Derivative-Free Optimization
Benchmarking and Performance Assessment

December 15, 2023

M2 Optimization, Université Paris-Saclay

Dimo Brockhoff

Inria Saclay – Ile-de-France

INVENTORS FOR THE DIGITAL WORLD

INSTITUT POLYTECHNIQUE DE PARIS

ÉCOLE POLYTECHNIQUE

IP PARIS

Practical Group Project

What do we want to achieve?

- understand some (new or not so new) algorithms
- learn how to conduct scientific testing, how to assess performance, and how to draw scientific conclusions from benchmarking data

How does it work?

- Today:
  - introduction to benchmarking
  - installation of COCO on your machines
  - first exercise to interpret (existing) benchmarking data
- In groups of 3
  - please use this to organize yourself: http://tinyurl.com/cocoproject2023

Practical Group Project

- We propose some possible algorithms (max. 1 group/algo)
  - but you can take others (e.g. your own)
  - benchmark this algorithm with the help of COCO (or other benchmarking software)
- 27th of January:
  - report (PDF) sent by email, 8 pages, template given by COCO
- 2nd of February:
  - 12 minutes presentation per group
  - + 10 minutes questions
  - about the algorithm
  - about its performance (compared to other algos)
  - order of the talks to be decided early next year
- 12th of January:
  - send a short progress statement by email (<10 lines)

Ideas for Algorithms/Papers

1) HypE: hypervolume estimation algorithm for multiobj. opt.
(C implementation available at `https://sop.tik.ee.ethz.ch/download/supplementary/hype/`)
2) ParEGO (multiobj. Efficient Global Optimization variant)
3) Gurobi (mixed-integer solver, academic license for free)
4) Nelder-Mead (on bbob-mixint)
5) Multiobjective NOMAD (`https://www.gerad.ca/en/software/nomad/`)
6a) Algorithms from PRIMA (`https://github.com/libprima/prima`)
6b) Algorithms from PDFO (`https://pdfo.net`)
7) Turbo (`https://github.com/uber-research/TuRBO`)
8) derivative-free Newton method
9) SLSQP (from scipy.optimize) for constrained problems

[you will see more information/links in the Google document]

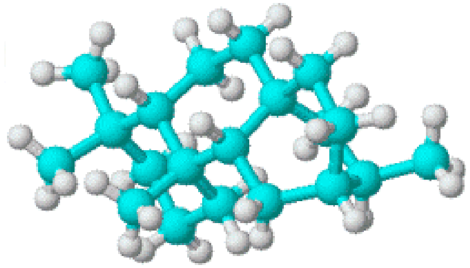
Ideas for Algorithms/Papers

1) HypE: hypervolume estimation algorithm for multiobj. opt.
(C implementation available at `https://sop.tik.ee.ethz.ch/download/supplementary/hype/`)
2) ParEGO (multiobj. Efficient Global Optimization variant)
3) Gurobi (mixed-integer solver, academic license for free)
4) Nelder-Mead (on bbob-mixint)
5) Multiobjective NOMAD (`https://www.gerad.ca/en/software/nomad/`)
6a) Algorithms from PRIMA (`https://github.com/libprima/prima`)
6b) Algorithms from PDFO (`https://pdfo.net`)
7) Turbo (`https://github.com/uber-research/TuRBO`)
8) derivative-free Newton method
9) SLSQP (from scipy.optimize) for constrained problems

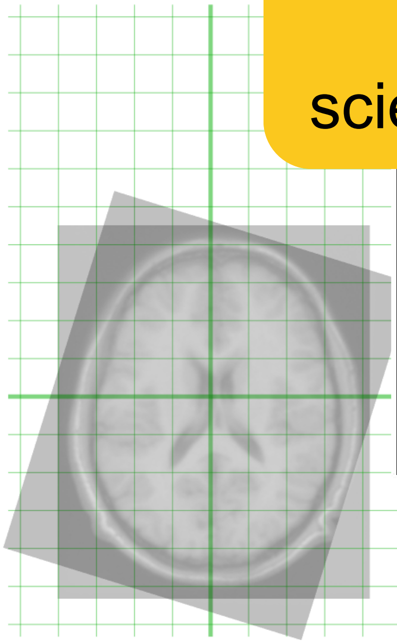
Note: for none of those algorithms,
benchmarking data is available with COCO
→ additional (workshop) paper project possible afterwards

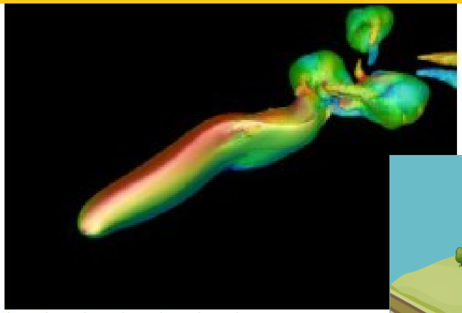
Benchmarking Optimization Algorithms

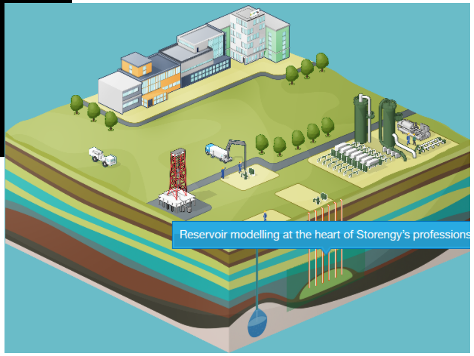
or: critical performance assessment



challenging optimization problems
appear in many
scientific, technological and industrial domains







Practical (Numerical) Blackbox Optimization

Given:

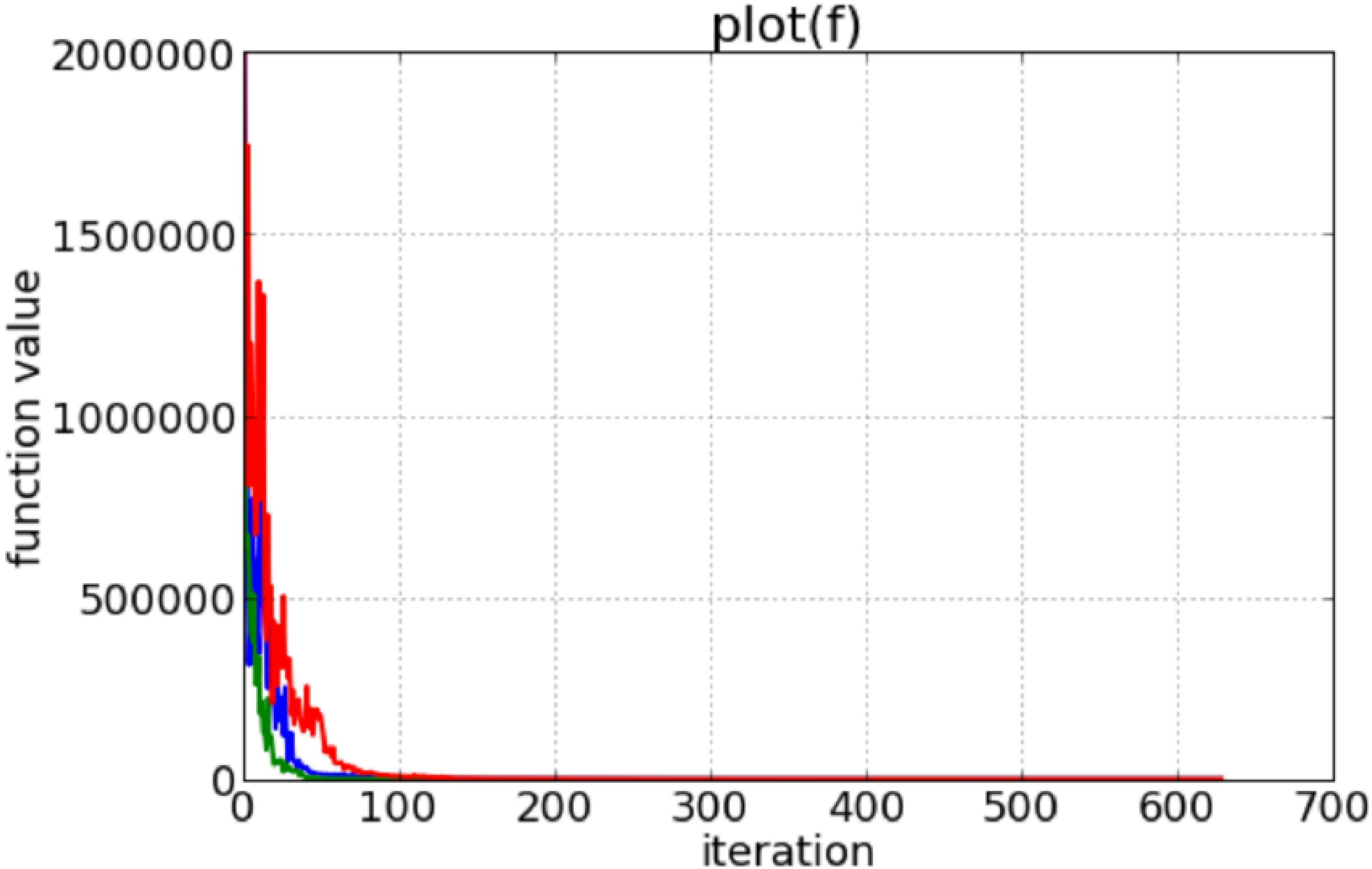
derivatives not available or not useful

Not clear:

which of the many algorithms should I use on my problem?

visualizing the raw data (single runs)

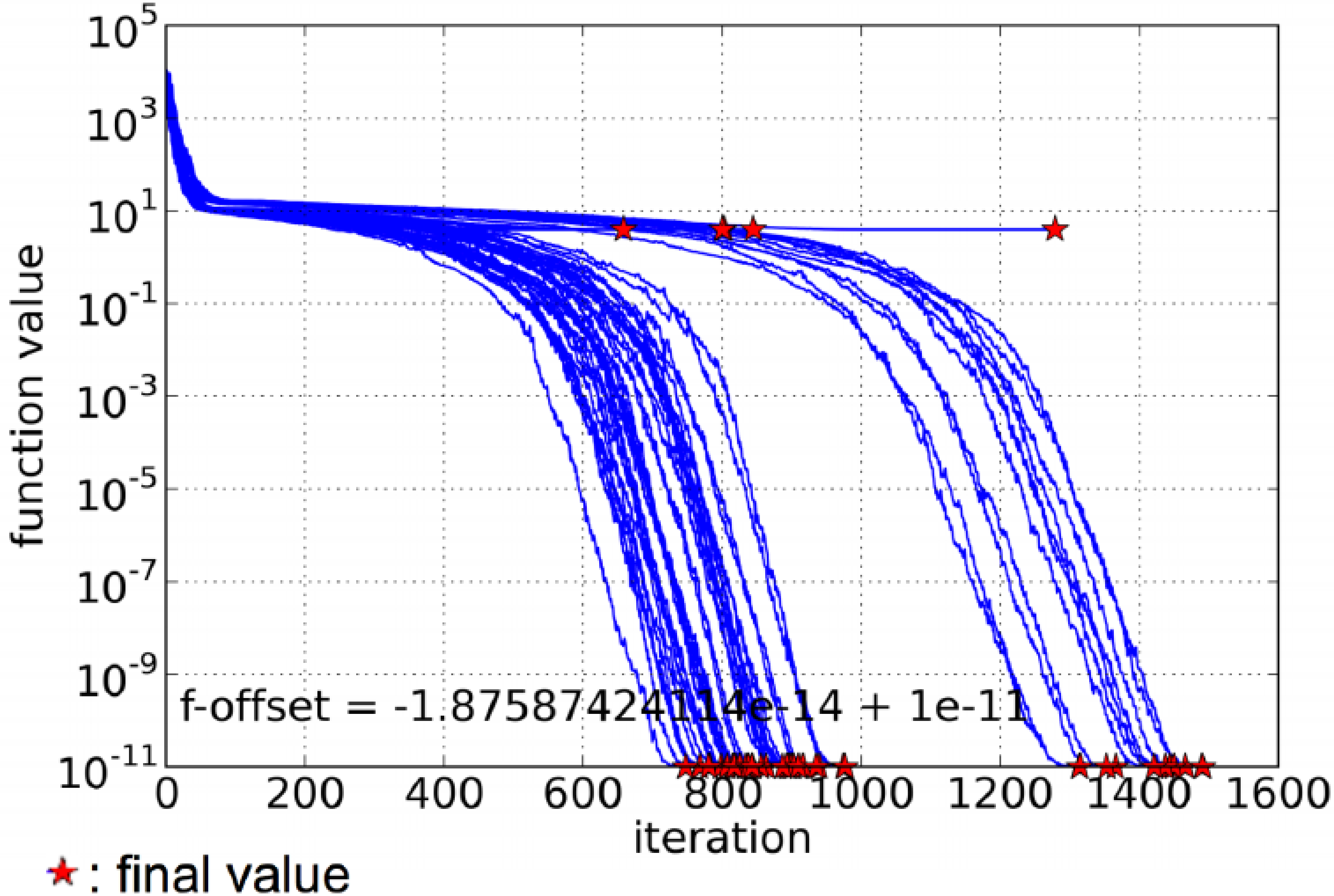
Displaying 3 runs (3 trials)



Copyright: A. Auger and N. Hansen

Displaying 51 runs

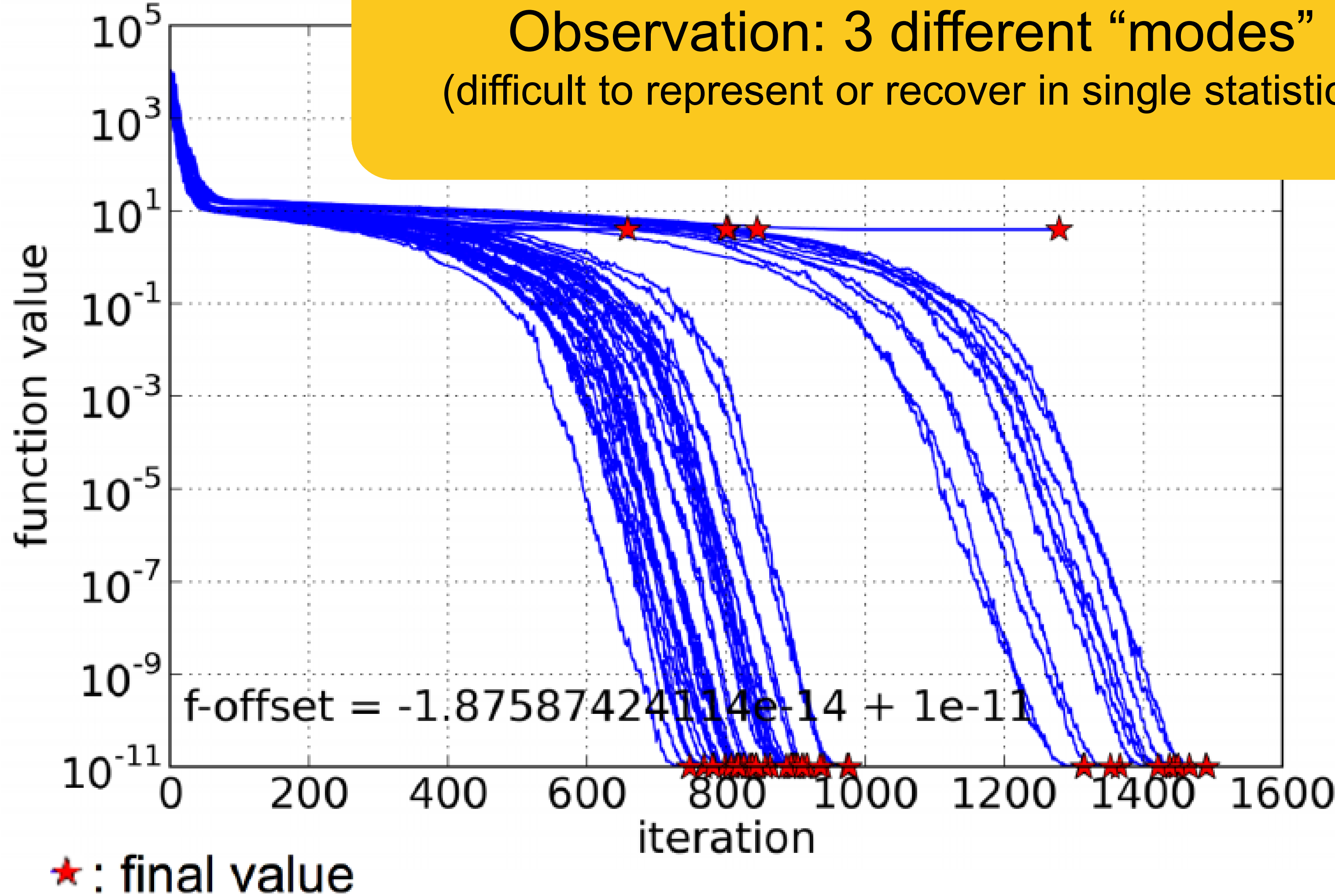
Don't hesitate to display all data (the appendix is your friend)



Copyright: A. Auger and N. Hansen

Displaying 51 runs

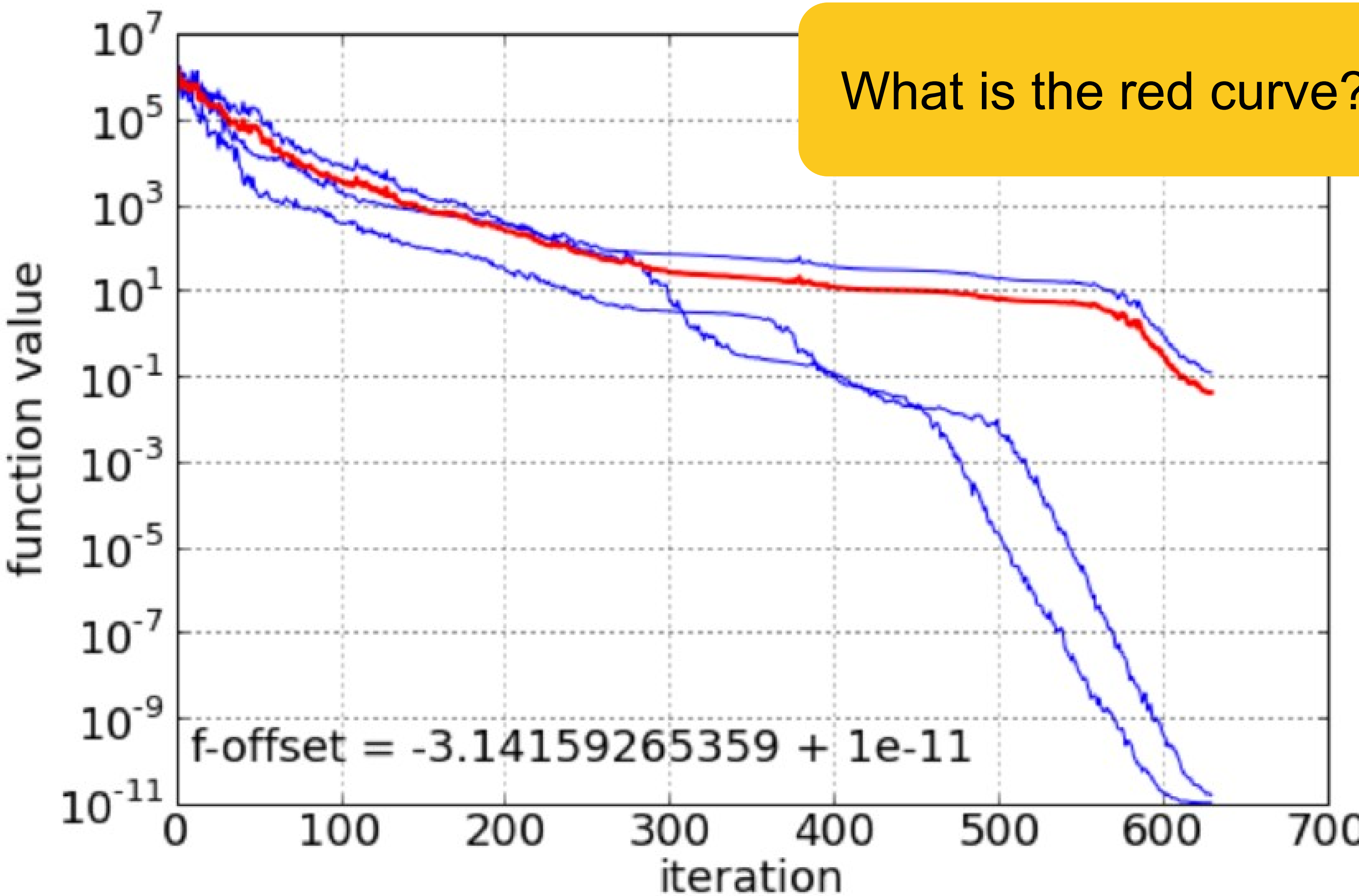
Don't hesitate to disp



Observation: 3 different "modes"
(difficult to represent or recover in single statistics)

Copyright: A. Auger and N. Hansen

Which Statistics?



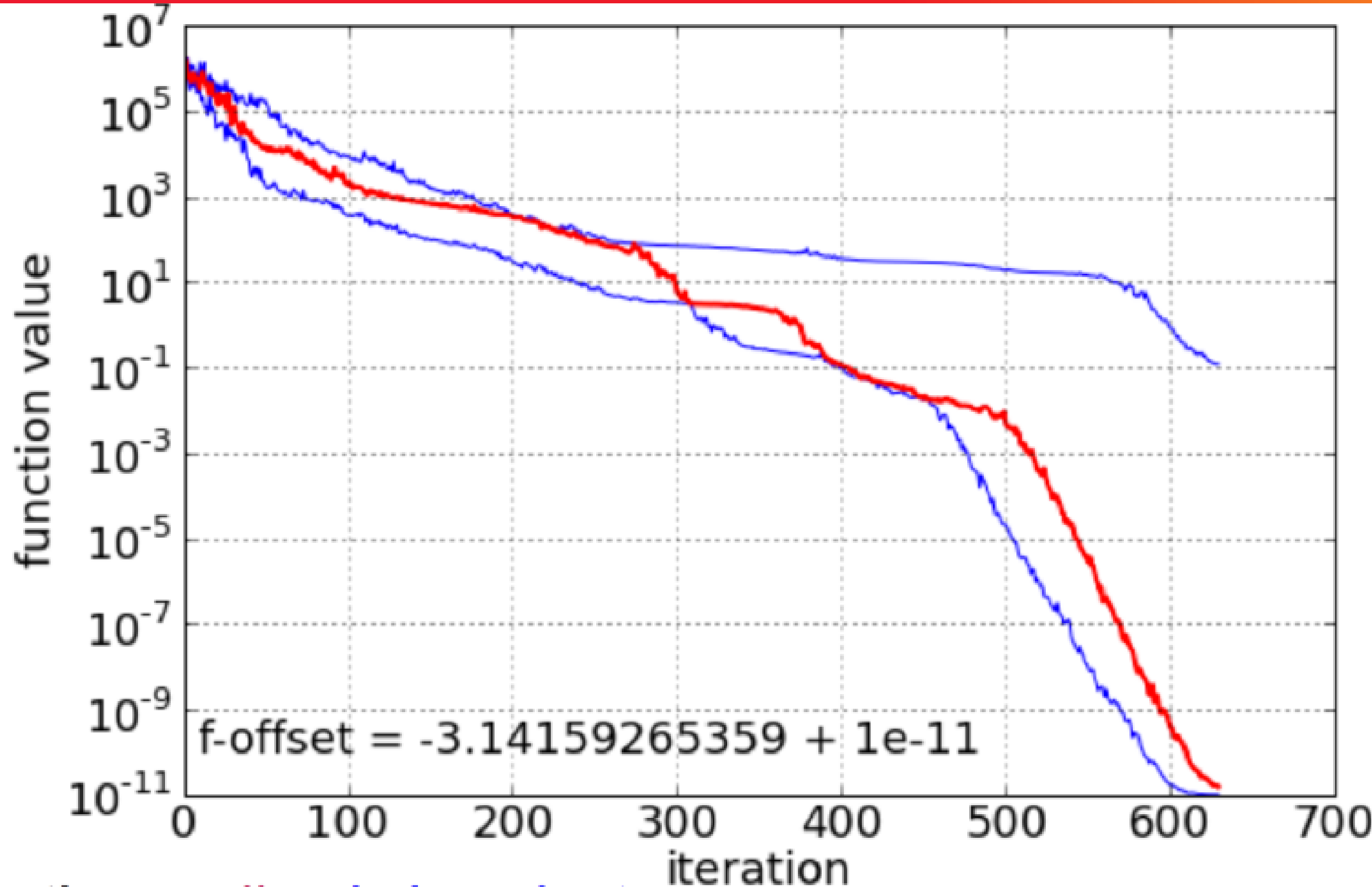
What is the red curve?

