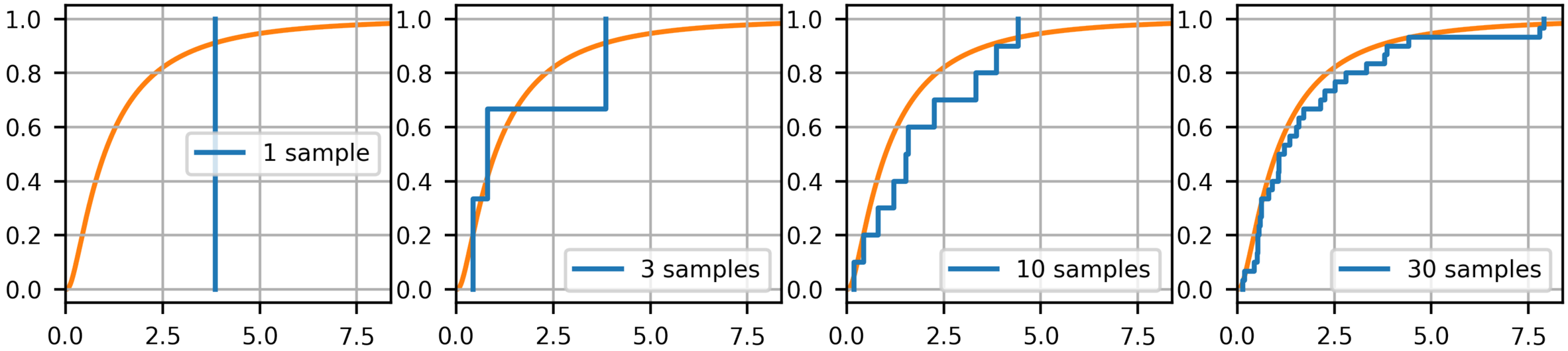
es the probability distribution of T

om variables have the same CDF



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.



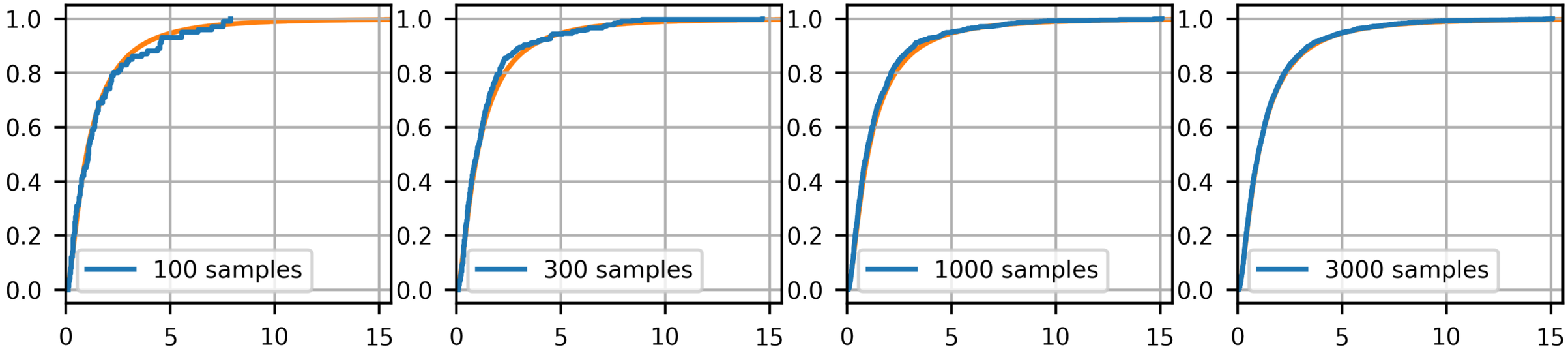
- It is an estimate of the CDF that generated the points in the sample.

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

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



- **How to assess performance**?
  - experimental setup

    depends to some extend on how we measure performance (e.g. max budget)
  - data collection

    depends to some extend on how we measure performance
  - measures used and presented

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

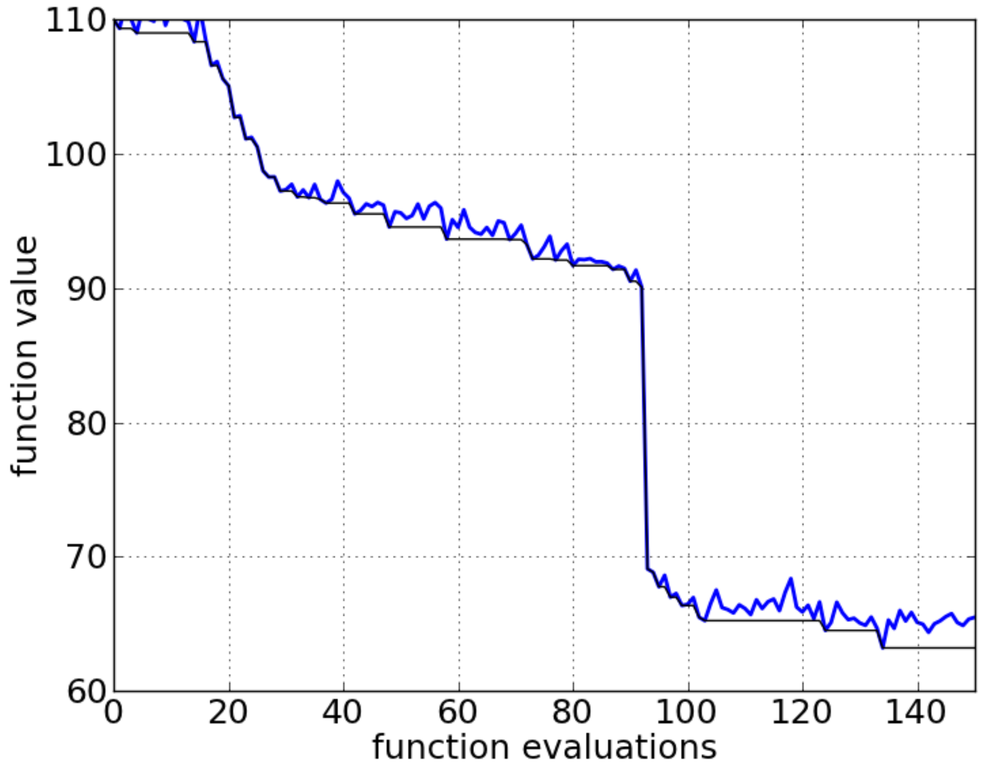
Scales of Measurement ("Quality" of Data)

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

Collecting Empirical Data

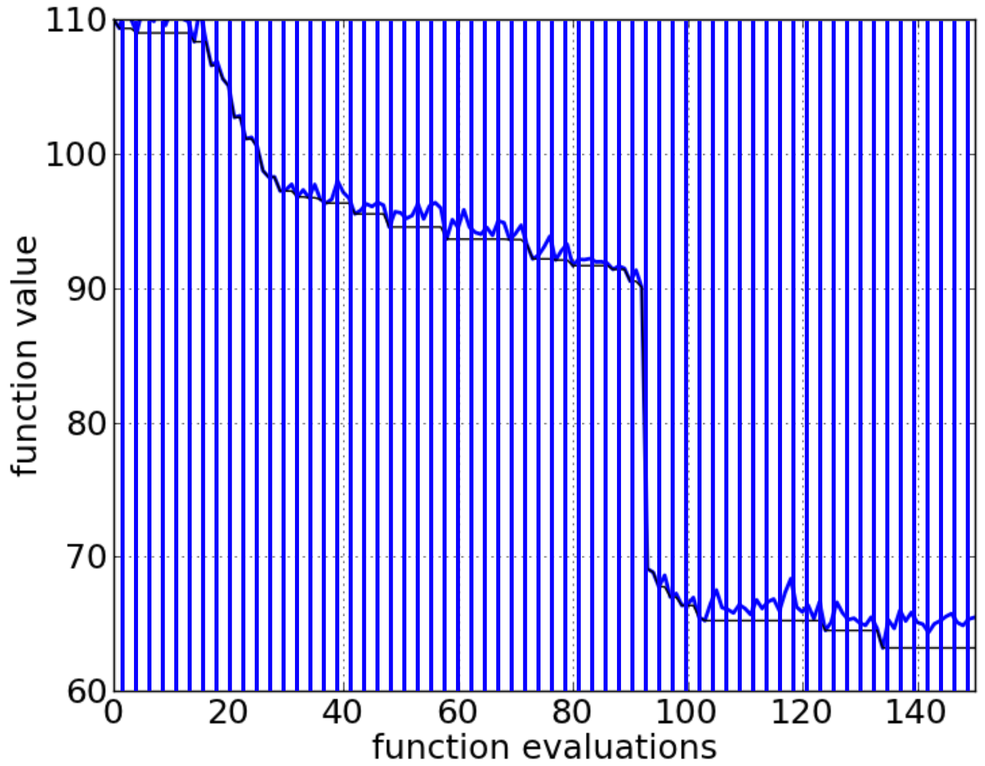
Convergence Graphs is All We Have



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

using the lower envelope is a practical choice

Discretization: Two Possibilities



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

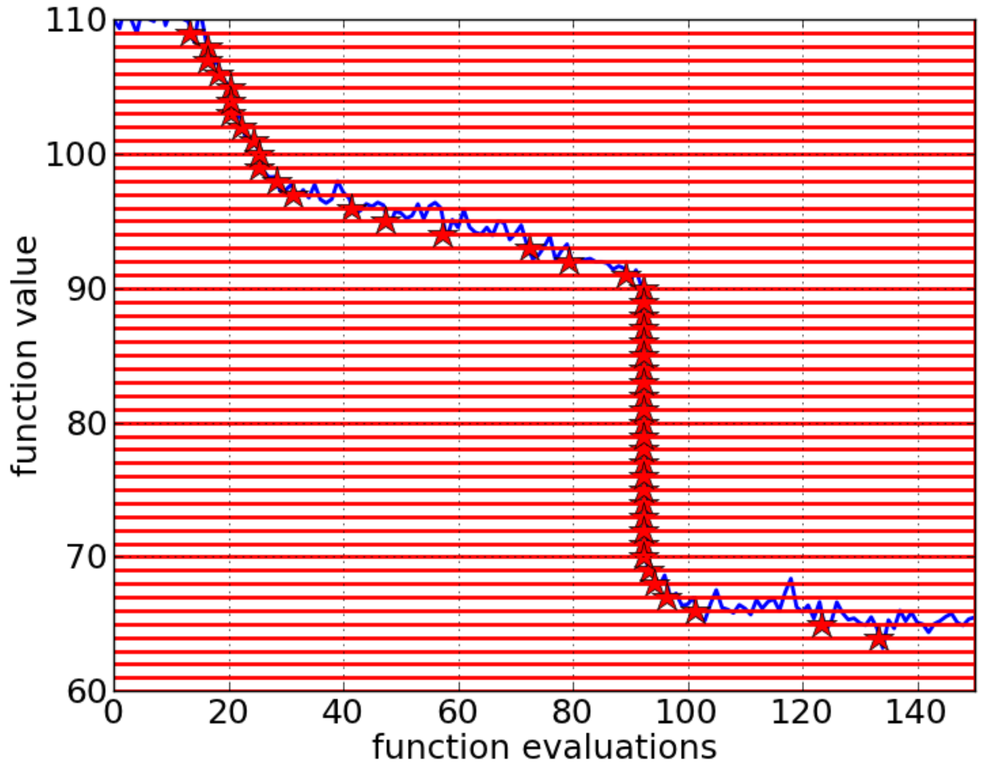
- **vertical**: by evaluation is a natural discretization

  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

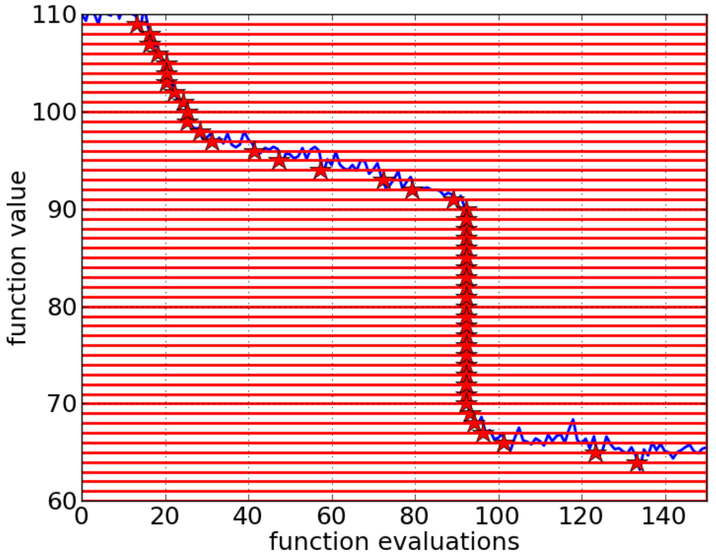
- function “target” values are the independent variable

