Introduction to Optimization

Derivative-Free Optimization II: Benchmarking

December 16, 2016

École Centrale Paris, Châtenay-Malabry, France

Dimo Brockhoff

Inria Saclay – Ile-de-France

Course Overview

Date		Topic
Fri, 7.10.2016		Introduction
Fri, 28.10.2016	D	Introduction to Discrete Optimization + Greedy algorithms I
Fri, 4.11.2016	D	Greedy algorithms II + Branch and bound
Fri, 18.11.2016	D	Dynamic programming
Mon, 21.11.2016 in S103-S105	D	Approximation algorithms ~~and heuristics~~
Fri, 25.11.2016 in S103-S105	C	Randomized Search Heuristics + Intro. to Continuous Opt. I
Mon, 28.11.2016 in S103-S105	C	Introduction to Continuous Optimization II
Mon, 5.12.2016 in S103-S105	C	Introduction to Continuous Optimization III
Fri, 9.12.2016	C	Constrained Optimization + Descent Methods
Mon, 12.12.2016 in S103-S105	C	Derivative Free Optimization I: CMA-ES
Fri, 16.12.2016	C	Derivative Free Optimization II: Benchmarking Optimizers with the COCO platform
Wed, 4.1.2017		Exam

if not indicated otherwise, classes take place in S115-S117

Experimental Considerations around CMA-ES

Experimentum Crucis (0)

What did we want to achieve? **with CMA-ES**

- reduce any convex-quadratic function

$$f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{H} \boldsymbol{x}$$

e.g. $f(\boldsymbol{x}) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} x_i^2$

to the sphere model

$$f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{x}$$

without use of derivatives

- lines of equal density align with lines of equal fitness

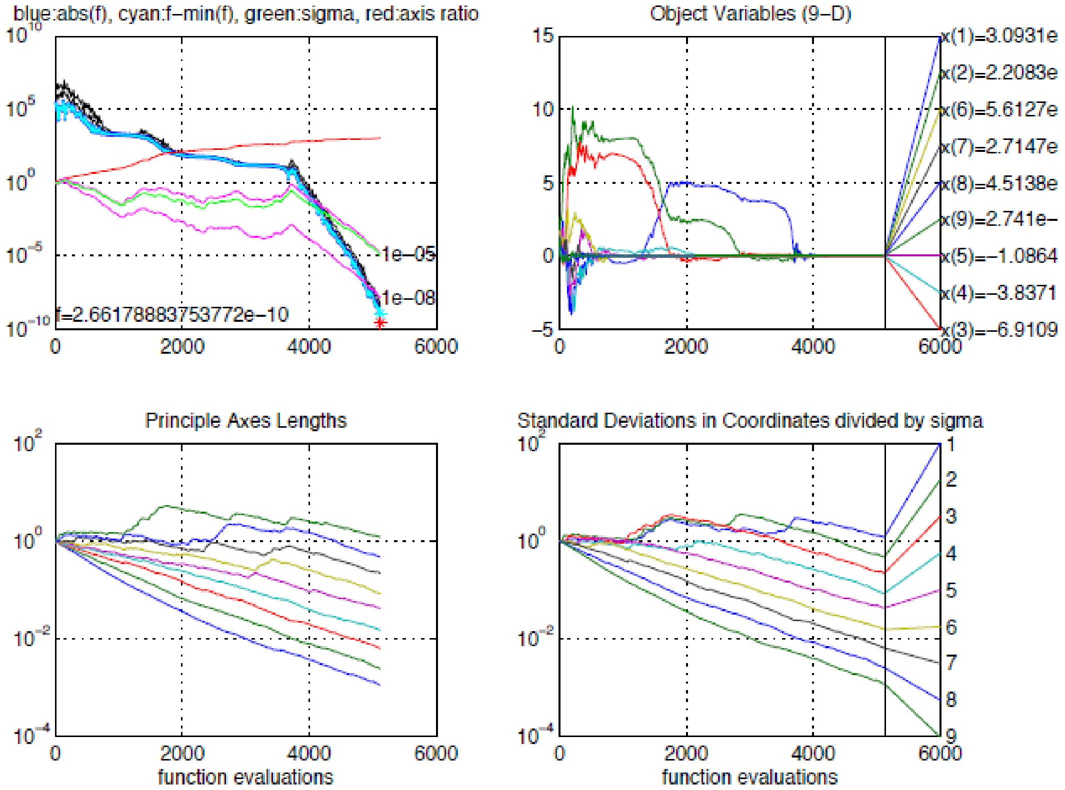
$$\mathbf{C} \propto \boldsymbol{H}^{-1}$$

in a stochastic sense

from [Hansen, p. 91]

Experimentum Crucis (1)

f convex quadratic, separable

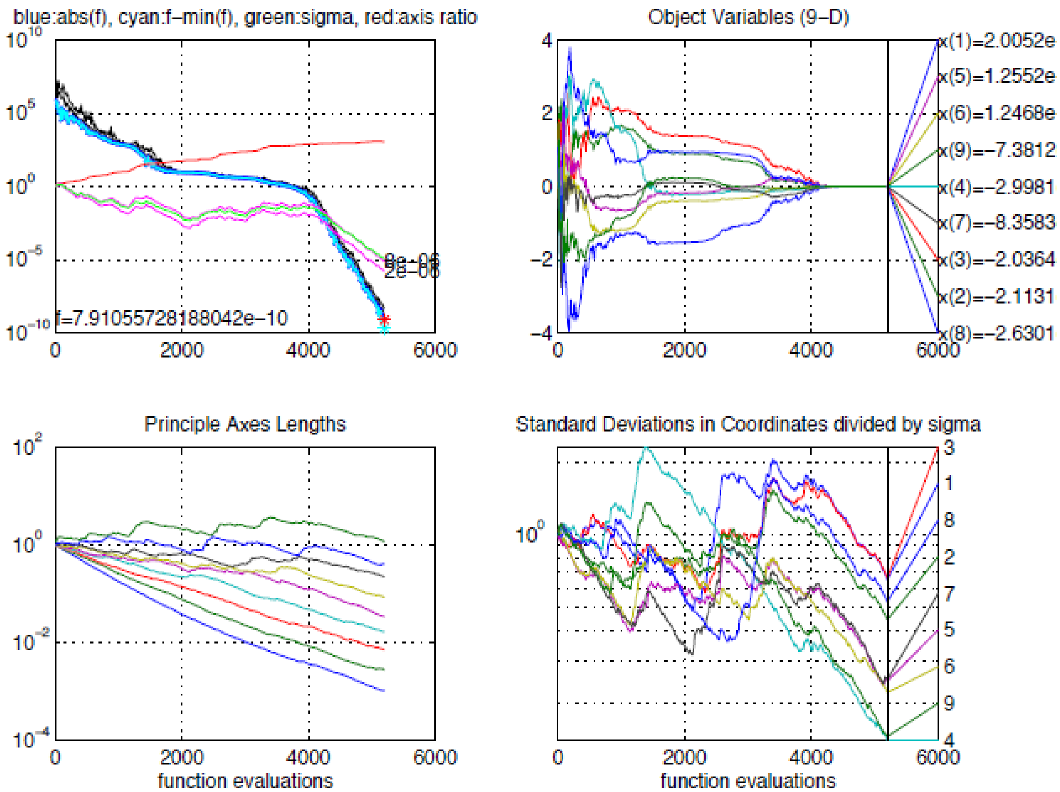


$$f(\boldsymbol{x}) = \sum_{i=1}^{n} 10^{\alpha \frac{i-1}{n-1}} x_i^2, \alpha = 6$$

from [Hansen, p. 92]

Experimentum Crucis (2)

f convex quadratic, as before but non-separable (rotated)



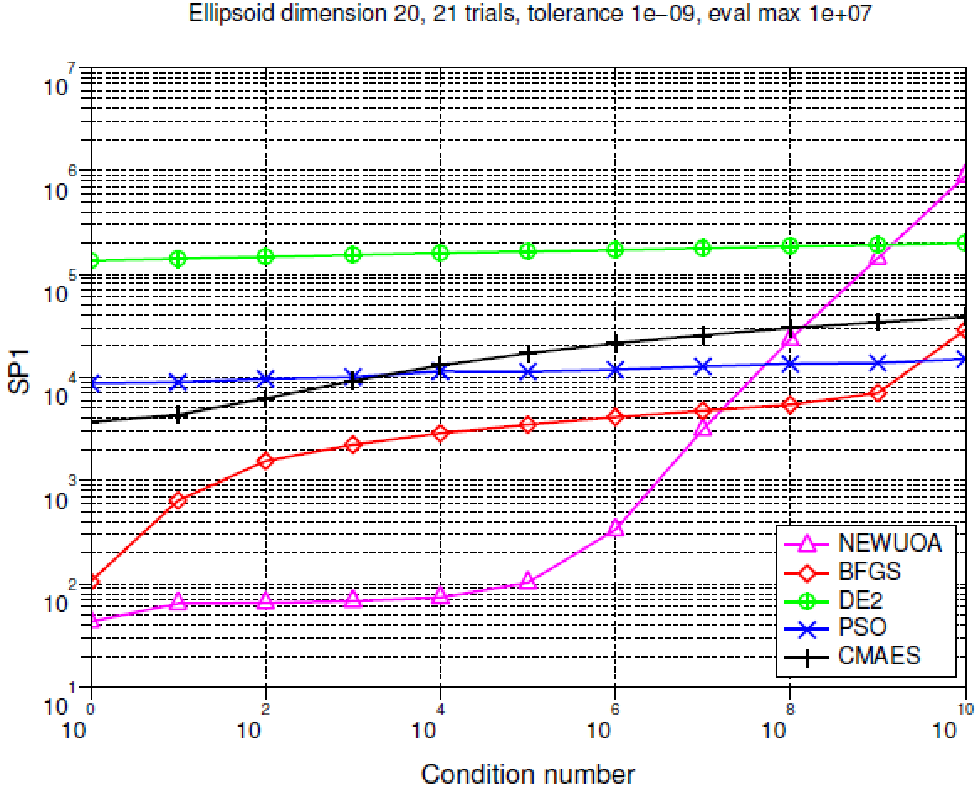
$\mathbf{C} \propto \boldsymbol{H}^{-1}$ for all $g, \mathbf{H}$

