Ten+ Years of Benchmarking with COCO/BBOB

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

Presented at the Lorentz Center Workshop *Benchmarked: Optimization meets Machine Learning,* Leiden 2020

COCO — Comparing Continuous Optimisers

- is a (software) platform for comparing continuous optimisers in a black-box scenario

  https://github.com/numbbo/coco

- *automatises* the tedious and repetitive task of *benchmarking numerical optimisation algorithms in a black-box setting*

- advantage: saves time and prevents common (and not so common) pitfalls

COCO provides

- experimental and measurement *methodology*

  main decision: what is the end point of measurement

- suites of benchmark functions

  single objective, bi-objective, noisy, mixed-integer, more to come…

- data of already benchmarked algorithms to compare with

Benchmarking: Related Goals

1. Understanding algorithms

2. Measuring performance in a systematic way (a performance “profile”)

3. Running a competition

Benchmarking: The Global Picture

Two *surprisingly* *(but not completely)* *independent* puzzles to solve

- **What to benchmark**: for example, 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 (2020), COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, published online, Aug 2020.

…feel free to ask questions…

COCO/BBOB: Test Suite(s)

- Functions are
  - Based on known (analytical) functions, modelling a “known” difficulty
  - Comprehensible
  - Scalable
  - Difficult (also: non-separable)

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

    with arbitrary shifts and smallish irregularities to avoid artificial exploits and mitigate overfitting, emulates repetition of experiments
- The bad
  - Rastrigin function type is somewhat overrepresented

    partly due to function pairing
  - 10% of the default targets for F23 Katsuuras are trivial to hit

    evaluating the domain middle at first is a good “algorithm"
- Require to define target values (function + target = problem)

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

Data Format

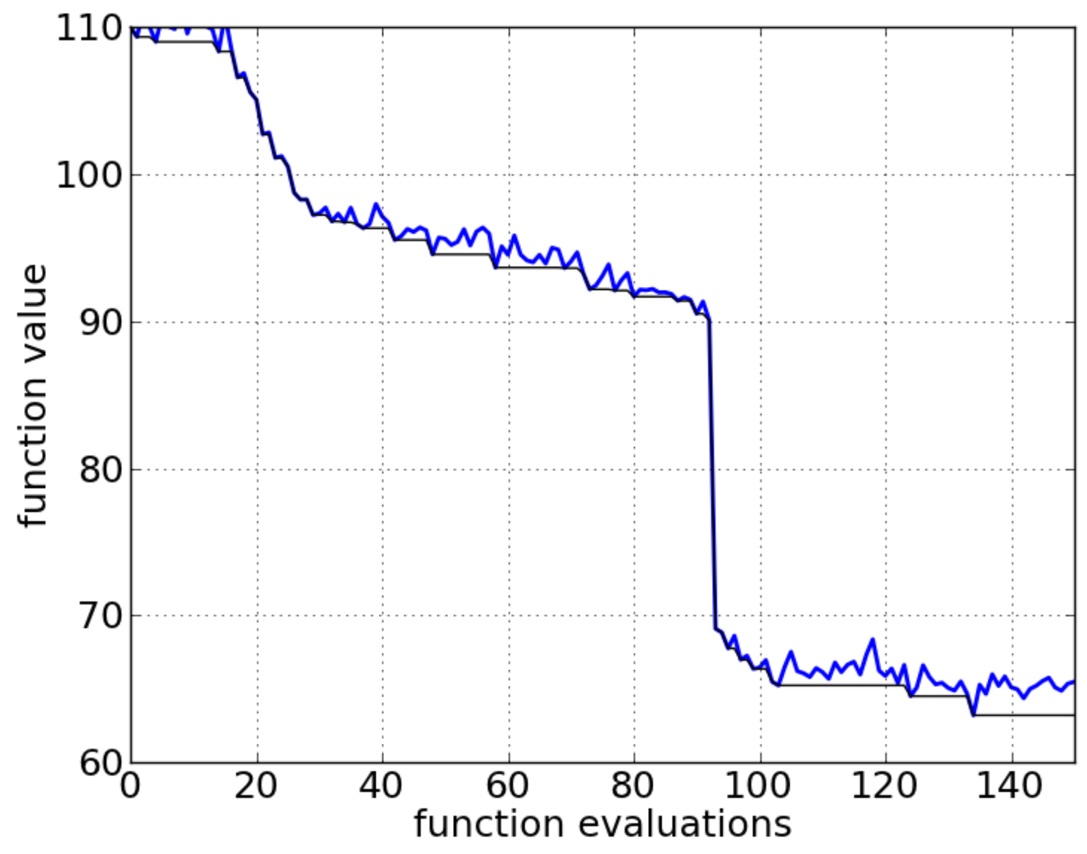
with hindsight 20/20

- The good:
  - scattered experiments can be “merged” (and “unmerged”) with a single “drag-and-drop”
  - separation between .info (meta- and summary-data) and .dat files is helpful
  - 10+ years old data are still smoothly usable
  - backwards compatible adjustments are/were possible
- The bad:
  - slightly too few targets (too coarse discretization, not a *format* issue though with backward compatible fix)
  - “handling” of restarts is suboptimal
  - meta-data are not json-style (key-value Python-dict-style) formatted
  - COCO maintains/writes *two* somewhat incompatible formats

…feel free to ask questions…

COCO/BBOB: Performance Assessment

“quality indicator” versus “time”
convergence graphs



is all we have (and all we use)

Specifically

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

  is invariant under changes of computer hardware, OS, programming language, compiler, …

- **quality indicator**:

  - SO: affine transformation of the function value (to be minimized)