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

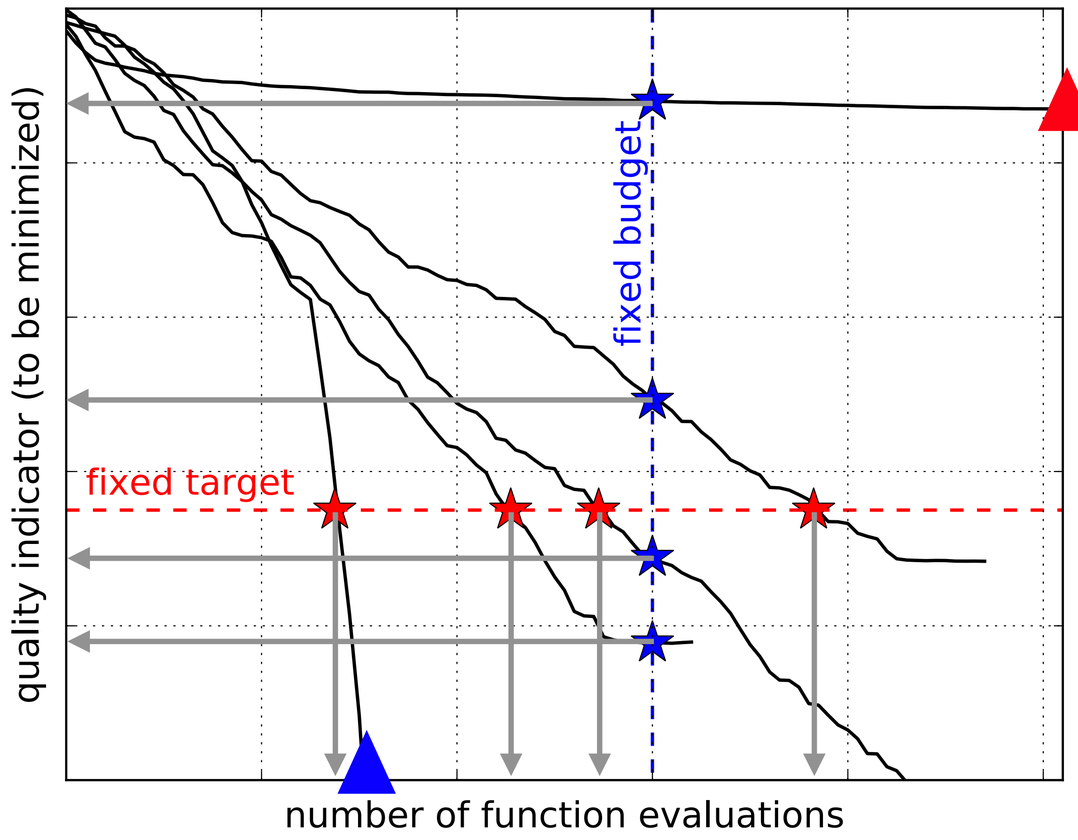
not

just

a technical subtlety

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

Fixed Target(s) versus Fixed Budget



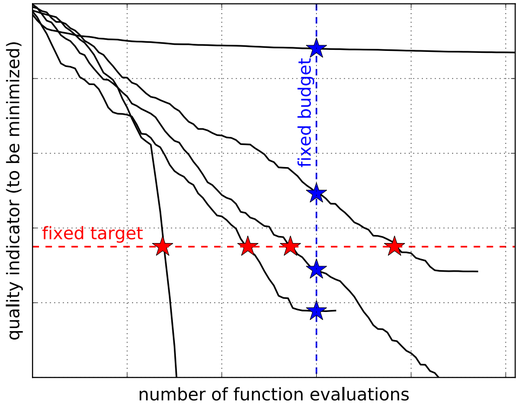
- five convergence graphs
  "quality indicator" versus "time"

- Leads to *different imprecise data* in both cases
  - "too" bad performance

    then the data only provide a lower bound estimate for the runtime (and a fixed budget measure at maximum budget)
  - "too" good performance

    (reached global optimum up to the relevant or numerical precision before the given budget)

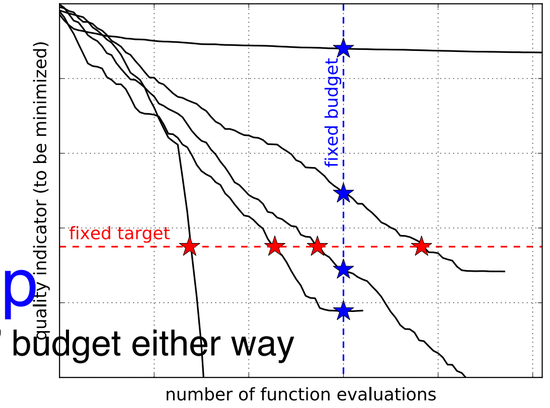
Fixed Target(s) versus Fixed Budget



The resulting measurement

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

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 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

- quantitative interpretation

  “Algorithm A is 100 times faster than Algorithm B”

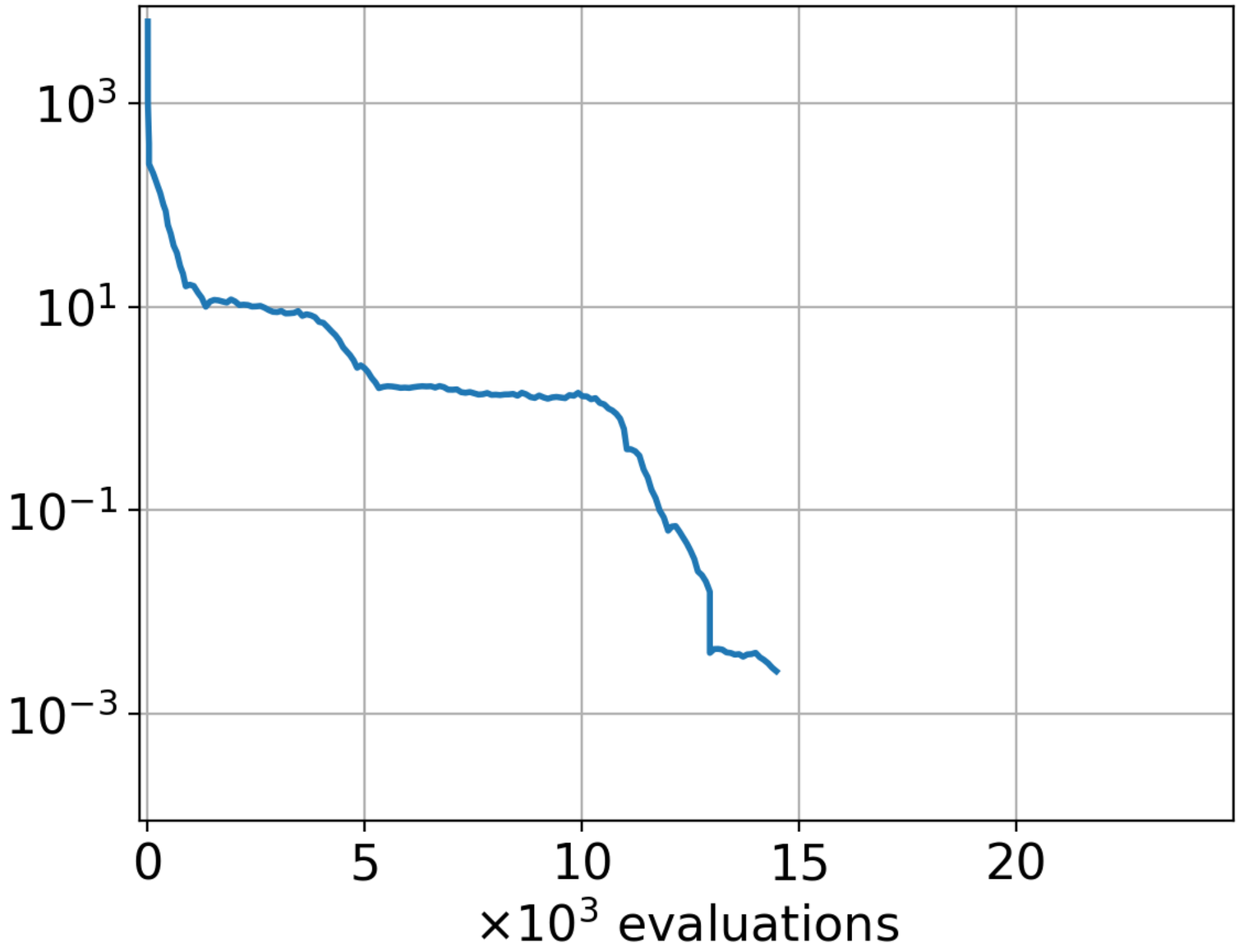
- Fixed target results are “budget-free”

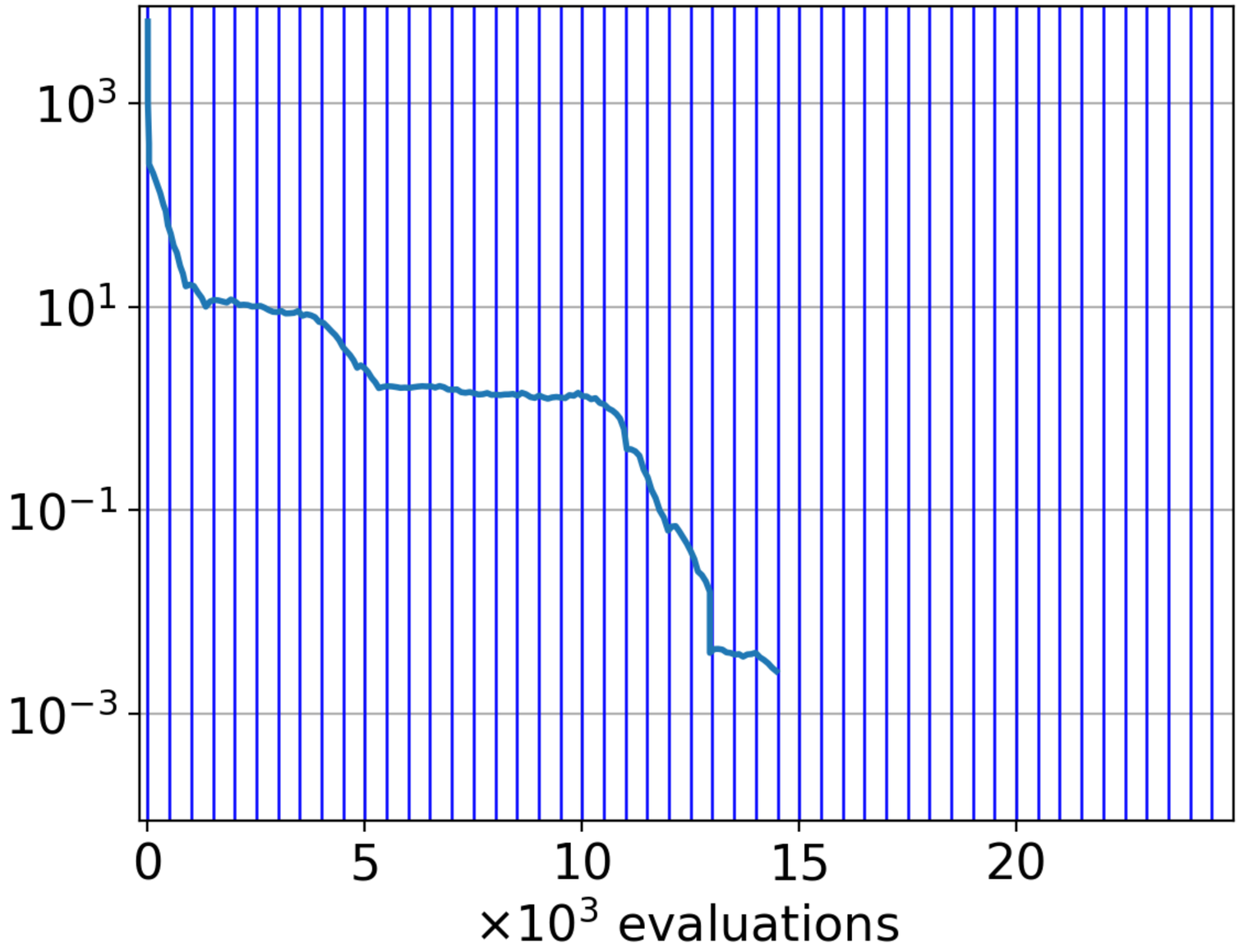
  we can compare results run with different maximal “timeout” budgets

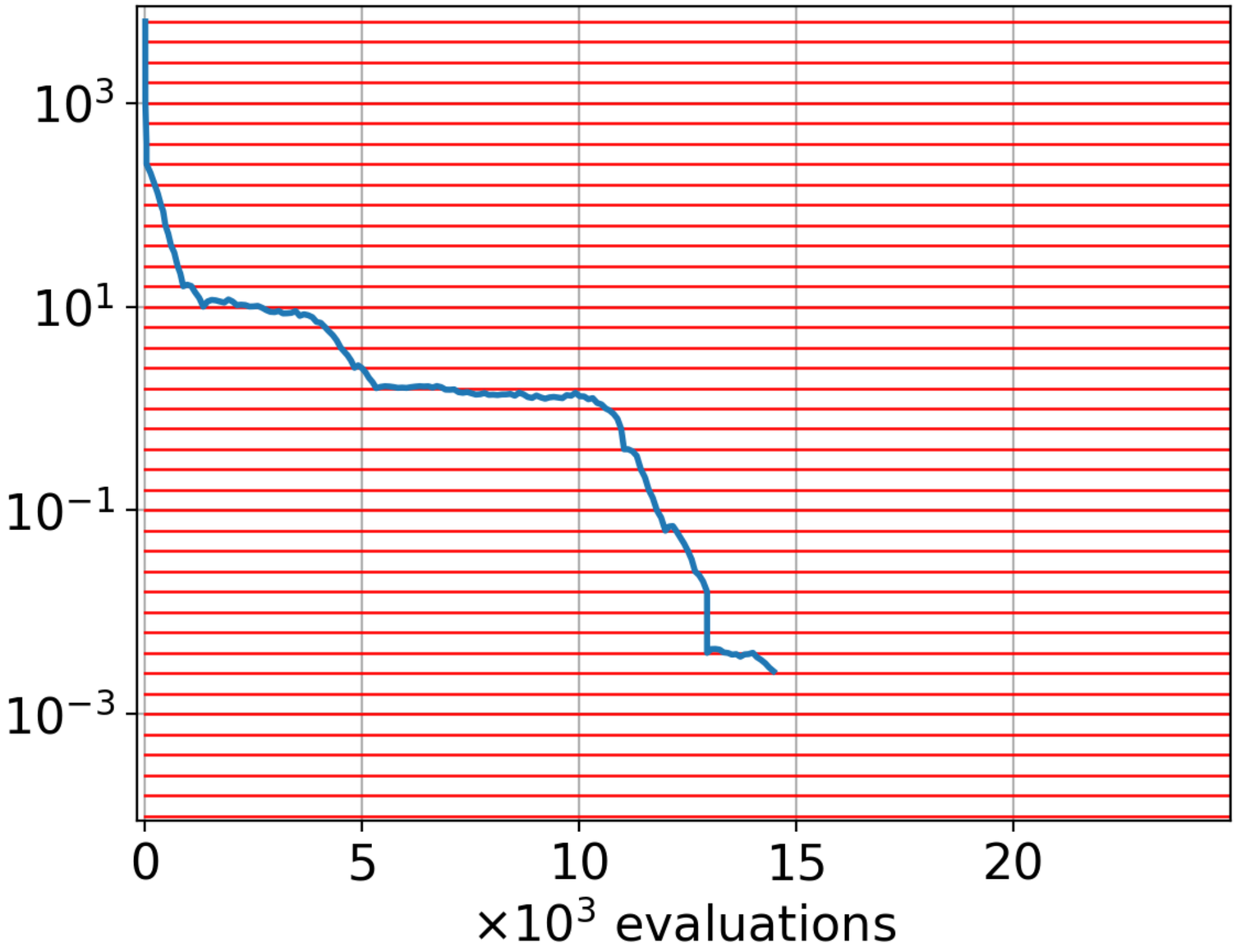
- Fixed target results can be meaningfully aggregated in ECDFs and geometric averages