$f(\boldsymbol{x}) = g\left(\boldsymbol{x}^{\mathrm{T}}\mathbf{H}\boldsymbol{x}\right)$, $g : \mathbb{R} \to \mathbb{R}$ stricly increasing

from [Hansen, p. 93]

Comparison to BFGS, NEWUOA, PSO and DE

f convex quadratic, separable with varying condition number α



BFGS (Broyden et al 1970)
NEWUAO (Powell 2004)
DE (Storn & Price 1996)
PSO (Kennedy & Eberhart 1995)
CMA-ES (Hansen & Ostermeier 2001)

$f(\boldsymbol{x}) = g(\boldsymbol{x}^{\mathrm{T}}\mathbf{H}\boldsymbol{x})$ with

$\boldsymbol{H}$ diagonal

g identity (for BFGS and NEWUOA)

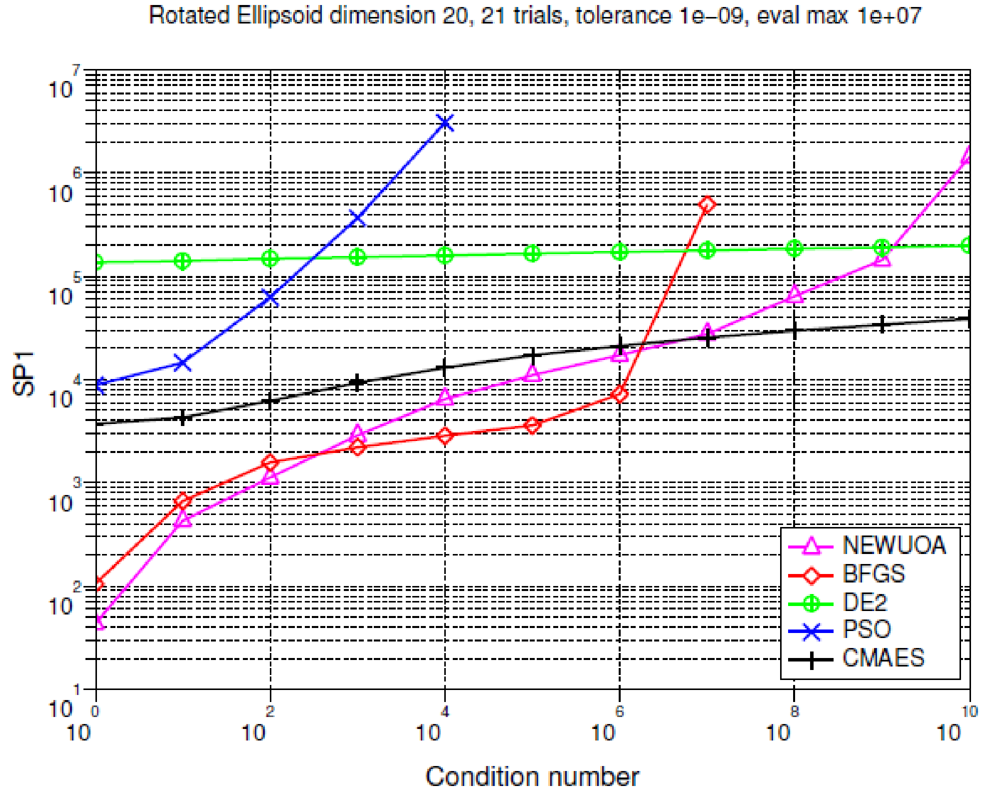
g any order-preserving = strictly increasing function (for all other)

SP1 = average number of objective function evaluations[14] to reach the target function value of $g^{-1}(10^{-9})$

[14] Auger et.al. (2009): Experimental comparisons of derivative free optimization algorithms, SEA

Comparison to BFGS, NEWUOA, PSO and DE

f convex quadratic, non-separable (rotated) with varying condition number α



BFGS (Broyden et al 1970)
NEWUAO (Powell 2004)
DE (Storn & Price 1996)
PSO (Kennedy & Eberhart 1995)
CMA-ES (Hansen & Ostermeier 2001)

$f(\mathbf{x}) = g(\mathbf{x}^T \mathbf{H} \mathbf{x})$ with

$\boldsymbol{H}$ full

g identity (for BFGS and NEWUOA)

g any order-preserving = strictly increasing function (for all other)

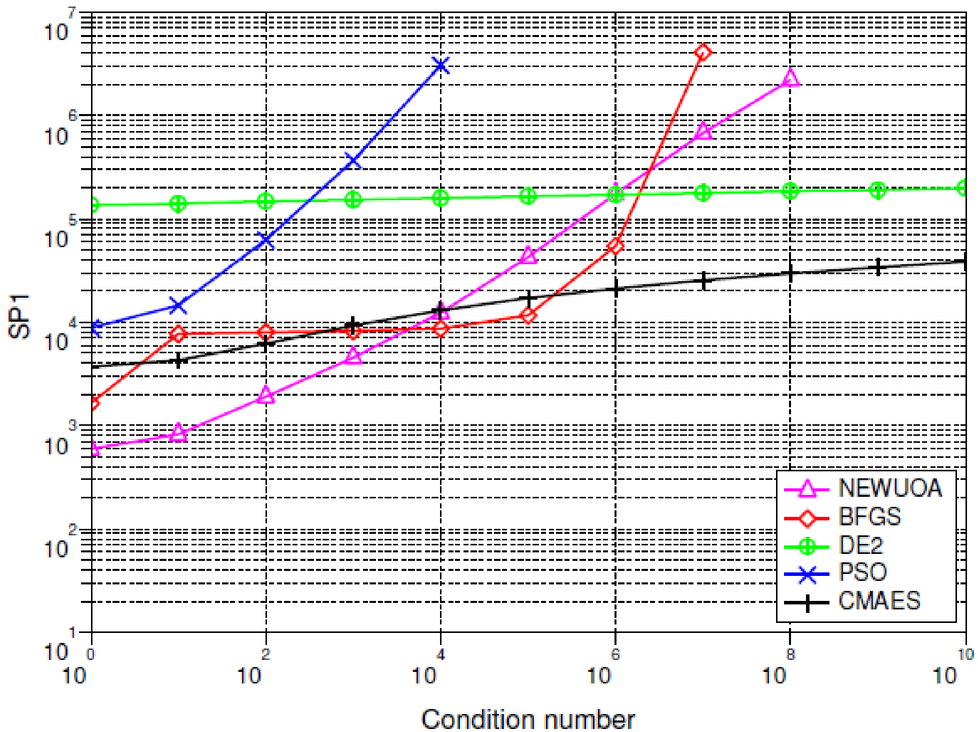
SP1 = average number of objective function evaluations[15] to reach the target function value of $g^{-1}(10^{-9})$

[15] Auger et.al. (2009): Experimental comparisons of derivative free optimization algorithms, SEA

Comparison to BFGS, NEWUOA, PSO and DE

f non-convex, non-separable (rotated) with varying condition number α

Sqrt of sqrt of rotated ellipsoid dimension 20, 21 trials, tolerance 1e−09, eval max 1e+07



BFGS (Broyden et al 1970)
NEWUAO (Powell 2004)
DE (Storn & Price 1996)
PSO (Kennedy & Eberhart 1995)
CMA-ES (Hansen & Ostermeier 2001)

$f(\boldsymbol{x}) = g(\boldsymbol{x}^{\mathrm{T}}\mathbf{H}\boldsymbol{x})$ with

$\boldsymbol{H}$ full

$g : x \mapsto x^{1/4}$ (for BFGS and NEWUOA)

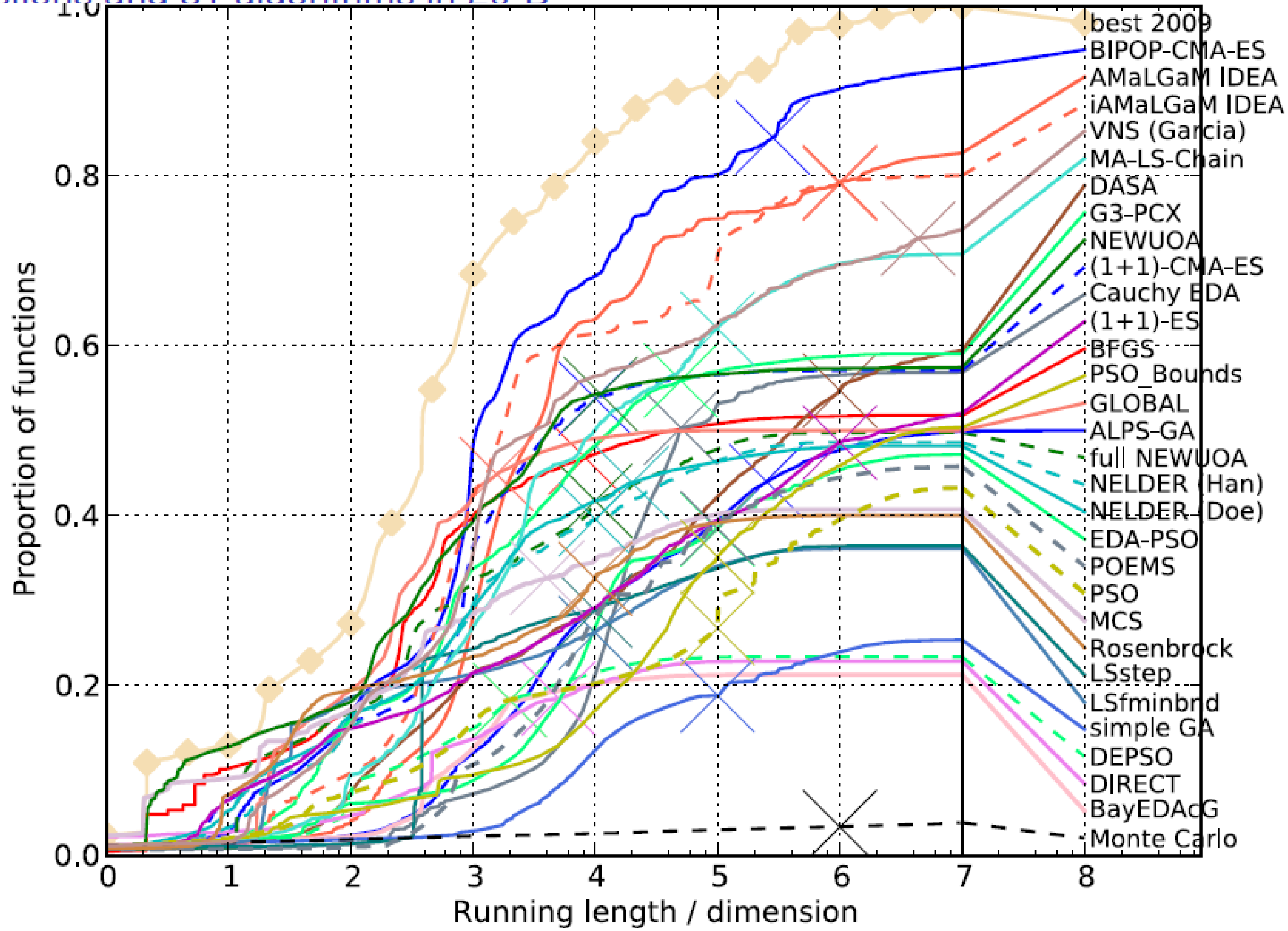
g any order-preserving = strictly increasing function (for all other)