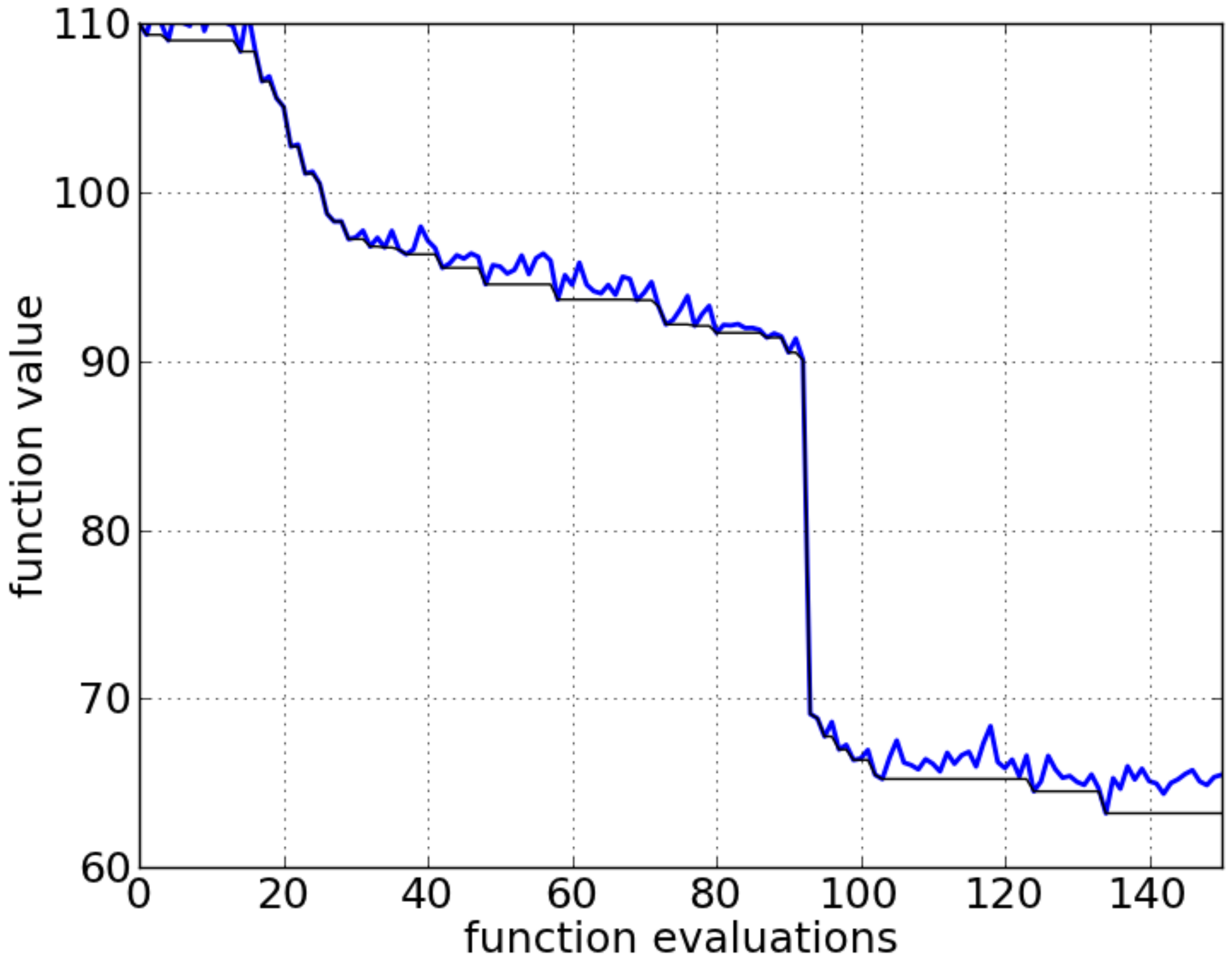
    different for each instance

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

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

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

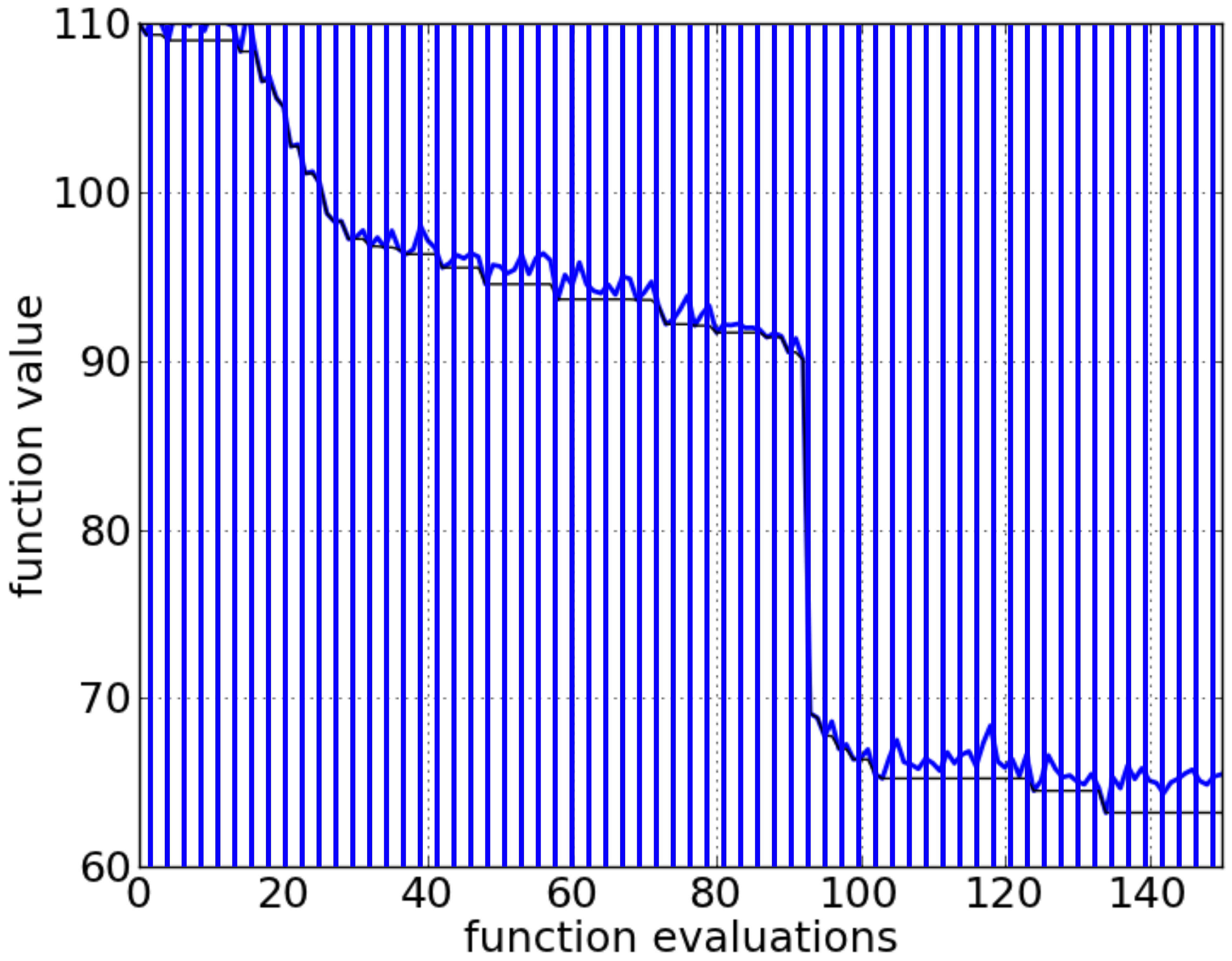
Convergence Graphs is All We Have



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

we only use the lower envelope

Discretization: Two Possibilities



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

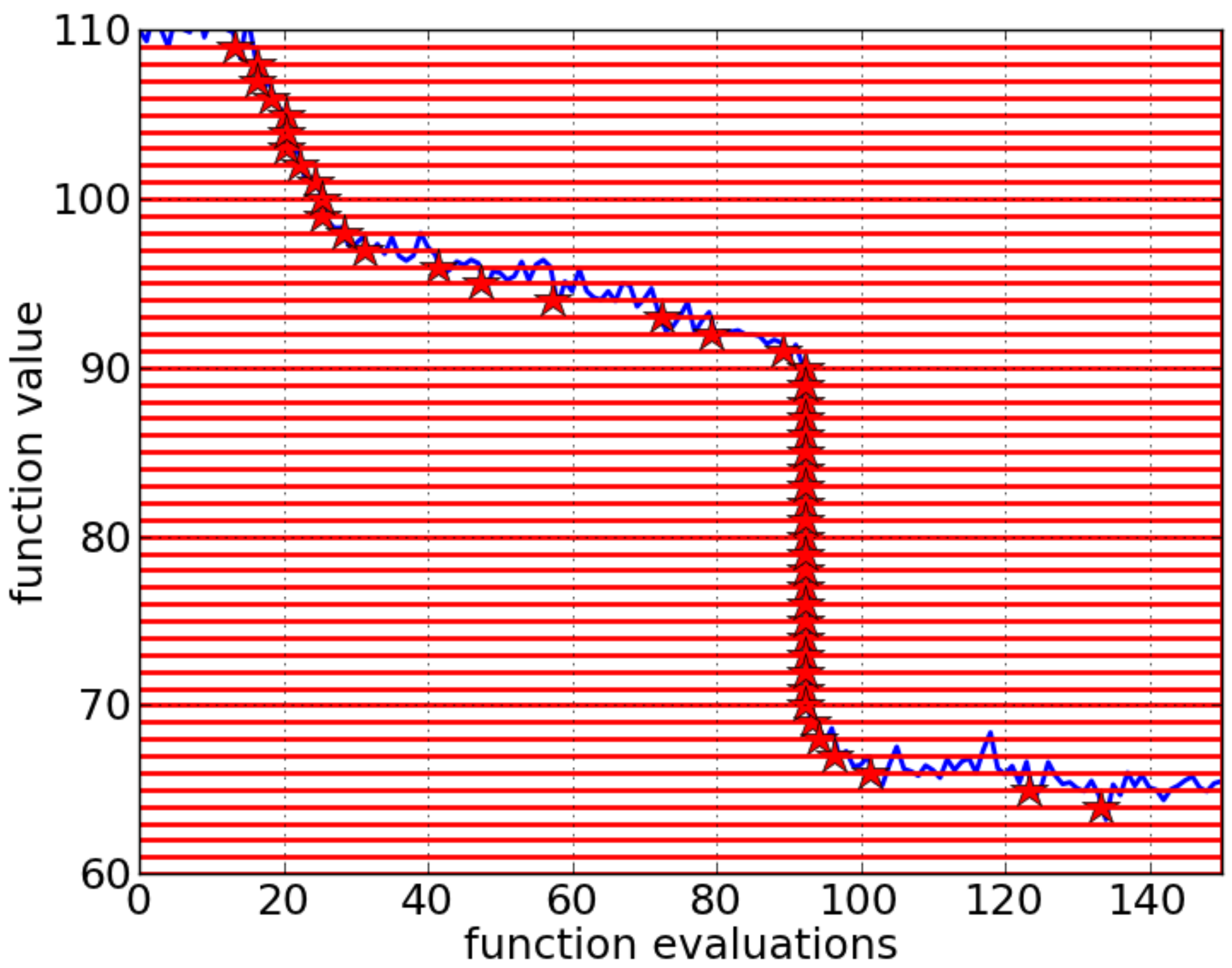