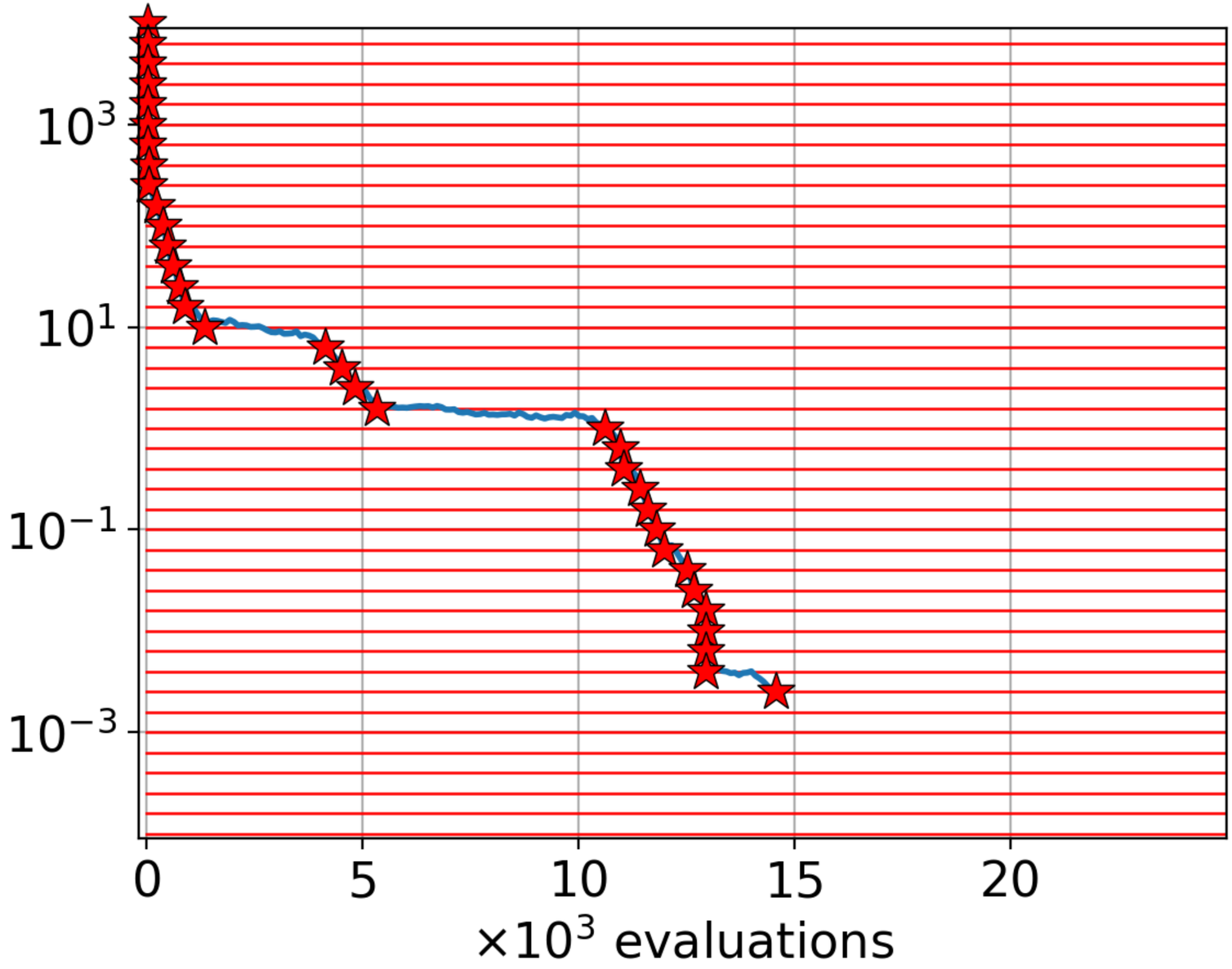
  whereas function values from different functions are in general not commensurable