SP1 = average number of objective function evaluations[16] to reach the target function value of $g^{-1}(10^{-9})$

[16] Auger et.al. (2009): Experimental comparisons of derivative free optimization algorithms, SEA

Comparison during BBOB at GECCO 2009

24 functions and 31 algorithms in 20-D



Main Characteristics of (CMA) Evolution Strategies

1. Multivariate normal distribution to generate new search points
   follows the maximum entropy principle
2. Rank-based selection
   implies invariance, same performance on $g(f(\boldsymbol{x}))$ for any increasing g
   more invariance properties are featured
3. Step-size control facilitates fast (log-linear) convergence and possibly linear scaling with the dimension
   in CMA-ES based on an evolution path (a non-local trajectory)
4. *Covariance matrix adaptation (CMA)* increases the likelihood of previously successful steps and can improve performance by orders of magnitude
   the update follows the natural gradient
   $\mathbf{C} \propto \boldsymbol{H}^{-1} \iff$ adapts a variable metric
   $\iff$ new (rotated) problem representation
   $\implies f : \boldsymbol{x} \mapsto g(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x})$ reduces to $\boldsymbol{x} \mapsto \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$

Limitations

of CMA Evolution Strategies

- internal CPU-time: $10^{-8}n^2$ seconds per function evaluation on a 2GHz PC, tweaks are available
  1 000 000 f-evaluations in 100-D take 100 seconds *internal* CPU-time
- better methods are presumably available in case of
  - partly separable problems
  - specific problems, for example with cheap gradients
    specific methods
  - small dimension ($n \ll 10$)
    for example Nelder-Mead
  - small running times (number of f-evaluations $< 100n$)
    model-based methods

Conclusions

I hope it became clear...

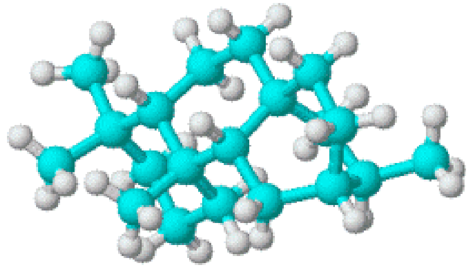
...that CMA-ES samples according to multivariate normal distributions

...how CMA-ES updates its mean, stepsize, and covariance matrix

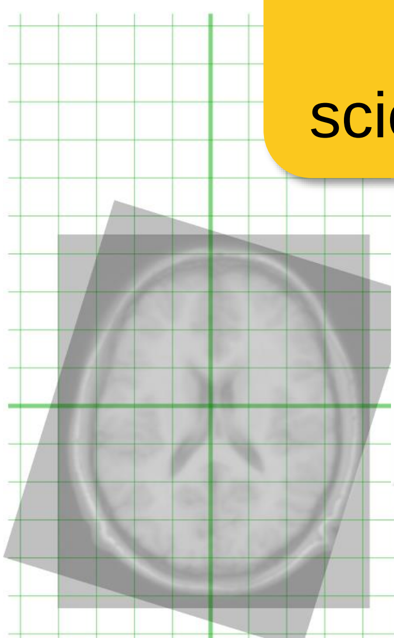
...what are the invariance properties of CMA-ES

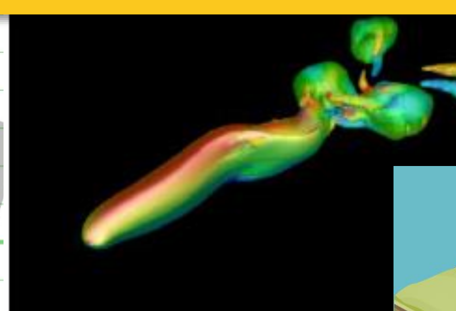
...and how to read the output of CMA-ES

Numerical Benchmarking
of Blackbox Optimization Algorithms



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





Numerical Blackbox Optimization

Optimize $f: \Omega \subset \mathbb{R}^n \mapsto \mathbb{R}^k$

derivatives not available or not useful

Practical Blackbox Optimization

Given:

Not clear:

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

Numerical Blackbox Optimizers

Deterministic algorithms

Quasi-Newton with estimation of gradient (BFGS) [Broyden et al. 1970]
Simplex downhill [Nelder & Mead 1965]
Pattern search [Hooke and Jeeves 1961]
Trust-region methods (NEWUOA, BOBYQA) [Powell 2006, 2009]

Stochastic (randomized) search methods

Evolutionary Algorithms (continuous domain)

- Differential Evolution [Storn & Price 1997]
- Particle Swarm Optimization [Kennedy & Eberhart 1995]
- **Evolution Strategies, CMA-ES** [Rechenberg 1965, Hansen&Ostermeier 2001]
- Estimation of Distribution Algorithms (EDAs) [Larrañaga, Lozano, 2002]
- Cross Entropy Method (same as EDA) [Rubinstein, Kroese, 2004]
- Genetic Algorithms [Holland 1975, Goldberg 1989]

Simulated annealing [Kirkpatrick et al. 1983]
Simultaneous perturbation stochastic approx. (SPSA) [Spall 2000]

Numerical Blackbox Optimizers

Deterministic algorithms

Quasi-Newton with estimation of gradient (BFGS) [Broyden et al. 1970]
Simplex downhill [Nelder & Mead 1965]
Pattern search [Hooke and Jeeves 1961]
Trust-region methods (NEWUOA, BOBYQA) [Powell 2006, 2009]

Stochastic (randomized) search methods

Evolutionary Algorithms (continuous domain)

- Differential Evolution [Storn & Price 1997]
- Particle Swarm Optimization [Kennedy & Eberhart 1995]
- **Evolution Strategies, CMA-ES** [Rechenberg 1965, Hansen&Ostermeier 2001]
- Estimation of Distribution Algorithms (EDAs) [Larrañaga, Lozano, 2002]
- Cross Entropy Method (same as EDA) [Rubinstein, Kroese, 2004]
- Genetic Algorithms [Holland 1975, Goldberg 1989]

Simulated annealing [Kirkpatrick et al. 1983]
Simultaneous perturbation stochastic approx. (SPSA) [Spall 2000]

- choice typically not immediately clear
- although practitioners have knowledge about problem difficulties (e.g. multi-modality, non-separability, ...)

Need: Benchmarking

- understanding of algorithms
- algorithm selection
- putting algorithms to a standardized test
  - simplify judgement
  - simplify comparison
  - regression test under algorithm changes

Kind of everybody has to do it (and it is tedious):

- choosing (and implementing) problems, performance measures, visualization, stat. tests, ...
- running a set of algorithms

that's where COCO comes into play

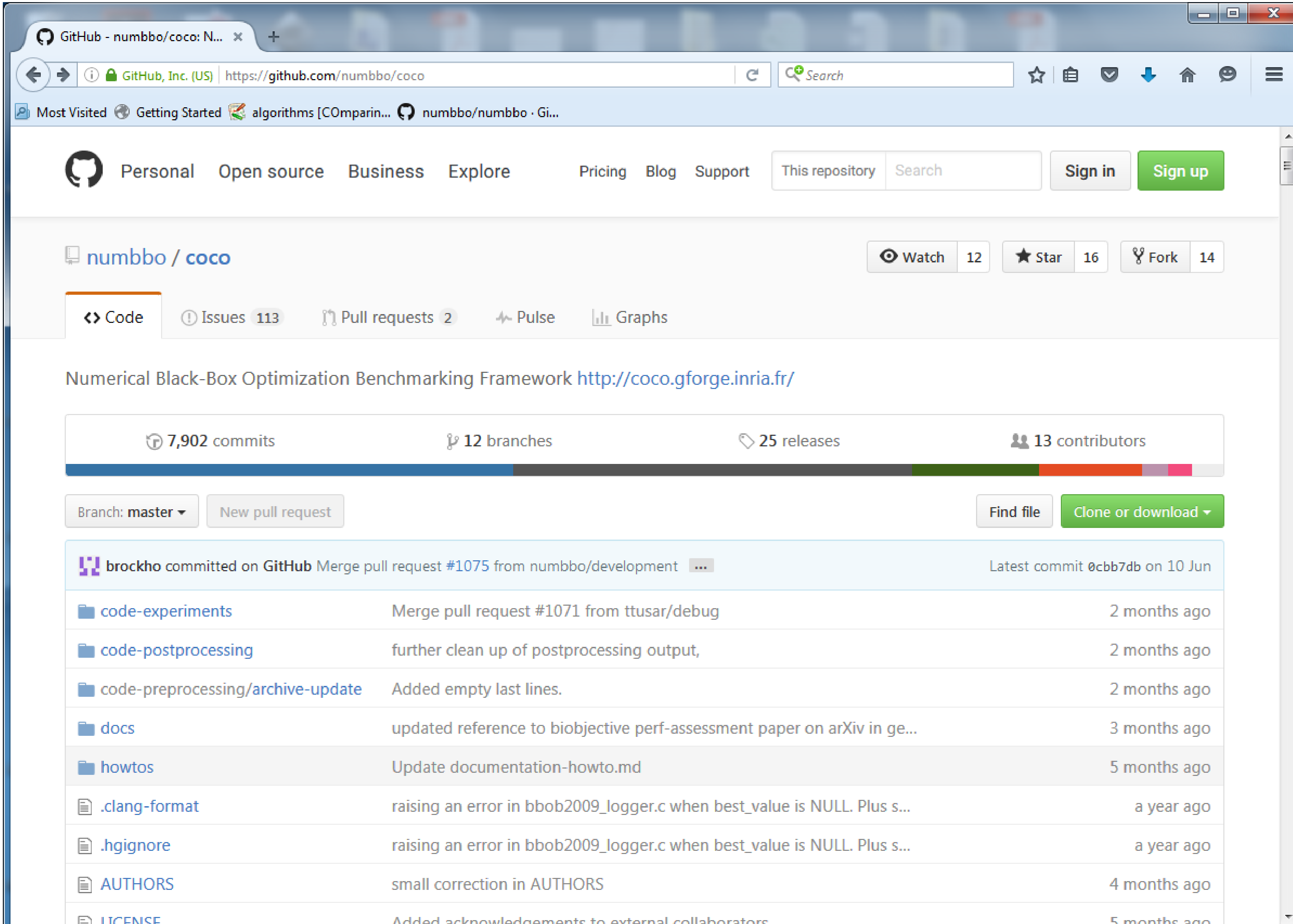
Comparing Continuous Optimizers Platform