- **vertical**: by evaluation is a natural discretization

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

- evaluations are the independent variable

  function value is the dependent variable, the measurement

Discretization: Two Possibilities



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

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

we need to determine discretization intervals

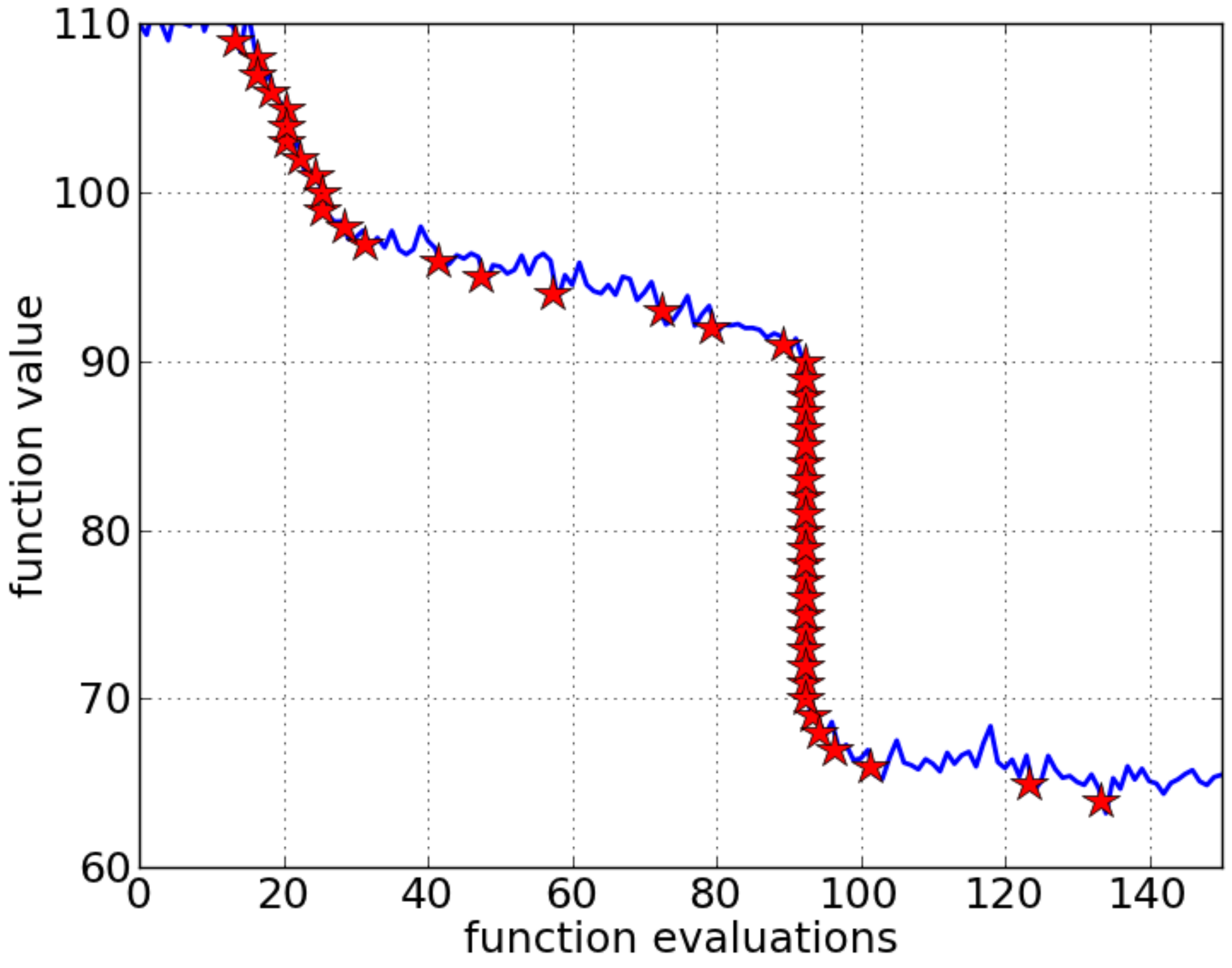
- function “target” values are the independent variable