More Problems with Averages/Expectations

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

from Hansen GECCO 2019 Experimentation tutorial

Which Statistics?



the median is invariant

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

median(log(data))=log(median(data))

- same when taken over x- or y-direction

Copyright: A. Auger and N. Hansen

Implications

- use the median as summary datum

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

- more general: use quantiles as summary data

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

from Hansen GECCO 2019 Experimentation tutorial

Benchmarking =

evaluate the performance of optimization algorithms

compare the performance of different algorithms

understand strengths and weaknesses of algorithms

help in design of new algorithms

How Do We Measure Performance?

Meaningful quantitative measure

- quantitative on the ratio scale (highest possible)

  "algo A is two *times* better than algo B" is a meaningful statement
- assume a wide range of values
- meaningful (interpretable) with regard to the real world

  possible to transfer from benchmarking to real world

How Do We Measure Performance?

Meaningful quantitative measure

- quantitative on the ratio scale (highest possible)

  "algo A is two *times* better than algo B" is a meaningful statement
- assume a wide range of values
- meaningful (interpretable) with regard to the real world

  possible to transfer from benchmarking to real world

CPU timing not a good candidate

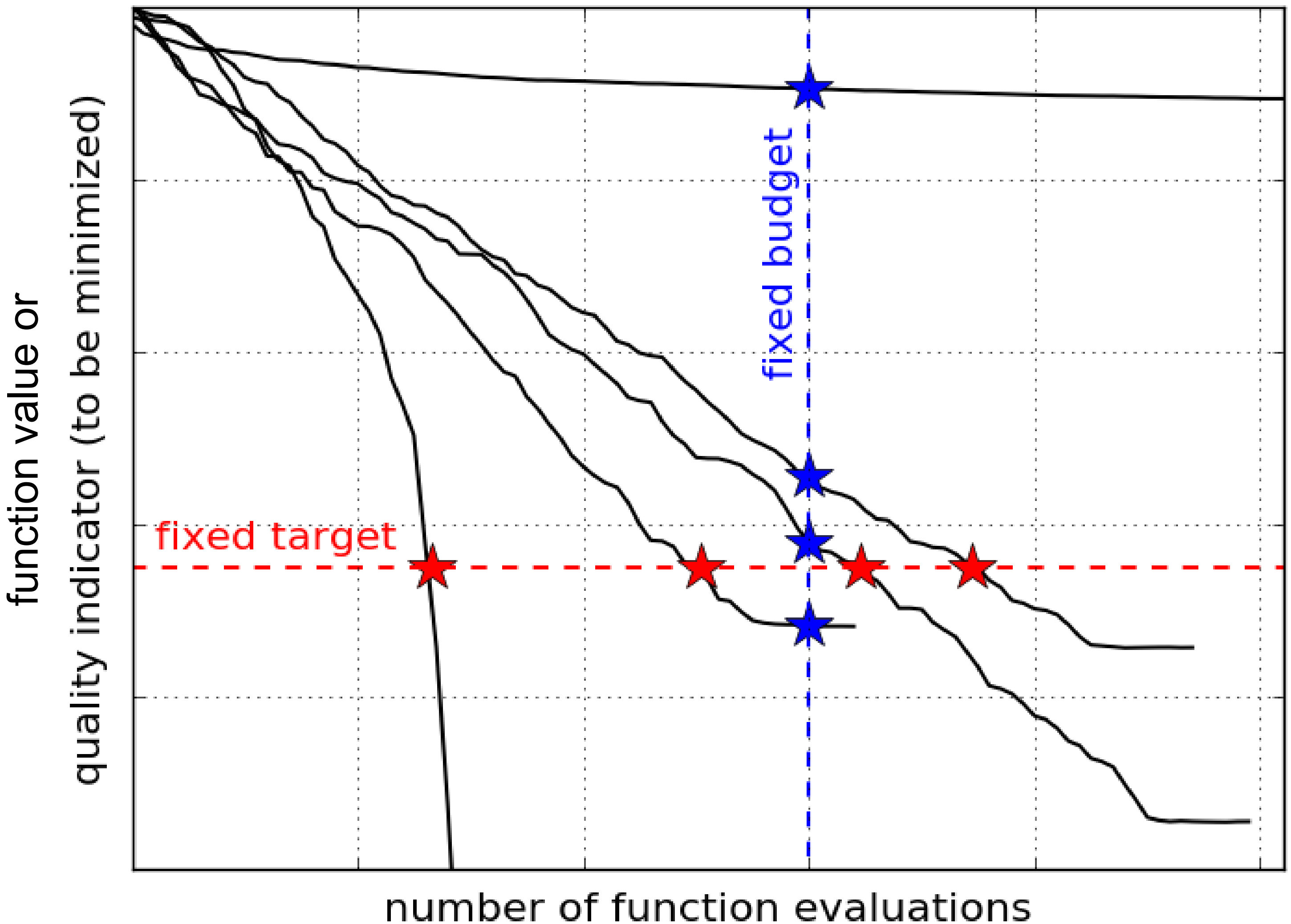
→depends on implementation/language/machine/…

time is spent on code optimization instead of science

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

Measuring Performance Empirically

convergence graphs is all we have to start with...



How Do We Measure Performance?

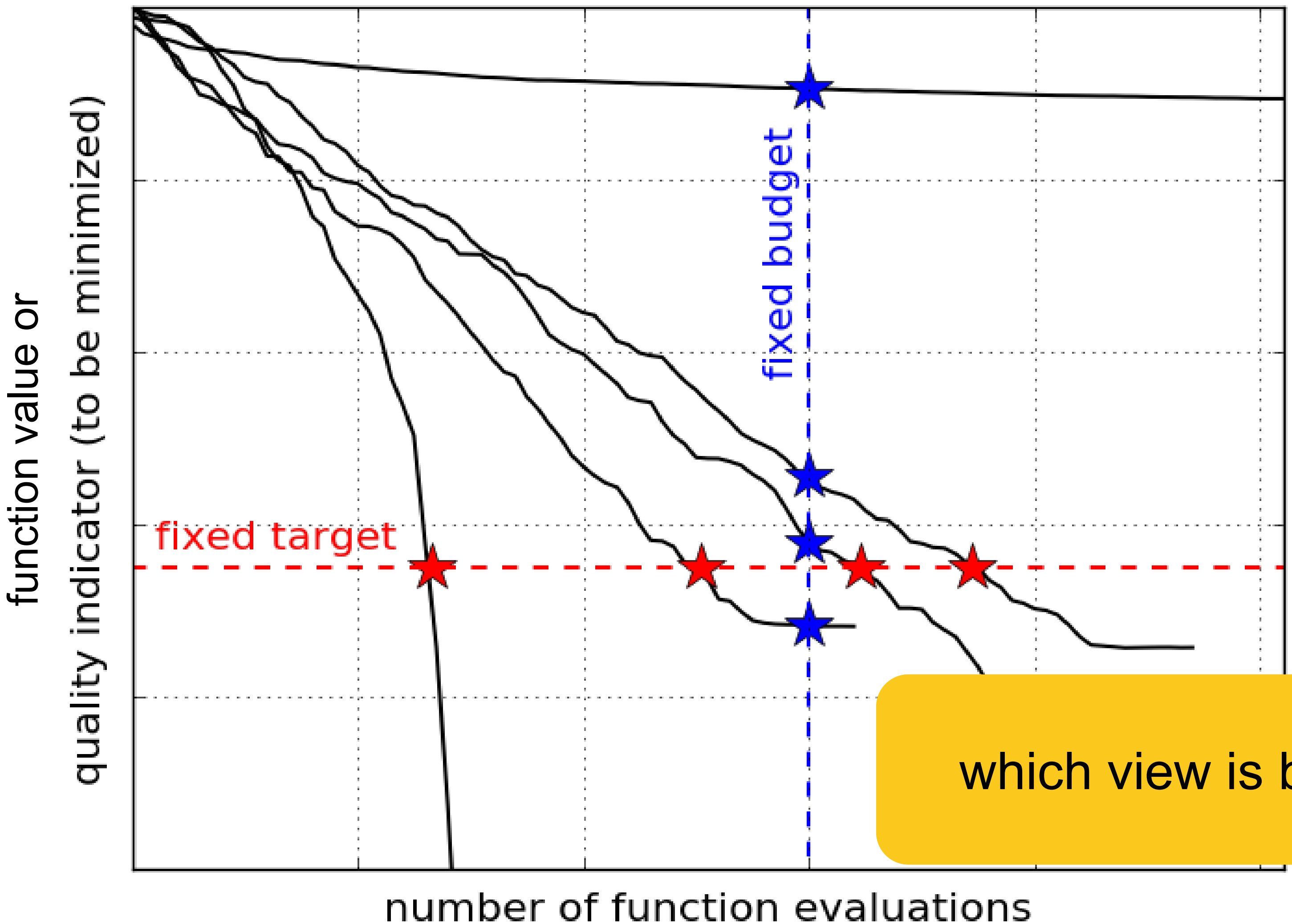
Two objectives:

- Find solution with small(est possible) function/indicator value
- With the least possible search costs (number of function evaluations)

For measuring performance: fix one and measure the other

Measuring Performance Empirically

convergence graphs is all we have to start with...



which view is better?

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

Repeat several times:

if algorithms are stochastic, never draw a conclusion from a single run

if deterministic algorithm, repeat by changing (randomly) the initial conditions

ECDF:

Empirical Cumulative Distribution Function of the Runtime

[aka data profile]

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

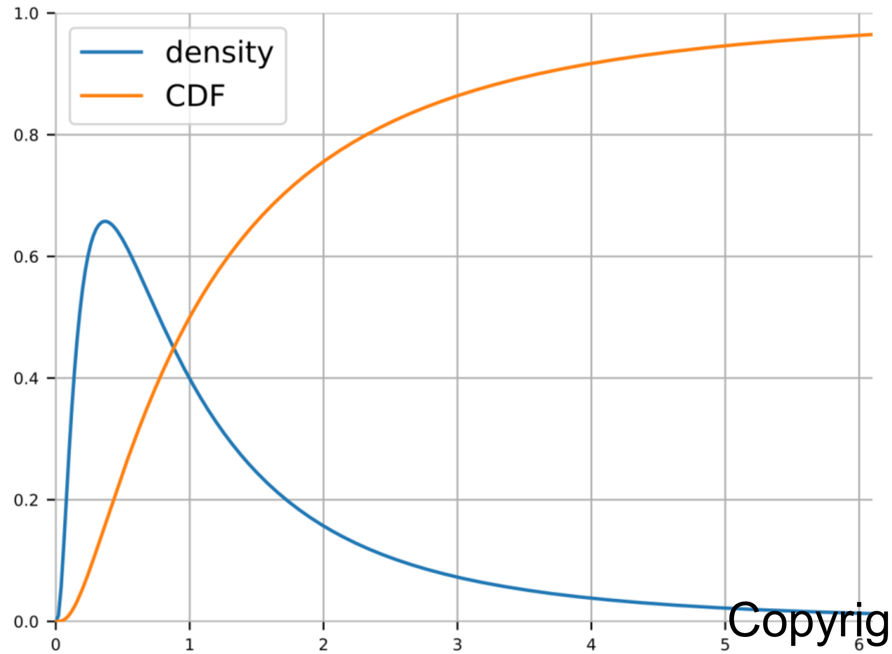
Cumulative Distribution Function (CDF)

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

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

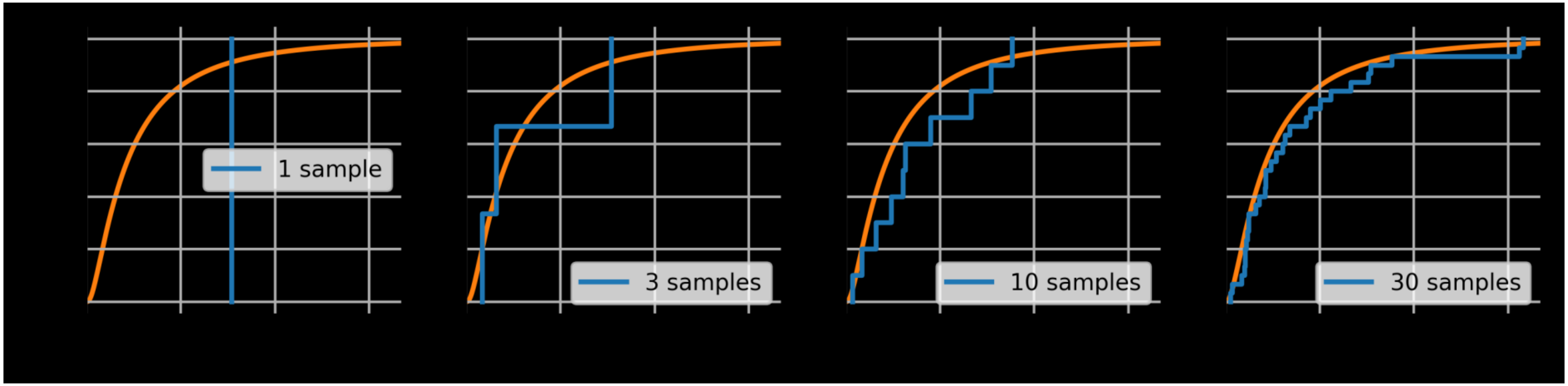
It characterizes the probability distribution of T

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



Copyright: A. Auger and N. Hansen

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.

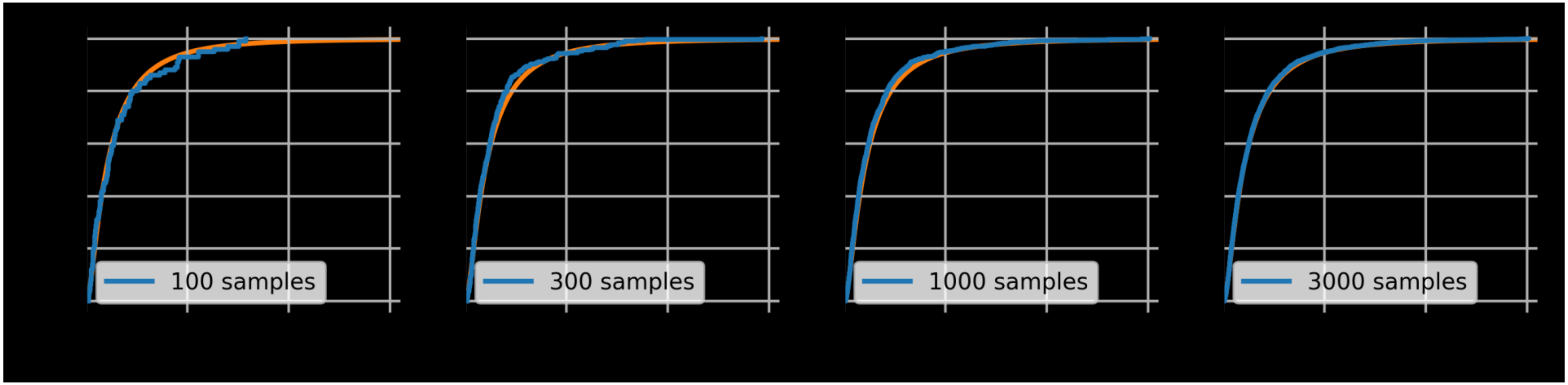


It is an estimate of the CDF that generated the points in the sample.

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

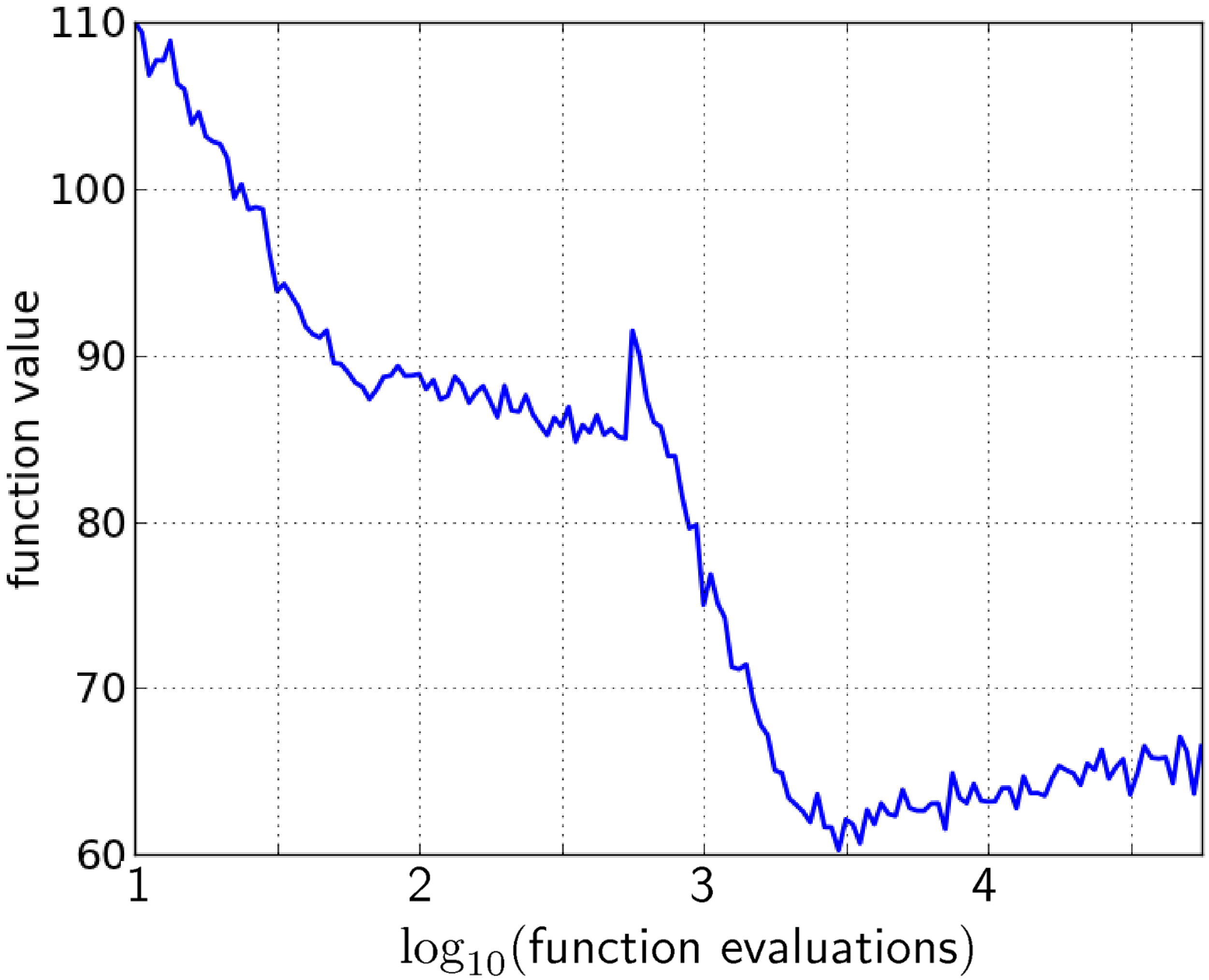
For $\{T_i : i \geq 1\}$ i.i.d. realization of a random variable T, by the LLN