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

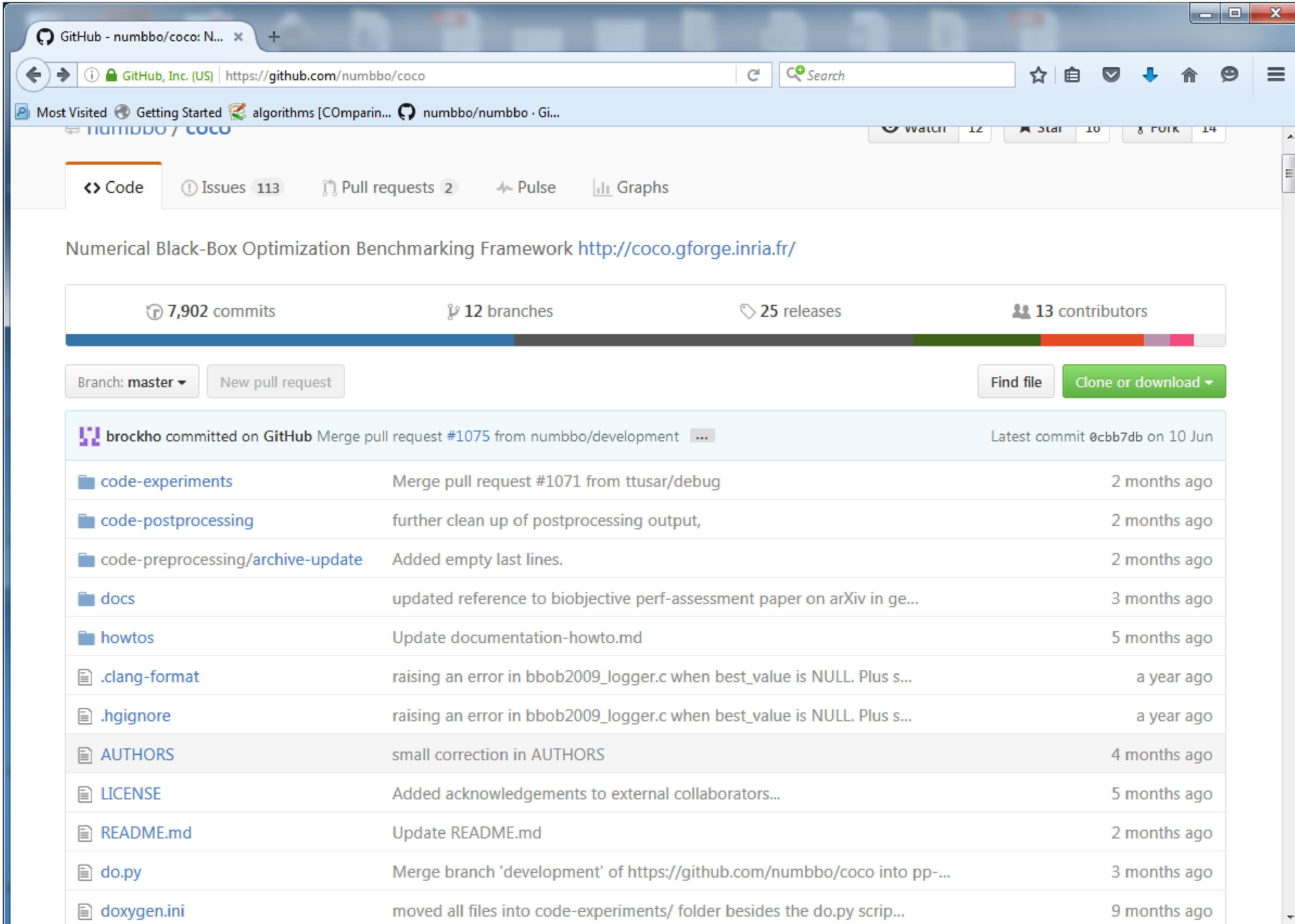
automatized benchmarking

How to benchmark algorithms with COCO?