time is the dependent variable, the measurement

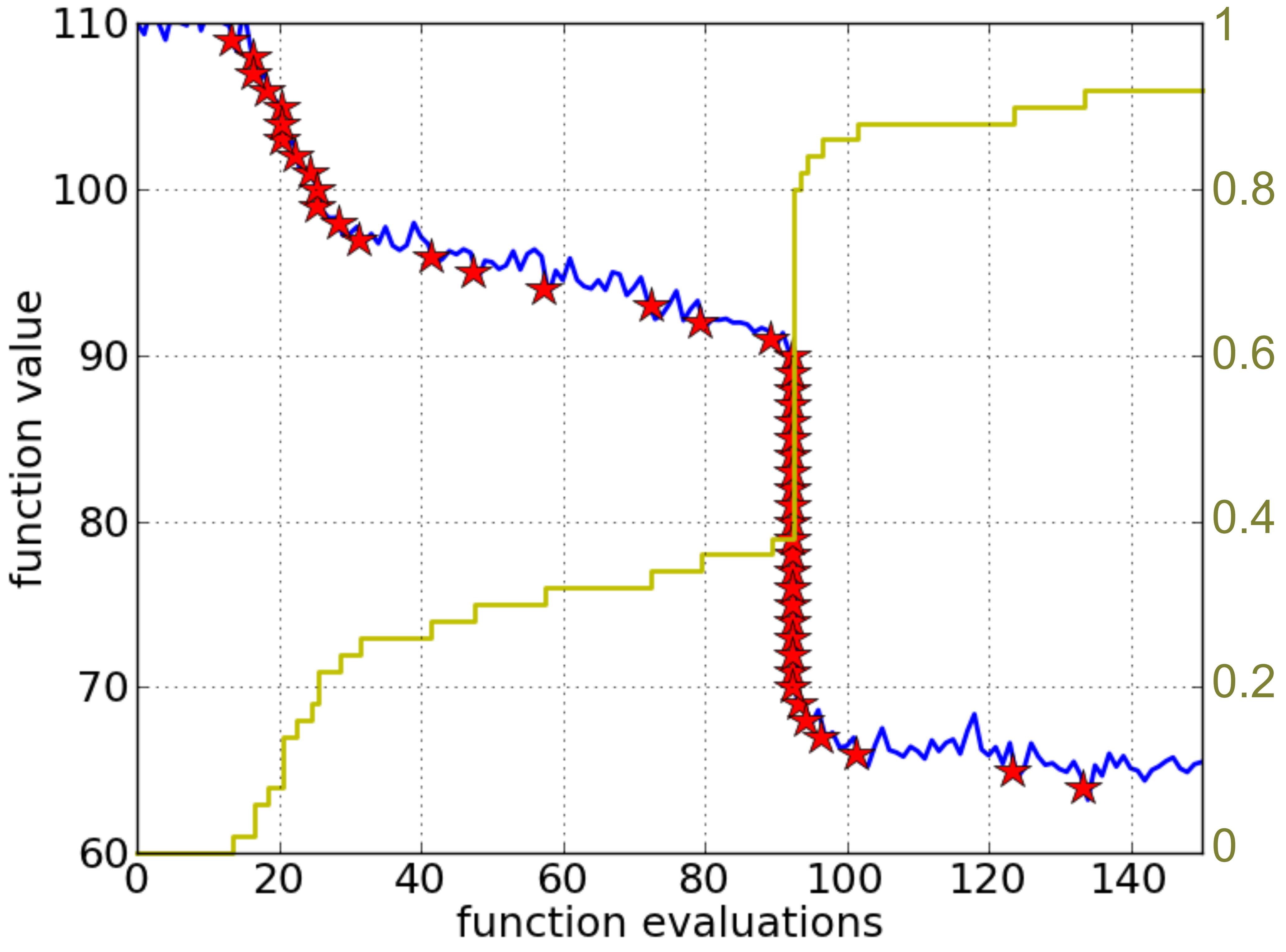
- still recovers the original data

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

Runtime distribution from a single graph



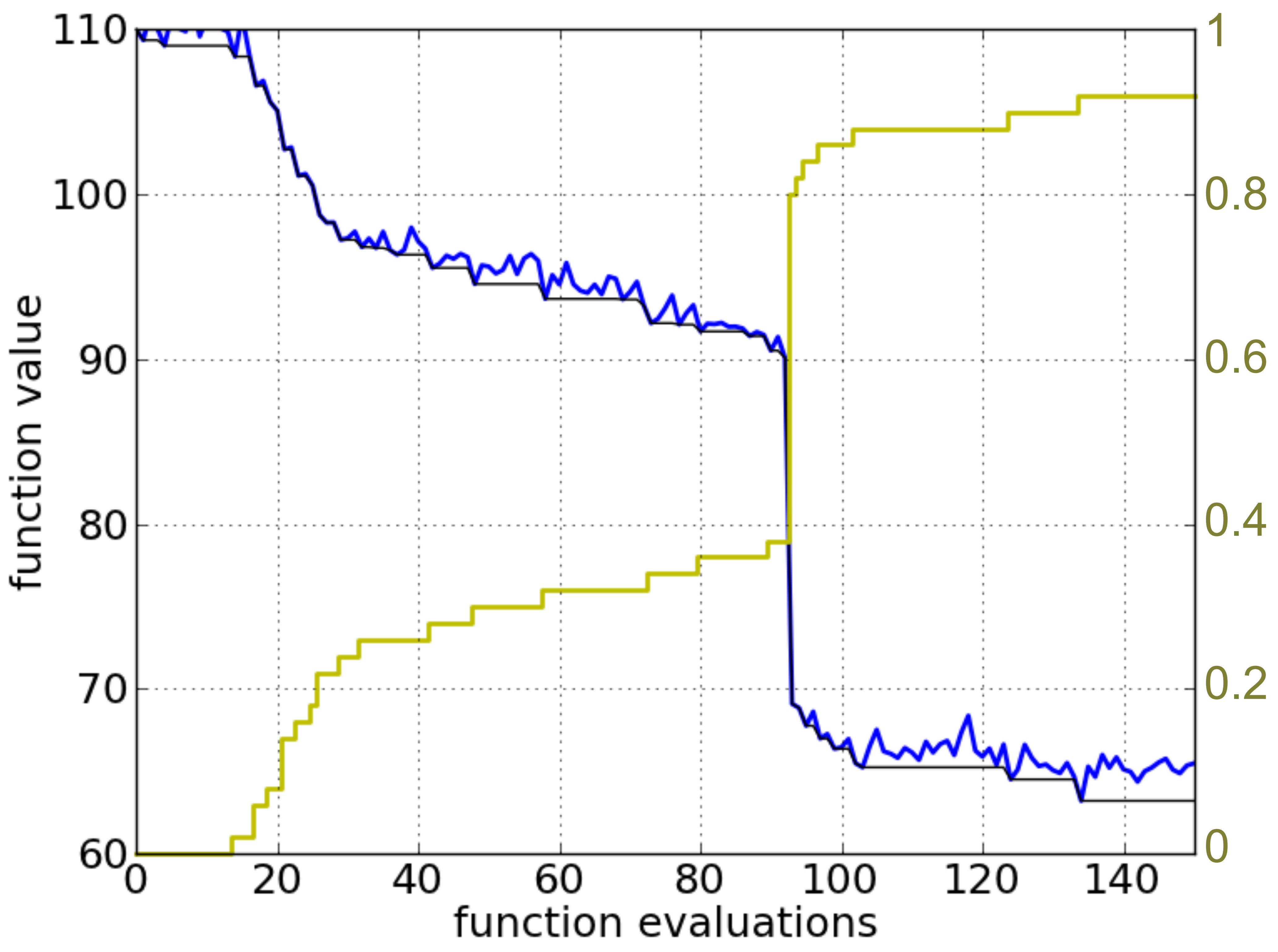
Runtime distribution from a single graph