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

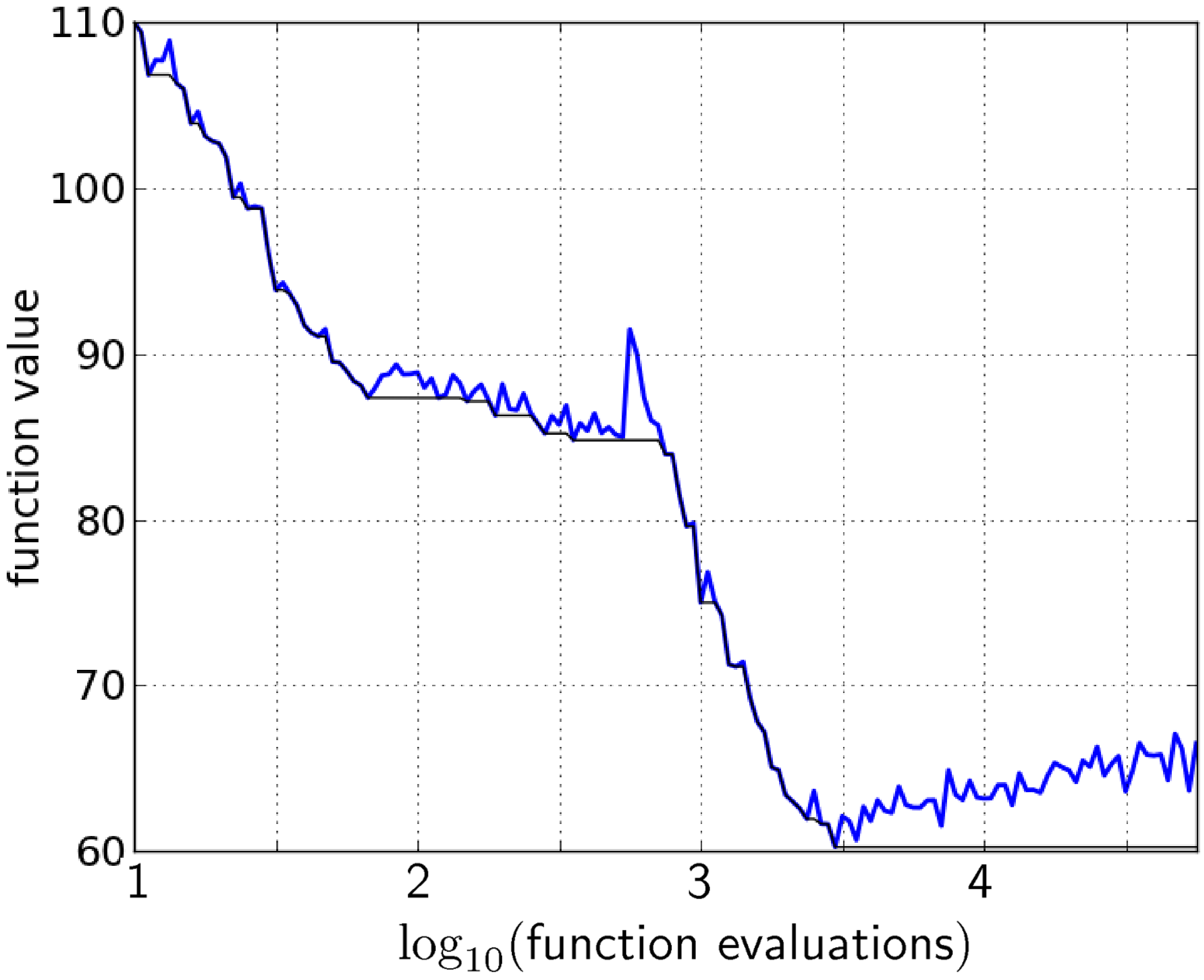


Copyright: A. Auger and N. Hansen

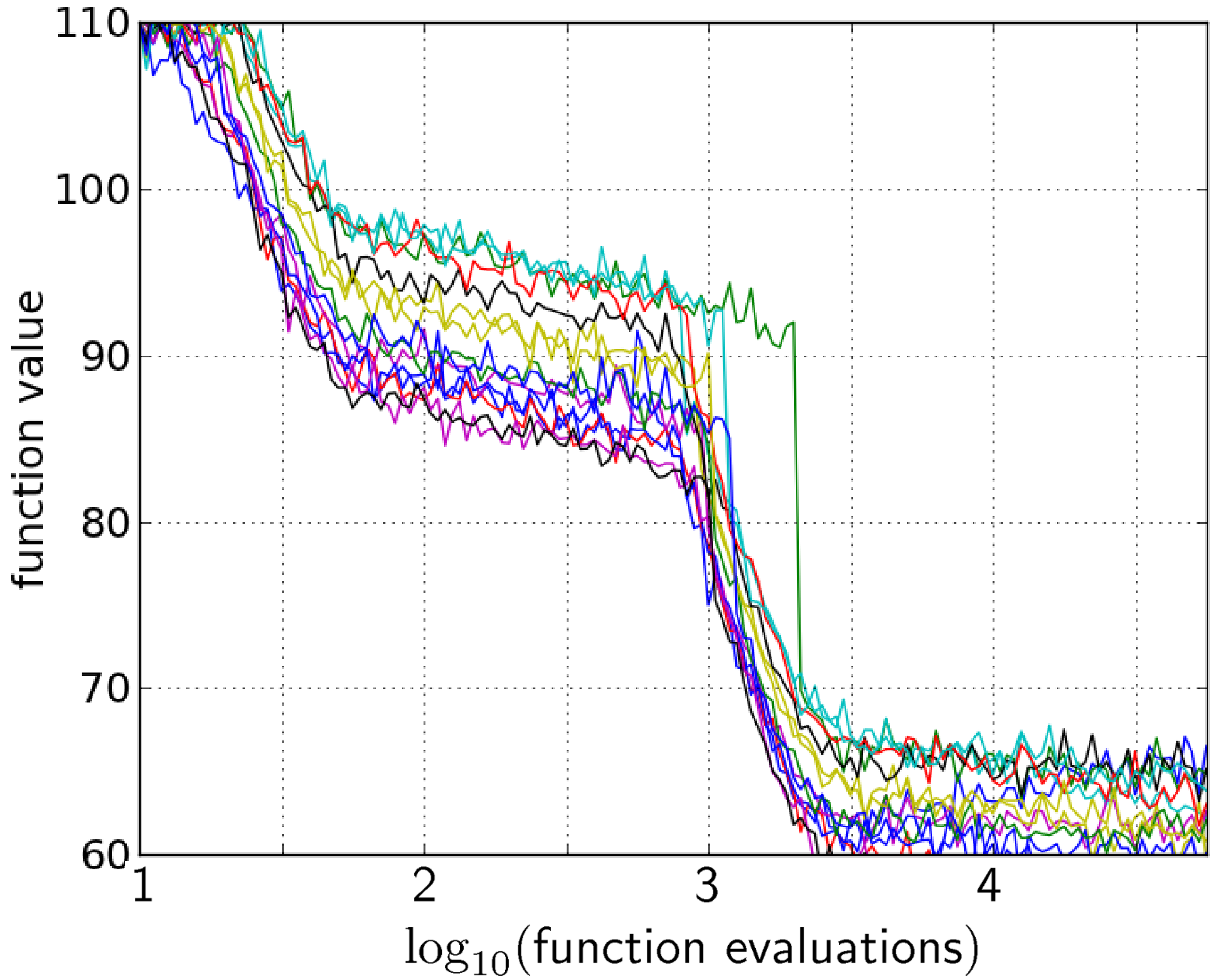
A Convergence Graph



First Hitting Time is Monotonous

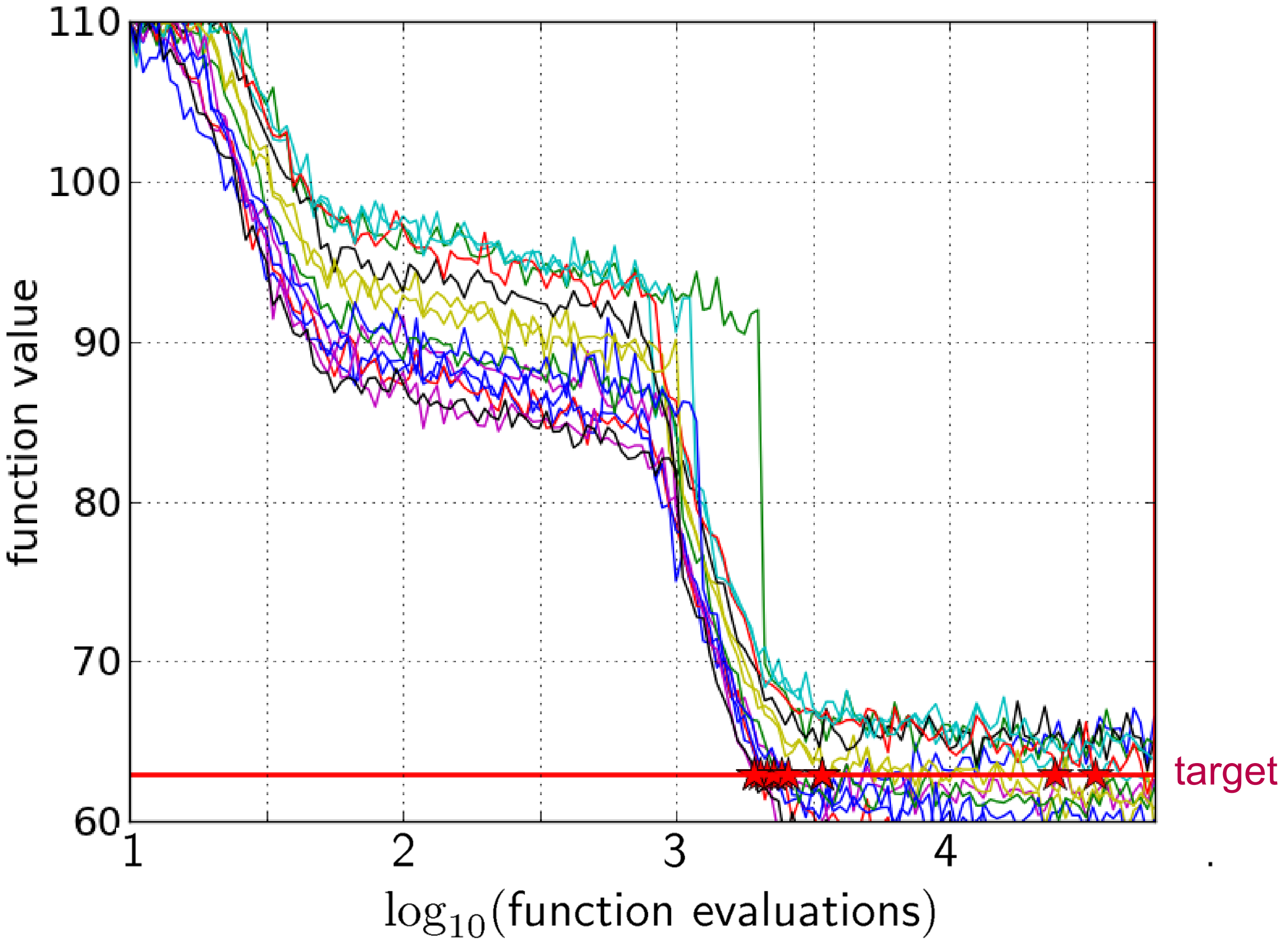


15 Runs

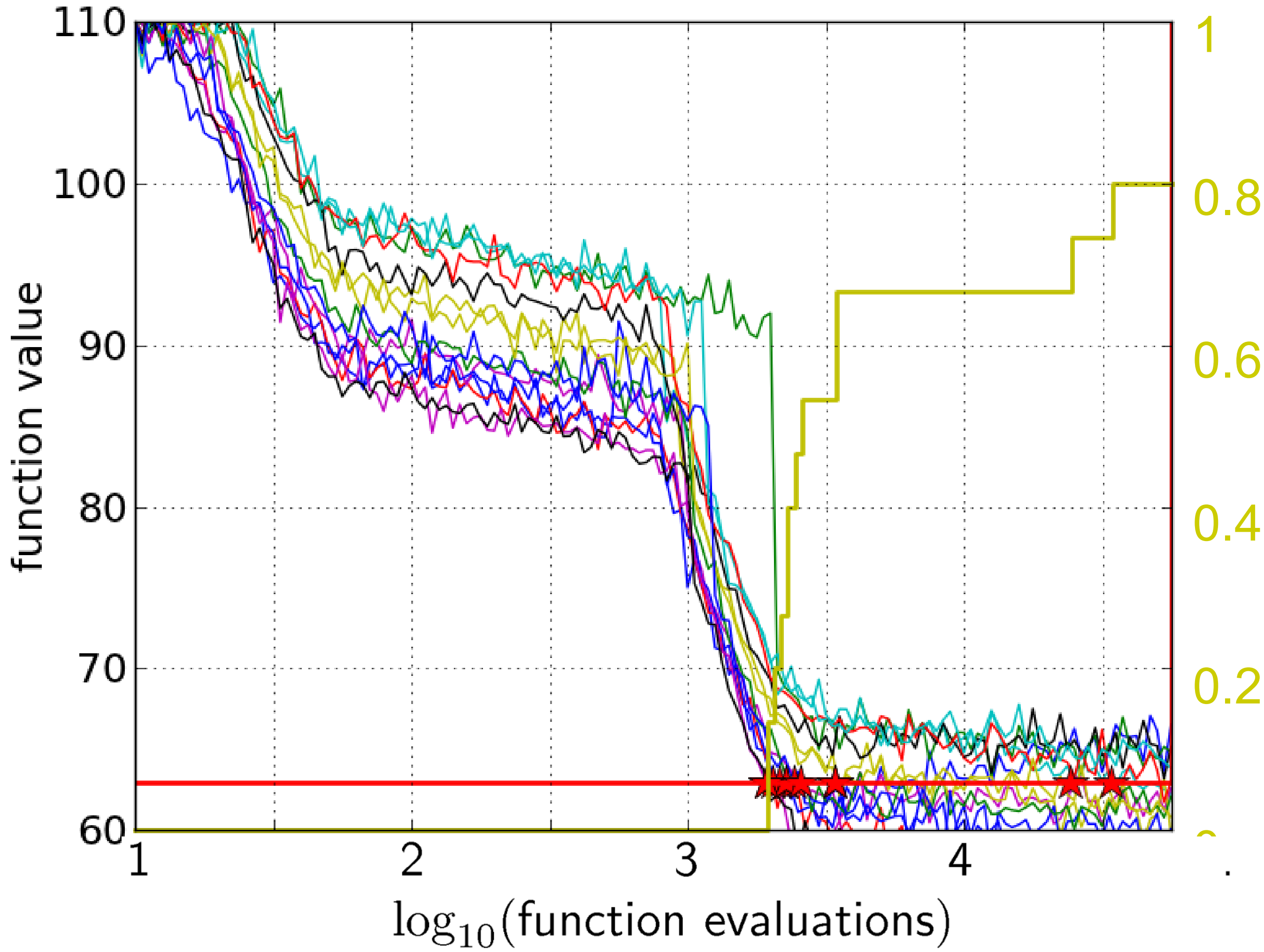


110
100
90
80
70
60
function value
1
2
3
4
$\log_{10}$(function evaluations)

15 Runs ≤ 15 Runtime Data Points



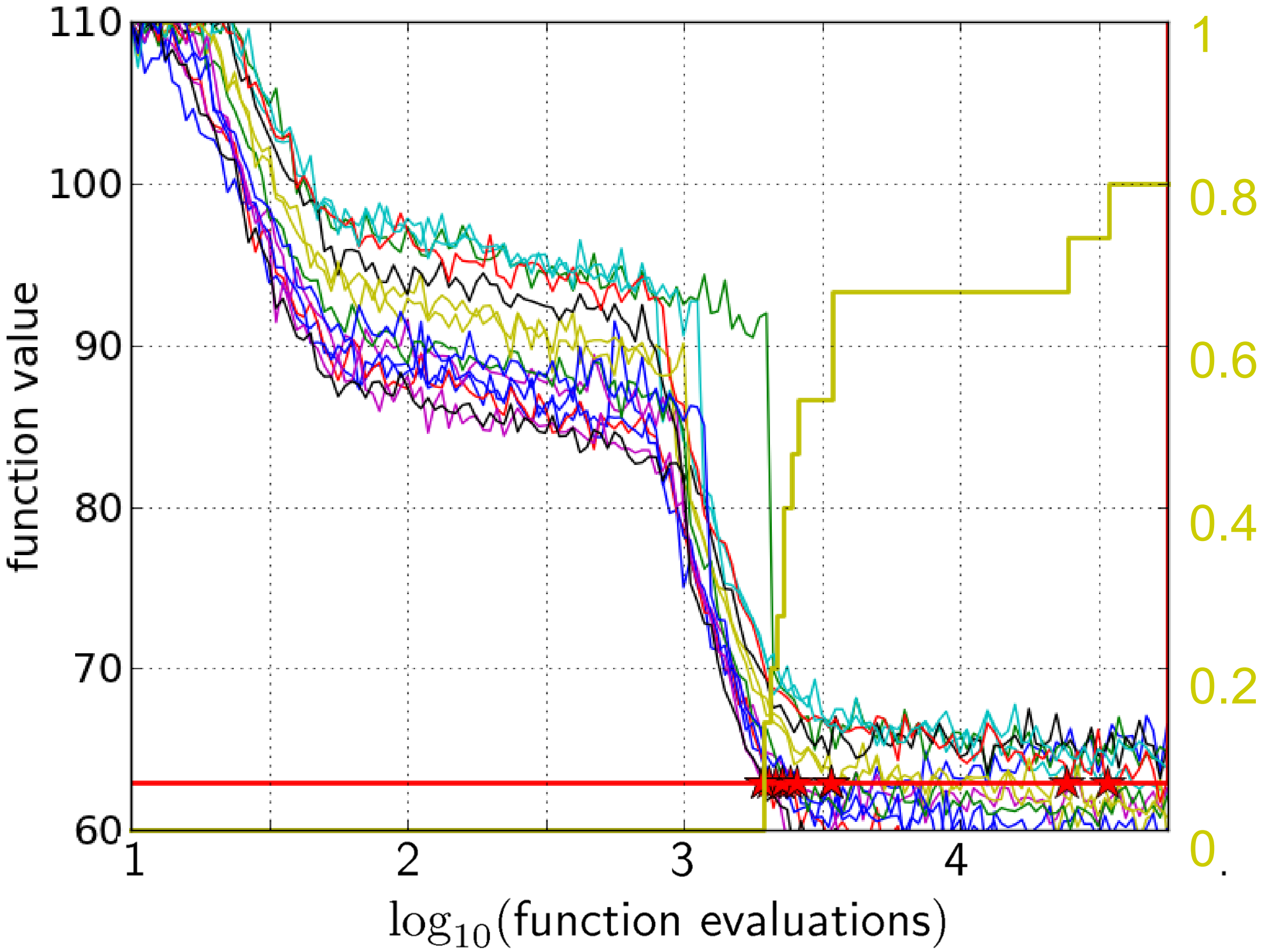
Empirical Cumulative Distribution



the ECDF of run lengths to reach the target

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

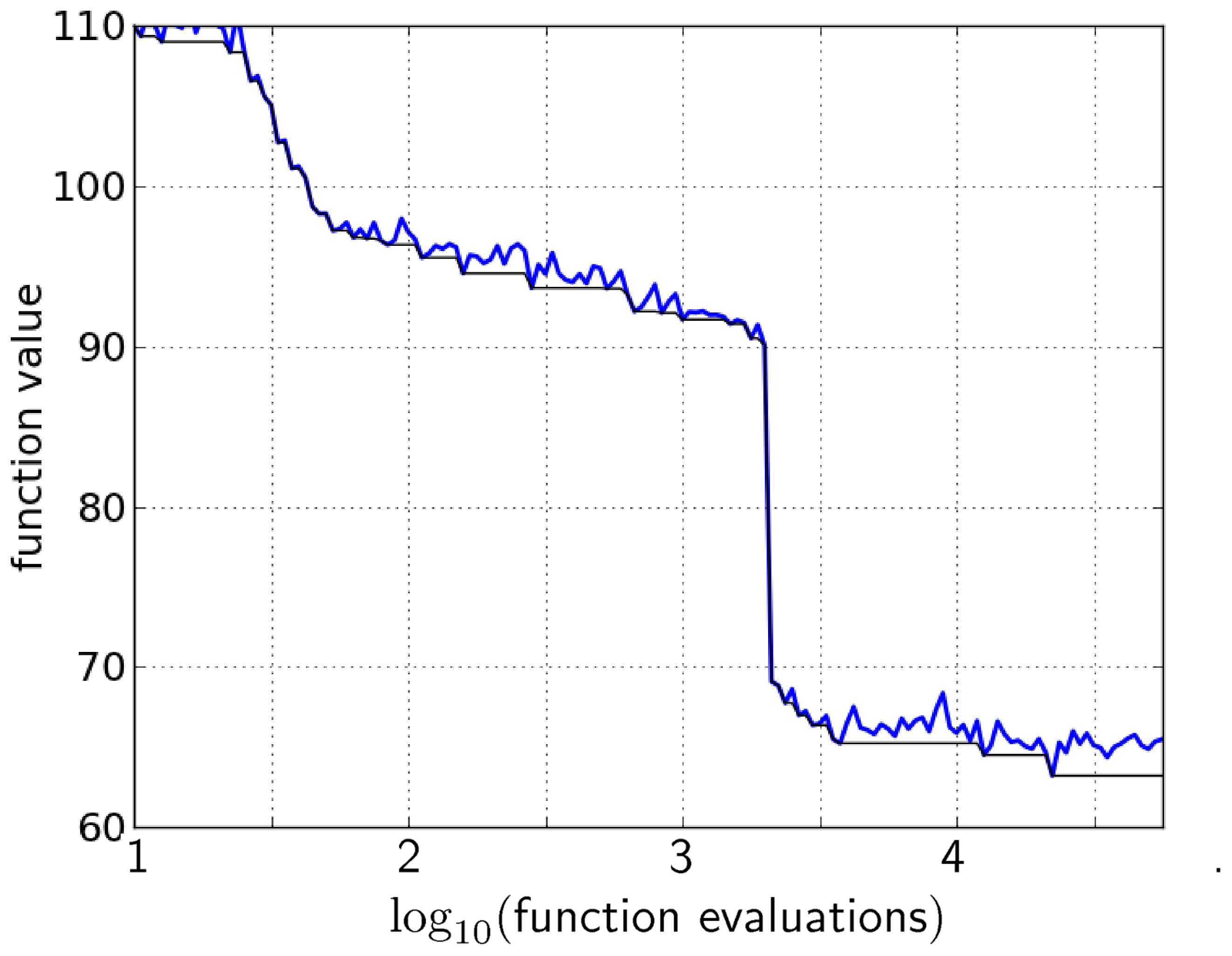
Empirical Cumulative Distribution



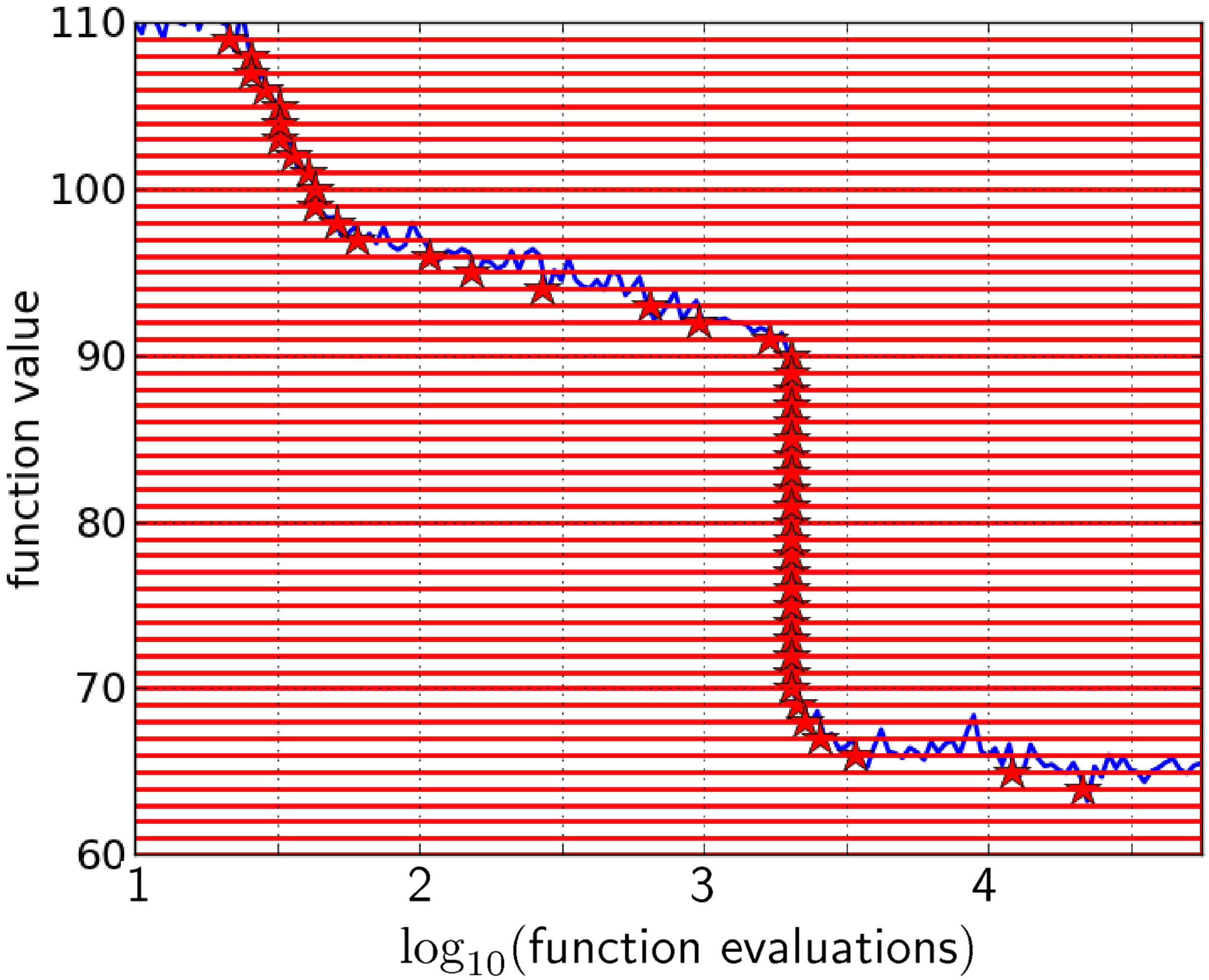
interpretations possible:

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

Reconstructing A Single Run

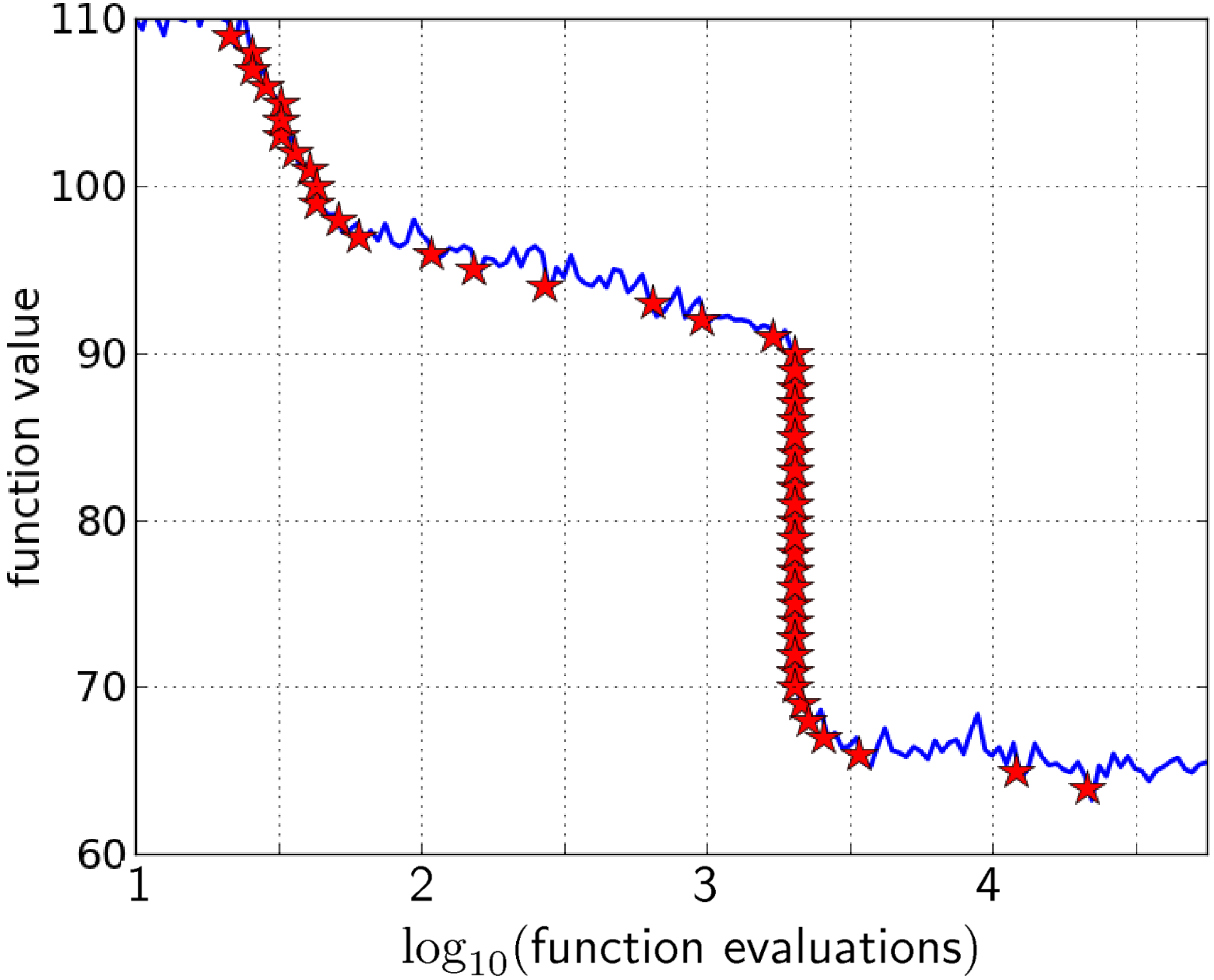


Reconstructing A Single Run



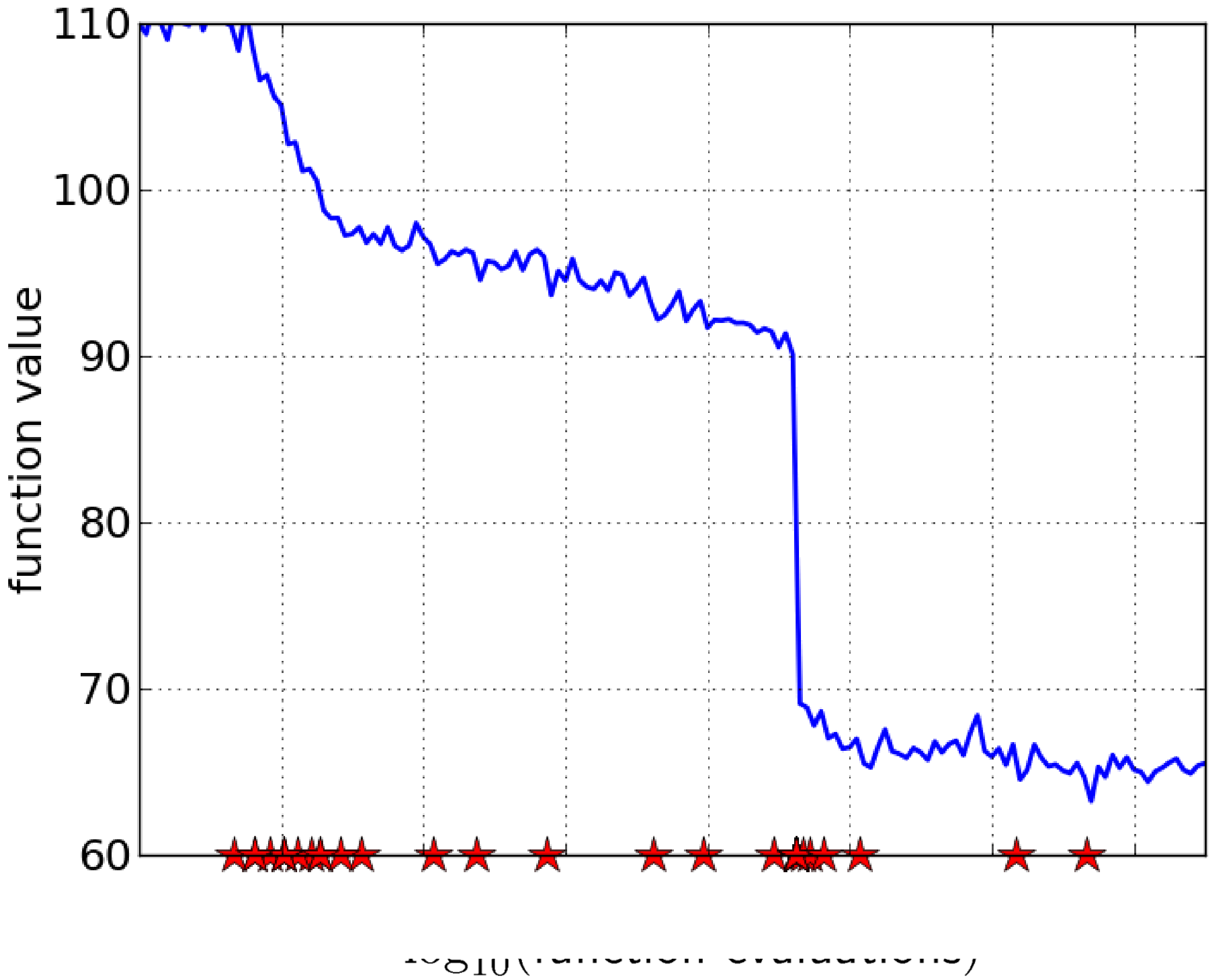
50 equally spaced targets

Reconstructing A Single Run

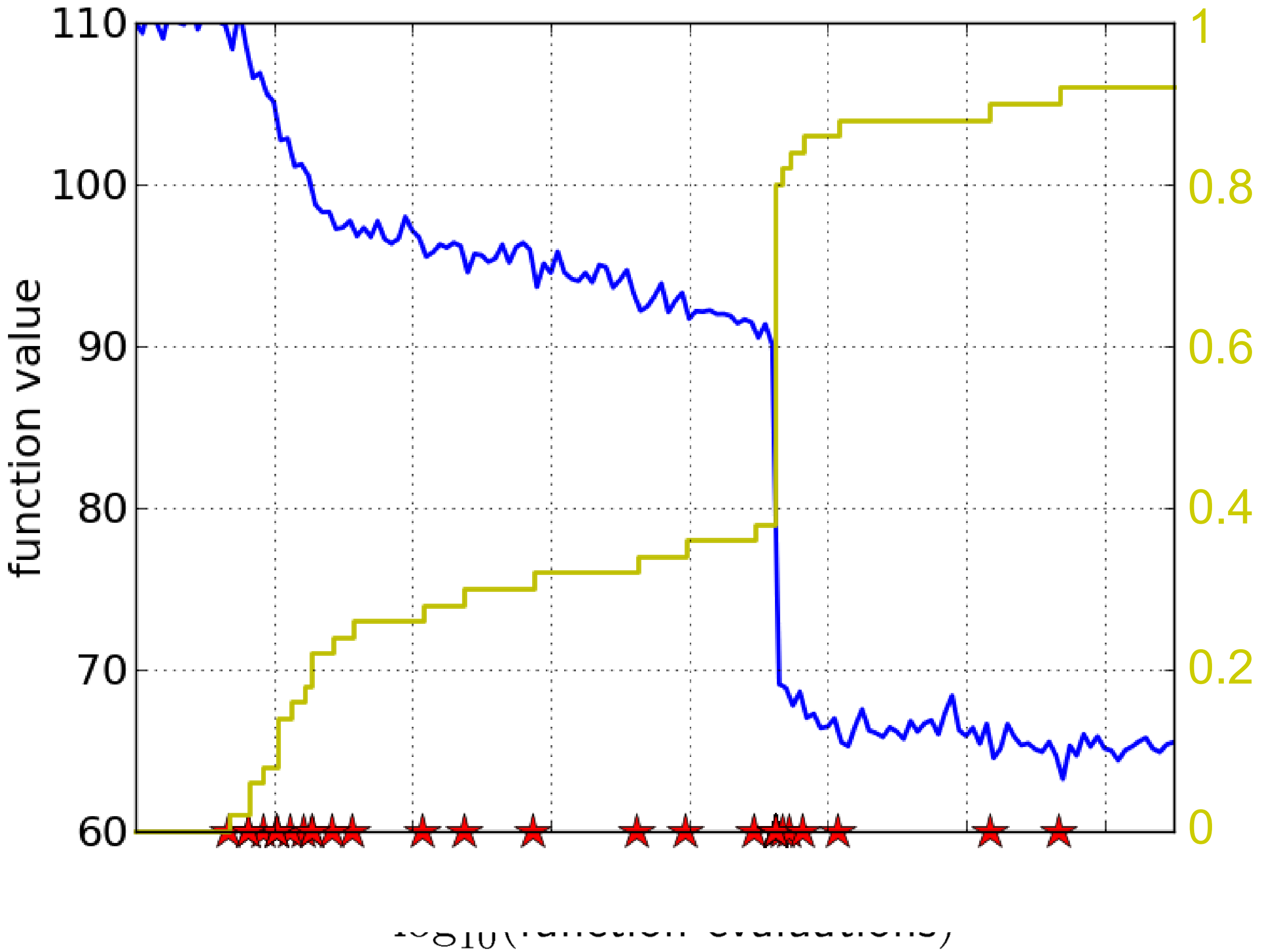


.

Reconstructing A Single Run

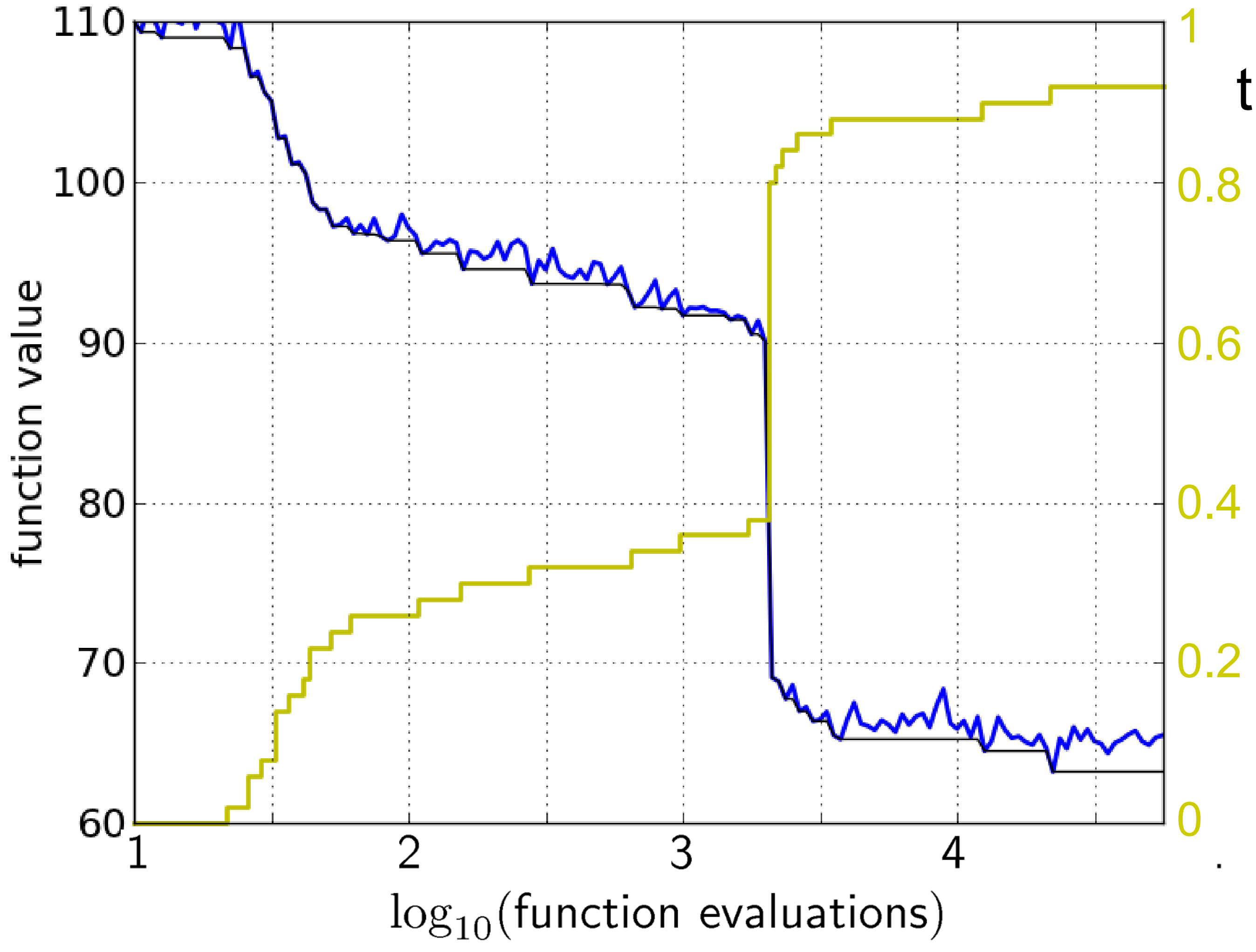


Reconstructing A Single Run



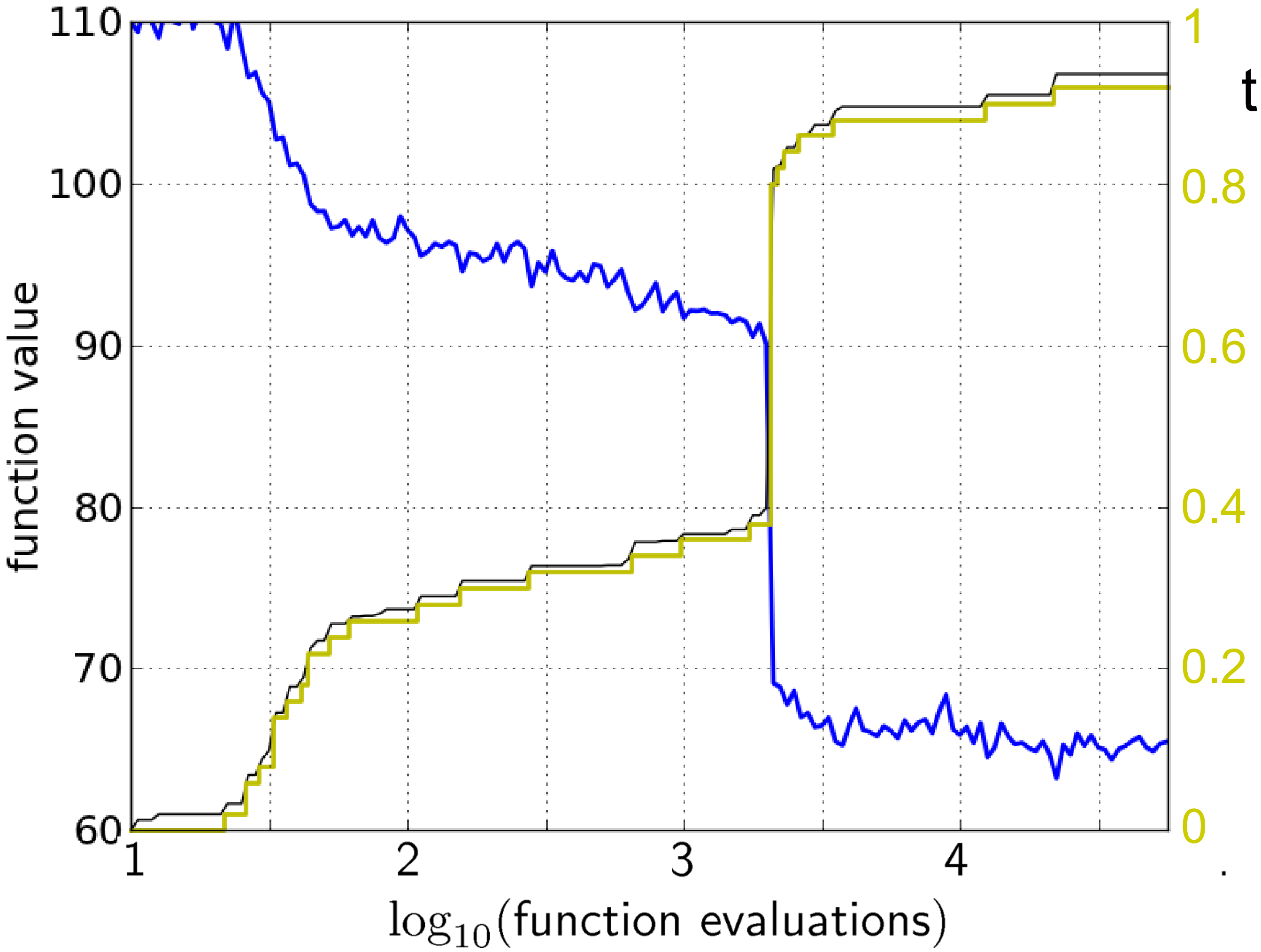
he empirical CDF makes a step for each star, is monotonous and displays for each budget the fraction of targets achieved within the budget

Reconstructing A Single Run



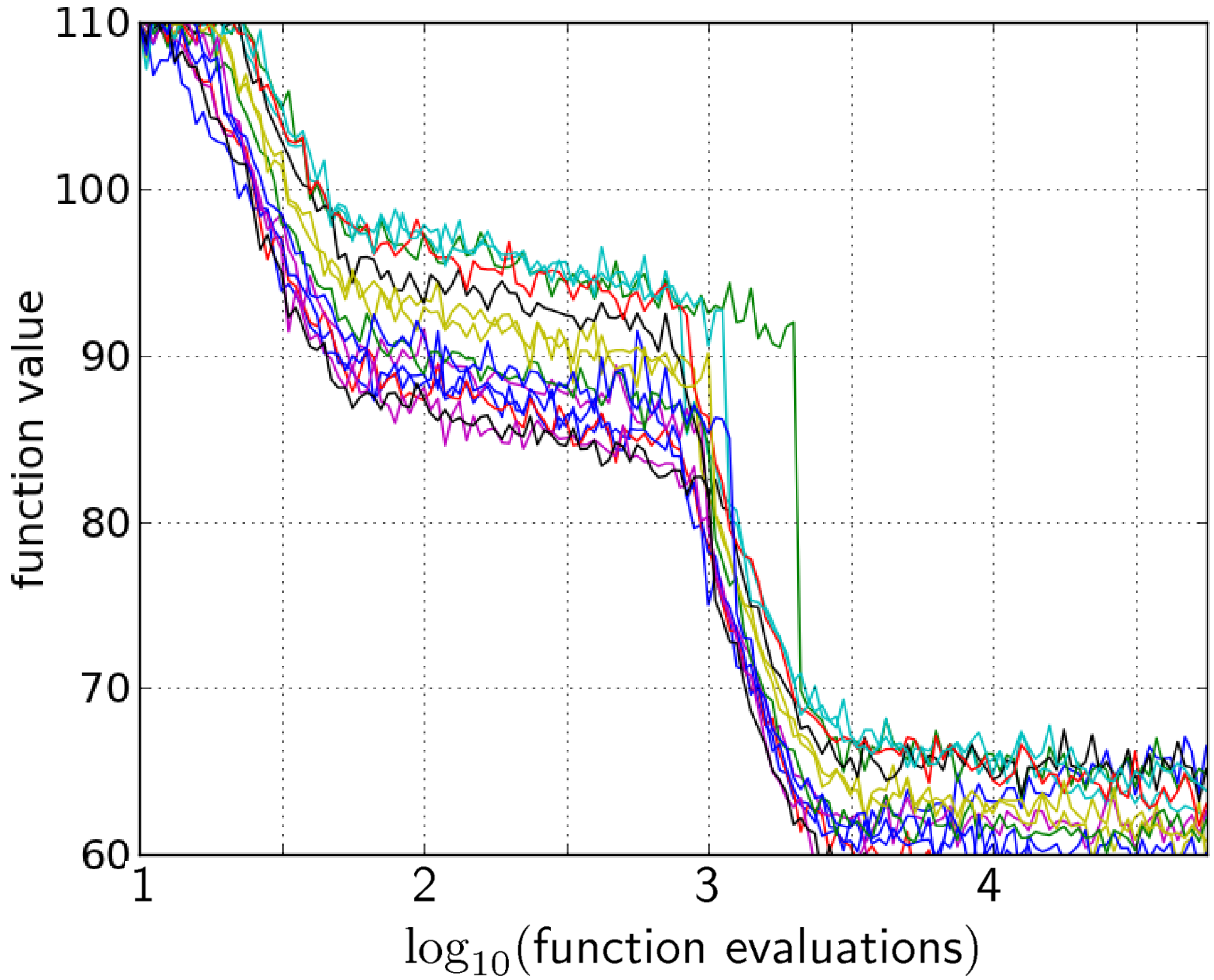
the ECDF recovers the monotonous graph, discretized and flipped