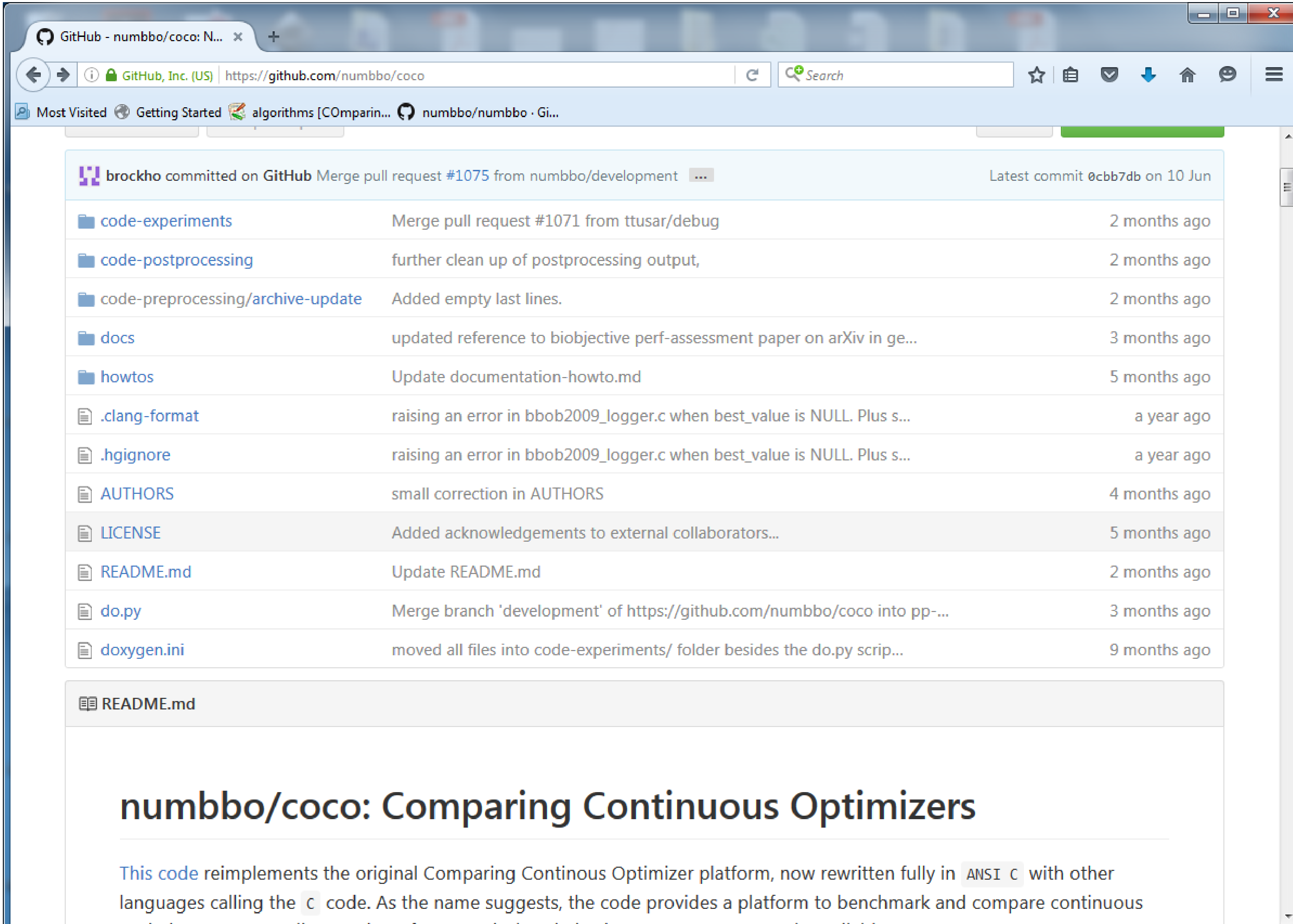
https://github.com/numbbo/coco



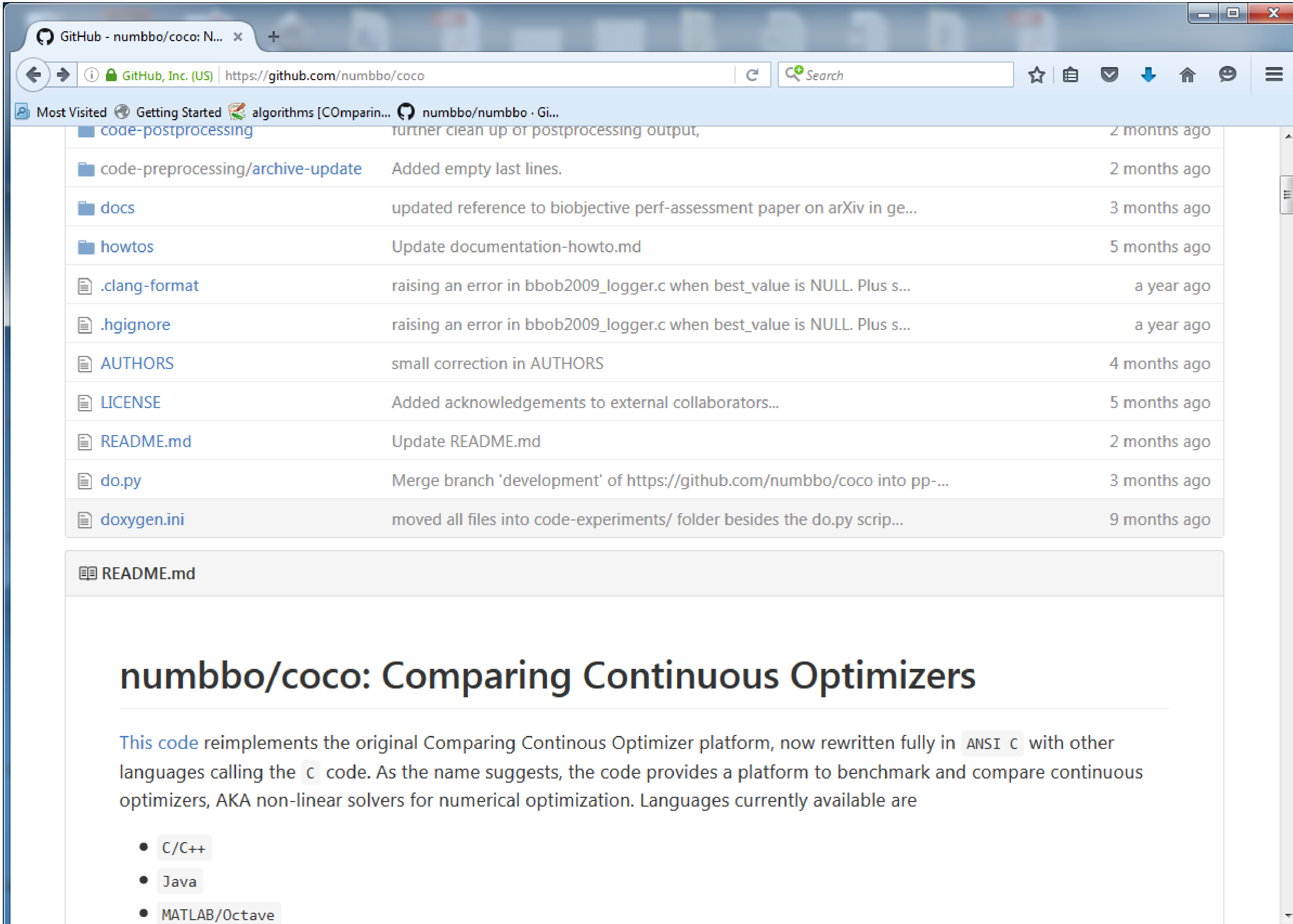
https://github.com/numbbo/coco



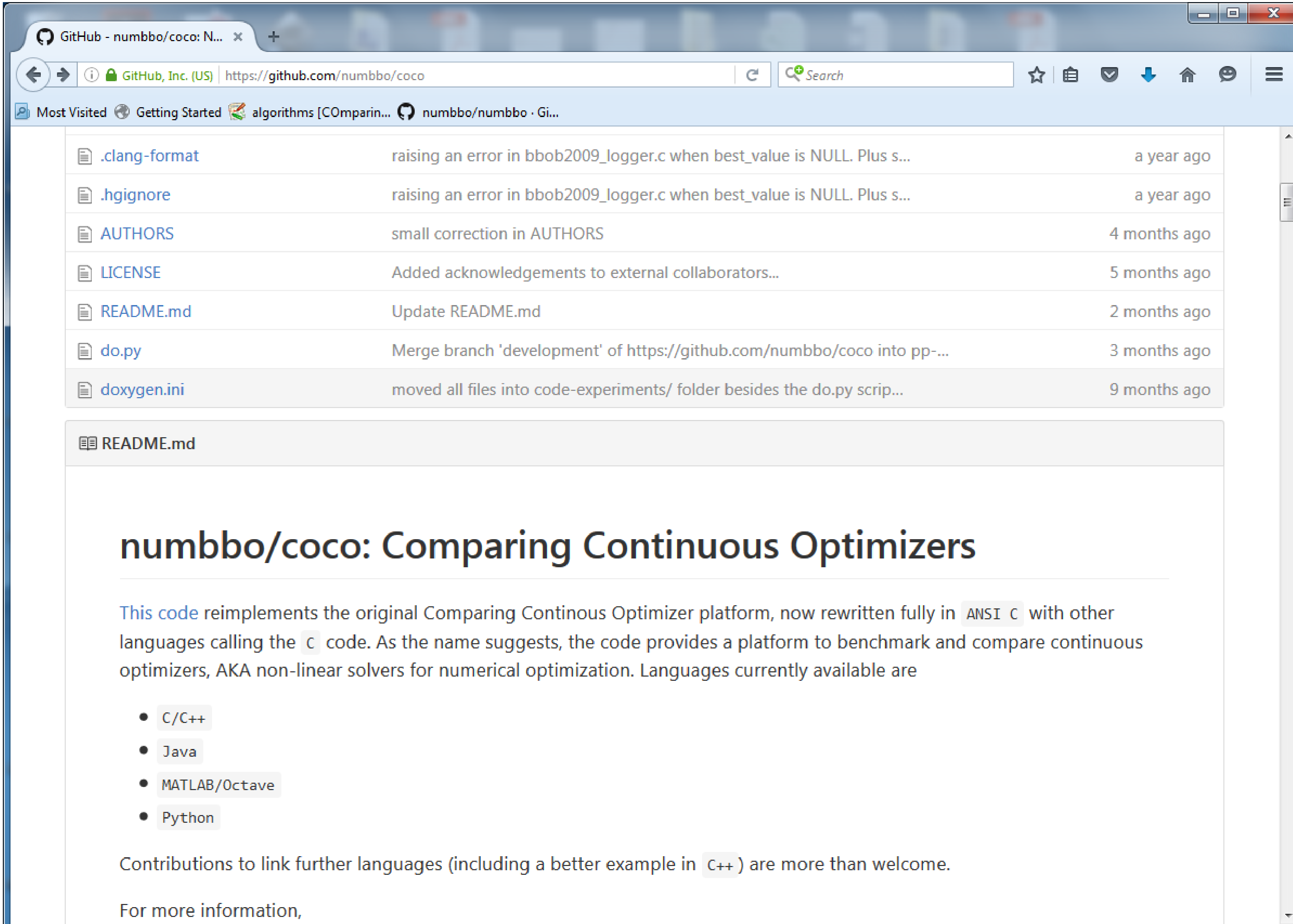
https://github.com/numbbo/coco



https://github.com/numbbo/coco



https://github.com/numbbo/coco



GitHub - numbbo/coco: N...

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

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

.clang-format	raising an error in bbob2009_logger.c when best_value is NULL. Plus s...	a year ago
.hgignore	raising an error in bbob2009_logger.c when best_value is NULL. Plus s...	a year ago
AUTHORS	small correction in AUTHORS	4 months ago
LICENSE	Added acknowledgements to external collaborators...	5 months ago
README.md	Update README.md	2 months ago
do.py	Merge branch 'development' of https://github.com/numbbo/coco into pp-...	3 months ago
doxygen.ini	moved all files into code-experiments/ folder besides the do.py scrip...	9 months ago

README.md

numbbo/coco: Comparing Continuous Optimizers

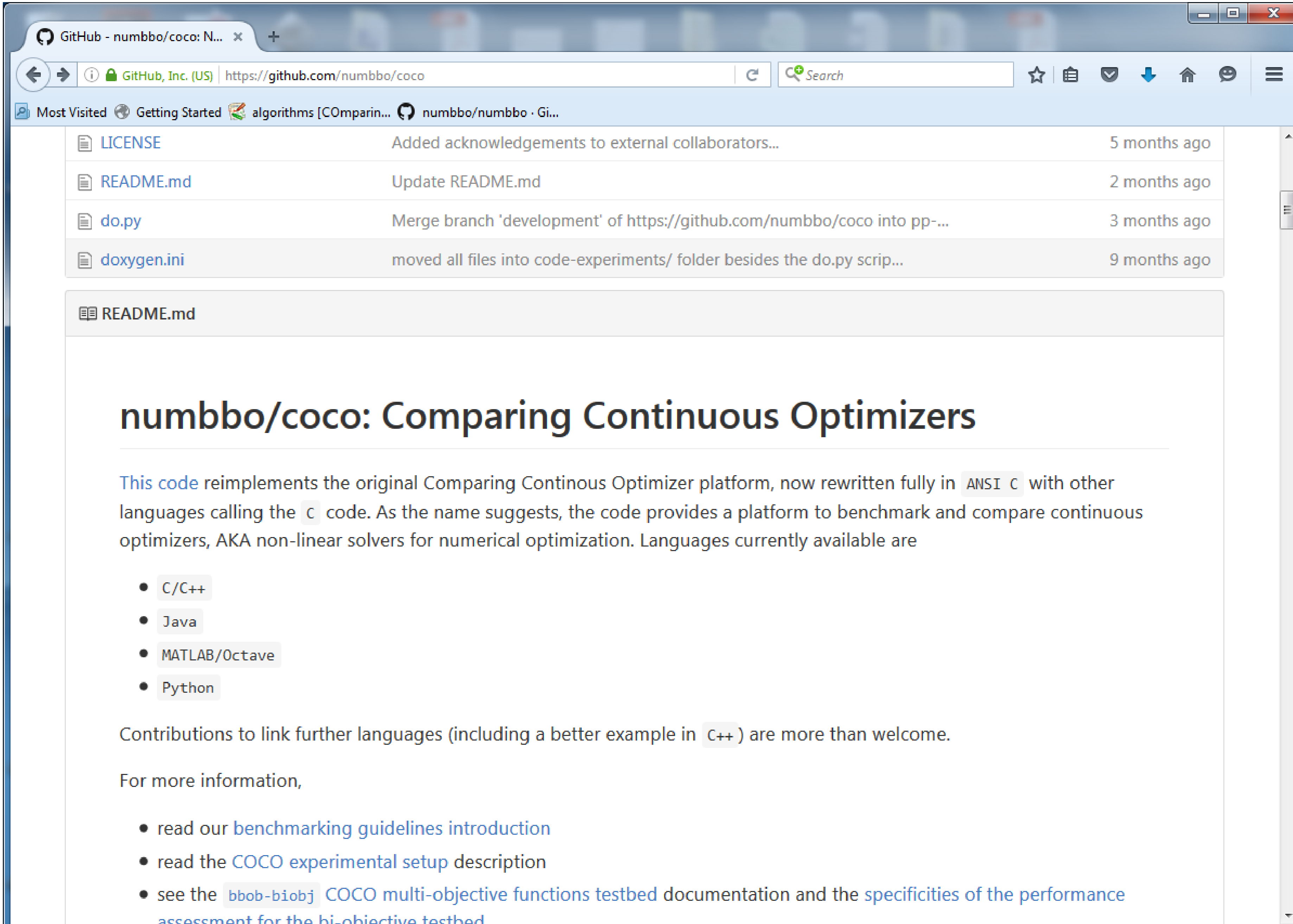
This code reimplements the original Comparing Continous Optimizer platform, now rewritten fully in `ANSI C` with other languages calling the `C` code. As the name suggests, the code provides a platform to benchmark and compare continuous optimizers, AKA non-linear solvers for numerical optimization.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