the ECDF recovers the monotonous graph

AKA runtime distribution

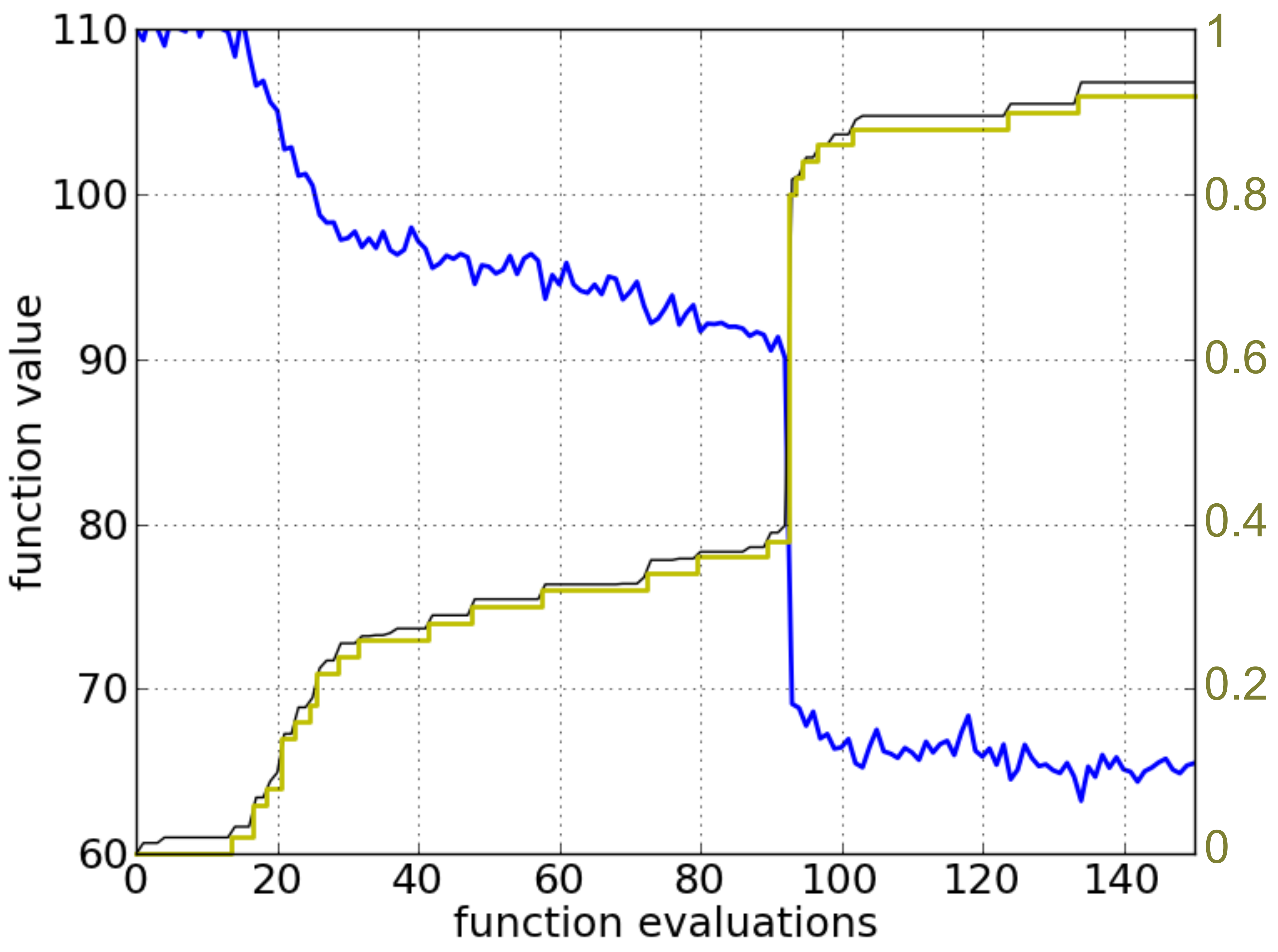
Runtime distribution from a single graph



the ECDF recovers the monotonous graph, discretised and flipped

AKA runtime distribution

Runtime distribution from a single graph

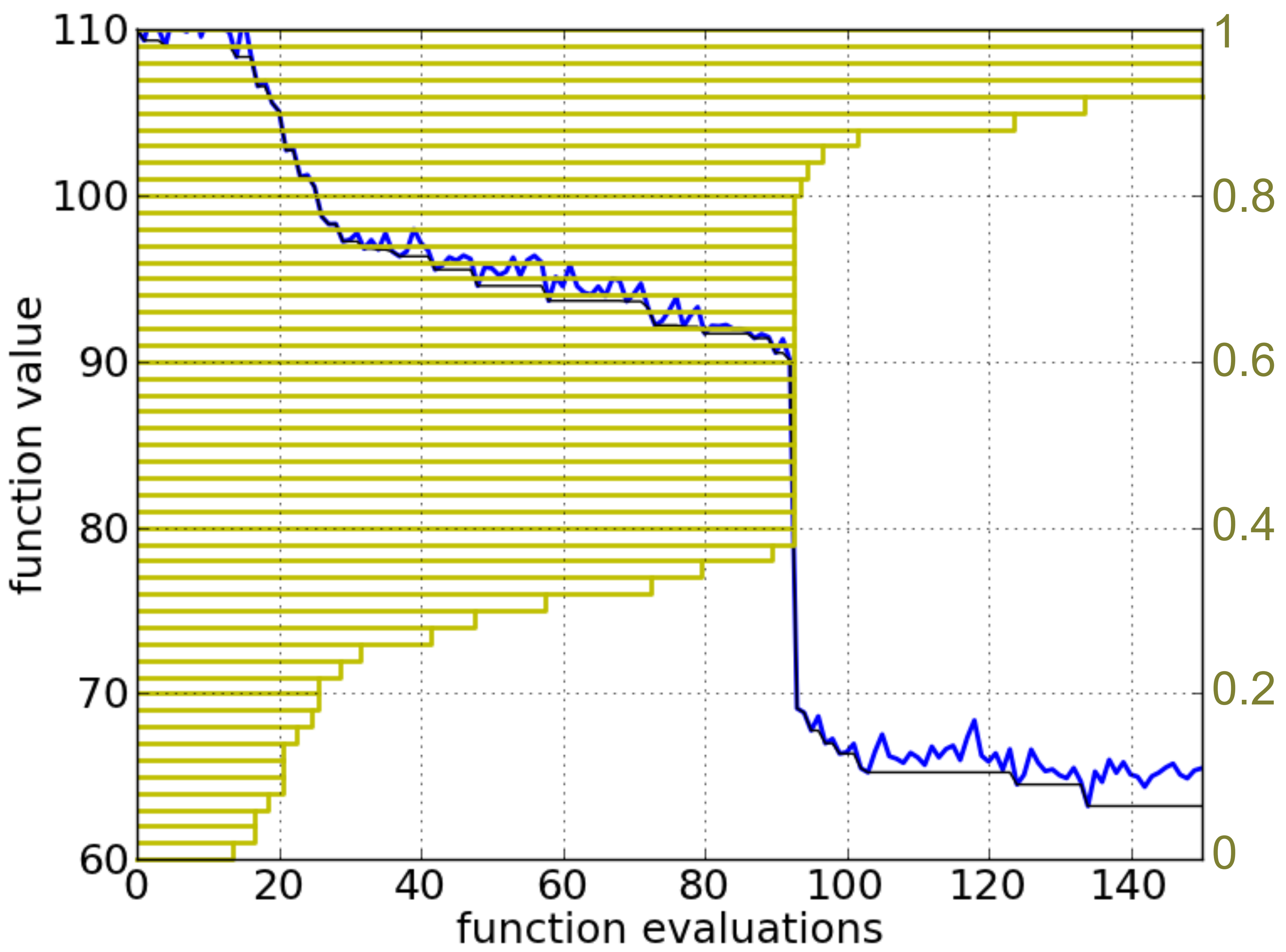


the ECDF recovers the monotonous graph, discretised and flipped

- recovering the convergence graph from discretized data
- collecting runtimes from a single experiments as ECDF