Reconstructing A Single Run



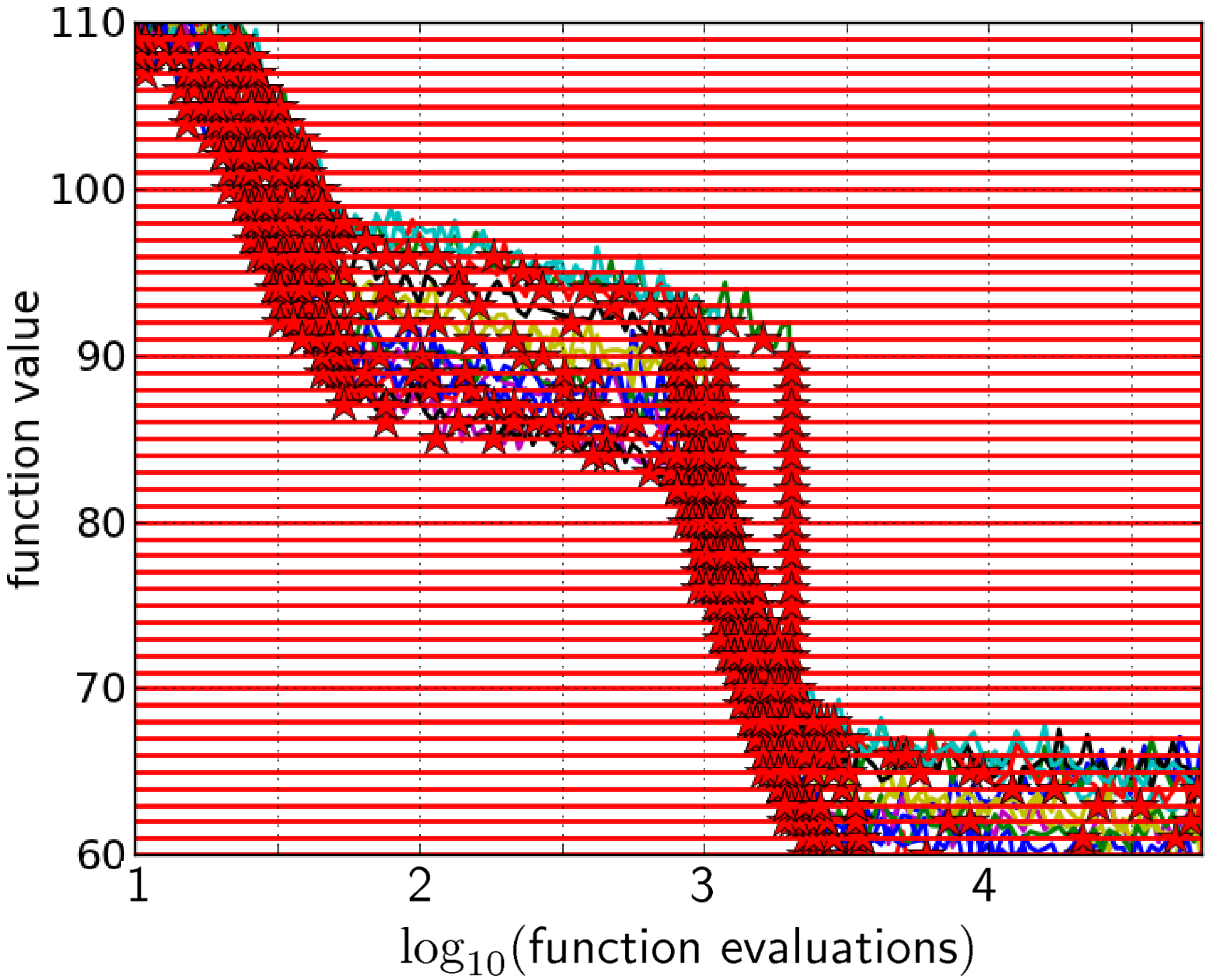
the ECDF recovers the monotonous graph, discretized and flipped

Aggregation



15 runs

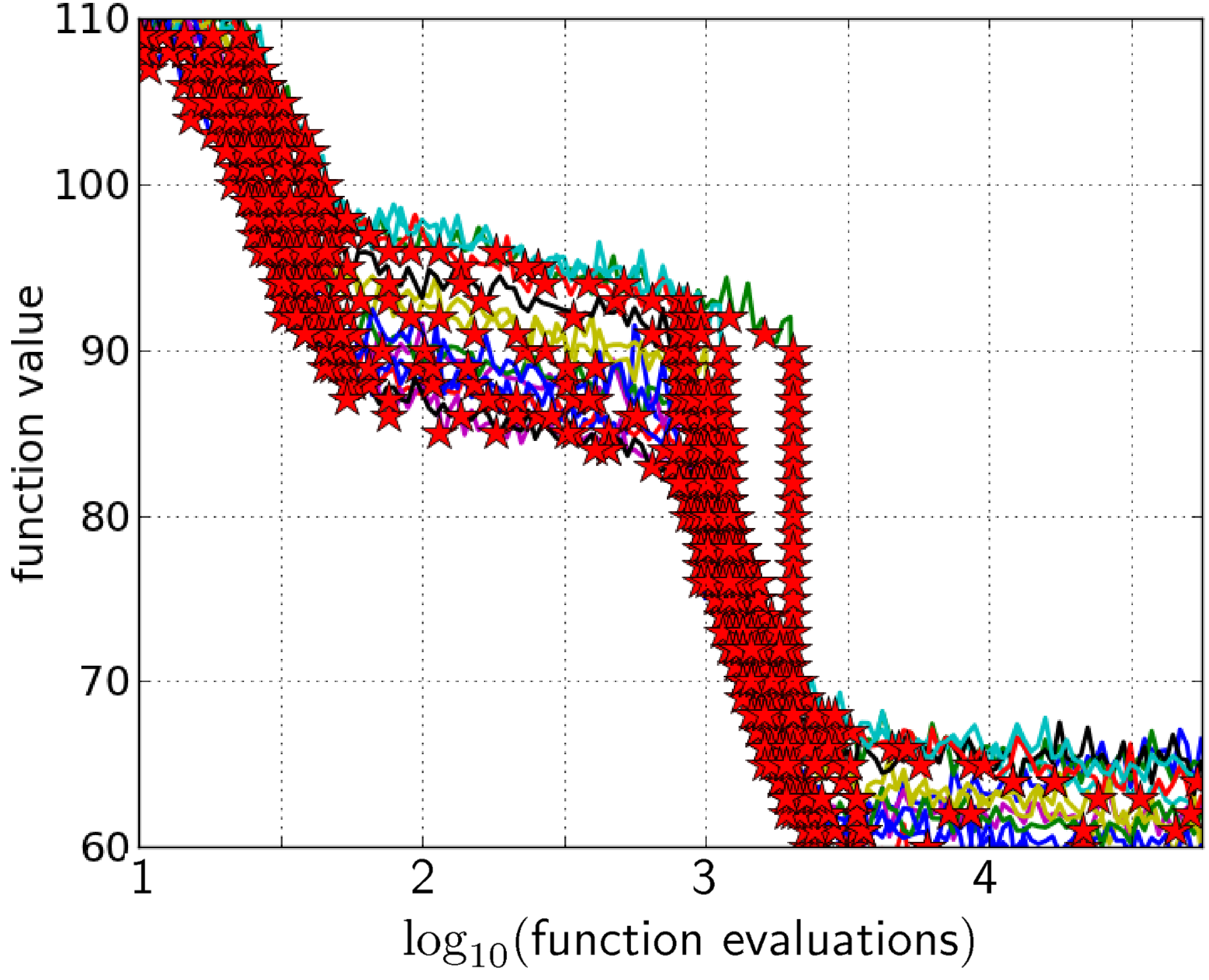
Aggregation



15 runs

50 targets

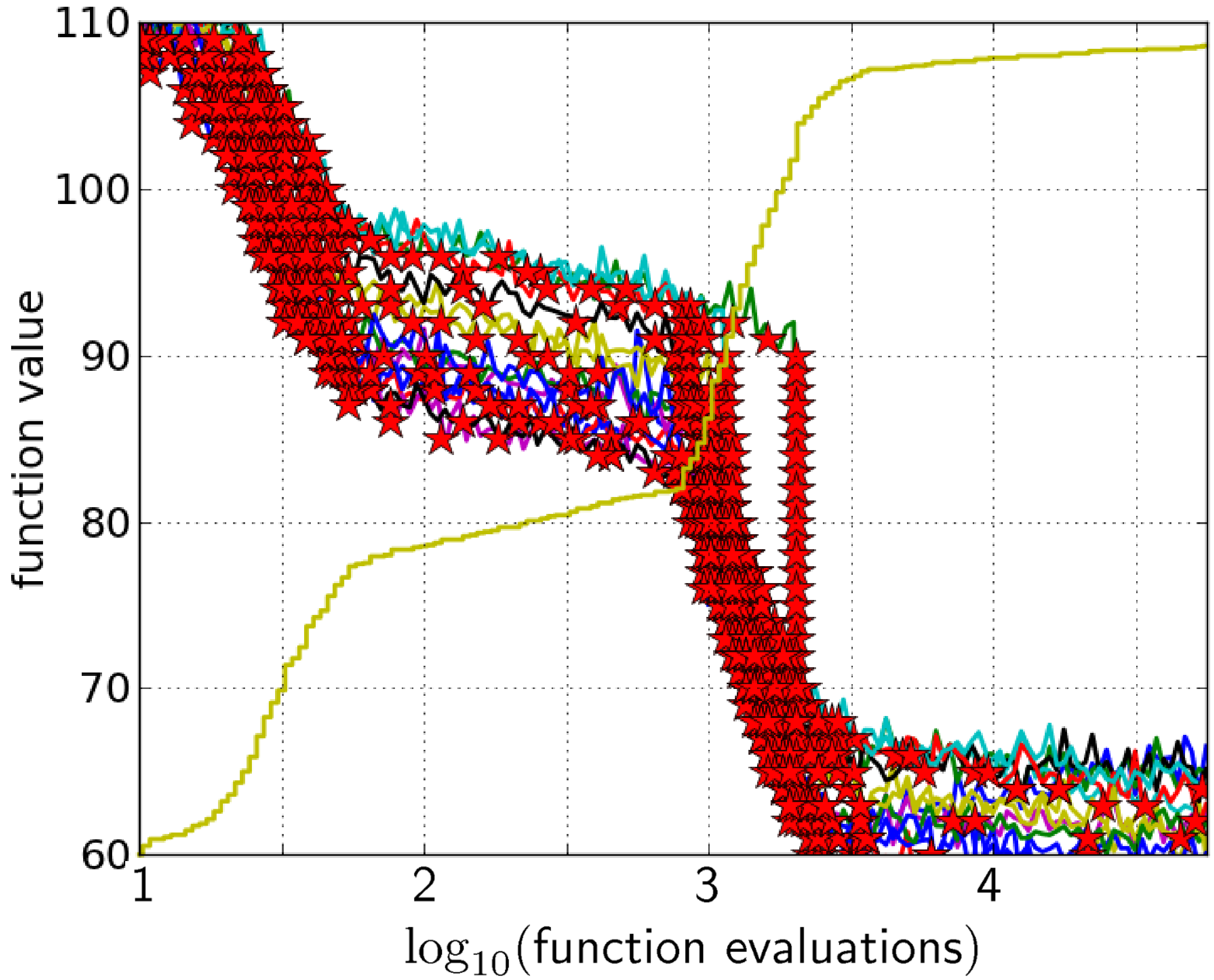
Aggregation



15 runs

50 targets

Aggregation

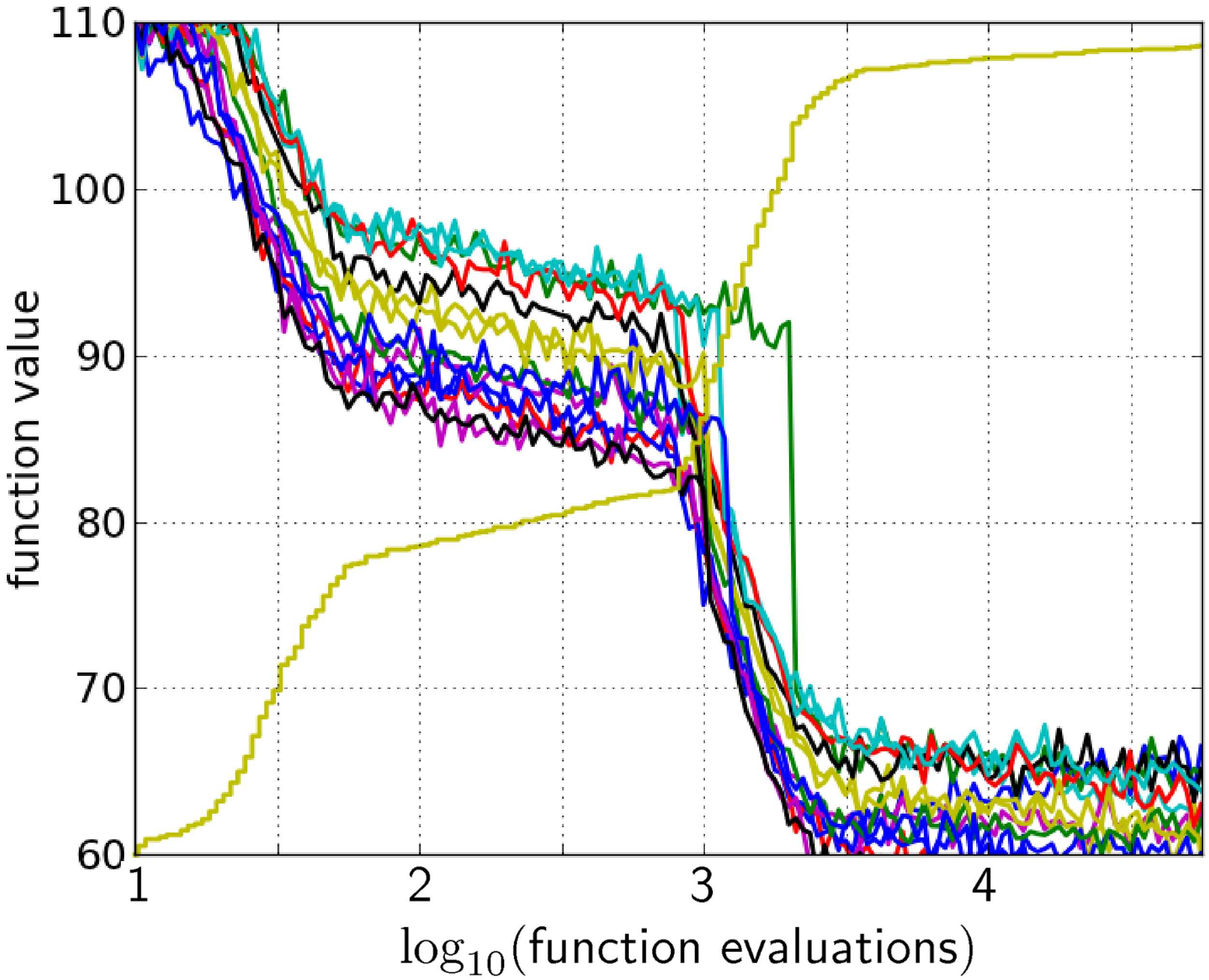


15 runs

50 targets

ECDF with
750 steps

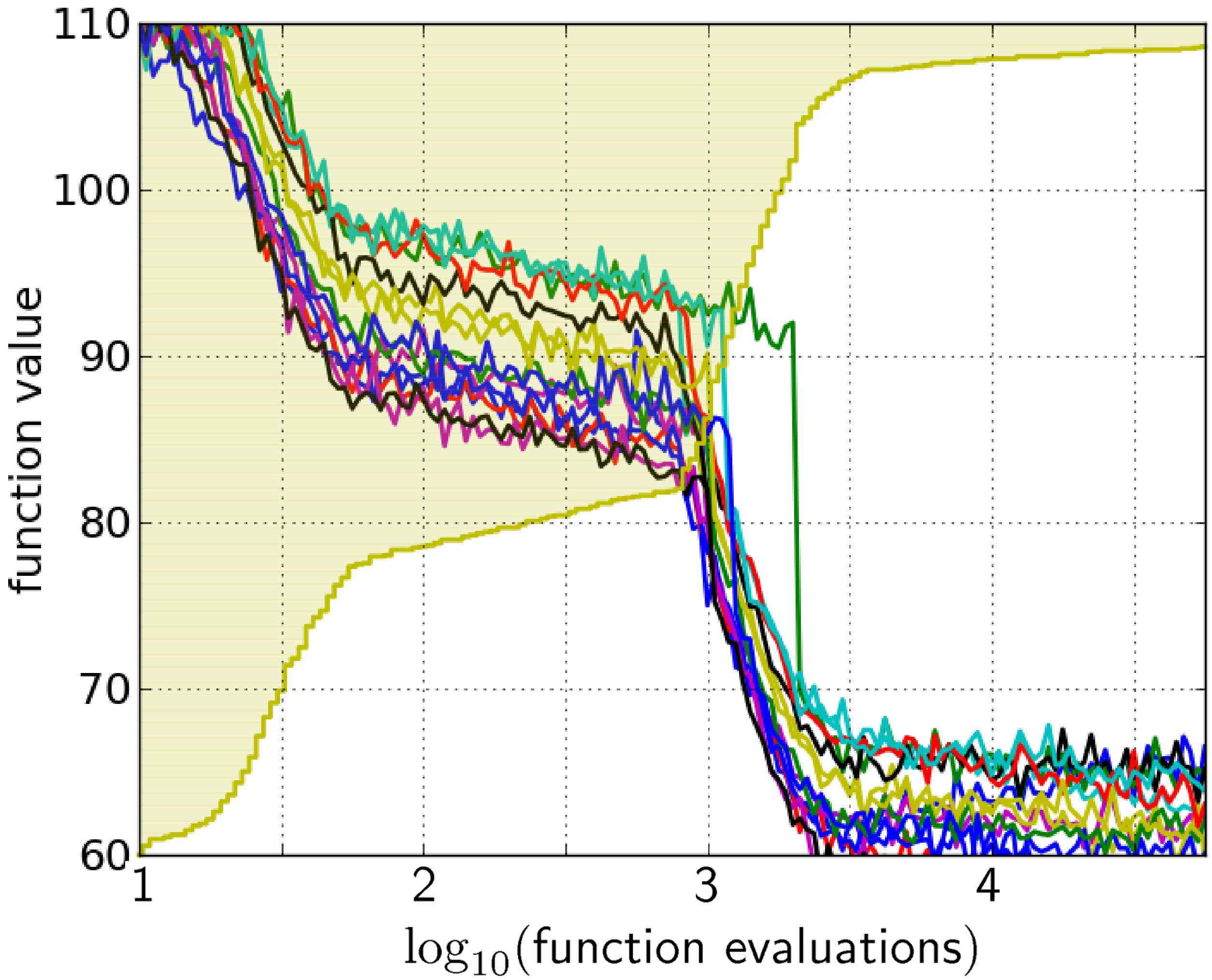
Aggregation



50 targets from 15 runs

...integrated in a single graph

Interpretation



area over the ECDF curve

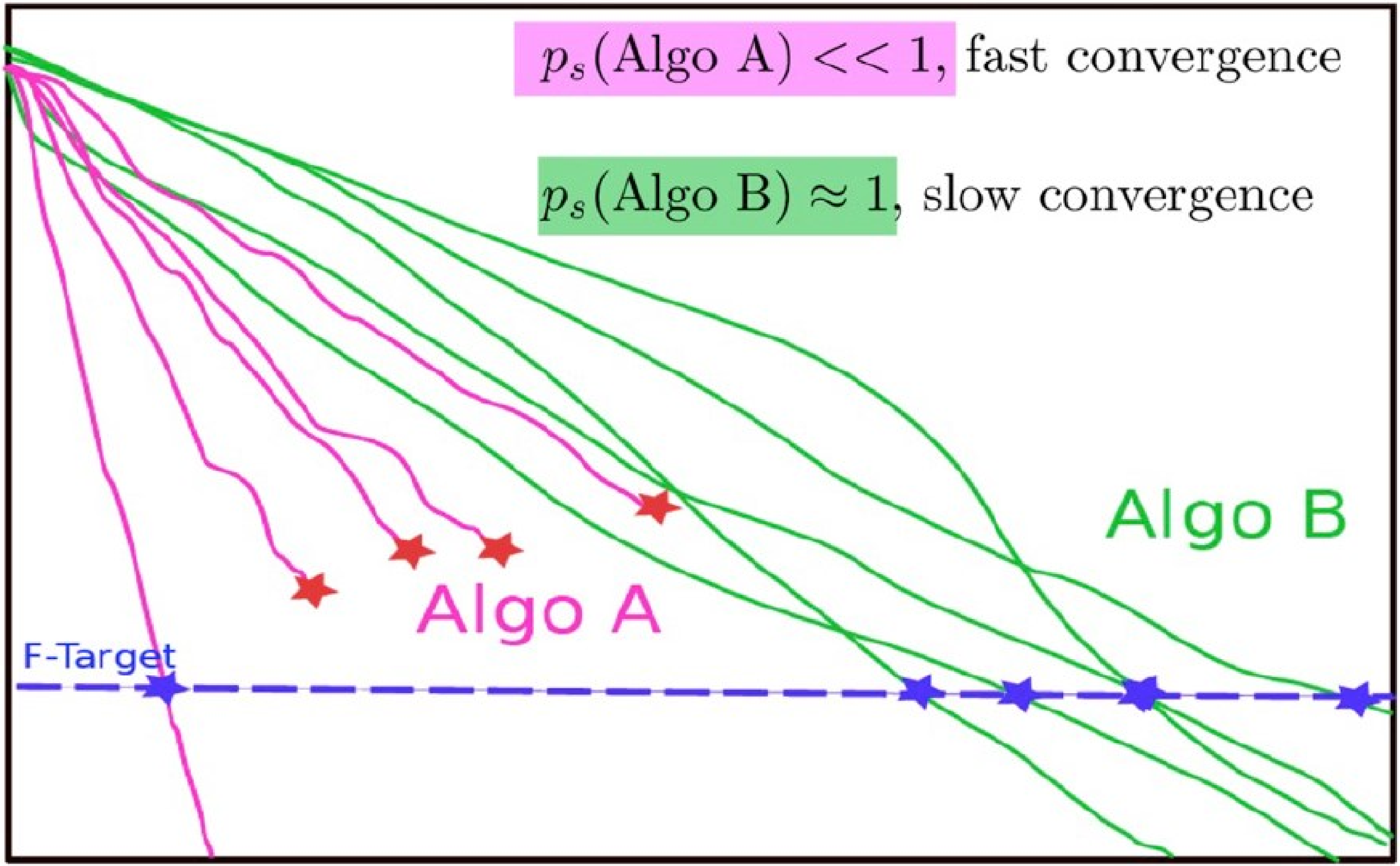
=

average log runtime

(or geometric avg. runtime) over all targets (difficult and easy) and all runs

.