https://github.com/numbbo/coco



https://github.com/numbbo/coco

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

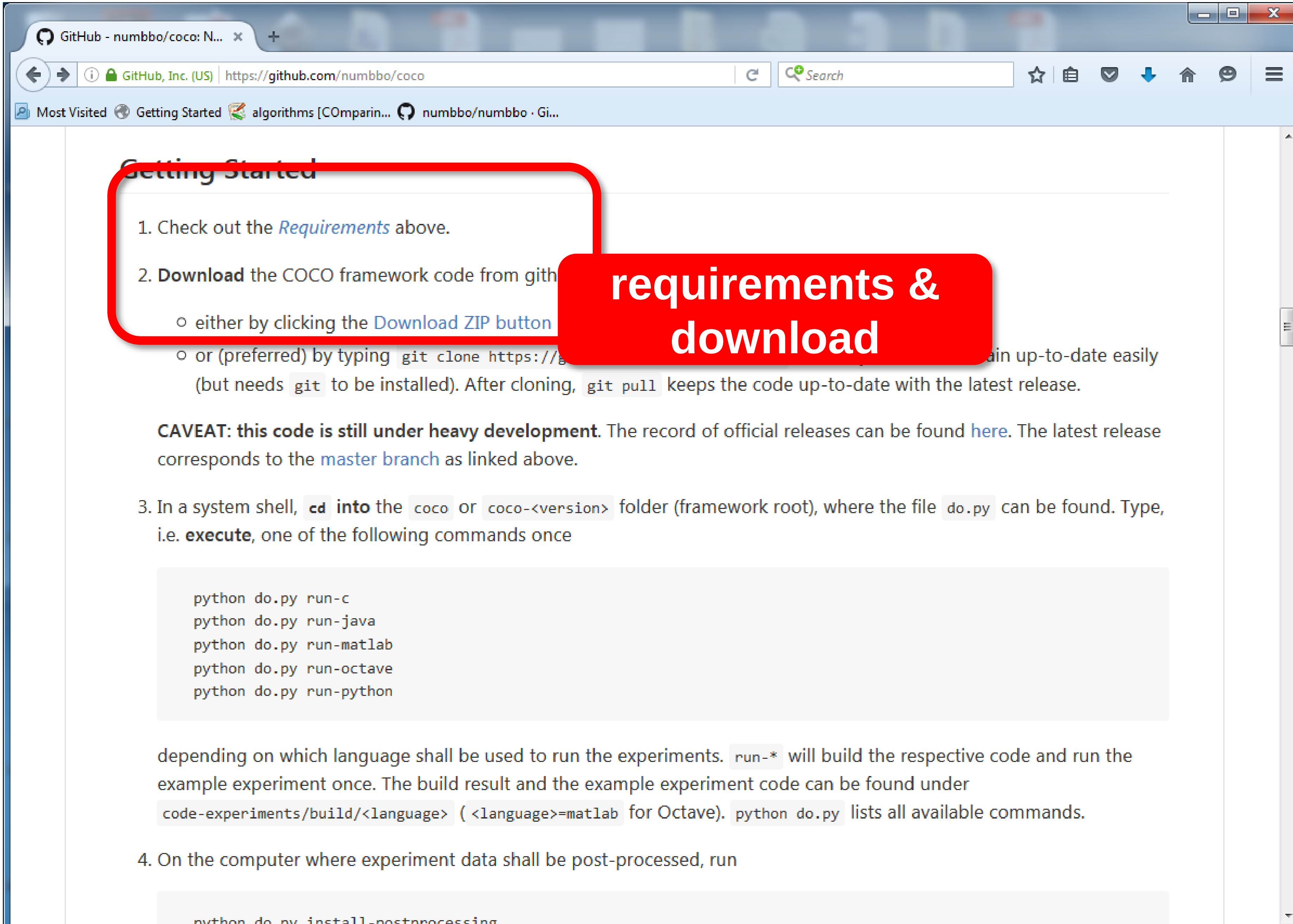
- `C/C++`
- `Java`
- `MATLAB/Octave`
- `Python`

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
- see the `bbob-biobj` COCO multi-objective functions testbed documentation and the specificities of the performance assessment for the bi-objective testbed.
- consult the BBOB workshops series,
- consider to register here for news,
- see the previous COCO home page here and

https://github.com/numbbo/coco



https://github.com/numbbo/coco

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

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

corresponds to the master branch as linked above.

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

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

installation I & test

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

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

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

installation II (post-processing)

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

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

https://github.com/numbbo/coco

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.

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

   - `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`

   example experiment

   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.

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

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

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

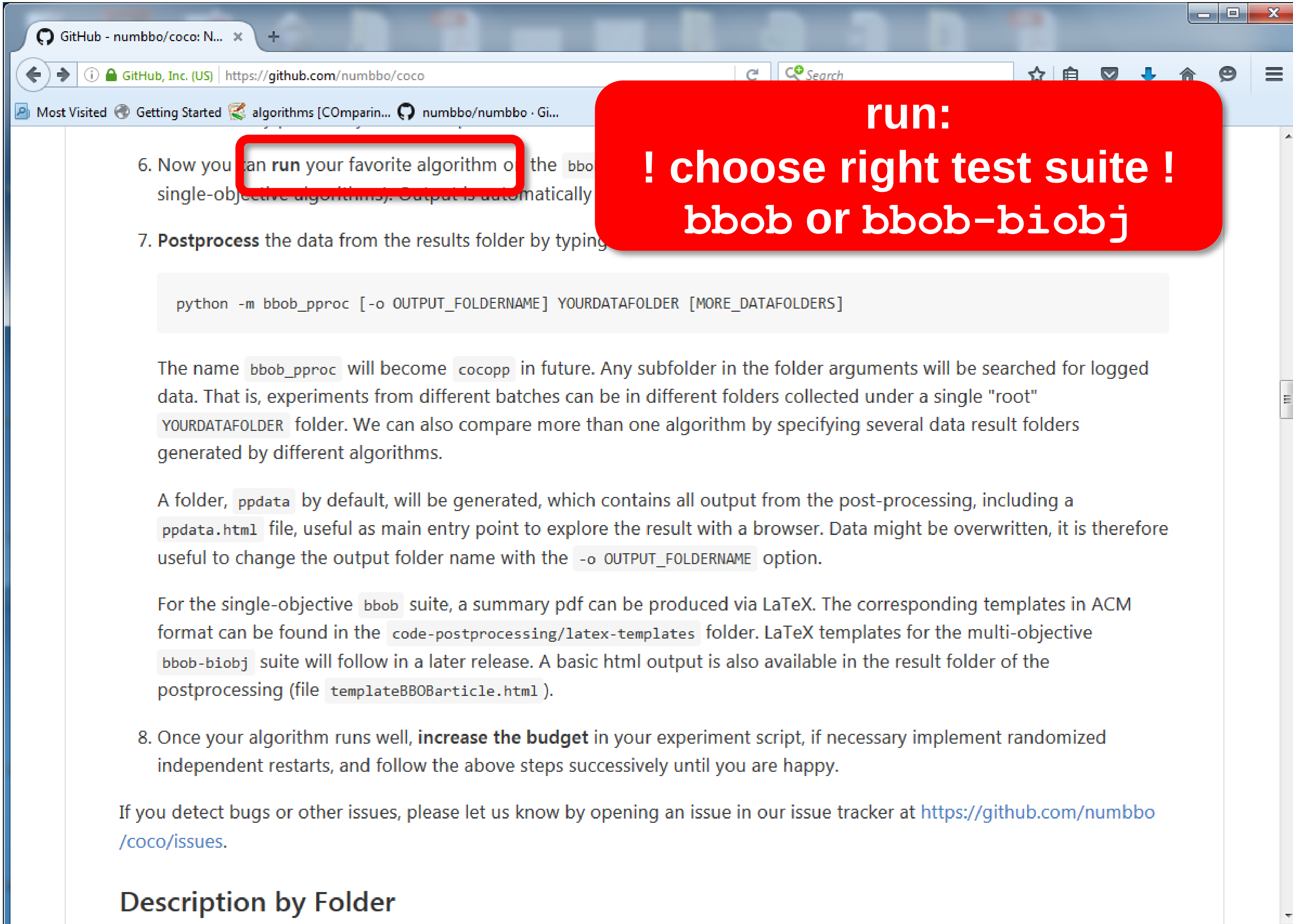
   The name `bbob_pproc` will become `cocopp` in future. Any subfolder in the folder arguments will be searched for logged data. That is, experiments from different batches can be in different folders collected under a single "root" `YOURDATAFOLDER` folder. We can also compare more than one algorithm by specifying several data result folders generated by different algorithms.

example_experiment.c

```c
/* Iterate over all problems in the suite */
while ((PROBLEM = coco_suite_get_next_problem(suite, observer)) != NULL)
{
    size_t dimension = coco_problem_get_dimension(PROBLEM);

    /* Run the algorithm at least once */
    for (run = 1; run <= 1 + INDEPENDENT_RESTARTS; run++) {

      size_t evaluations_done = coco_problem_get_evaluations(PROBLEM);
      long evaluations_remaining =
        (long)(dimension * BUDGET_MULTIPLIER) - (long)evaluations_done;

      if (... || (evaluations_remaining <= 0))
        break;

      my_random_search(evaluate_function, dimension,
                       coco_problem_get_number_of_objectives(PROBLEM),
                       coco_problem_get_smallest_values_of_interest(PROBLEM),
                       coco_problem_get_largest_values_of_interest(PROBLEM),
                       (size_t) evaluations_remaining,
                       random_generator);
}
```

generated by different algorithms.

https://github.com/numbbo/coco



GitHub - numbbo/coco: N...