are two interpretations of the same thing

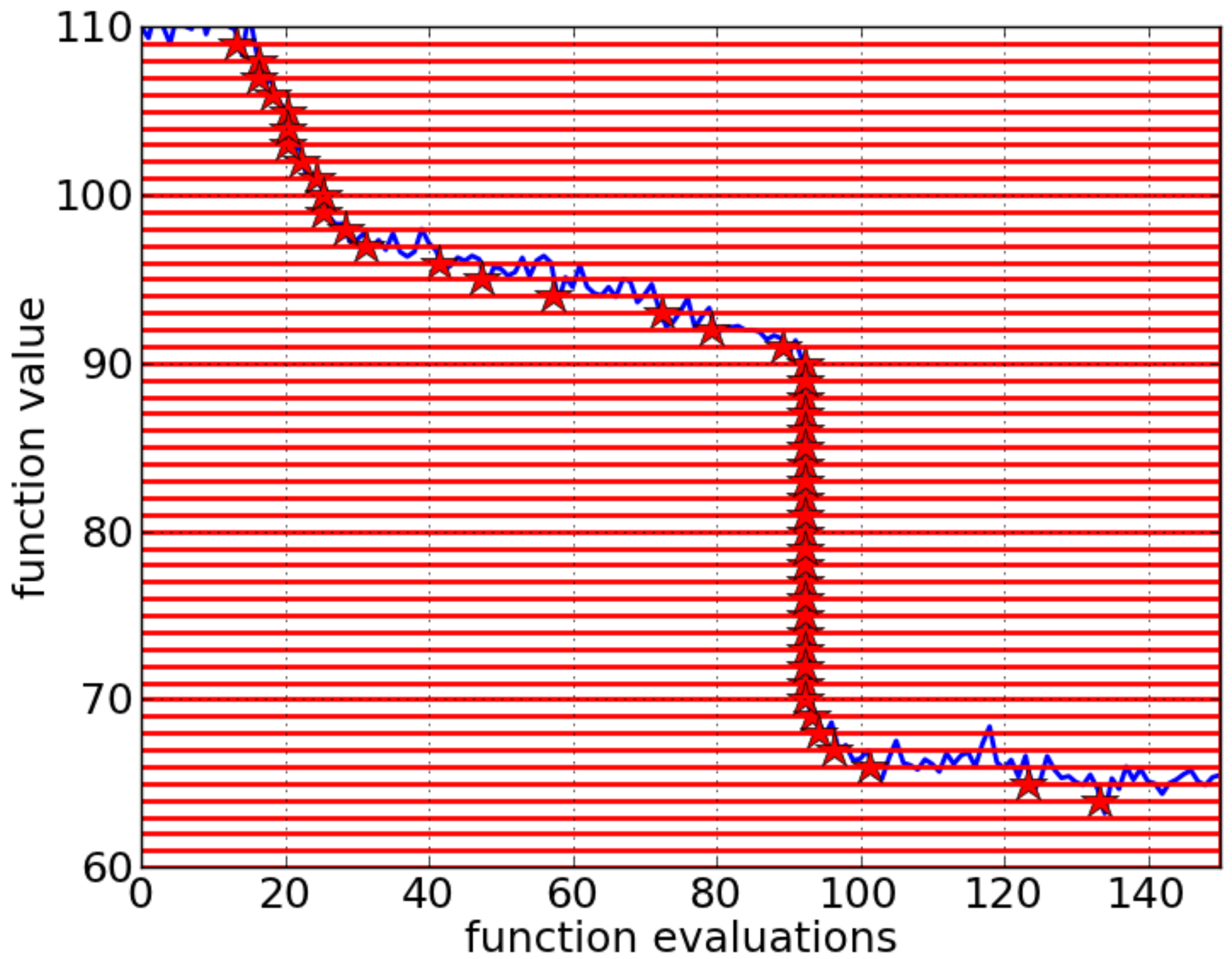
Runtime distribution from a single graph



the ECDF recovers the monotonous graph, discretised and flipped

the area over the ECDF curve is the average runtime (the geometric average if the x-axis is in log scale)