Fixed-target: Measuring Runtime



Fixed-target: Measuring Runtime

Idea: Simulated restarts by bootstrapping from measured runtimes until we see a success

Algo Restart A:

RT_A^r

p_s(Algo Restart A) = 1

Algo Restart B:

RT_B^r

p_s(Algo Restart A) = 1

Fixed-target: Measuring Runtime

- Expected running time of the restarted algorithm:

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

- Estimator average running time (aRT/ERT/Enes/SP2*, w/o proof):

$$\widehat{p_s} = \frac{\#\text{successes}}{\#\text{runs}}$$

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

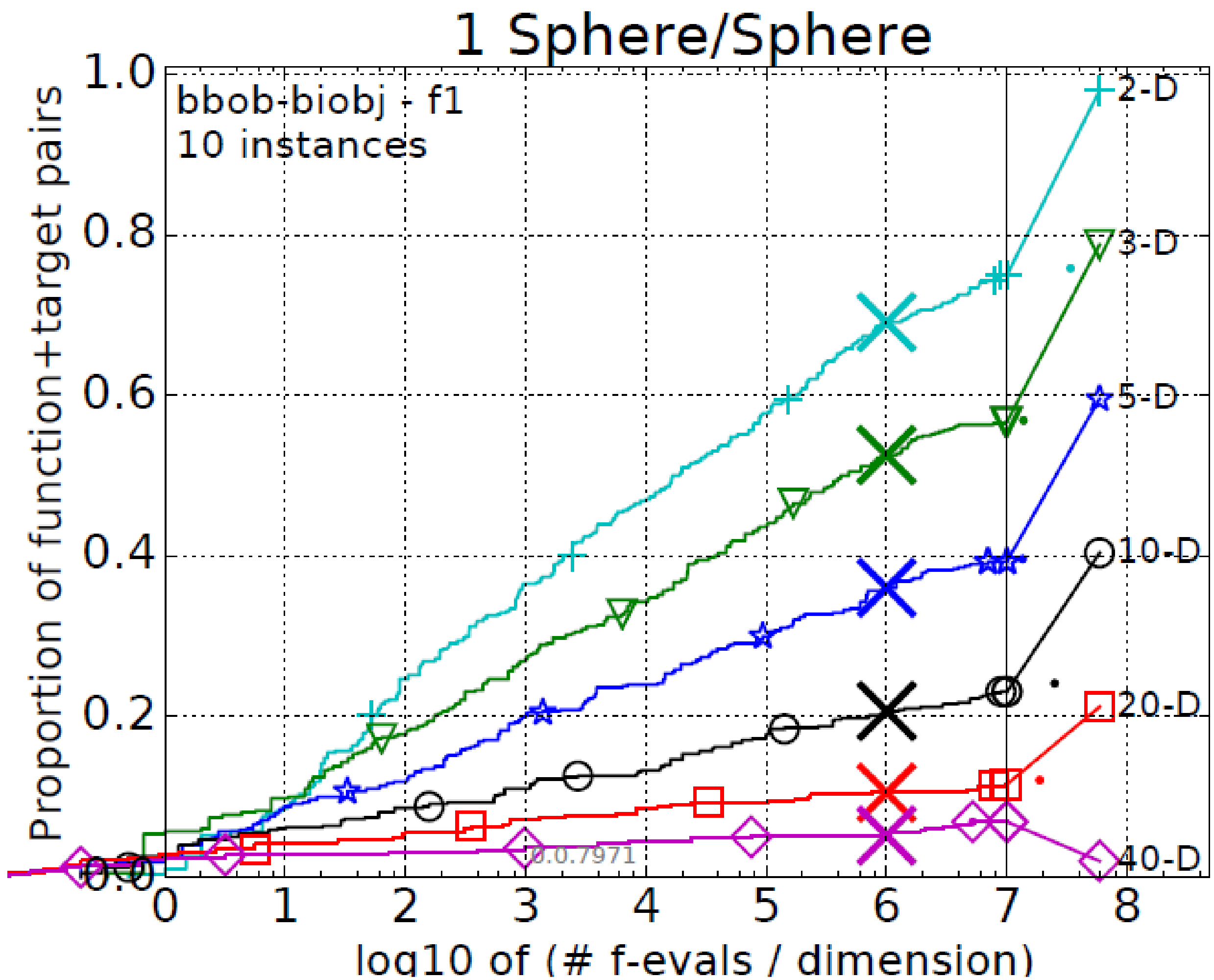
$$\widehat{RT_{succ}} = \text{Average evals of successful runs}$$

$$aRT = \frac{\text{total }\#\text{evals}}{\#\text{successes}}$$

* The concept is so essential that it has been discovered/proposed multiple times under different names in the past.

ECDFs with Simulated Restarts

What we typically plot are ECDFs of the simulated restarted algorithms:



Measuring Performance

On

- real world problems
  - expensive
  - comparison typically limited to certain domains
  - experts have limited interest to publish
- "artificial" benchmark functions
  - cheap
  - controlled
  - data acquisition is comparatively easy
  - problem of representativeness

Test Functions

- define the "scientific question"

the relevance can hardly be overestimated

- should represent "reality"
- are often too simple?

remind separability

- a number of testbeds are around

- account for invariance properties

prediction of performance is based on “similarity”,
ideally equivalence classes of functions

What to Benchmark?

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

- Taking all possible functions from a repository?

What to Benchmark?

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

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

What to Benchmark?

- test functions should be representative of difficulties we want to test
  therefore NFL has no relevance as assumption of being closed under permutation has no relevance wrt real world problems

- related to real-word difficulties
  for performance to be generalizable to RW

- scalable
  dimension plays a big role in performance
  curse of dimensionality

- comprehensible but not too easy
  BB optimization does not mean BB benchmarking

- we should still hide properties from the solver (hide optimum, …)
  solvers should not be able to exploit the benchmark intentionally or not

automated benchmarking: COCO

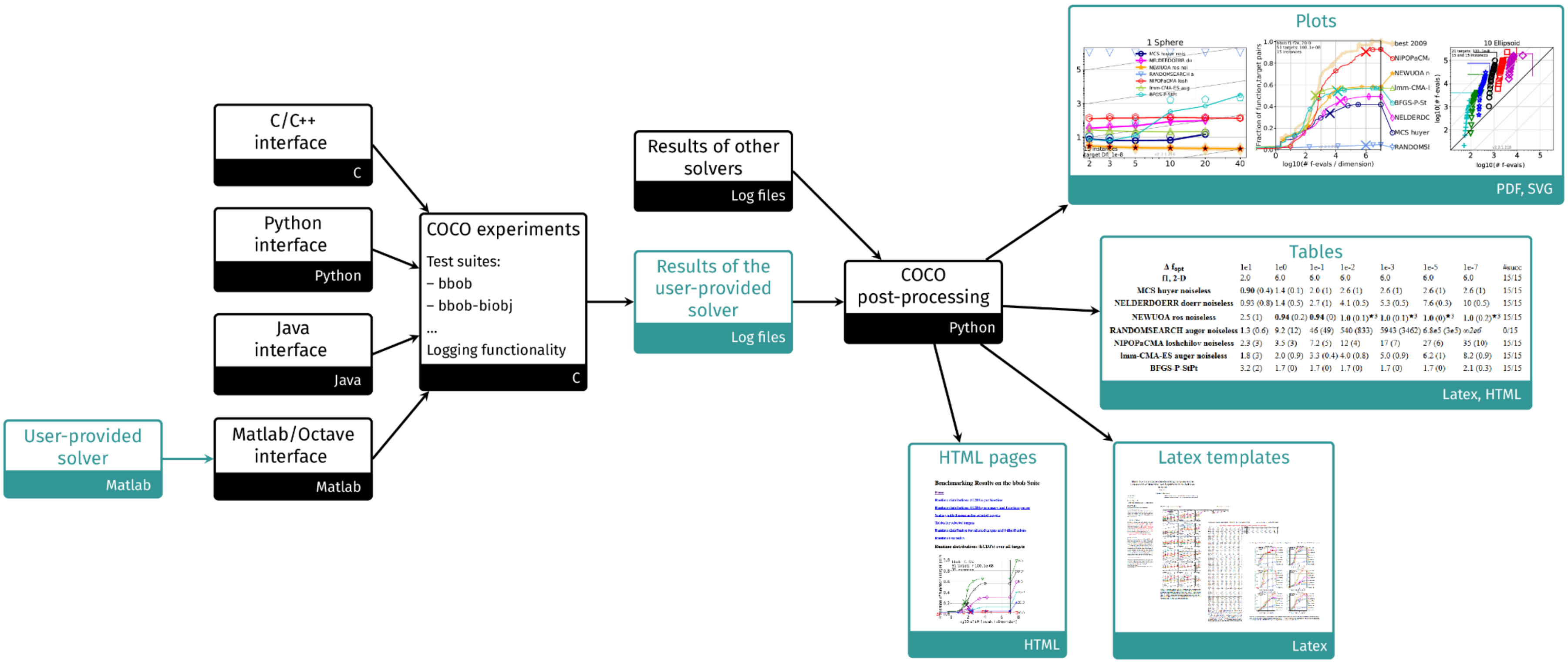
Comparing Continuous Optimizers Platform

`https://github.com/numbbo/coco`

COCO implements a reasonable, well-founded, and well-documented pre-chosen methodology

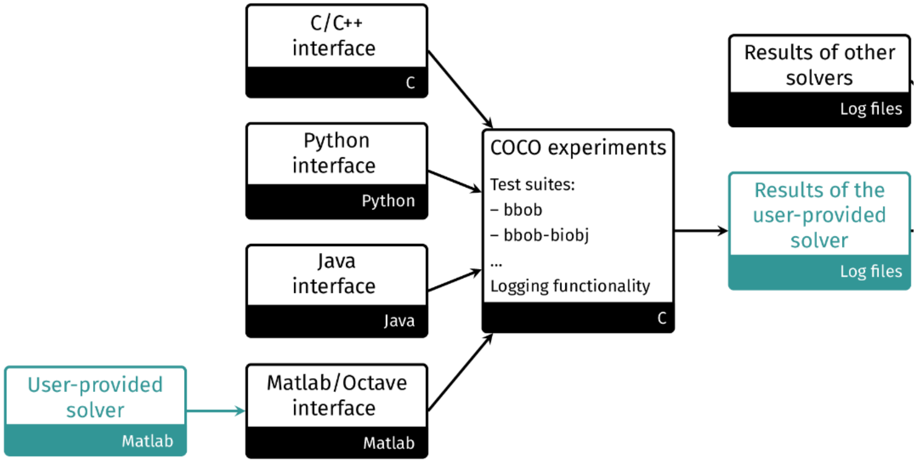
The Comparing Continuous Optimizers Platform

https://github.com/numbbo/coco



The Comparing Continuous Optimizers Platform

https://github.com/numbbo/coco

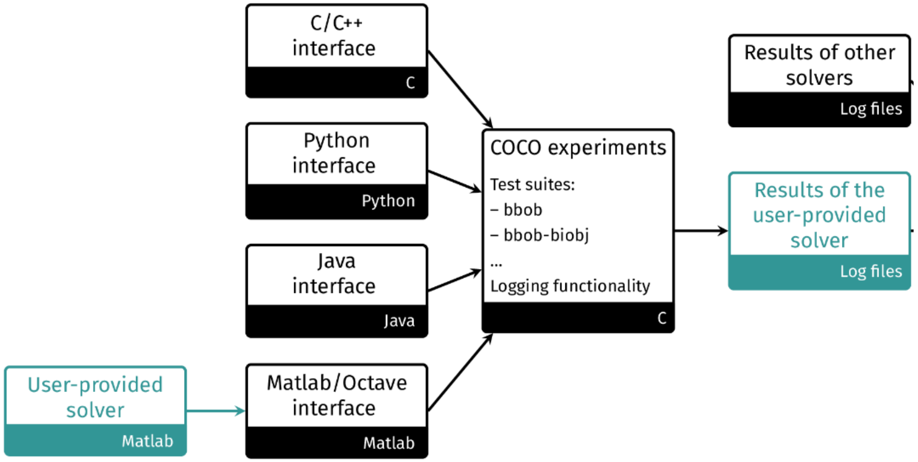


Experiments are as simple as (after installation)…

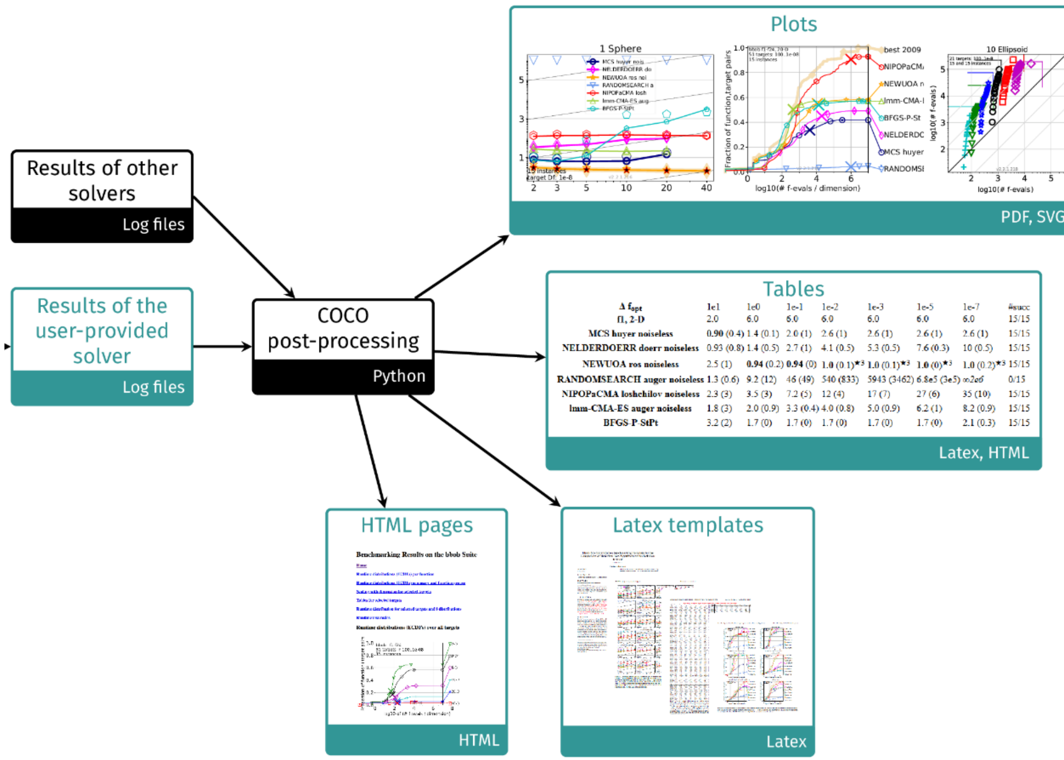
```
import cocoex
import scipy.optimize

### input
suite_name = "bbob"  # or "bbob-biobj" or ...
output_folder = "scipy-optimize-fmin"

### 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
    problem.observe_with(observer)  # generates the data for
                                    # cocopp post-processing
    scipy.optimize.fmin(problem, problem.initial_solution)
```

The Comparing Continuous Optimizers Platform

https://github.com/numbbo/coco

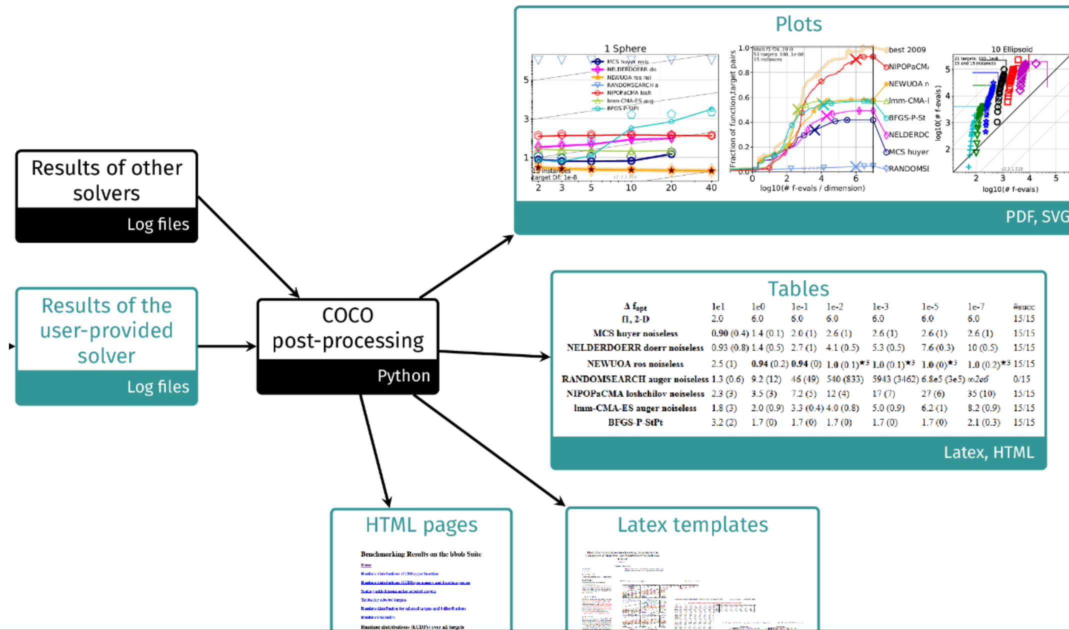


https://github.com/numbbo/coco



The Comparing Continuous Optimizers Platform

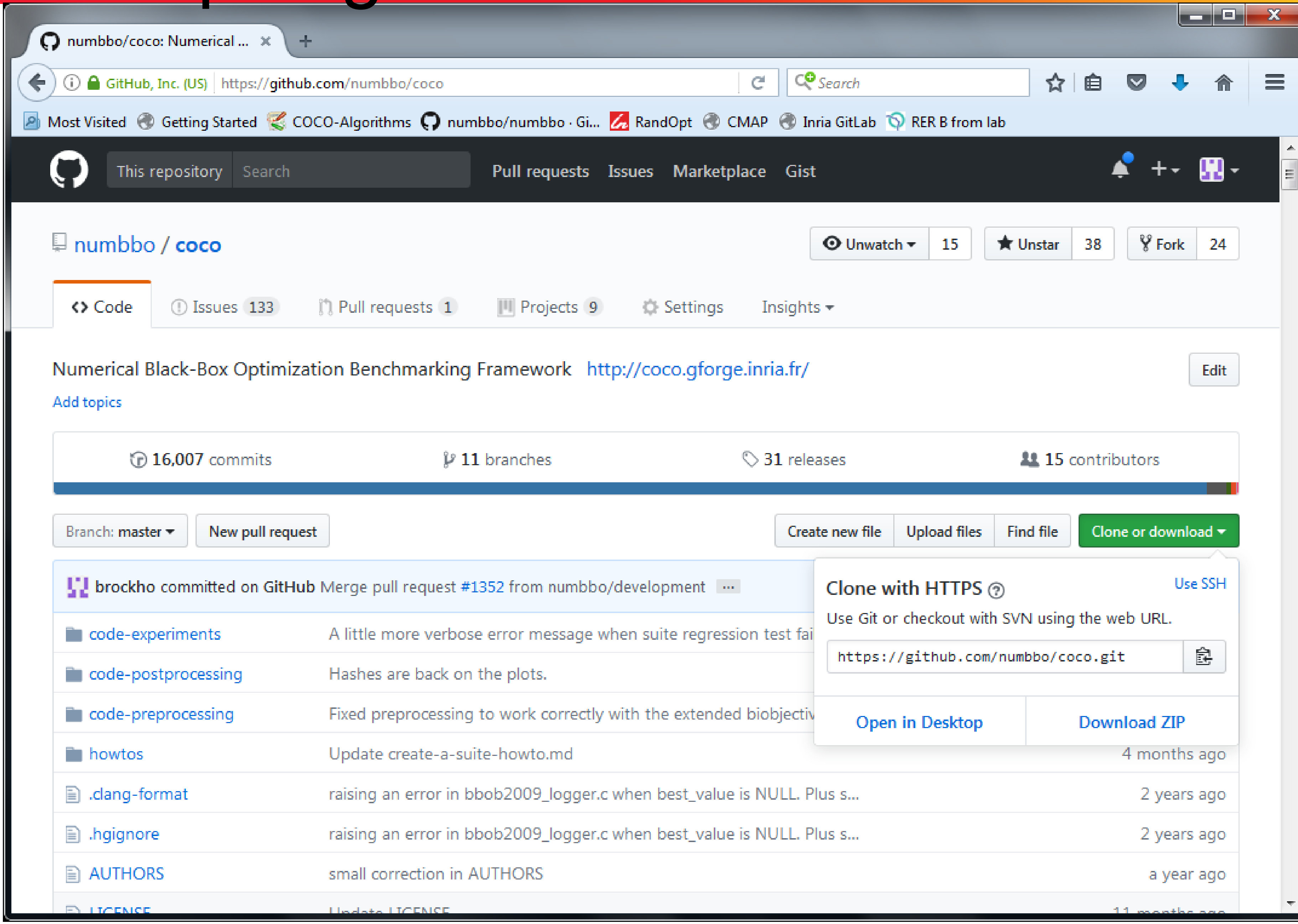
https://github.com/numbbo/coco



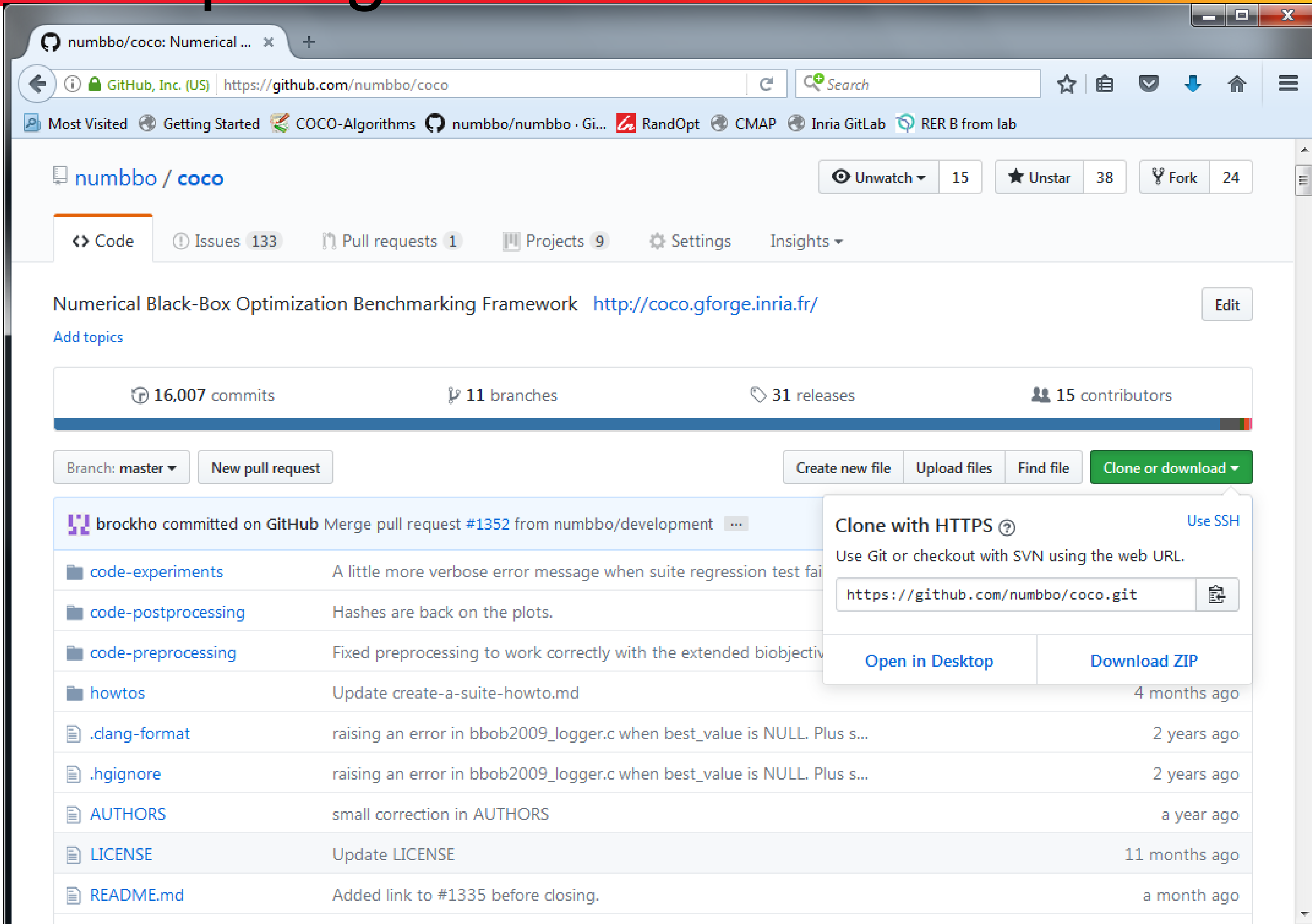
data from 300+ algorithms can be accessed directly through their name (see `https://numbbo.github.io/data-archive/`)

How to benchmark algorithms with COCO?