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

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

run:
! choose right test suite !
bbob or bbob-biobj

6. Now you can **run** your favorite algorithm on the bbo... single-objective algorithms). Output is automatically ...

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

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

The name `bbob_pproc` will become `cocopp` in future. Any subfolder in the folder arguments will be searched for logged data. That is, experiments from different batches can be in different folders collected under a single "root" `YOURDATAFOLDER` folder. We can also compare more than one algorithm by specifying several data result folders generated by different algorithms.

A folder, `ppdata` by default, will be generated, which contains all output from the post-processing, including a `ppdata.html` file, useful as main entry point to explore the result with a browser. Data might be overwritten, it is therefore useful to change the output folder name with the `-o OUTPUT_FOLDERNAME` option.

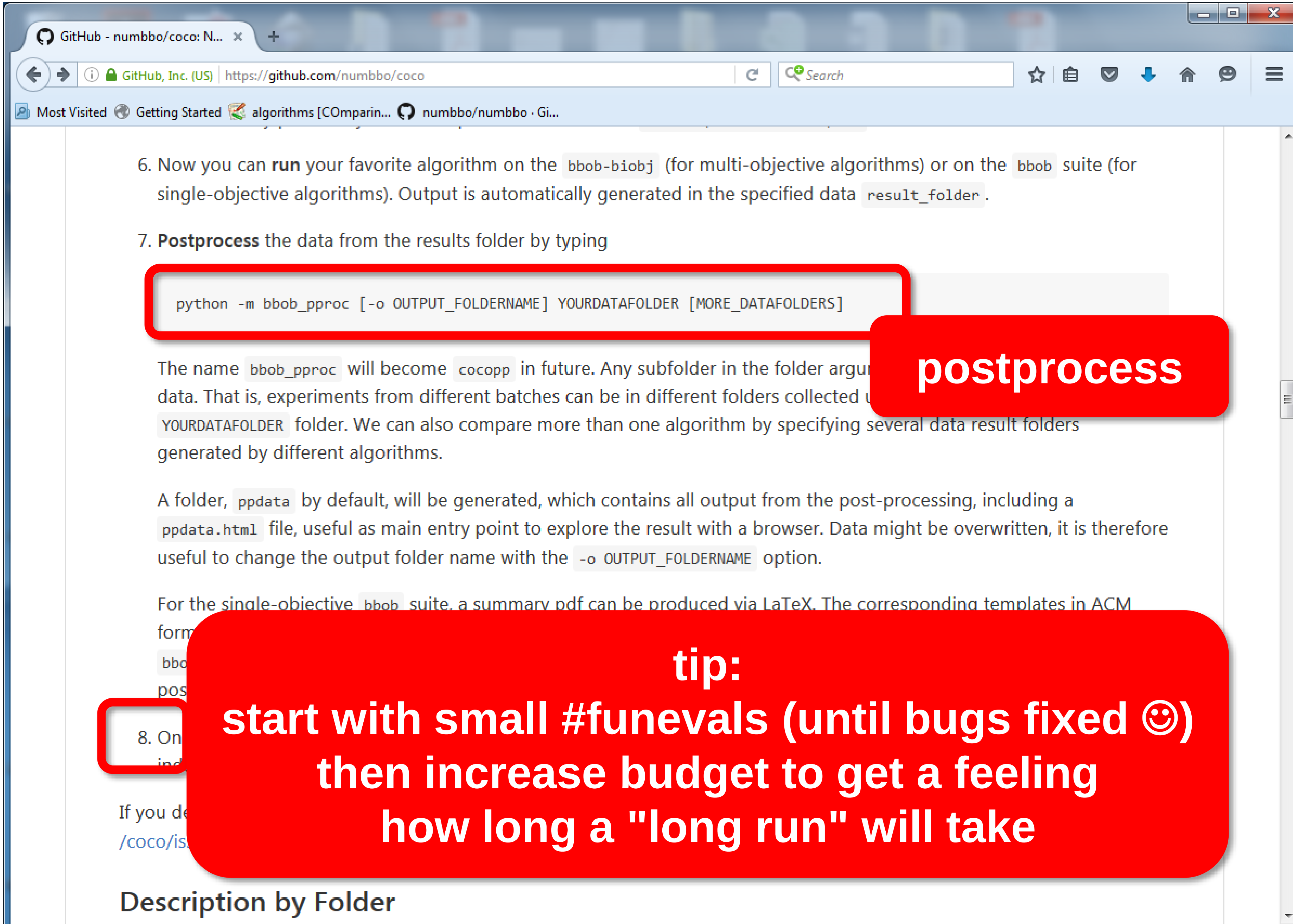
For the single-objective `bbob` suite, a summary pdf can be produced via LaTeX. The corresponding templates in ACM format can be found in the `code-postprocessing/latex-templates` folder. LaTeX templates for the multi-objective `bbob-biobj` suite will follow in a later release. A basic html output is also available in the result folder of the postprocessing (file `templateBBOBarticle.html`).

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

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

Description by Folder

https://github.com/numbbo/coco



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

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

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

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

A folder, `ppdata` by default, will be generated, which contains all output from the post-processing, including a `ppdata.html` file, useful as main entry point to explore the result with a browser. Data might be overwritten, it is therefore useful to change the output folder name with the `-o OUTPUT_FOLDERNAME` option.

For the single-objective `bbob` suite, a summary pdf can be produced via LaTeX. The corresponding templates in ACM form... bbo... pos...

8. On... ind...

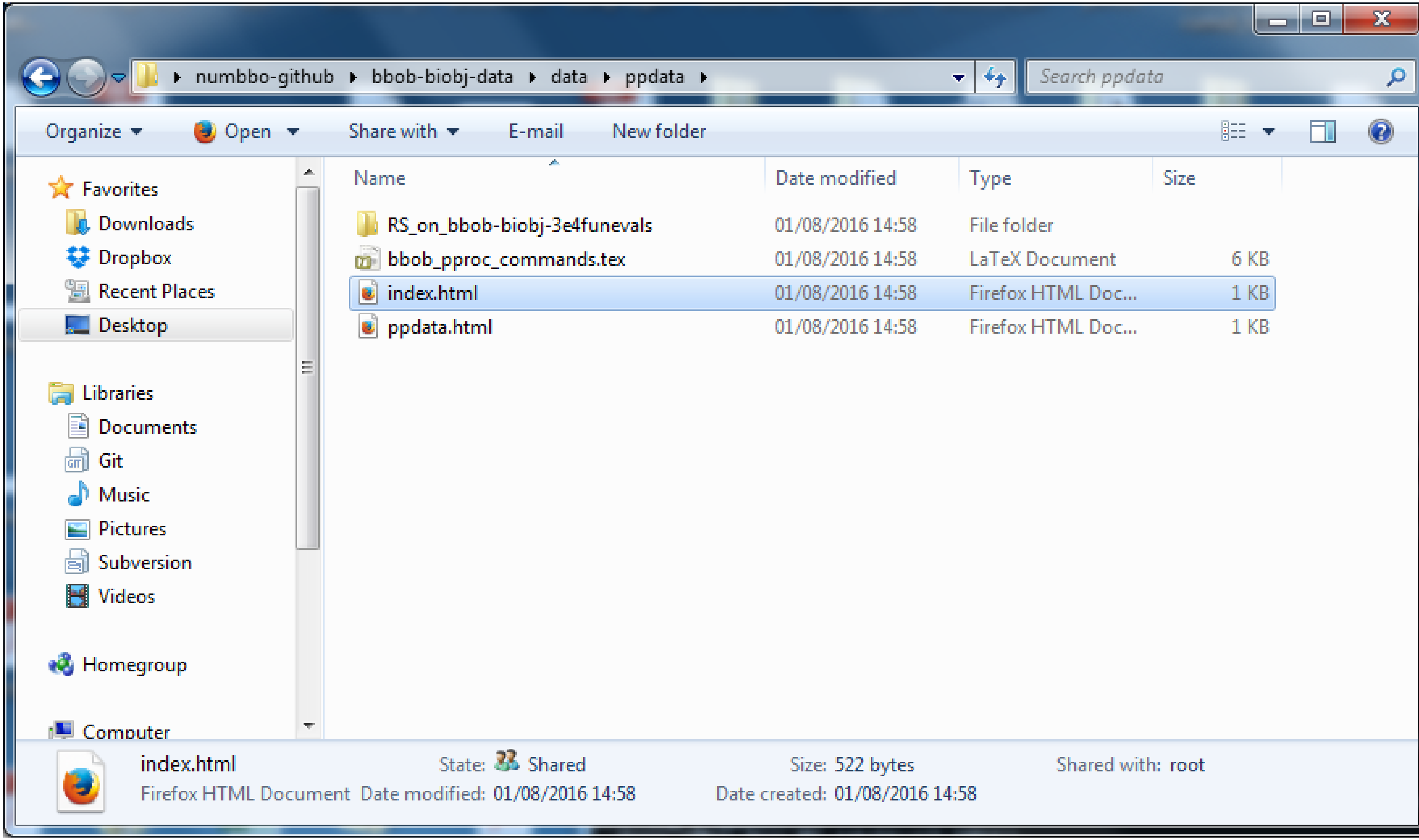
If you d... /coco/is...