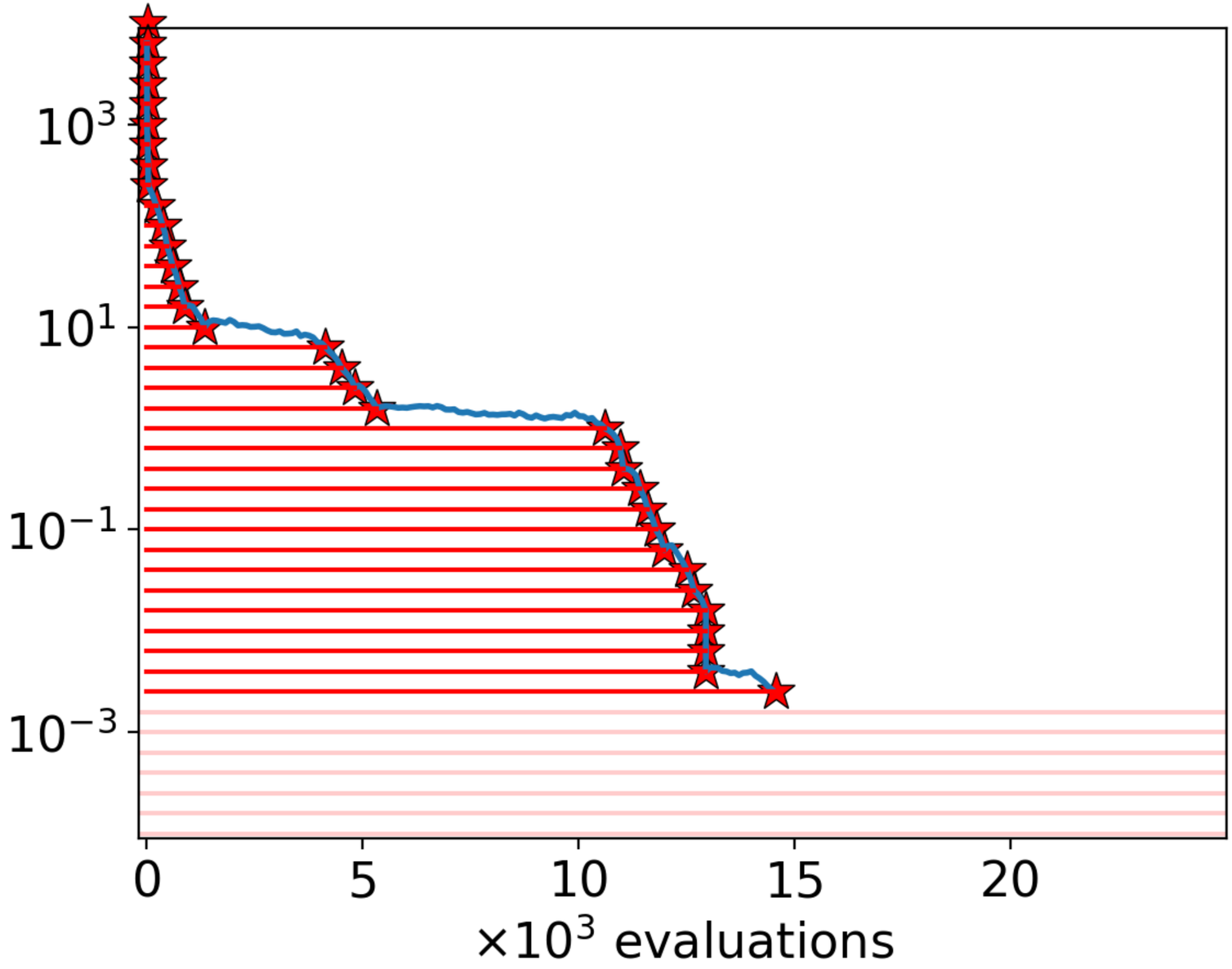
Convergence Graphs is All We Have

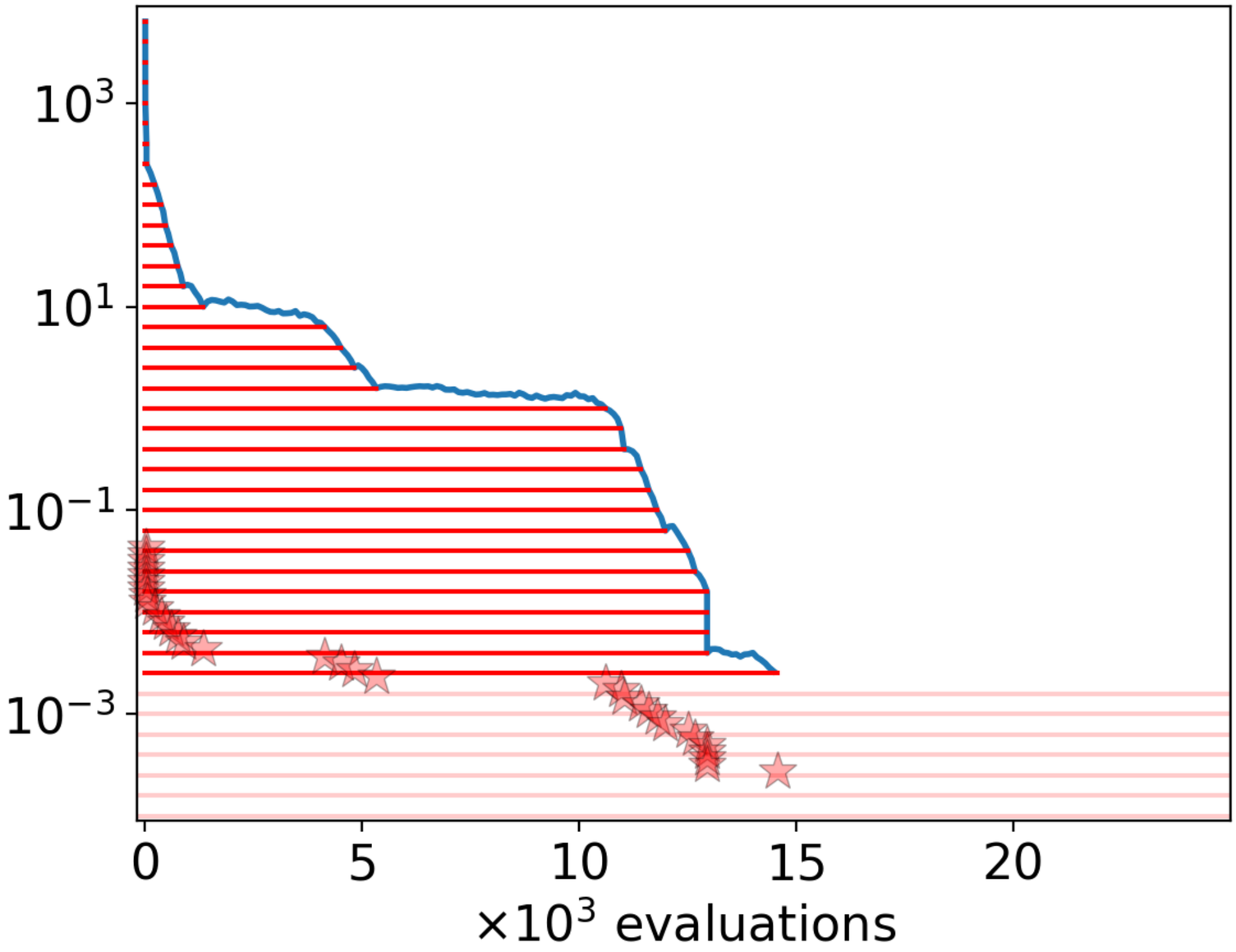


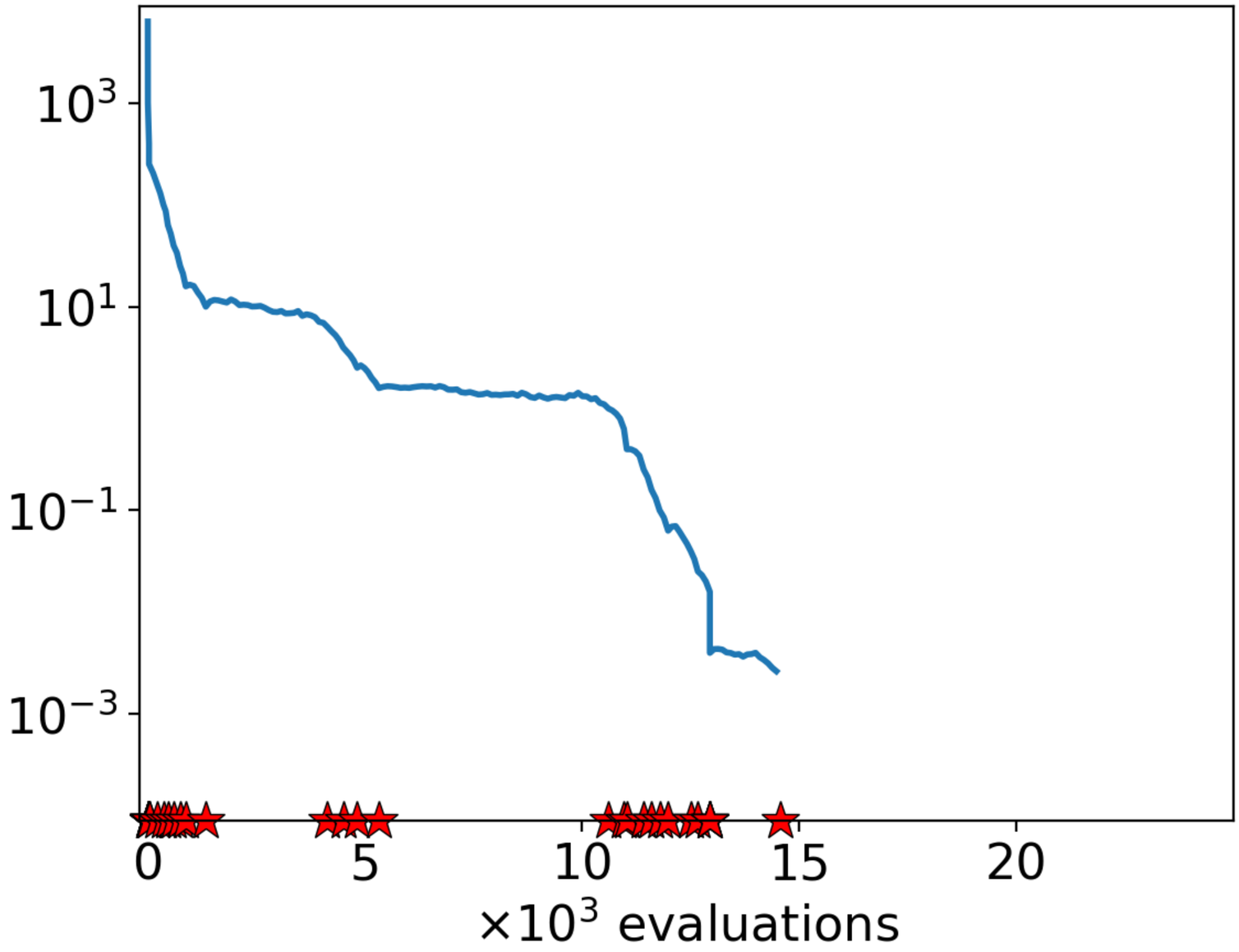

10^3
10^1
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^3$ evaluations

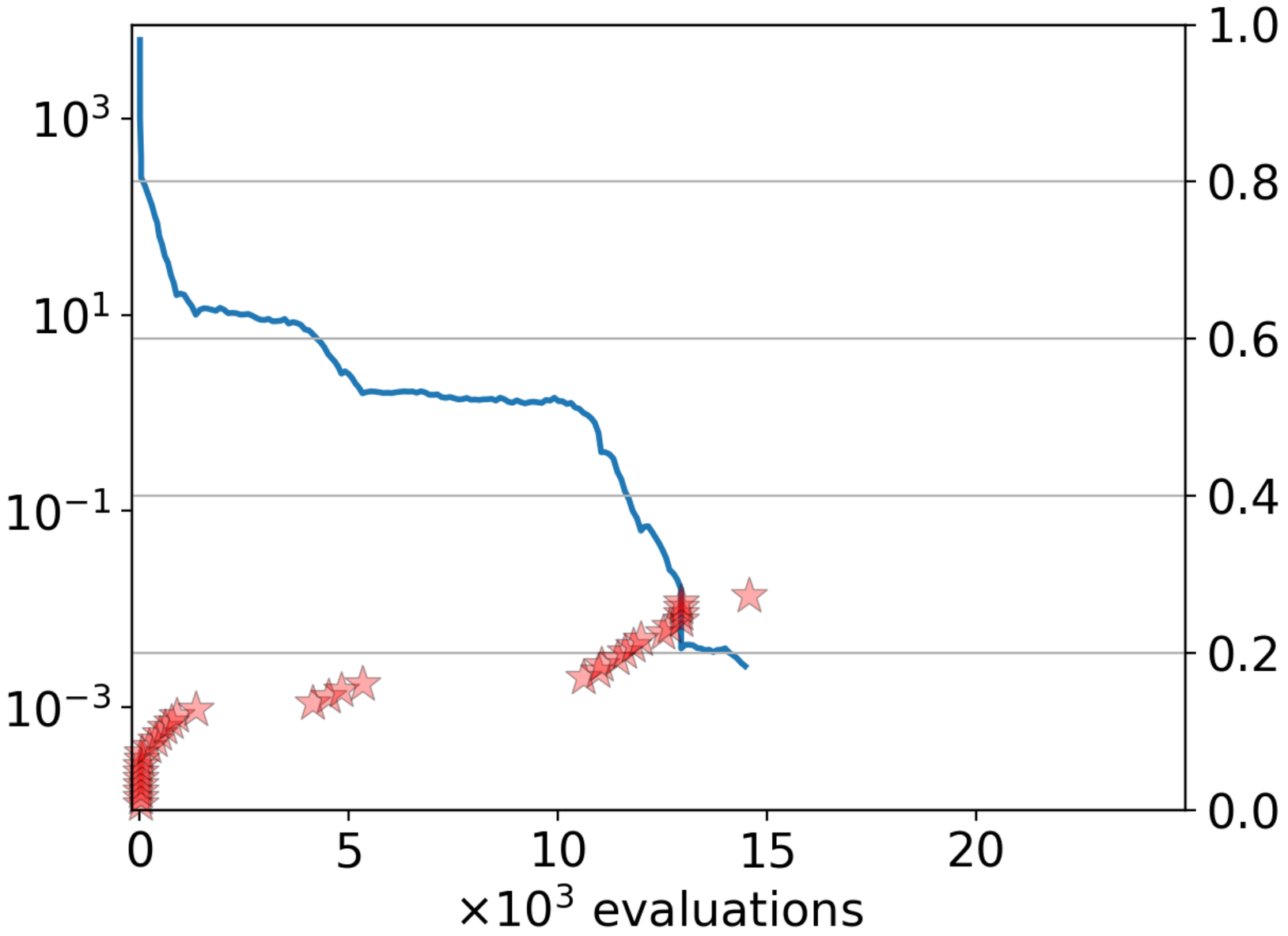

10^3
10^1
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^3$ evaluations

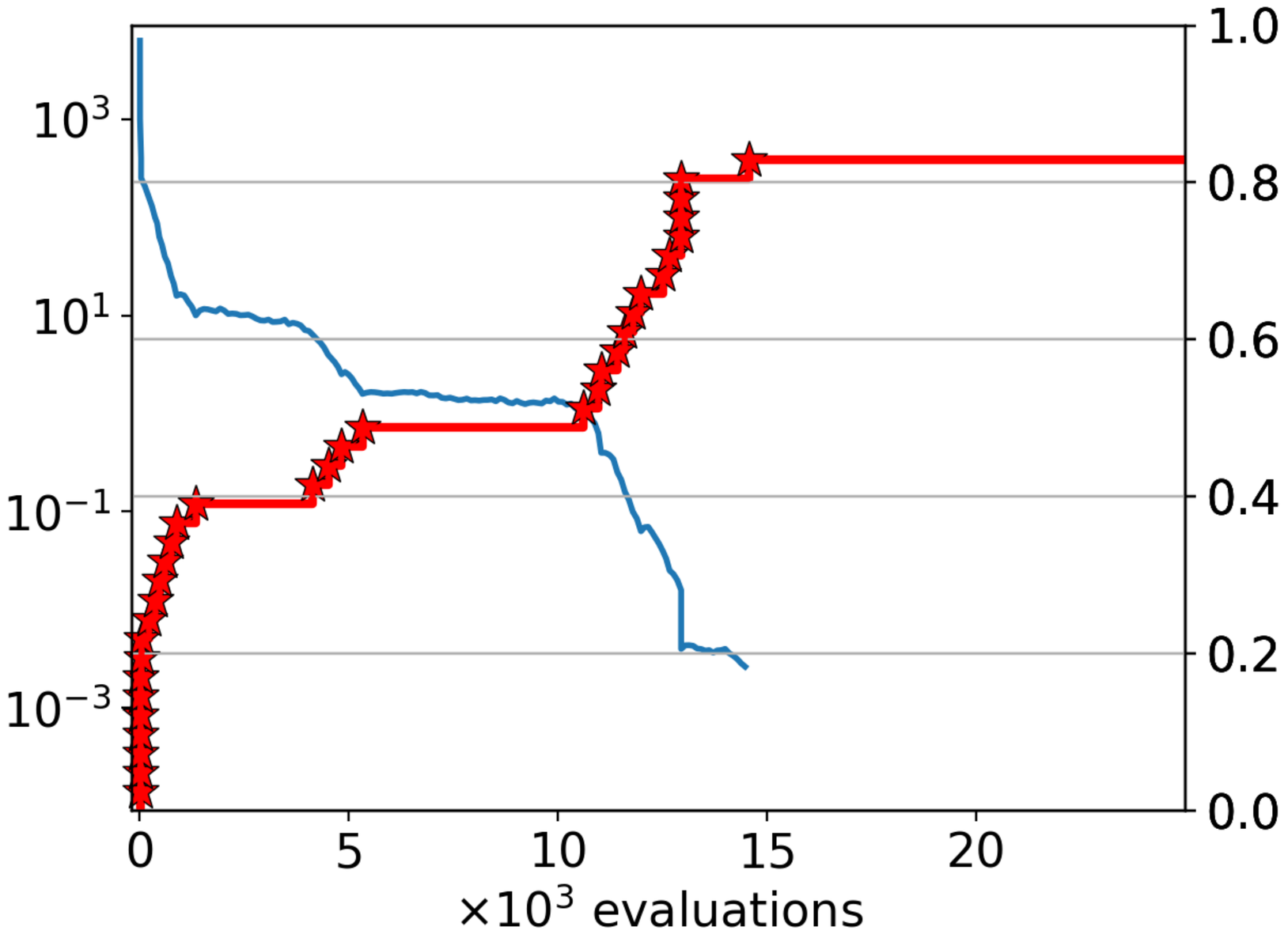

10^3
10^1
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^3$ evaluations

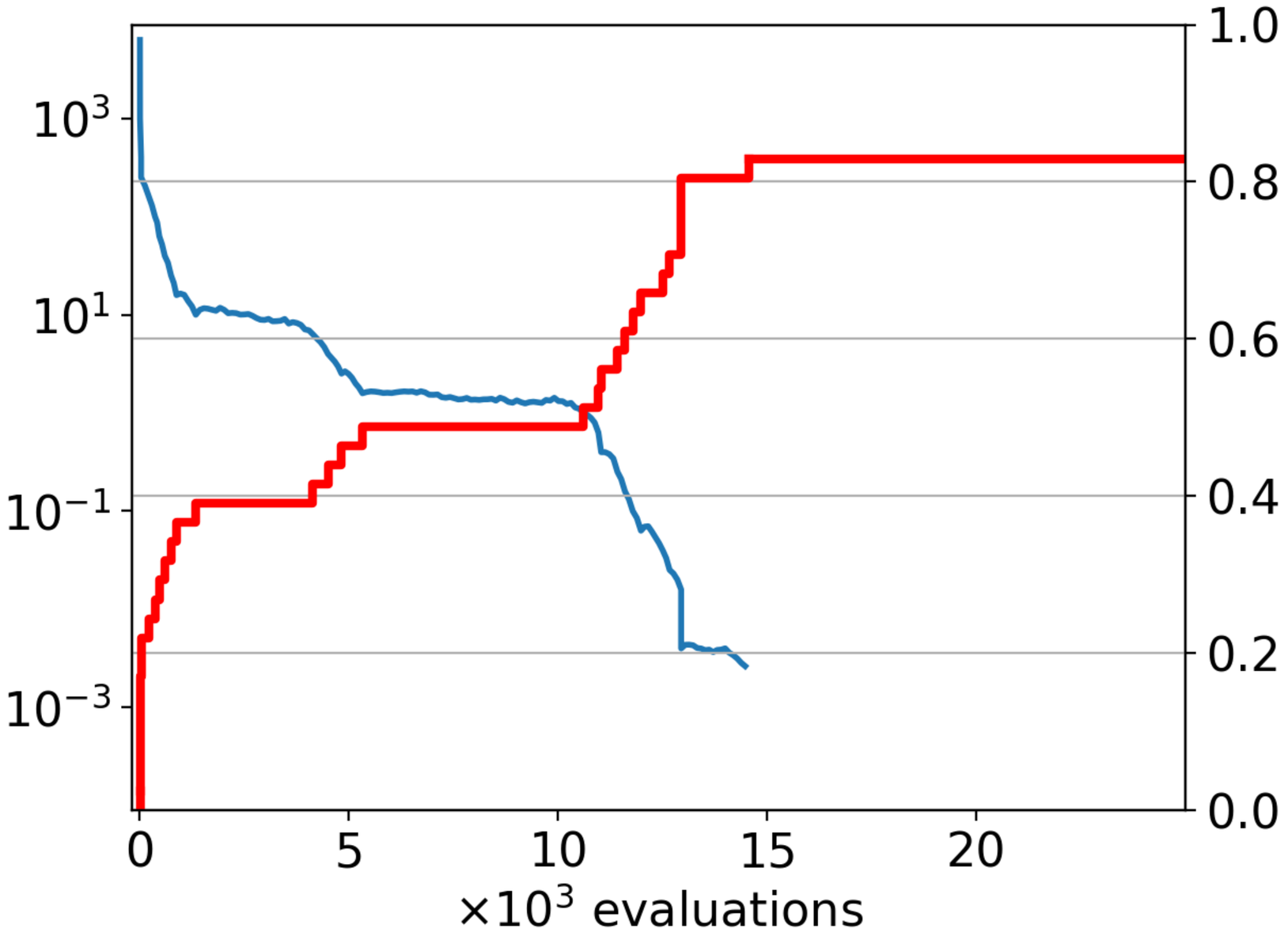

10^3
10^1
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^3$ evaluations


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