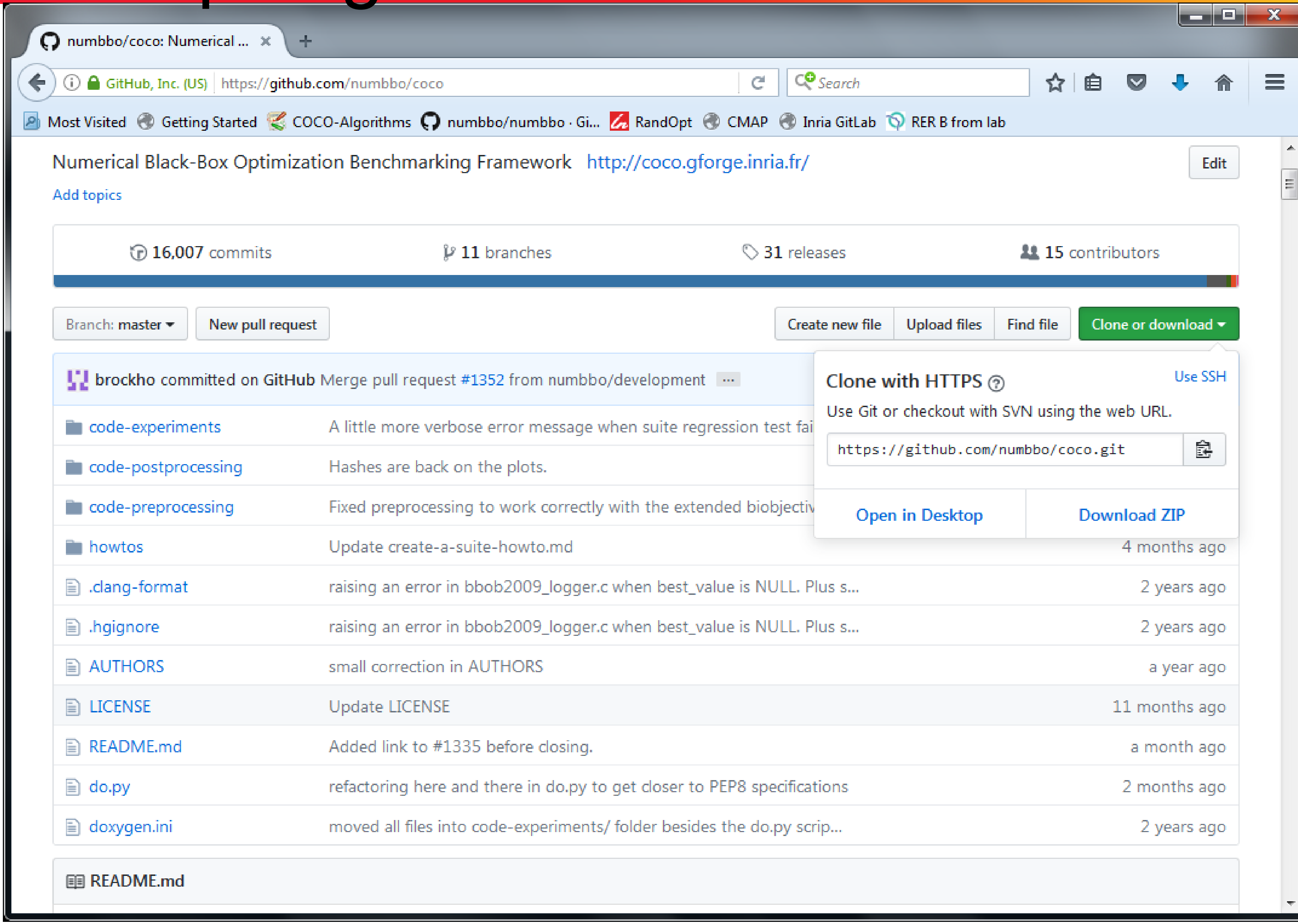
https://github.com/numbbo/coco



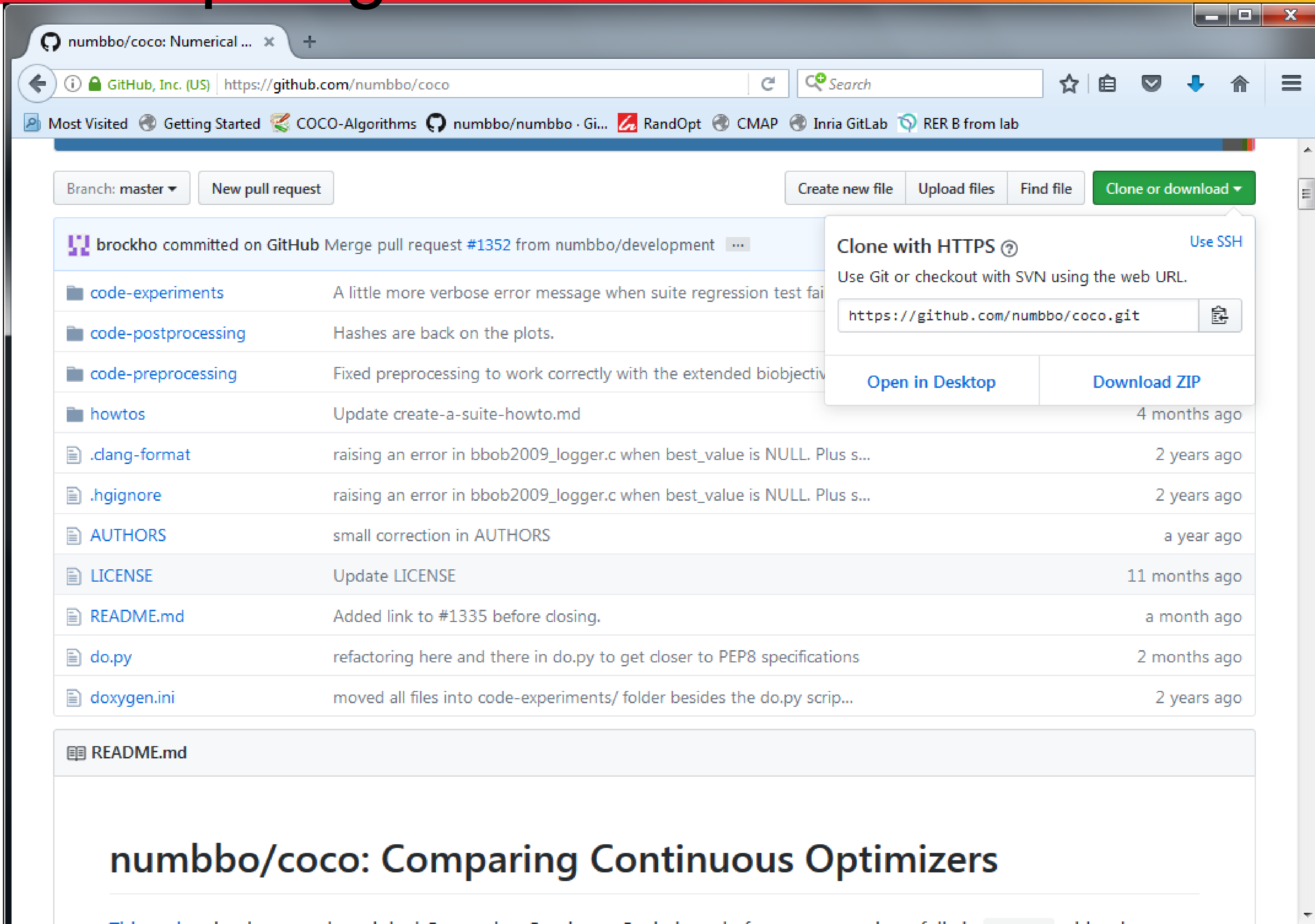
https://github.com/numbbo/coco



https://github.com/numbbo/coco



https://github.com/numbbo/coco



https://github.com/numbbo/coco

numbbo/coco: Numerical ...

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

Most Visited | Getting Started | COCO-Algorithms | numbbo/numbbo · Gi... | RandOpt | CMAP | Inria GitLab | RER B from lab

Open in Desktop | Download ZIP

code-preprocessing	Fixed preprocessing to work correctly with the extended biobjectiv	
howtos	Update create-a-suite-howto.md	4 months ago
.clang-format	raising an error in bbob2009_logger.c when best_value is NULL. Plus s...	2 years ago
.hgignore	raising an error in bbob2009_logger.c when best_value is NULL. Plus s...	2 years ago
AUTHORS	small correction in AUTHORS	a year ago
LICENSE	Update LICENSE	11 months ago
README.md	Added link to #1335 before closing.	a month ago
do.py	refactoring here and there in do.py to get closer to PEP8 specifications	2 months ago
doxygen.ini	moved all files into code-experiments/ folder besides the do.py scrip...	2 years ago

README.md

numbbo/coco: Comparing Continuous Optimizers

This code reimplements the original Comparing Continous Optimizer platform, now rewritten fully in `ANSI C` with other languages calling the `C` code. As the name suggests, the code provides a platform to benchmark and compare continuous optimizers, AKA non-linear solvers for numerical optimization. Languages currently available are

- `C/C++`
- `Java`
- `MATLAB/Octave`

https://github.com/numbbo/coco

numbbo/coco: Numerical ...

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

Search

Most Visited · Getting Started · COCO-Algorithms · numbbo/numbbo · Gi... · RandOpt · CMAP · Inria GitLab · RER B from lab

LICENSE	Update LICENSE	11 months ago
README.md	Added link to #1335 before closing.	a month ago
do.py	refactoring here and there in do.py to get closer to PEP8 specifications	2 months ago
doxygen.ini	moved all files into code-experiments/ folder besides the do.py scrip...	2 years ago

README.md

numbbo/coco: Comparing Continuous Optimizers

This code reimplements the original Comparing Continous Optimizer platform, now rewritten fully in `ANSI C` with other languages calling the `C` code. As the name suggests, the code provides a platform to benchmark and compare continuous optimizers, AKA non-linear solvers for numerical optimization. Languages currently available are

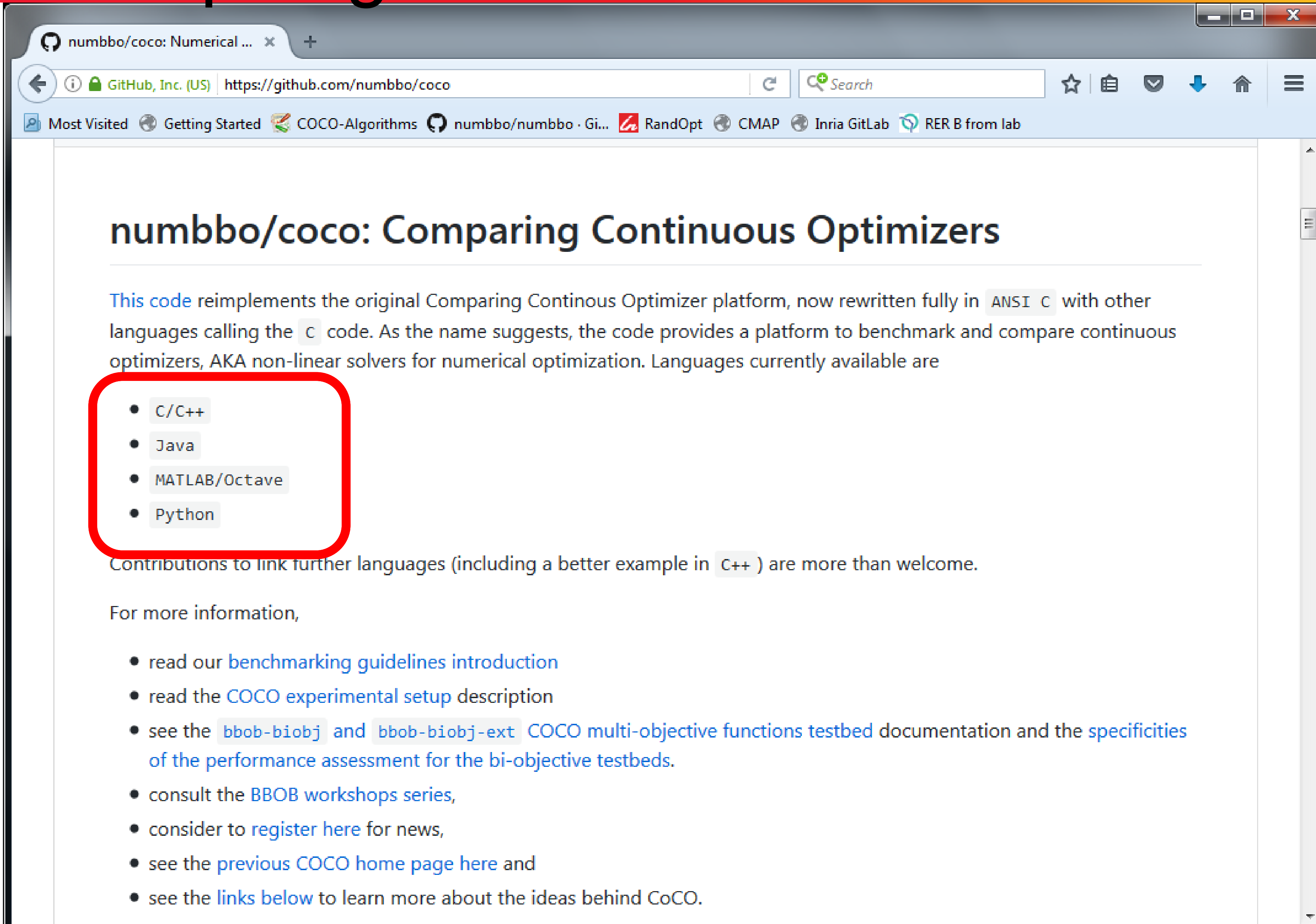
- `C/C++`
- `Java`
- `MATLAB/Octave`
- `Python`

Contributions to link further languages (including a better example in `C++`) are more than welcome.

For more information,

- read our benchmarking guidelines introduction
- read the COCO experimental setup description

https://github.com/numbbo/coco



numbbo/coco: Numerical ...

https://github.com/numbbo/coco

Most Visited | Getting Started | COCO-Algorithms | numbbo/numbbo · Gi... | RandOpt | CMAP | Inria GitLab | RER B from lab

numbbo/coco: Comparing Continuous Optimizers

This code reimplements the original Comparing Continous Optimizer platform, now rewritten fully in `ANSI C` with other languages calling the `C` code. As the name suggests, the code provides a platform to benchmark and compare continuous optimizers, AKA non-linear solvers for numerical optimization. Languages currently available are

- `C/C++`
- `Java`
- `MATLAB/Octave`
- `Python`

Contributions to link further languages (including a better example in `C++`) are more than welcome.

For more information,

- read our benchmarking guidelines introduction
- read the COCO experimental setup description
- see the `bbob-biobj` and `bbob-biobj-ext` COCO multi-objective functions testbed documentation and the specificities of the performance assessment for the bi-objective testbeds.
- consult the BBOB workshops series,
- consider to register here for news,
- see the previous COCO home page here and
- see the links below to learn more about the ideas behind CoCO.

https://github.com/numbbo/coco

installation I: experiments

As easy (in python) as:

```
pip install cocoex
```

installation II: postprocessing

```
pip install cocopp
```

for other languages, you can use the development branch

https://github.com/numbbo/coco

numbbo/coco: Numerical ...

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

Most Visited · Getting Started · COCO-Algorithms · numbbo/numbbo · Gi... · RandOpt · CMAP · Inria GitLab · RER B from lab

example experiment once. The build result and the example experiment code can be found under code-experiments/build/<language> (<language>=matlab for Octave). python do.py lists all available commands.

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

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

to (user-locally) install the post-processing. From here on, do.py has done its job and is only needed again for updating the builds to a new release.

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

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

coupling algo + COCO

If the example experiment runs, **connect** your favorite algorithm to Coco: replace the call to the random search optimizer in the example experiment file by a call to your algorithm (see above). **Update** the output result_folder, the algorithm_name and algorithm_info of the observer options in the example experiment file.

Another entry point for your own experiments can be the code-experiments/examples folder.

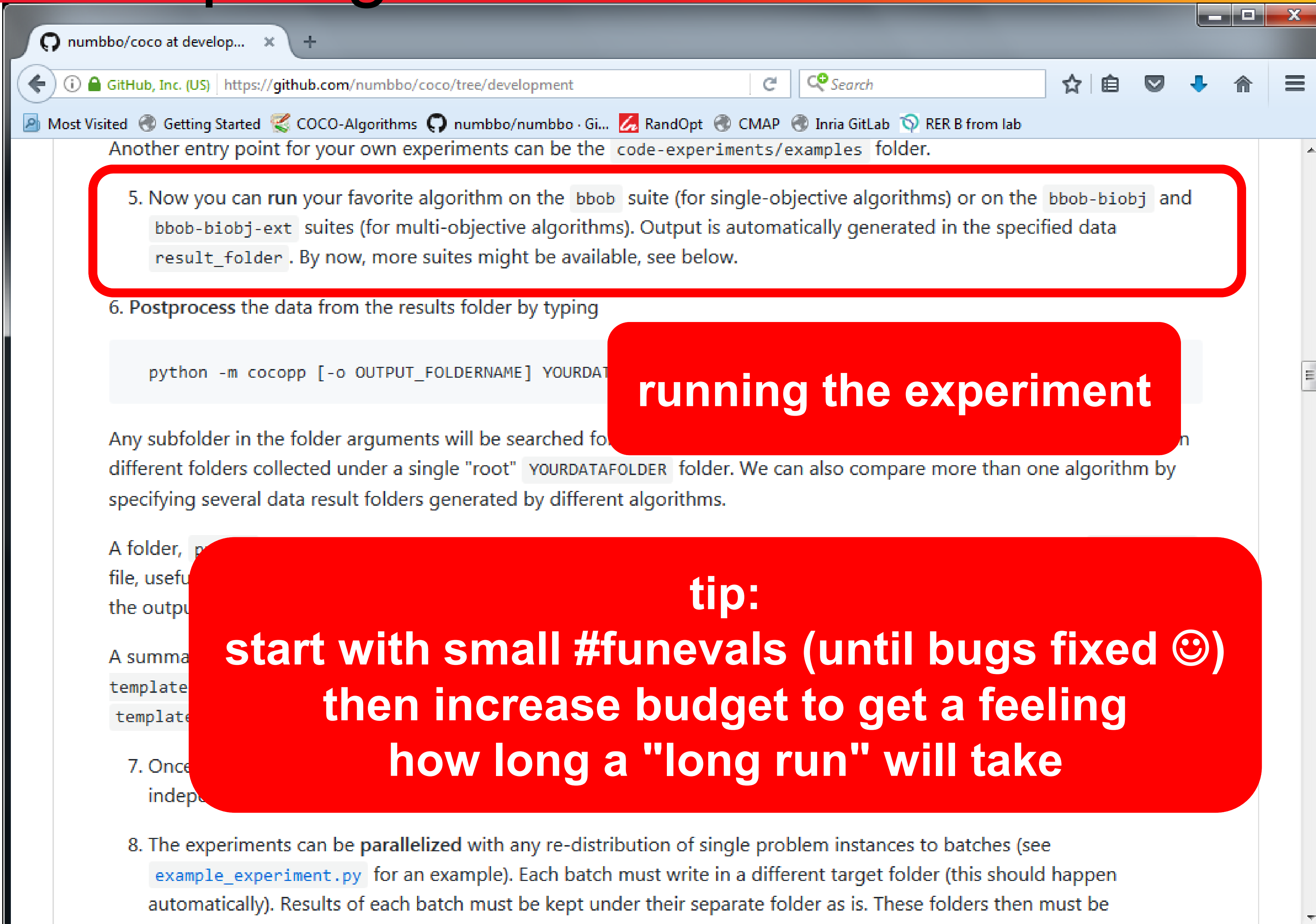
5. Now you can **run** your favorite algorithm on the bbob suite (for single-objective algorithms) or on the bbob-biobj and bbob-biobj-ext suites (for multi-objective algorithms). Output is automatically generated in the specified data result_folder. By now, more suites might be available, see below.

Simplified Example Experiment in Python

```
import cocoex
import scipy.optimize

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

### 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
    problem.observe_with(observer)  # generates the data for
                                    # cocopp post-processing
    fmin(problem, problem.initial_solution)
```

Note: the actual example_experiment.py contains more advanced things like restarts, batch experiments, other algorithms (e.g. CMA-ES), etc.

https://github.com/numbbo/coco



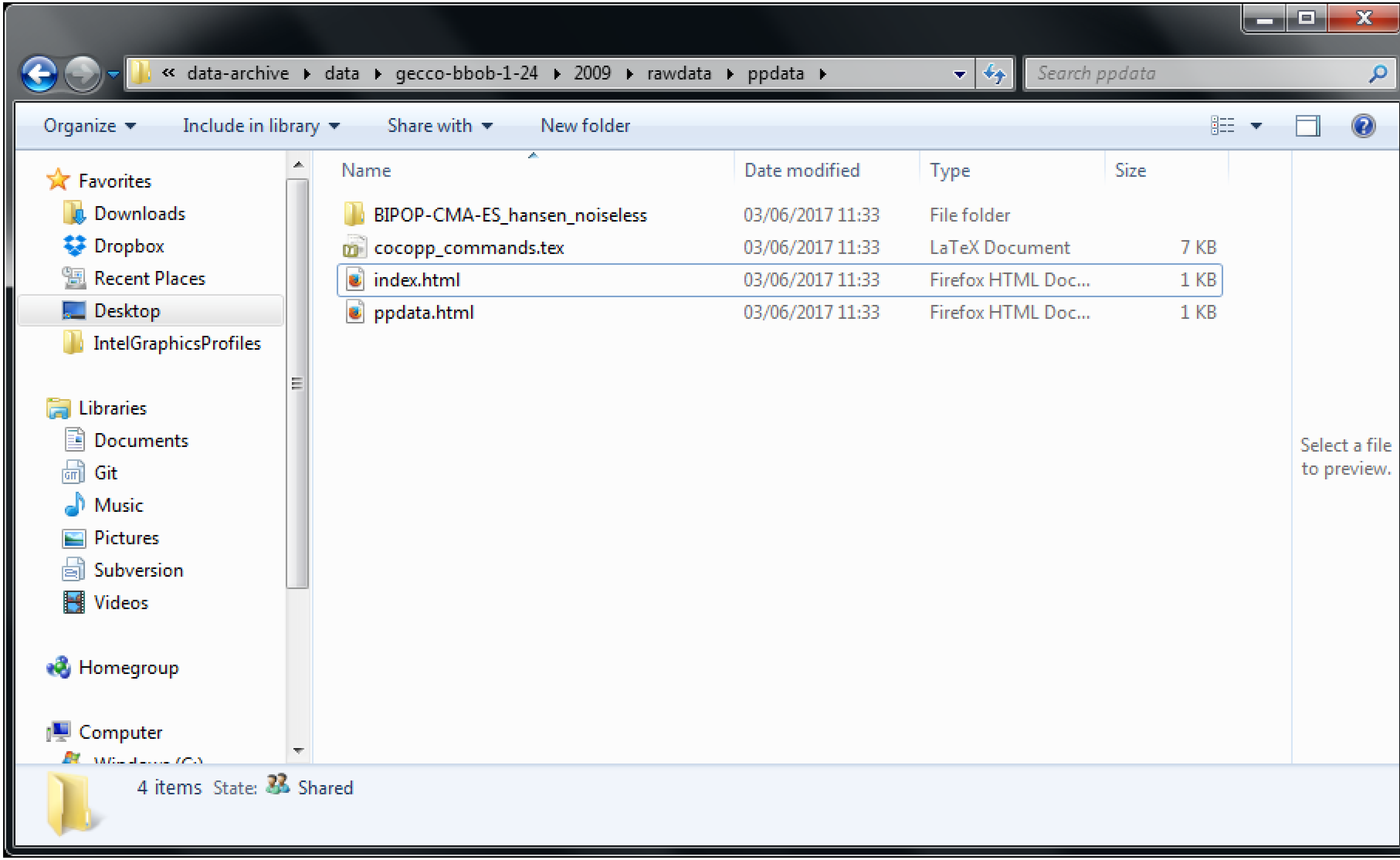
running the experiment

tip:
start with small #funevals (until bugs fixed ☺)
then increase budget to get a feeling
how long a "long run" will take