COCO/BBOB



uses only

horizontal discretization

this is

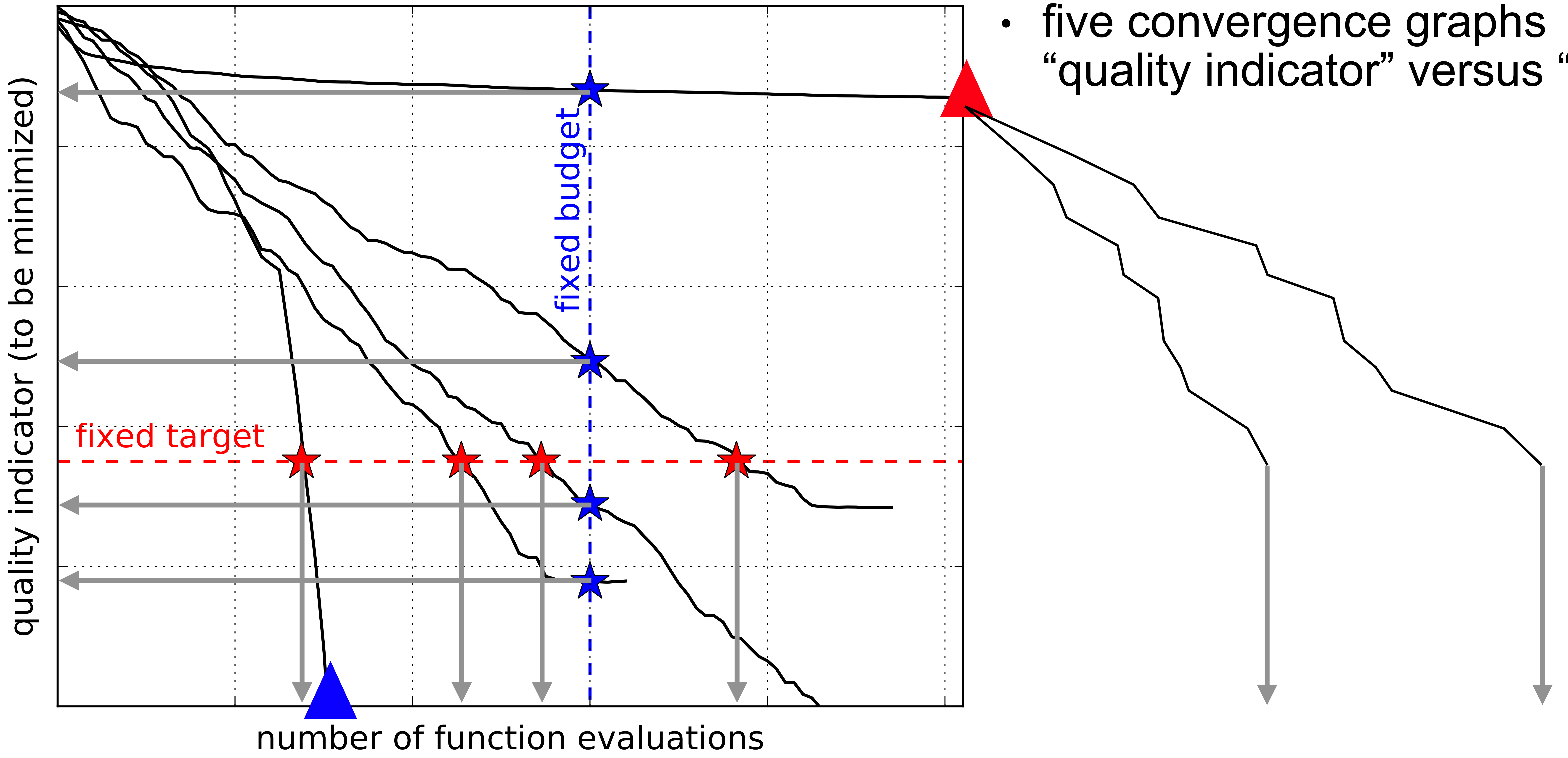
not

just

a technical subtlety

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

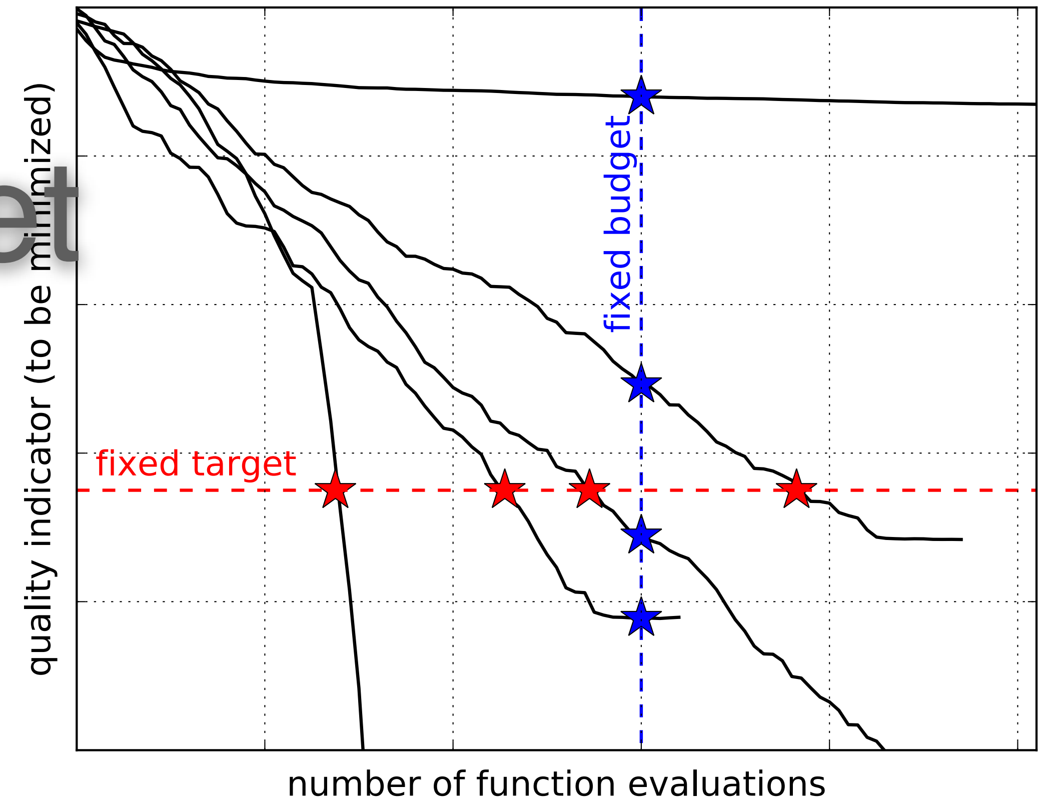
COCO/BBOB: Fixed Target(s) versus Fixed Budget



- 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)