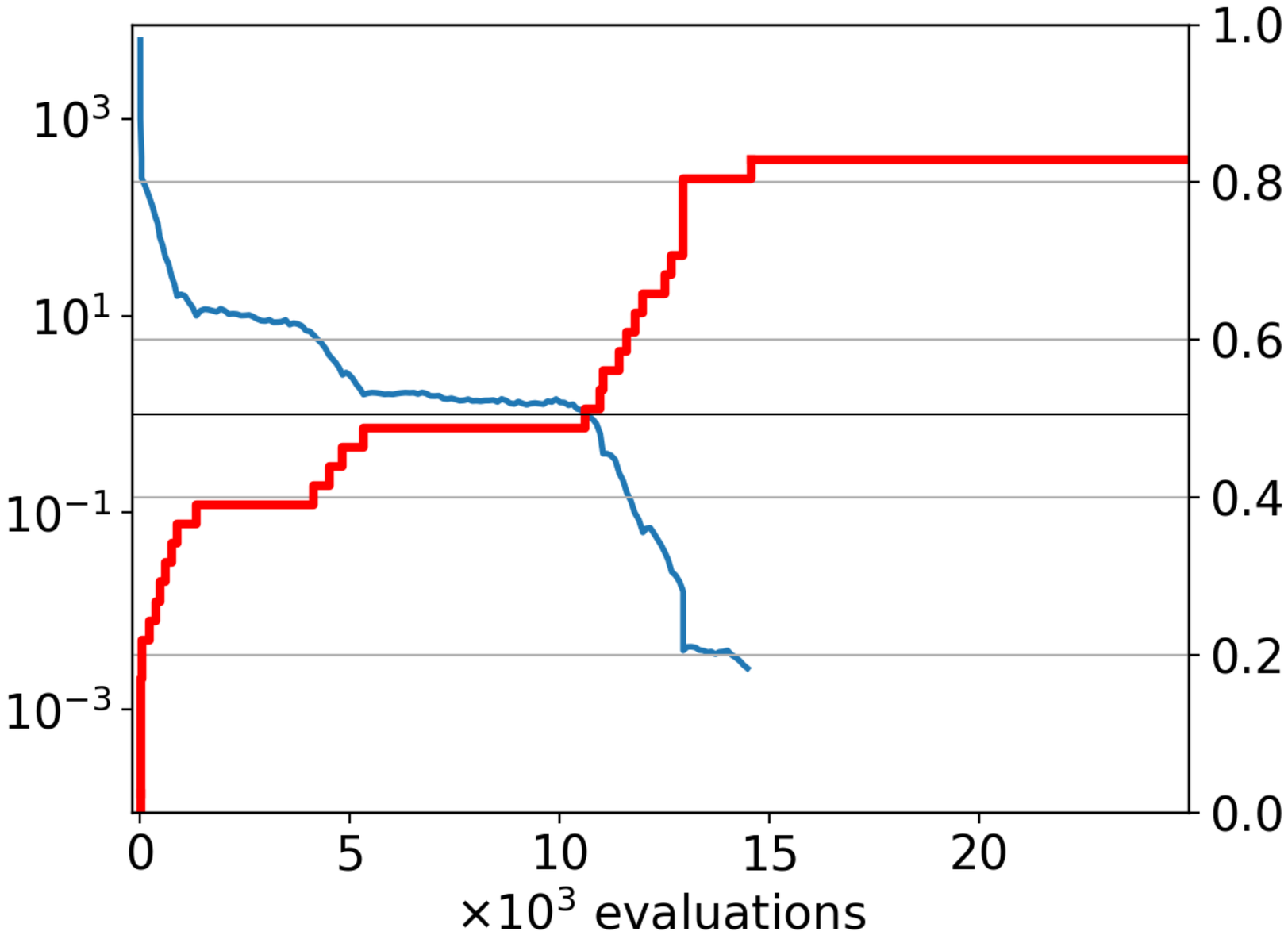

10^3
10^1
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^3$ evaluations


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

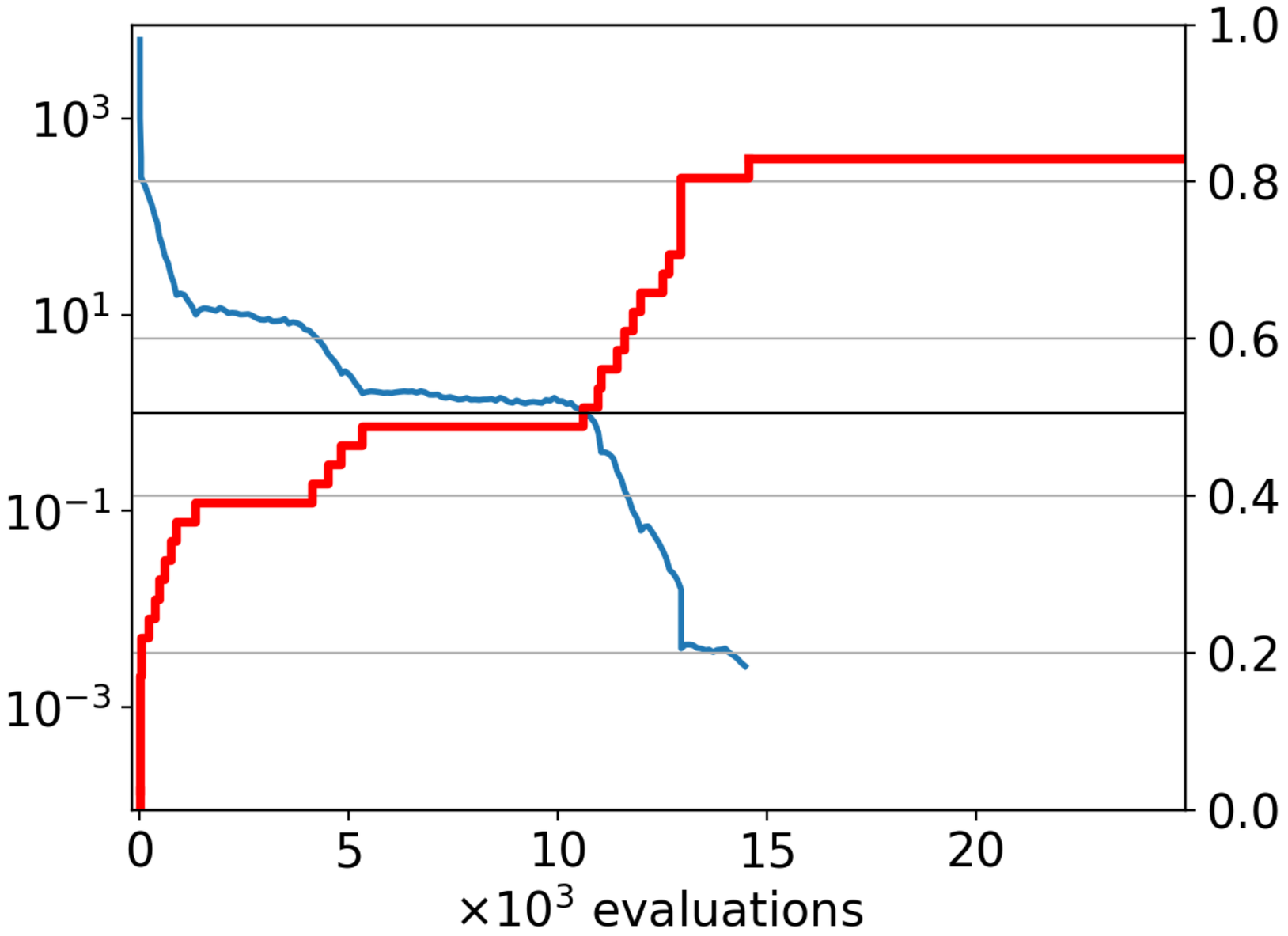

10^3
10^1
10^{-1}
10^{-3}
0
5
10
15
20
$\times 10^3$ evaluations
1.0
0.8
0.6
0.4
0.2
0.0

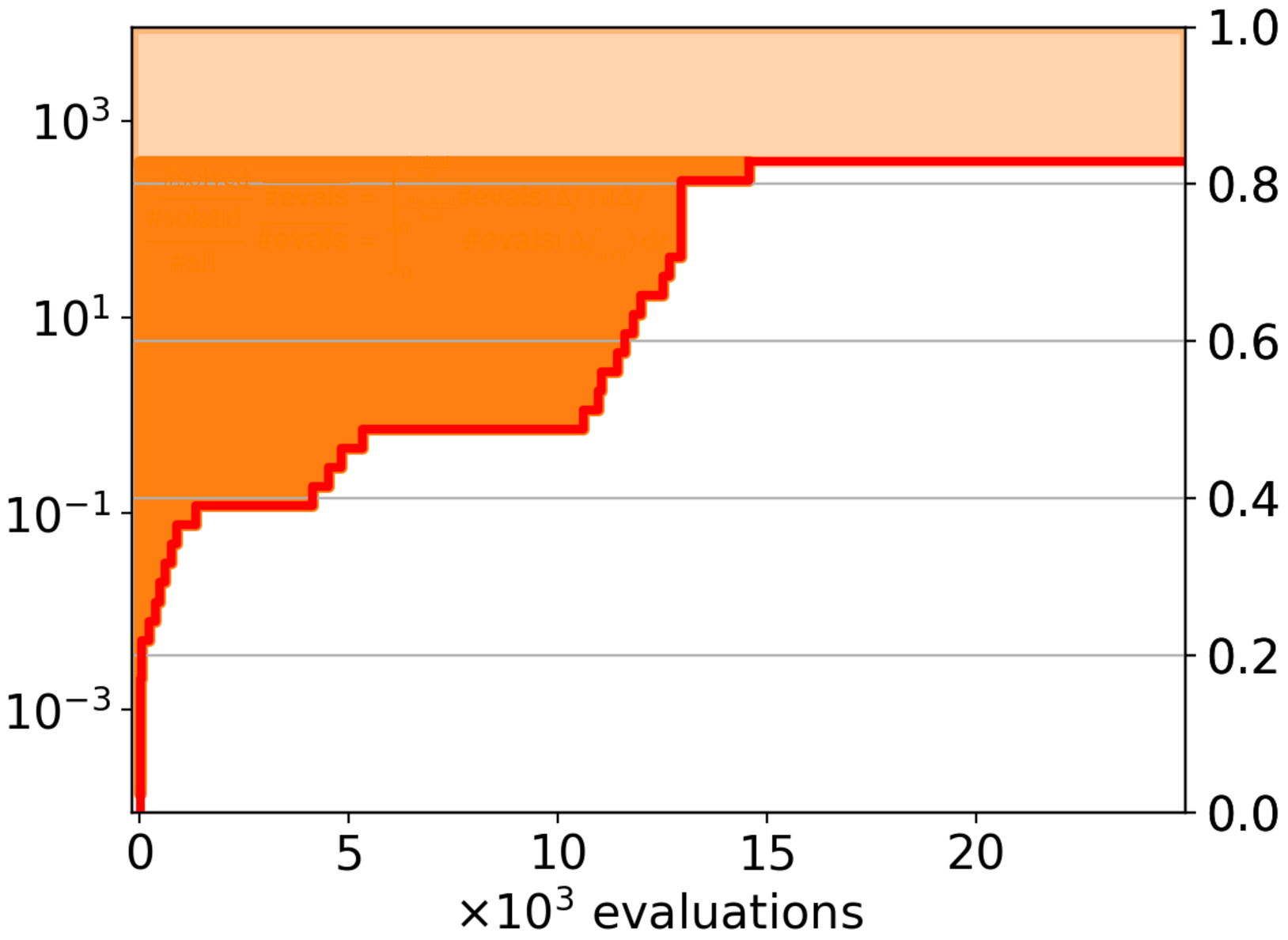

10^3
10^1
10^{-1}
10^{-3}
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
$\times 10^3$ evaluations

0
5
10
15
20
10^{-3}
10^{-1}
10^{1}
10^{3}
0.0
2.0
4.0
6.0
8.0
10

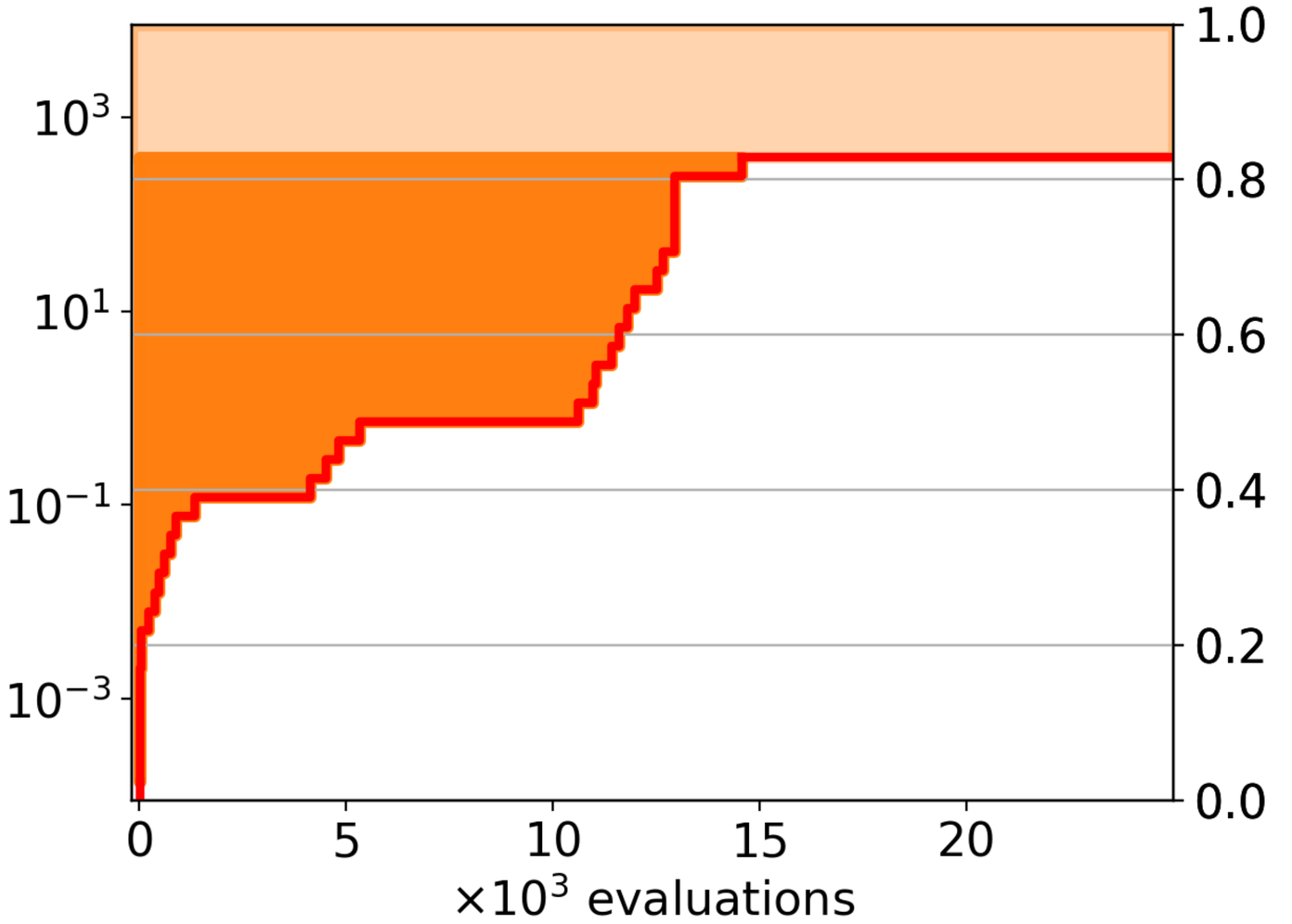

10^3
10^1
10^{-1}
10^{-3}
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
$\times 10^3$ evaluations