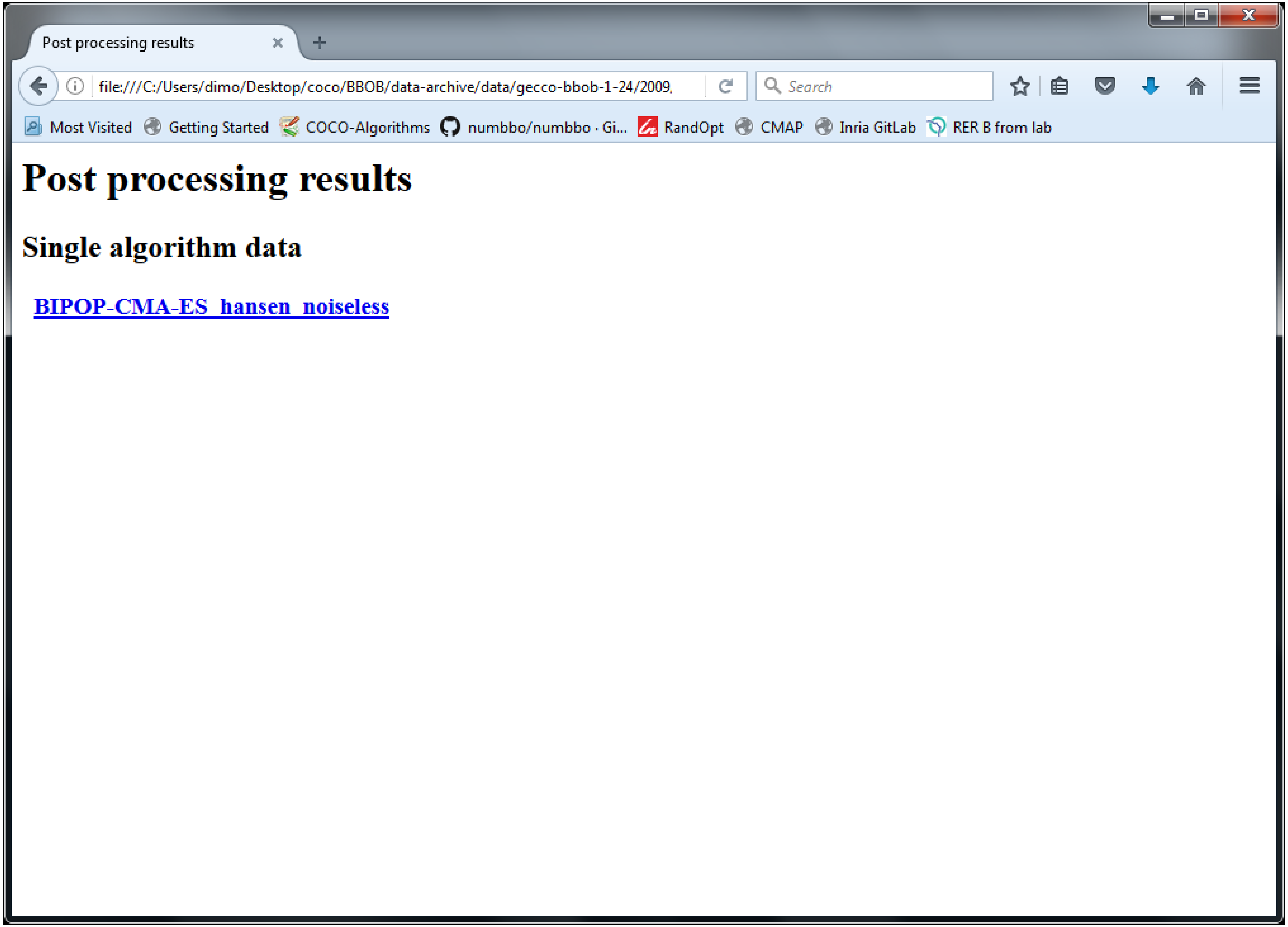
https://github.com/numbbo/coco

postprocessing

data from 300+ algorithms can be accessed directly through its name (see `https://numbbo.github.io/data-archive/`)


« data-archive ▸ data ▸ gecco-bbob-1-24 ▸ 2009 ▸ rawdata ▸ ppdata ▸
Search ppdata
Organize
Include in library
Share with
New folder
Favorites
Downloads
Dropbox
Recent Places
Desktop
IntelGraphicsProfiles
Libraries
Documents
Git
Music
Pictures
Subversion
Videos
Homegroup
Computer
Name
Date modified
Type
Size
BIPOP-CMA-ES_hansen_noiseless
03/06/2017 11:33
File folder
cocopp_commands.tex
03/06/2017 11:33
LaTeX Document
7 KB
index.html
03/06/2017 11:33
Firefox HTML Doc...
1 KB
ppdata.html
03/06/2017 11:33
Firefox HTML Doc...
1 KB
Select a file to preview.
4 items State: Shared

Automatically Generated Results



Automatically Generated Results

BIPOP-CMA-ES, templateBBOB...

file:///C:/Users/dimo/Desktop/coco/BBOB/data-archive/data/gecco-bbob-1-24/2009,

Most Visited · Getting Started · COCO-Algorithms · numbbo/numbbo · Gi... · RandOpt · CMAP · Inria GitLab · RER B from lab

BIPOP-CMA-ES

[Home]

[Runtime distributions (ECDFs) per function]

[Runtime distributions (ECDFs) summary and function groups]

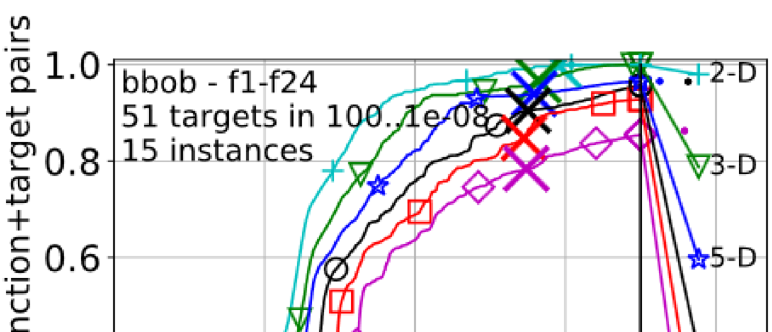
[Scaling with dimension for selected targets]

[Tables for selected targets]

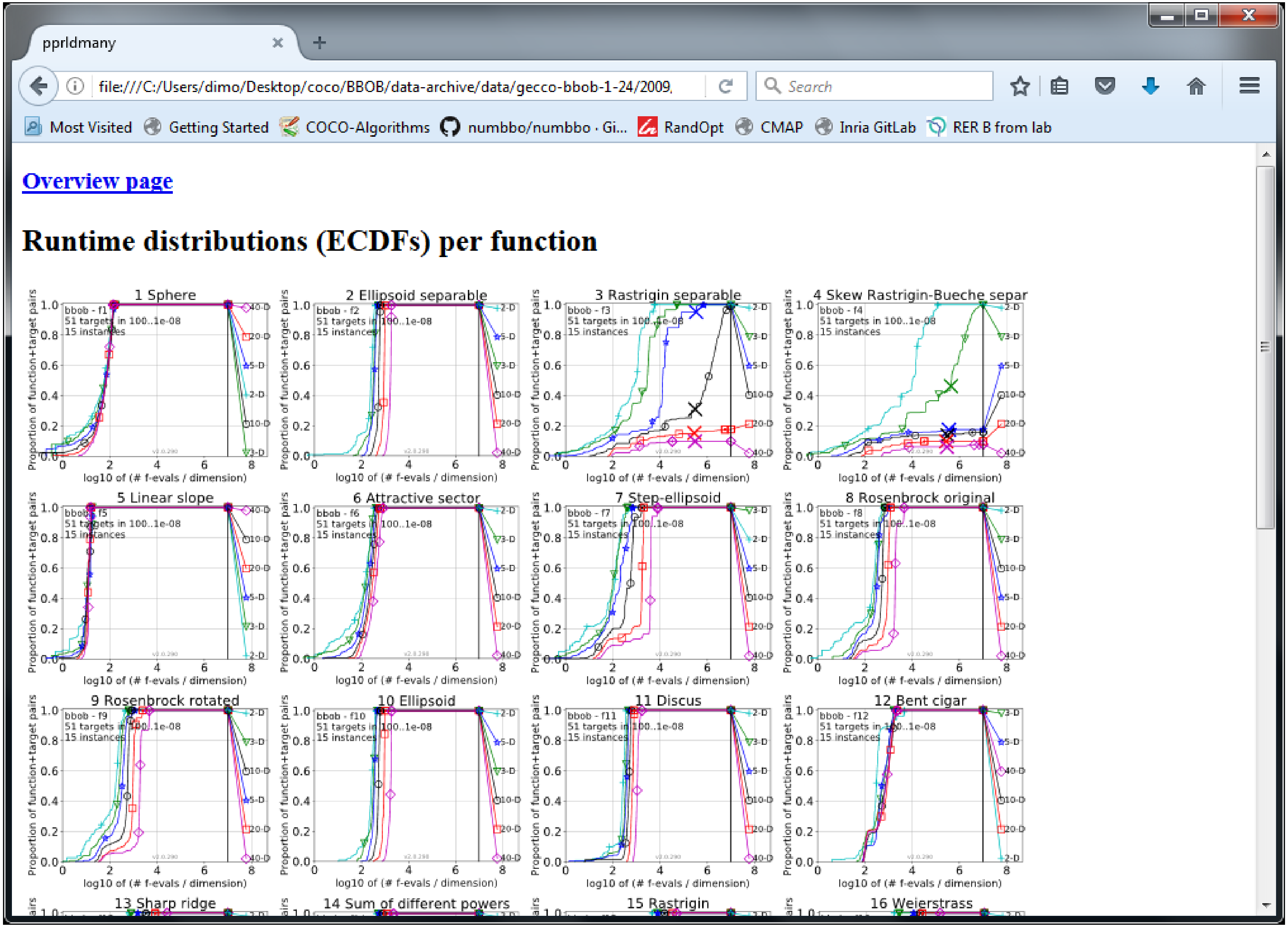
[Runtime distribution for selected targets and f-distributions]

[Runtime loss ratios]

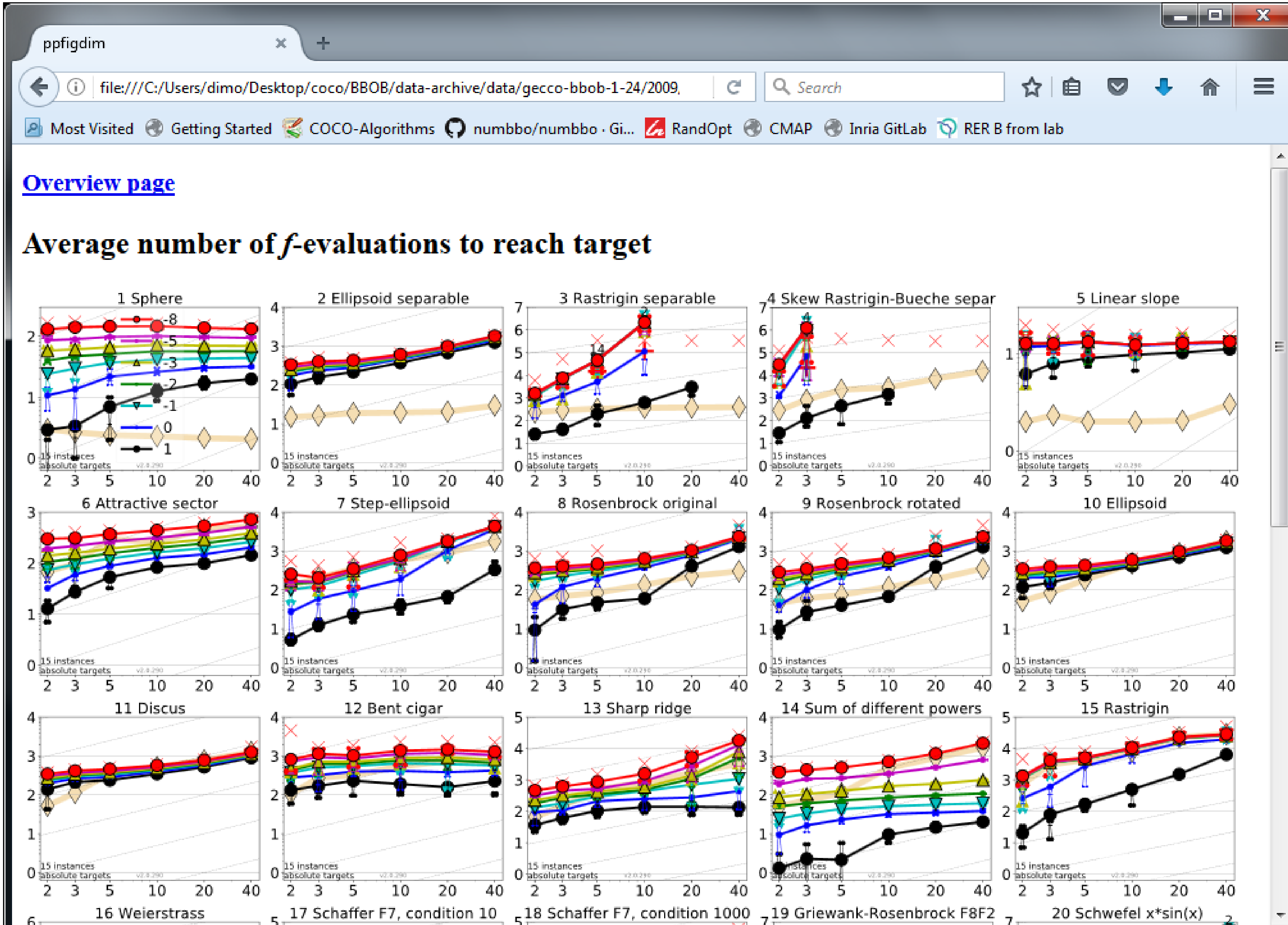
Runtime distributions (ECDFs) over all targets



Automatically Generated Results



Automatically Generated Results



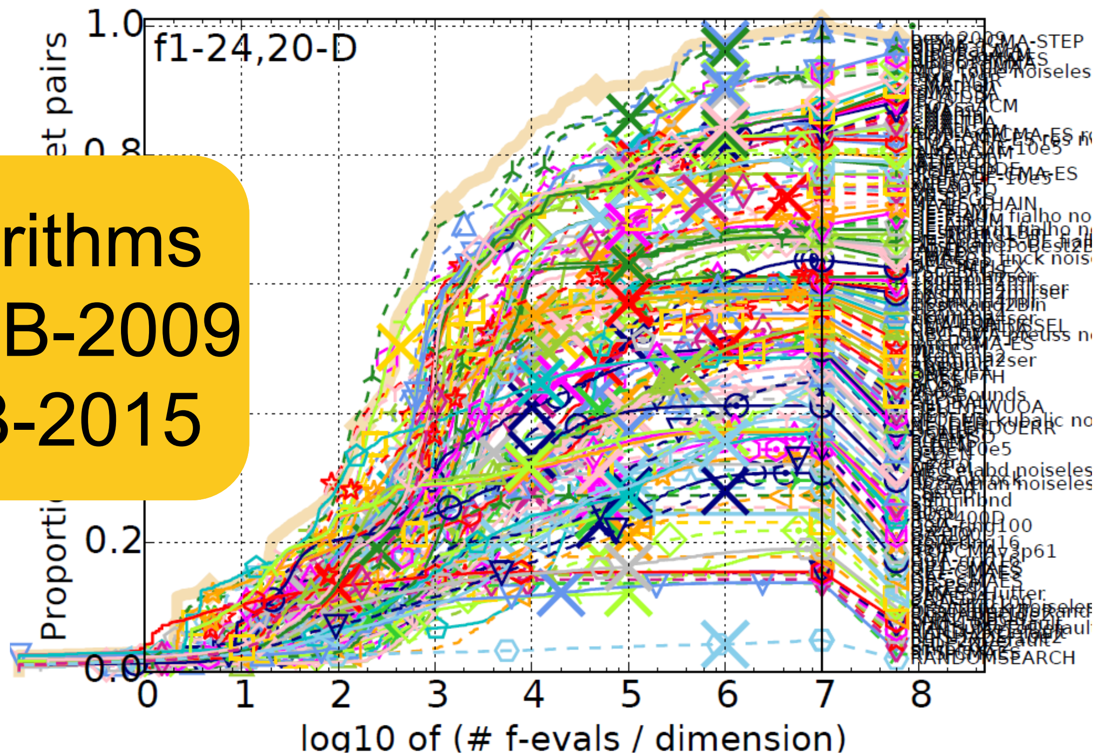
Available Test Suites in COCO

▪ bbob	24 noiseless fcts	240+ algo data sets
▪ bbob-noisy	30 noisy fcts	40+ algo data sets
▪ bbob-biobj	55 bi-objective fcts	30+ algo data sets
▪ bbob-largescale	24 noiseless fcts	16 algo data sets
▪ bbob-mixint	24 mixed-int. fcts	5 algo data sets
▪ bbob-constrained	54 fcts w/ varying constr.	9 algo data sets

Worth to Note: ECDFs in COCO

In COCO, ECDF graphs

- never aggregate over dimension
    - but often over targets and functions
- can show data of more than 1 algorithm at a time



150 algorithms from BBOB-2009 till BBOB-2015

The single-objective BBOB functions

https://numbbo.github.io/gforge/downloads/download16.00/bbobdocfunctions.pdf

The bbob Testbed

24 functions in 5 groups:

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

6 dimensions: 2, 3, 5, 10, 20, (40 optional)

Notion of Instances

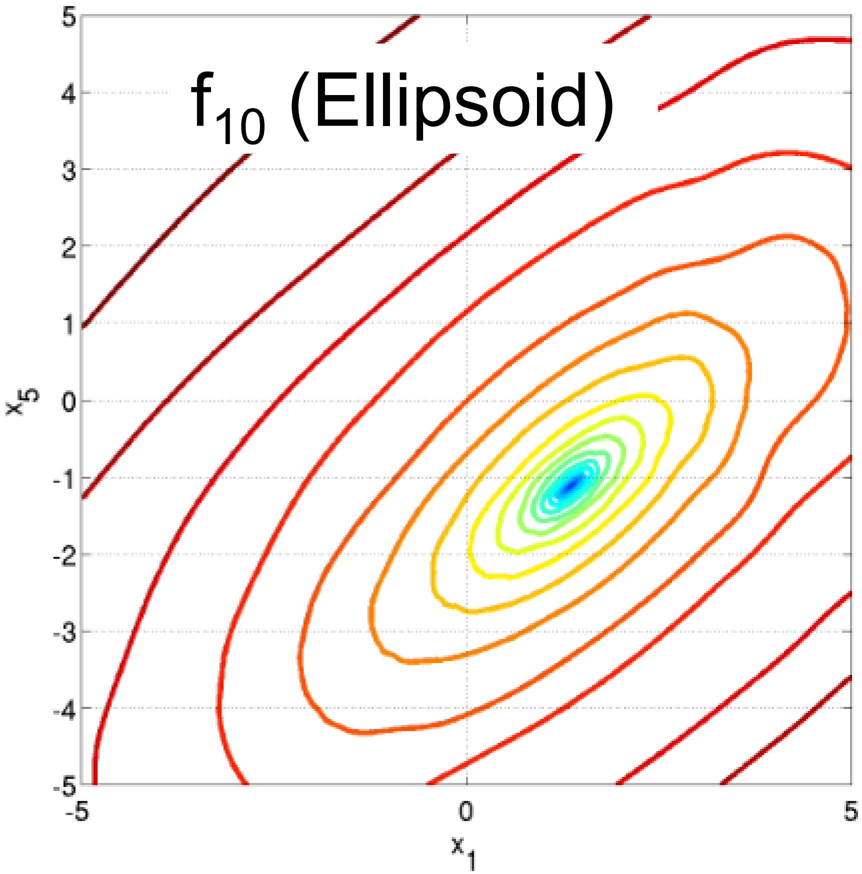
- All COCO problems come in form of instances
  - e.g. as translated/rotated versions of the same function
- Prescribed instances typically change from year to year
  - avoid overfitting
  - 5 instances are always kept the same

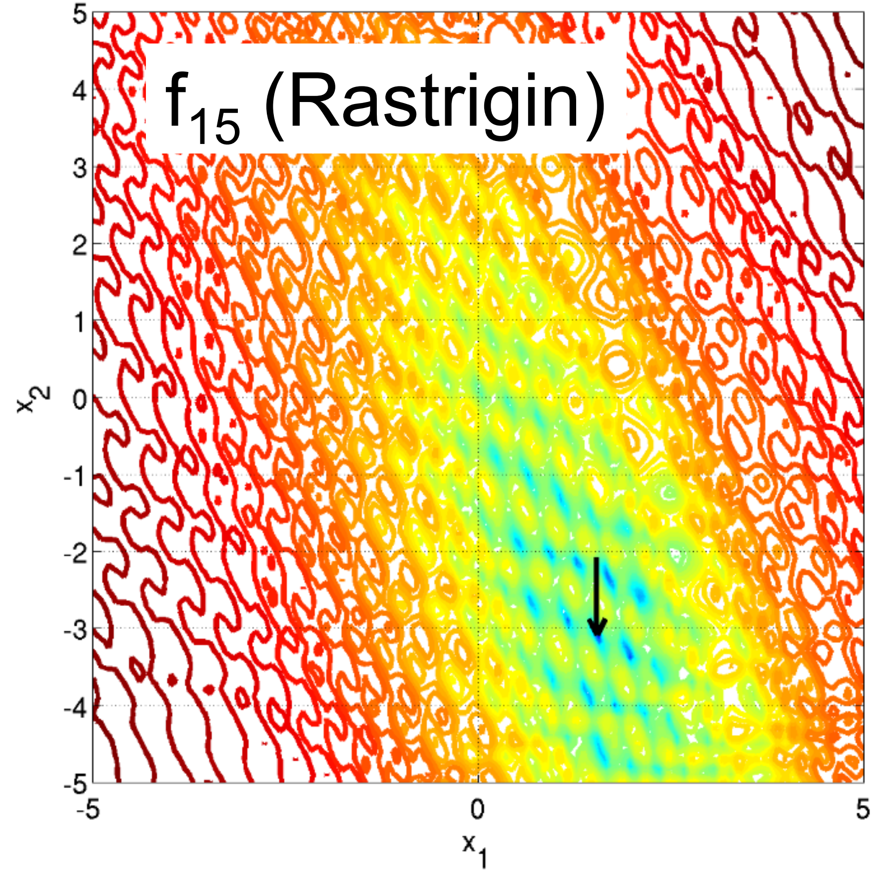
Plus:

- the bbob functions are locally perturbed by non-linear transformations

Notion of Instances

- All COCO problems come in form of instances
  - e.g. as translated/rotated versions of the same





Exercise

Objectives:

- investigate the performance of these 6 algorithms:
  - CMA-ES ("IPOP-CMA-ES" version)
  - CMA-ES ("BIPOP-CMA-ES" version)
  - Nelder-Mead simplex (use "NelderDoerr" version here)
  - BFGS quasi-Newton
  - Genetic Algorithm: discretization of cont. variables ("GA")
  - ONEFIFTH: (1+1)-ES with 1/5 rule
- postprocessed available here: **`http://www.cmap.polytechnique.fr/~dimo.brockhoff/advancedOptSaclay/2019/exercises/coco-results/`**
- so now: investigate the data!

Exercise

Objective:

investigate the data:

a) which algorithms are the best ones?
b) does this depend on the dimension? Or on other things?
c) look at single graphs: can we say something about the algorithms' invariances, e.g. wrt. rotations of the search space?
d) what's the impact of covariance-matrix-adaptation?
e) what do you think: are the displayed algorithms well-suited for problems with larger dimension?