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

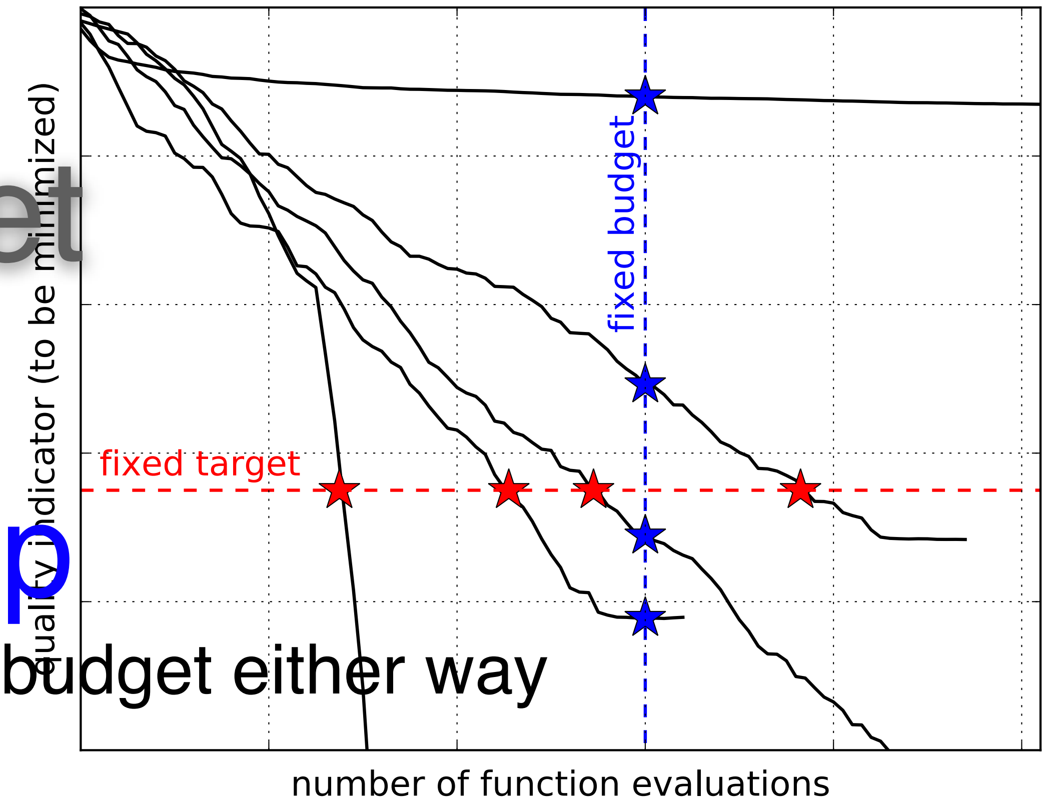


The resulting measurement

- Fixed budget (vertical) design: function values

- Fixed target design: evaluations

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



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

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

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

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

- 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 not commensurable

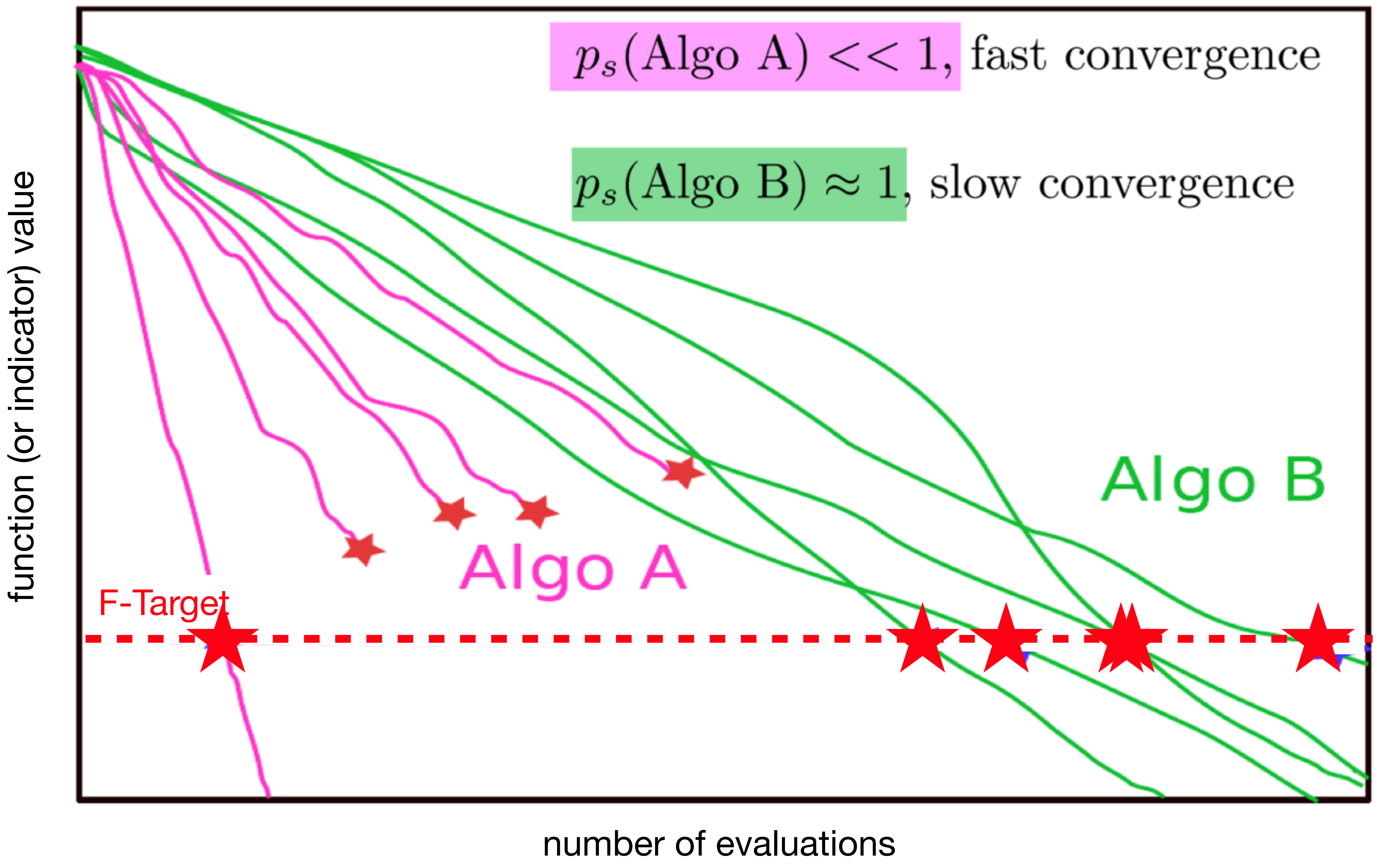
Scales of Measurement ("Quality" of Data)

- 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 (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*.

Treating Success Probabilities

Solving the fast-versus-successful comparison dilemma



Treating Success Probabilities

Solving the fast-versus-successful comparison dilemma

Short answer: consider as runtime

$$\frac{\text{something}}{p_{\text{success}}}$$

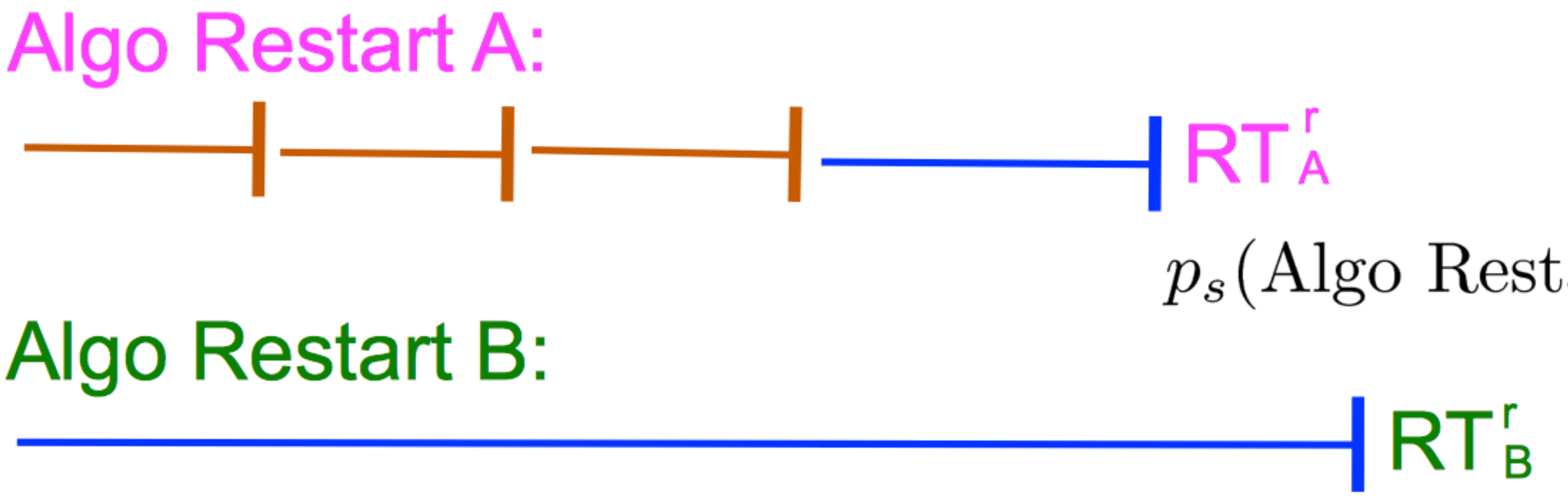
that is, roughly,

$$\text{runtime} \propto \frac{1}{p_{\text{success}}}$$

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



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

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

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

Treating Success Probabilities

Solving the fast-versus-successful comparison dilemma

Replacing the success probability with the expected runtime (ERT, aka Enes, SP2, aRT) to hit a target value in #evaluations is computed (estimated) as:

$$
\begin{aligned}
\mathrm{ERT} &= \frac{\#\text{evaluations(until to hit the target)}}{\#\text{successes}} \\
&= \text{avg}(\text{evals}_{\text{succ}}) + \overbrace{\frac{N_{\text{unsucc}}}{N_{\text{succ}}}}^{\text{odds ratio}} \times \text{avg}(\text{evals}_{\text{unsucc}}) \\
&\approx \text{avg}(\text{evals}_{\text{succ}}) + \frac{N_{\text{unsucc}}}{N_{\text{succ}}} \times \text{avg}(\text{evals}_{\text{succ}}) \\
&= \frac{N_{\text{succ}} + N_{\text{unsucc}}}{N_{\text{succ}}} \times \text{avg}(\text{evals}_{\text{succ}}) \\
&= \frac{1}{\text{success rate}} \times \text{avg}(\text{evals}_{\text{succ}})
\end{aligned}
$$

unsuccessful runs count (only) in the nominator

defined (only) for #successes > 0. The last three lines are AKA Q-measure or SP1 (success performance).

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

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].