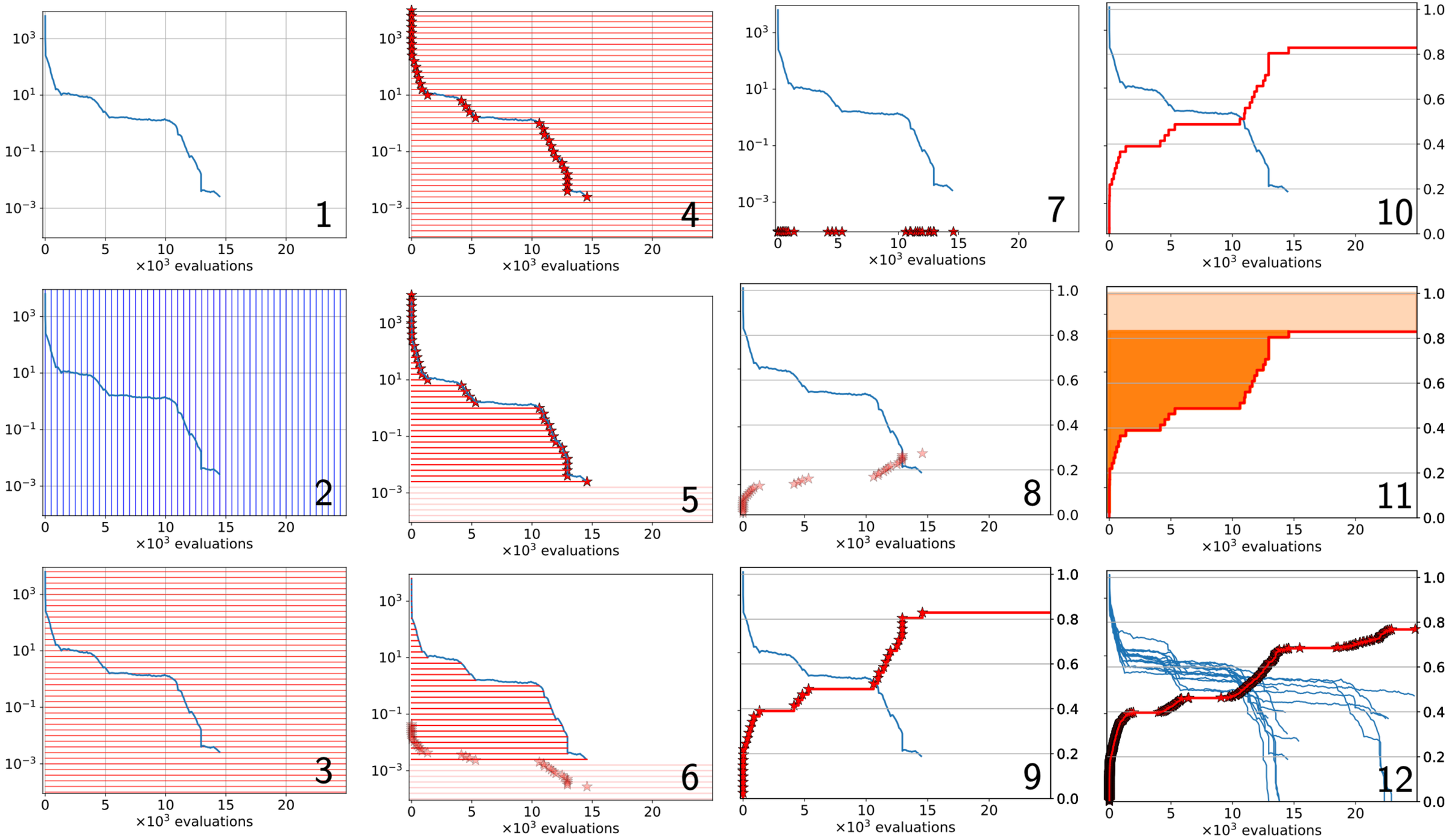
0
5
10
15
20
10^{-3}
10^{-1}
10^{1}
10^{3}
0.0
2.0
4.0
6.0
8.0
10


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

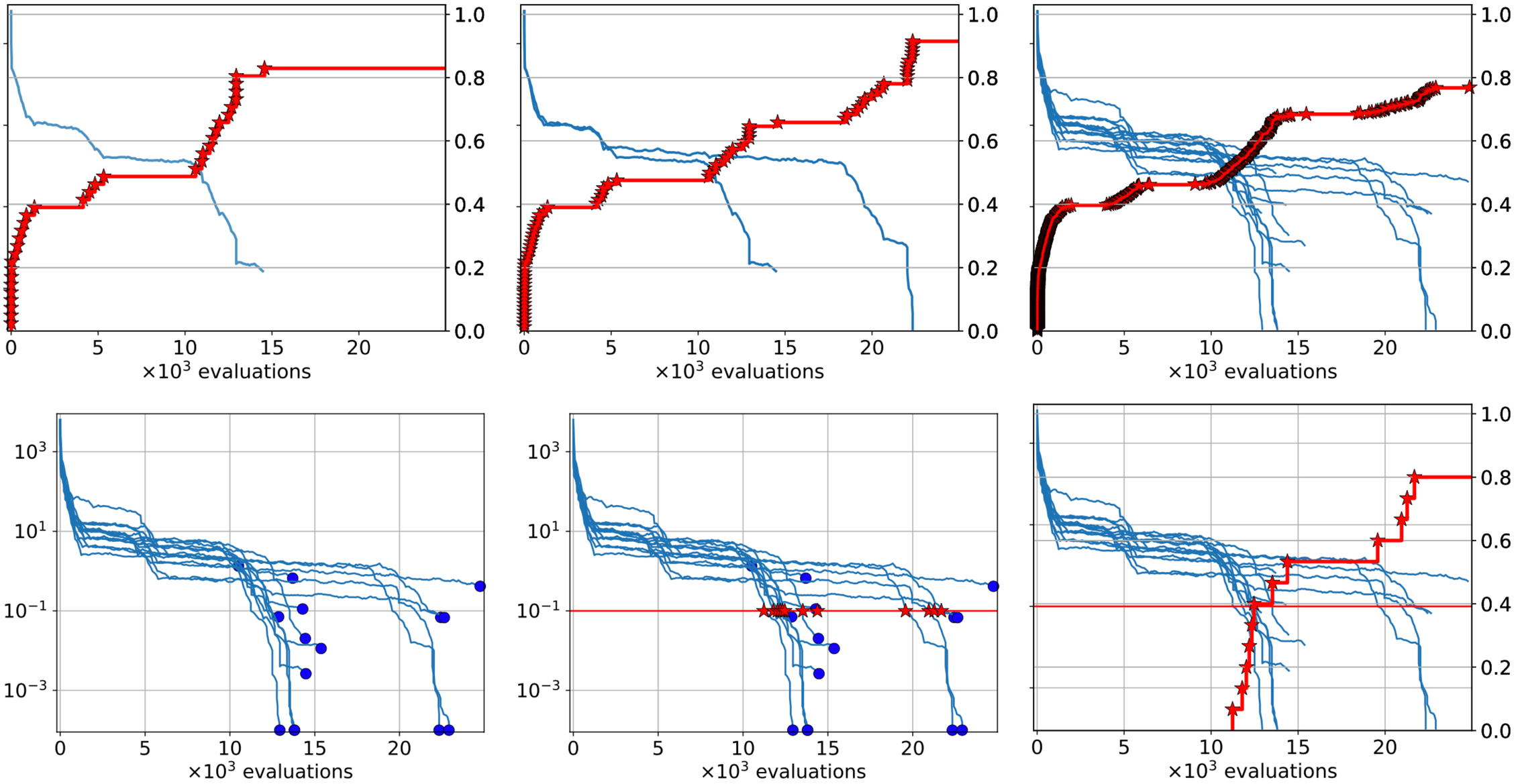

10^3
10^1
10^{-1}
10^{-3}
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
$\times 10^3$ evaluations

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




1
2
3
4
5
6
7
8
9
10
11
12
×10^3 evaluations

Aggregation of Several Convergence Graphs



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 $\{T_{p,s}/(n+1), p \in \mathcal{P}\}$:

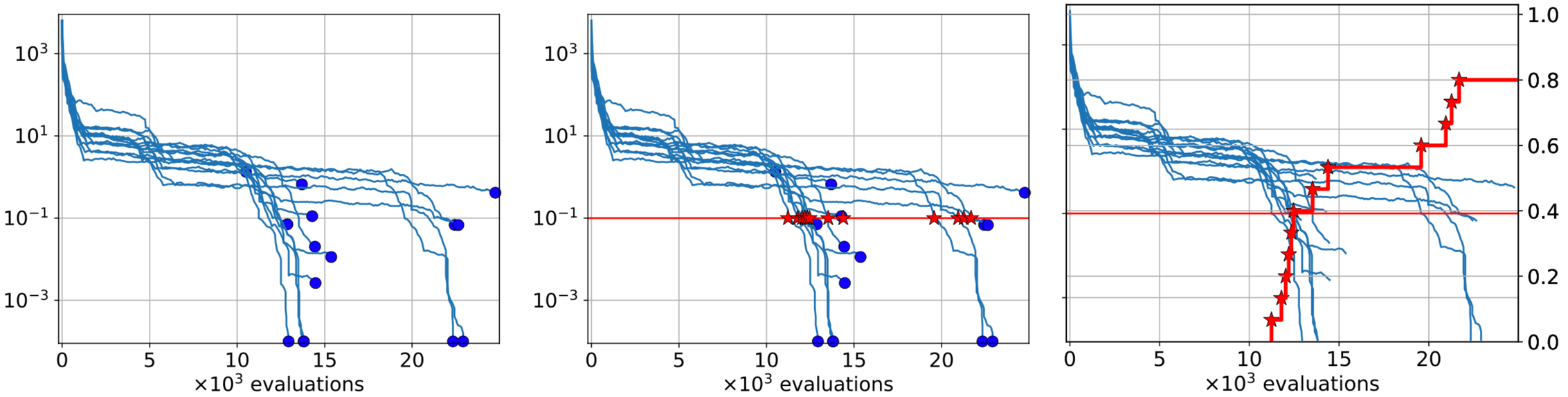
Normalization is done because runtime associated to different dimensions are put together

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

•

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 $\{r_{p,s}, p \in \mathcal{P}\}$:

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

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

  we have like 25 x 15 x 100 ≈ 40,000 single measurements for each algorithm in each dimension

- implicit assumption: uniform distribution over all aggregated problems

  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 need to 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: ECDFs

- 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

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

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

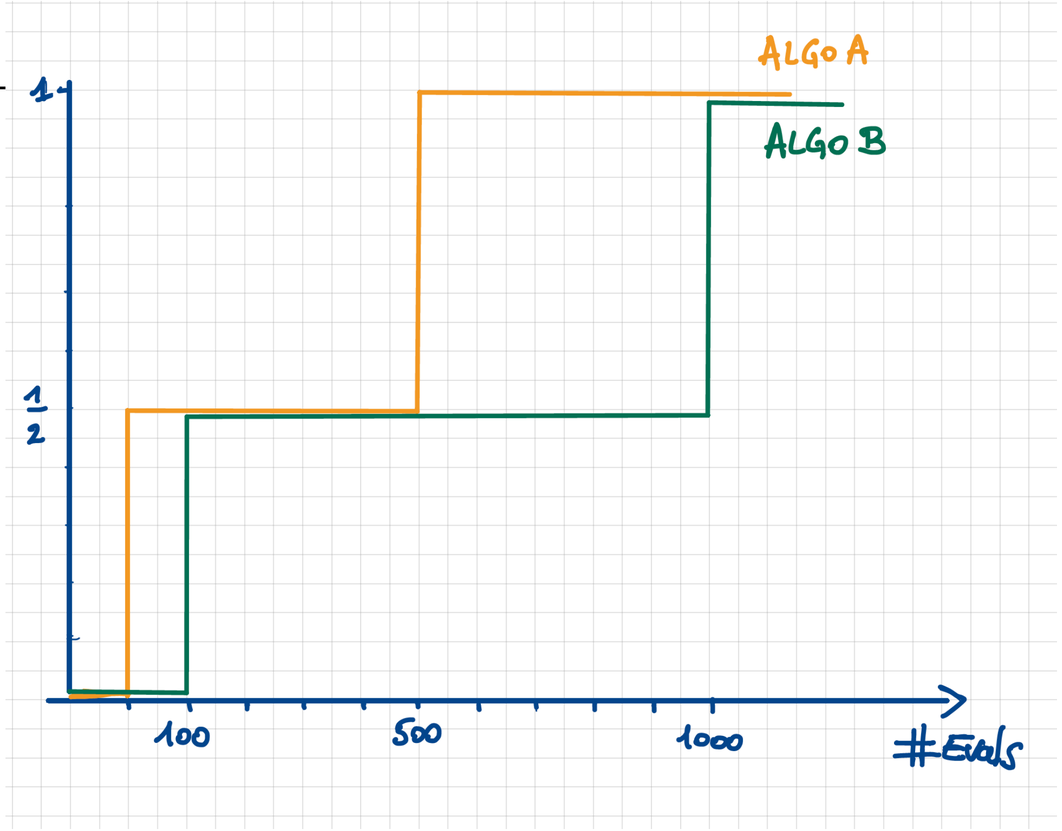
when the x-axis is in log-scale
is invariant under reordering
(whereas ECDFs are constrained to a unique order).

Discussion of Aggregation

	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



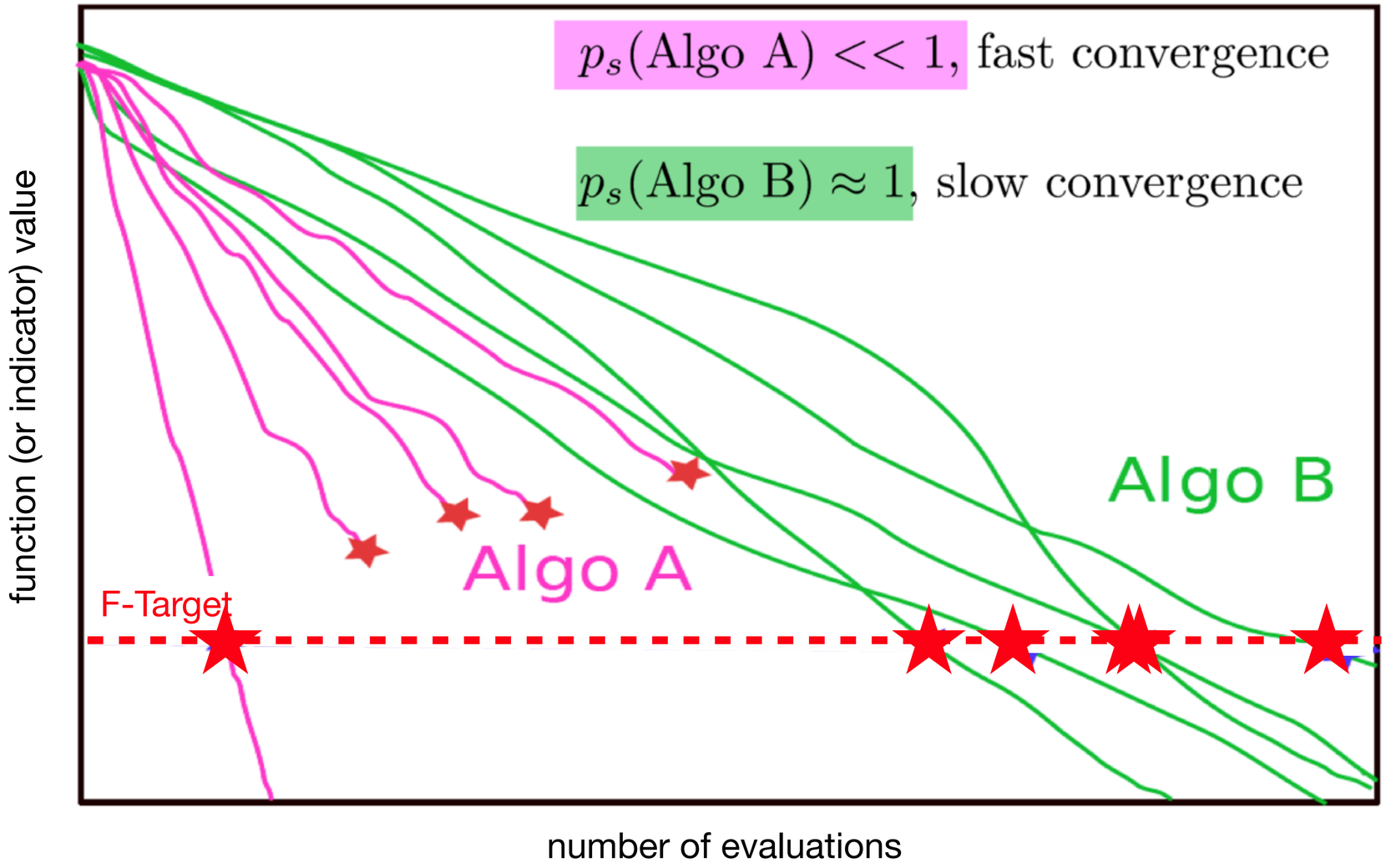
Domination in each point of an empirical runtime distribution does not imply equal or better performance on each problem !

Expected RunTime (ERT)

Aggregated measurement

Treating Success Probabilities

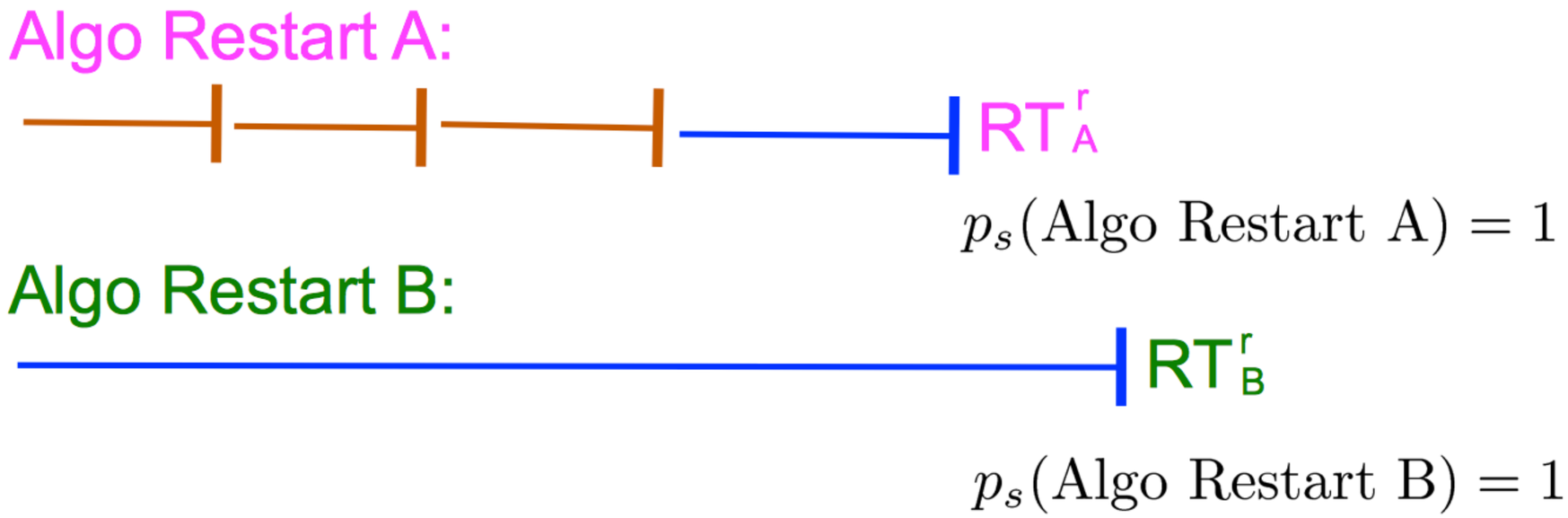
Solving the fast-versus-successful comparison dilemma



Treating Success Probabilities

Solving the fast-versus-successful comparison dilemma

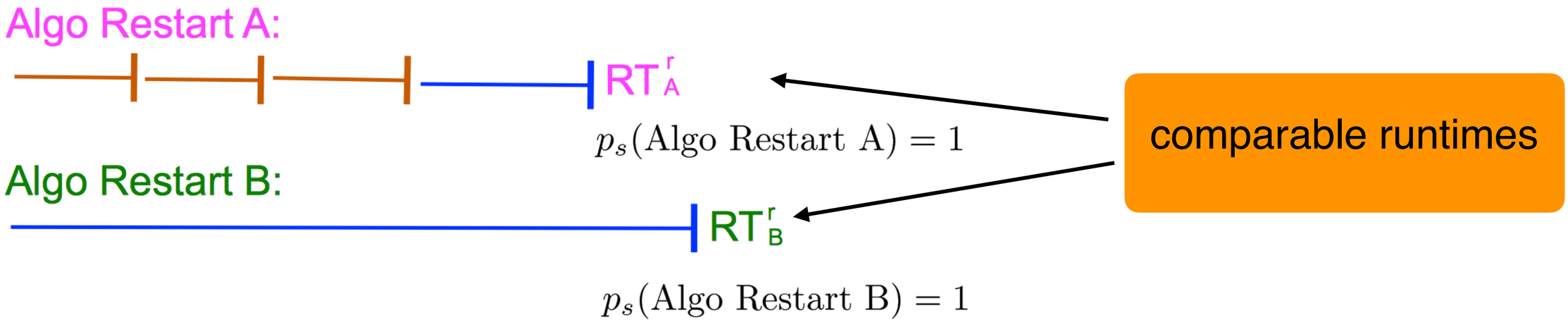
We can **simulate a runtime distribution** by simulated (artificial) restarts using the given independent runs



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



Expected Runtime of Restart Algorithm:

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

Expected time to see the first success

Expected RunTime - ERT

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

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

unsuccessful runs count (only) in the nominator

defined (only) for #successes > 0

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

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

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

odds ratio

ERT Related Performance Measures

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

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

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

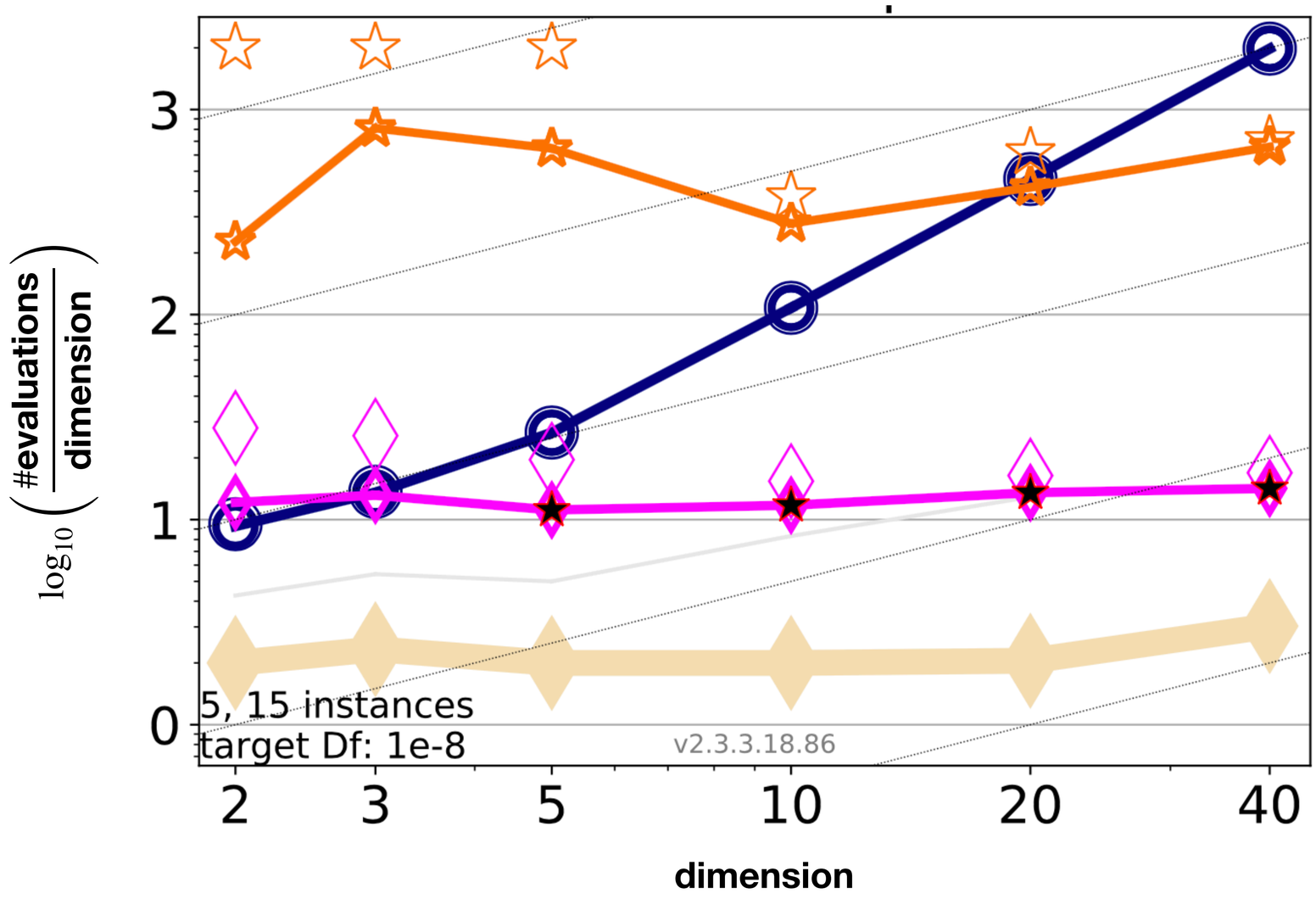
$$= \left(\frac{1}{\mathrm{success\ rate}}\right) \mathrm{avg}(\mathrm{eval}_{\mathrm{succ}})$$

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


$\log_{10}\left(\frac{\#\text{evaluations}}{\text{dimension}}\right)$
3
2
1
0
5, 15 instances
target Df: 1e-8
v2.3.3.18.86
2
3
5
10
20
40
dimension

Take Home Messages (1)

- Select a balanced testbed
  - using “all functions” is likely to introduce a bias

    like too many simple or low dimensional problems
- Use quantitative measurements

  also: empirical CDFs are a very useful tool
- 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
```

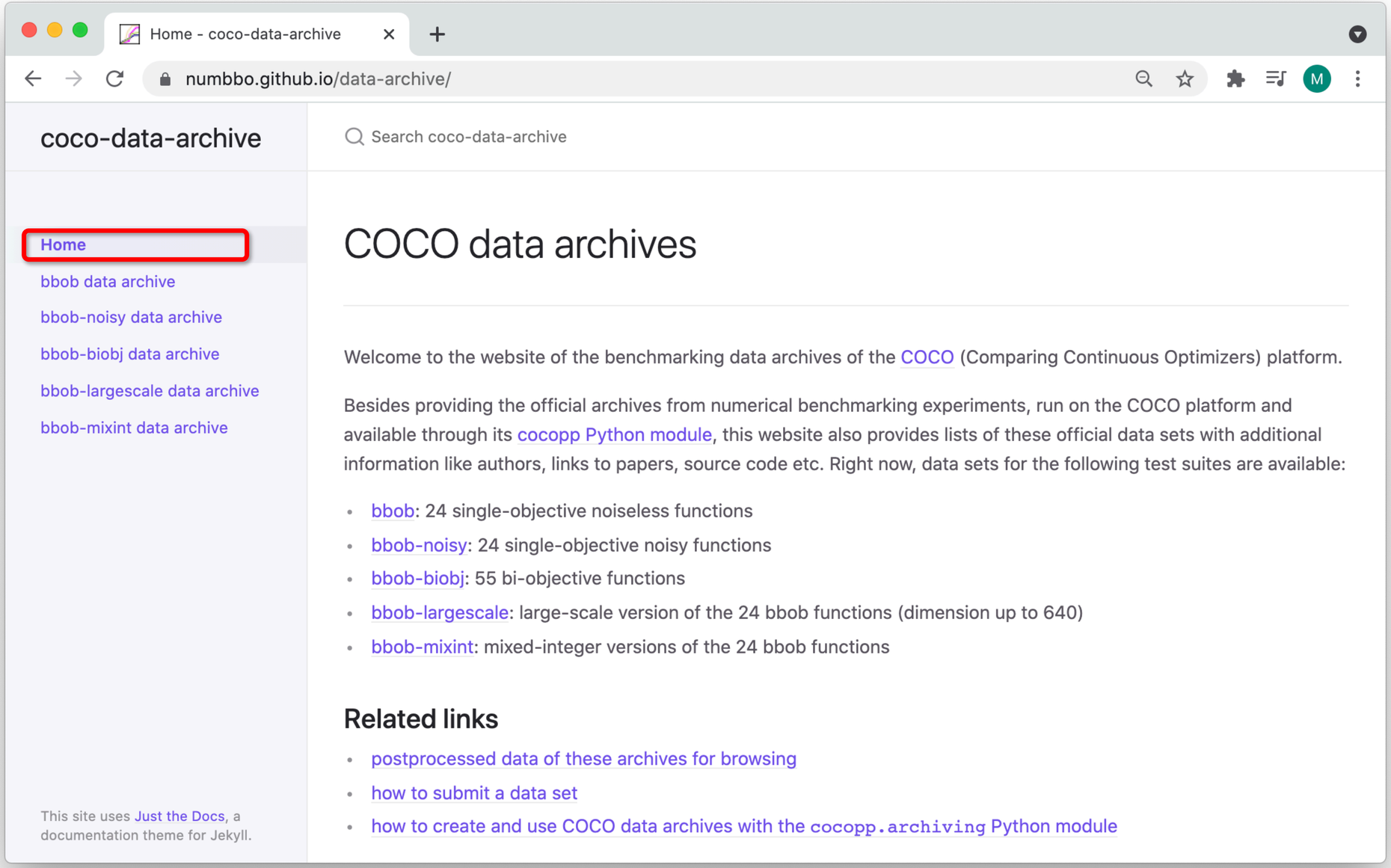
```
#!/usr/bin/env python
"""Python script to benchmark fmin of scipy.optimize"""
from __future__ import division  # not needed in Python 3
import cocoex, cocopp  # experimentation and post-processing modules
import scipy.optimize # to define the solver to be benchmarked

### 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
```

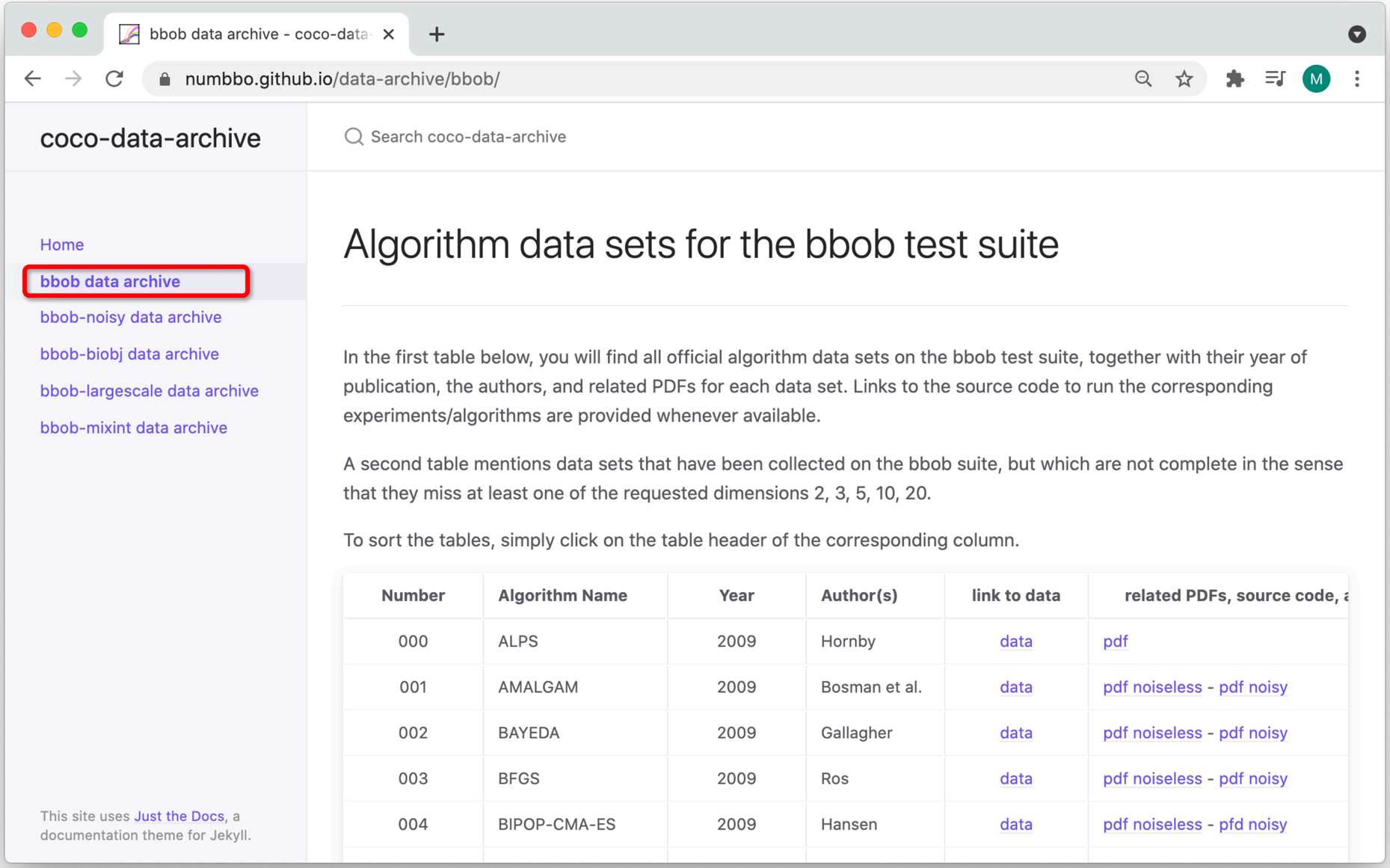
```
import cocoex, cocopp  # experimentation and post-processing modules
import scipy.optimize # to define the solver to be benchmarked

### 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

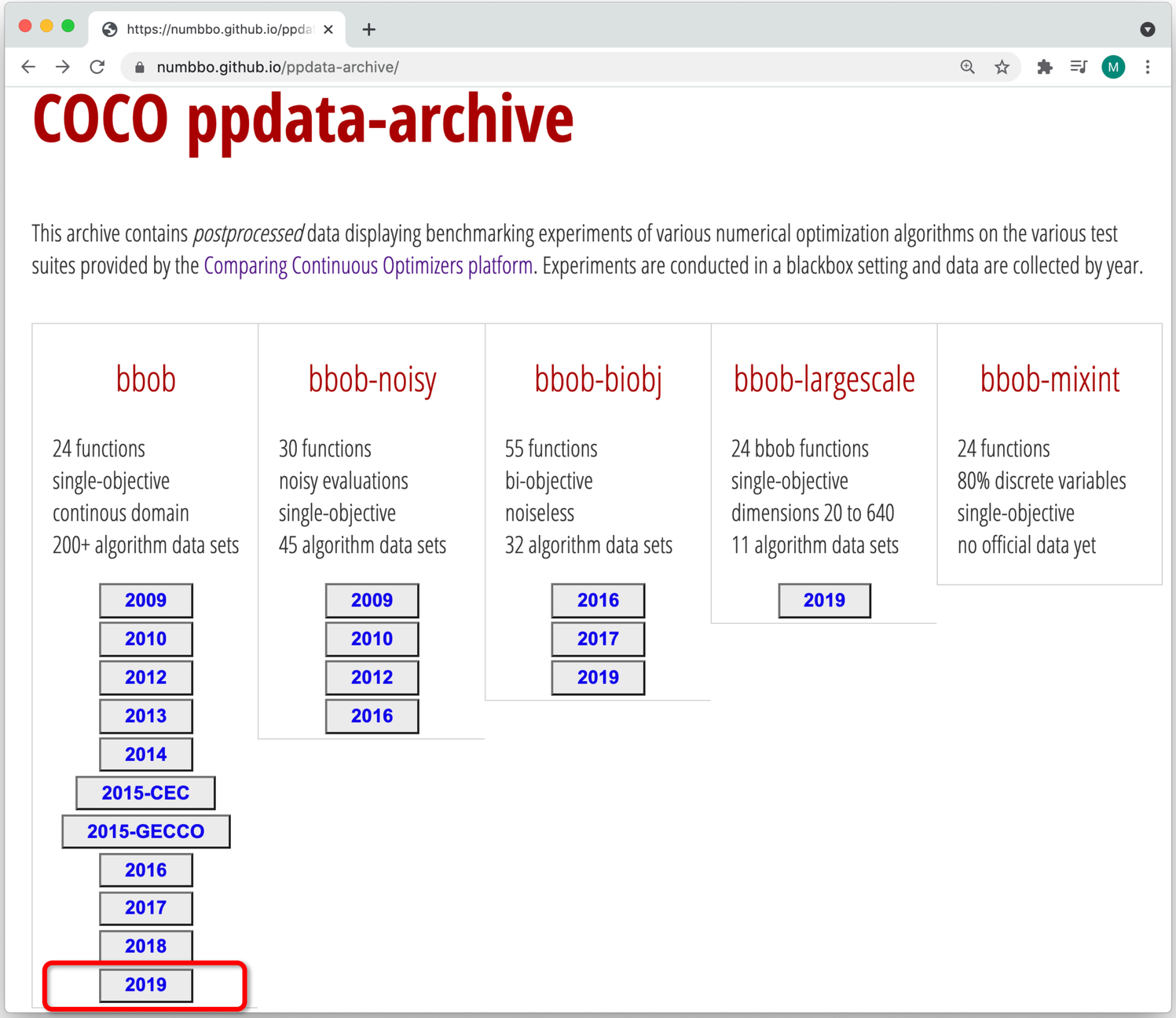
Using COCO


Home - coco-data-archive
numbbo.github.io/data-archive/
coco-data-archive
Search coco-data-archive
Home
bbob data archive
bbob-noisy data archive
bbob-biobj data archive
bbob-largescale data archive
bbob-mixint 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



Visit https://numbbo.github.io/data-archive/



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

index

numbbo.github.io/ppdata-archive/bbob/2019-all/index.html

Home

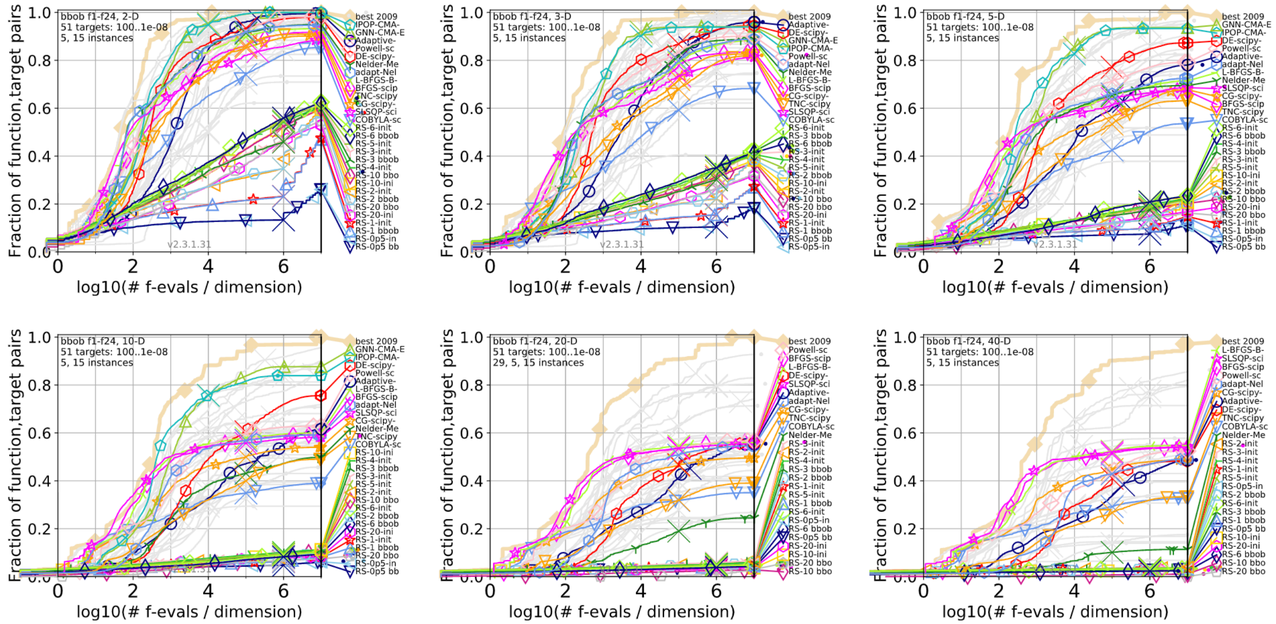
Runtime distributions (ECDFs) per function

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 | benchmarking-tutorial-20 | convergence-to-ecdf-plot

Code | Python 3

[1]:
```
%pylab
```
```
Using matplotlib backend: TkAgg
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
try:
    import cocopp
except ImportError:
    !pip install cocopp
    import cocopp
```

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

[*]:
```
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
```

benchmarking-tutorial-20 | benchmarking-tutorial-20 | convergence-to-ecdf-plot

Code | Python 3

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

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

[2]:
```
try:
    import cocopp
except ImportError:
    !pip install cocopp
    import cocopp
```

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

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

[*]:
```
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
```

file:///Users/hansen/gitlab/nhansen/convergence-grap

JupyterLab | index

[Home]

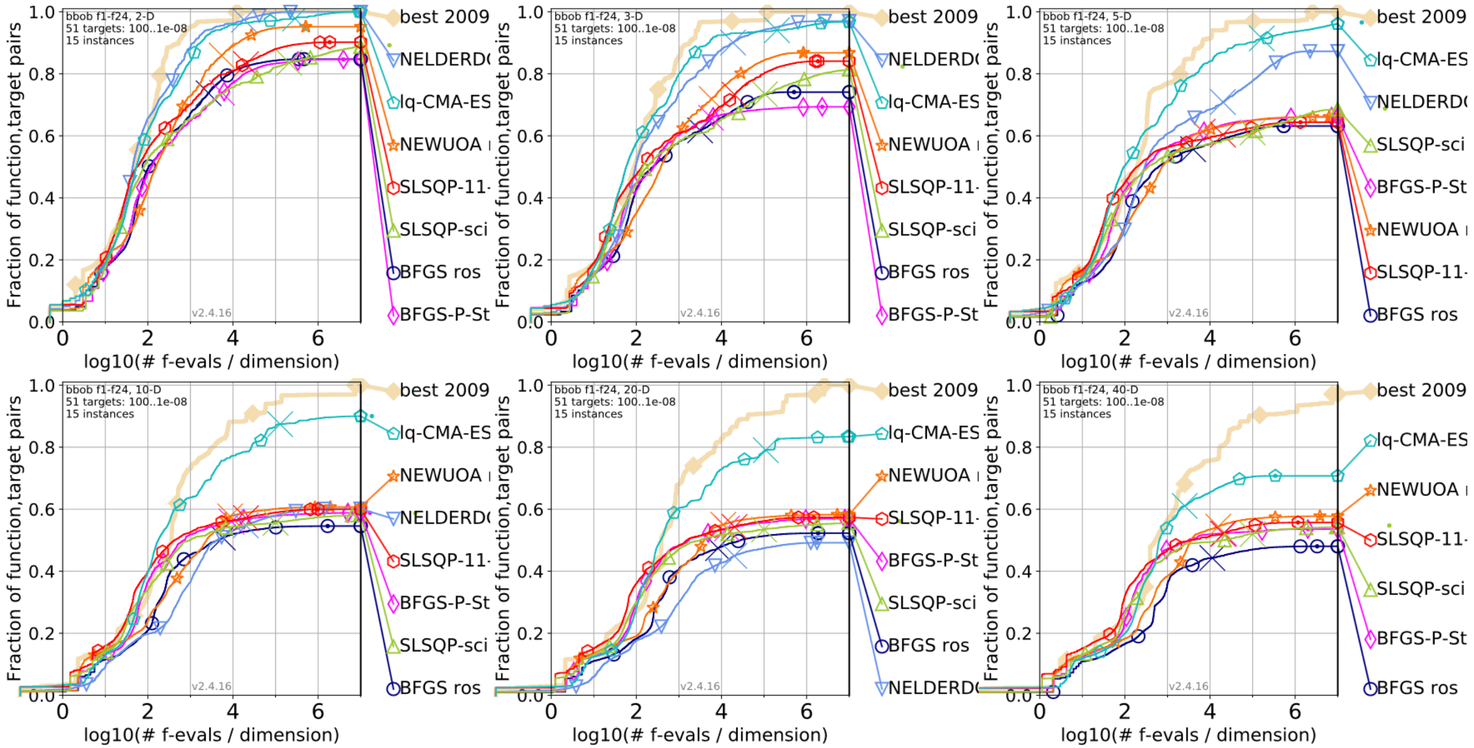
[Runtime distributions (ECDFs) per function]

[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 *"comparing continuous optimizers" OR "black-box optimization benchmarking (BBOB)"*		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 (final)

- Benchmarking is tedious but necessary
- Select a balanced testbed

  furious activity is no substitute for understanding
- Use quantitative measurements
- Don’t aggregate over attributes that are simple to determine

  like dimension

Your questions!