Description by Folder

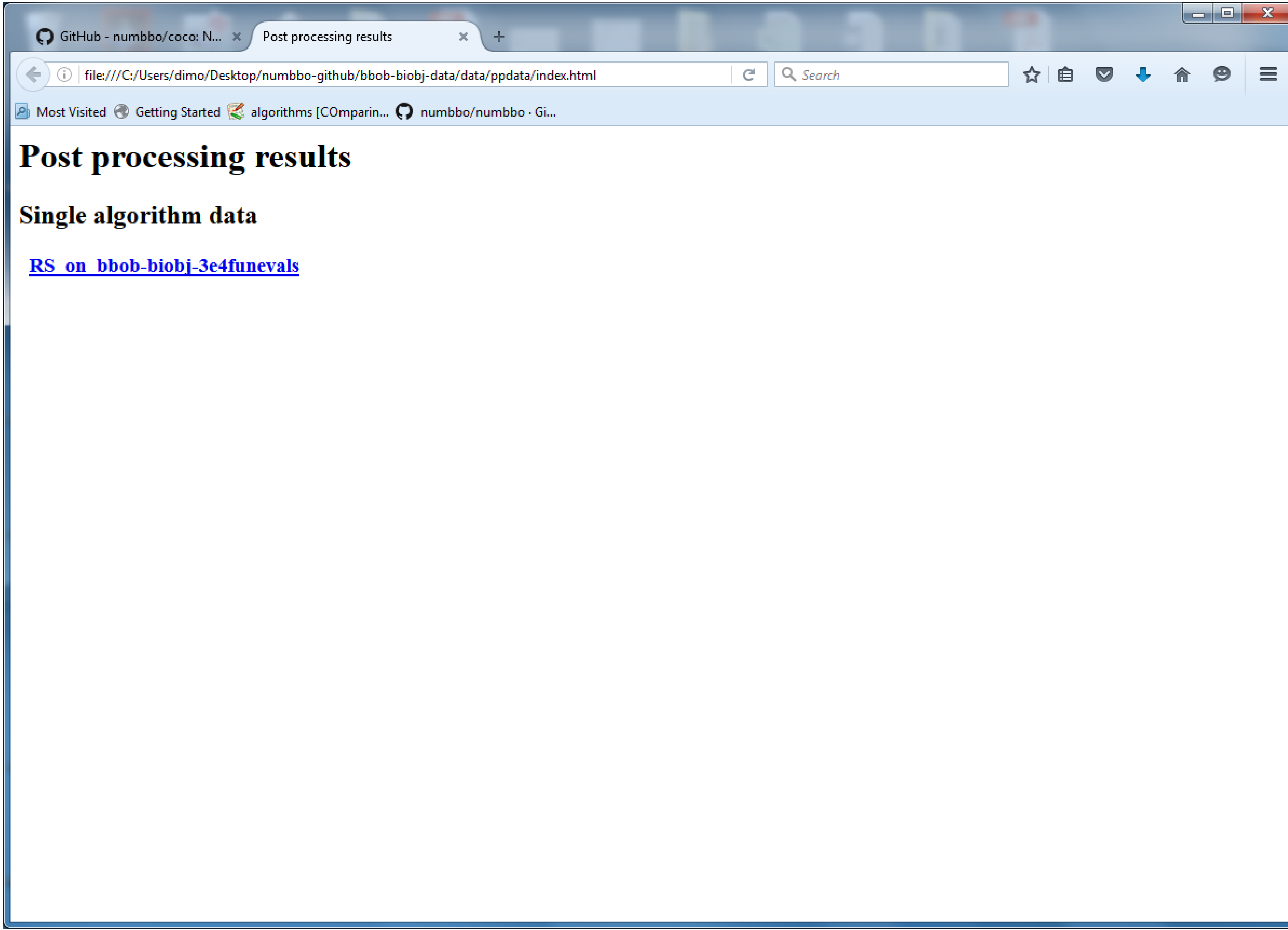
postprocess

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

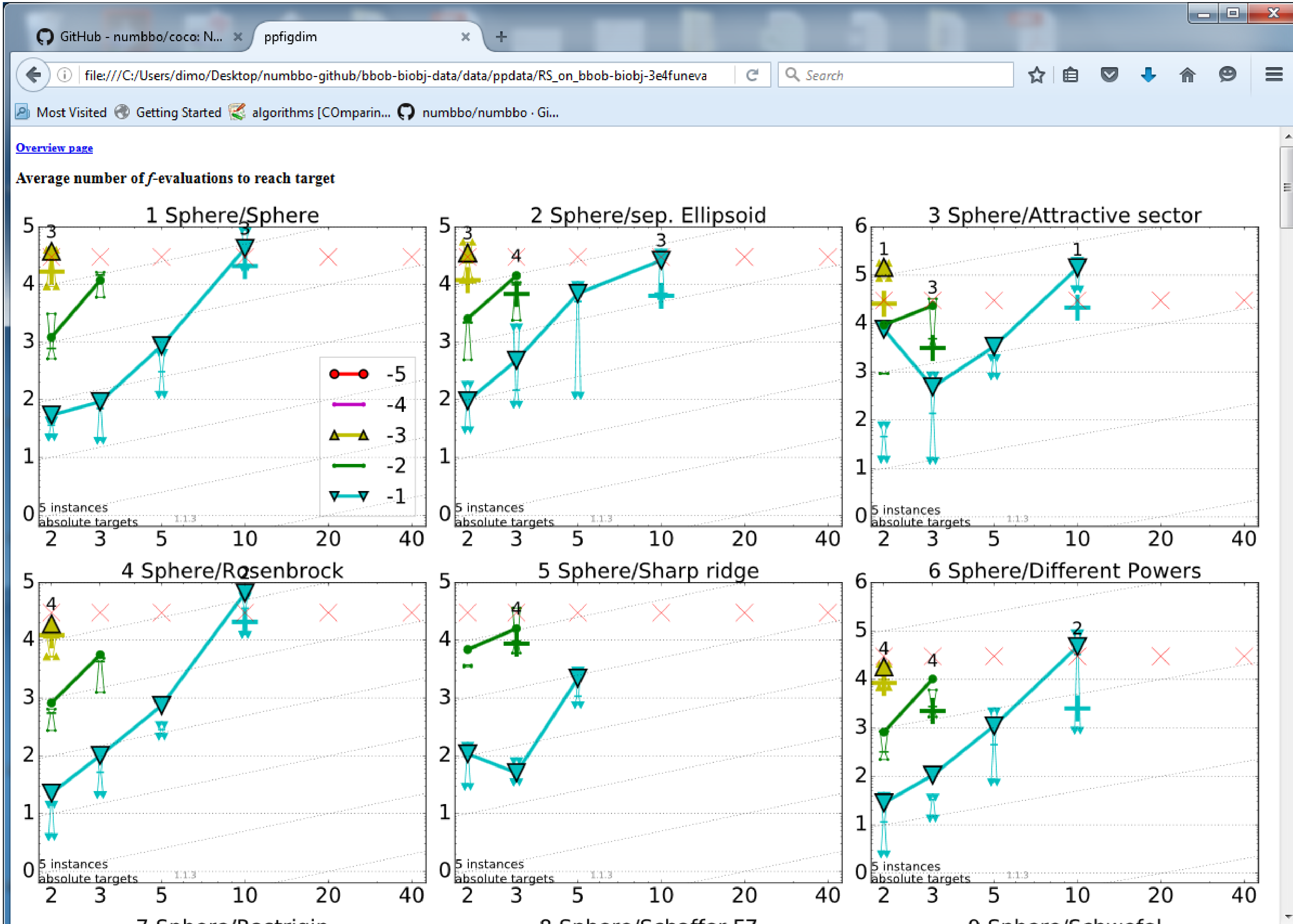
result folder



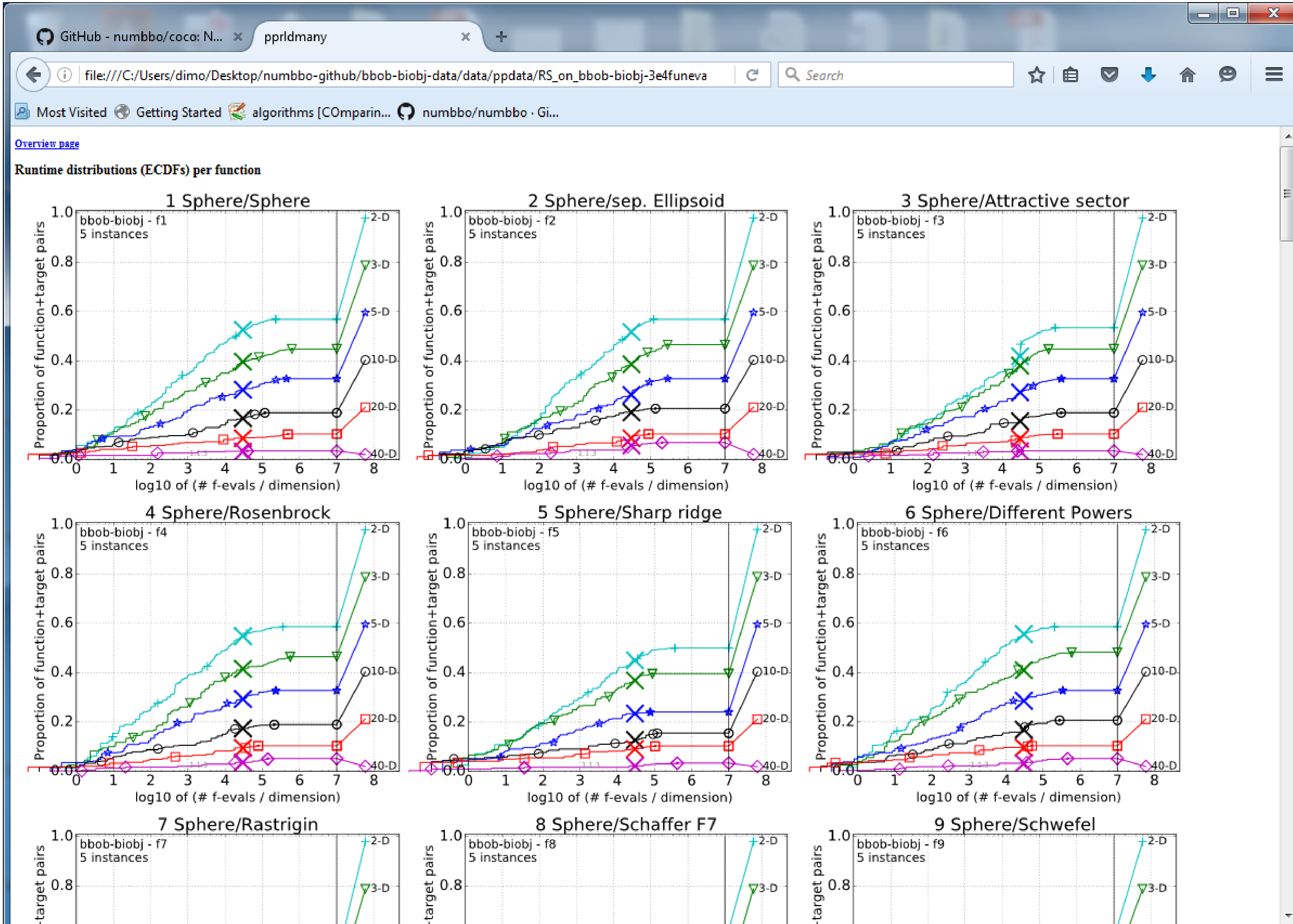
automatically generated results



automatically generated results



automatically generated results



so far:

data for about 165 algorithm variants

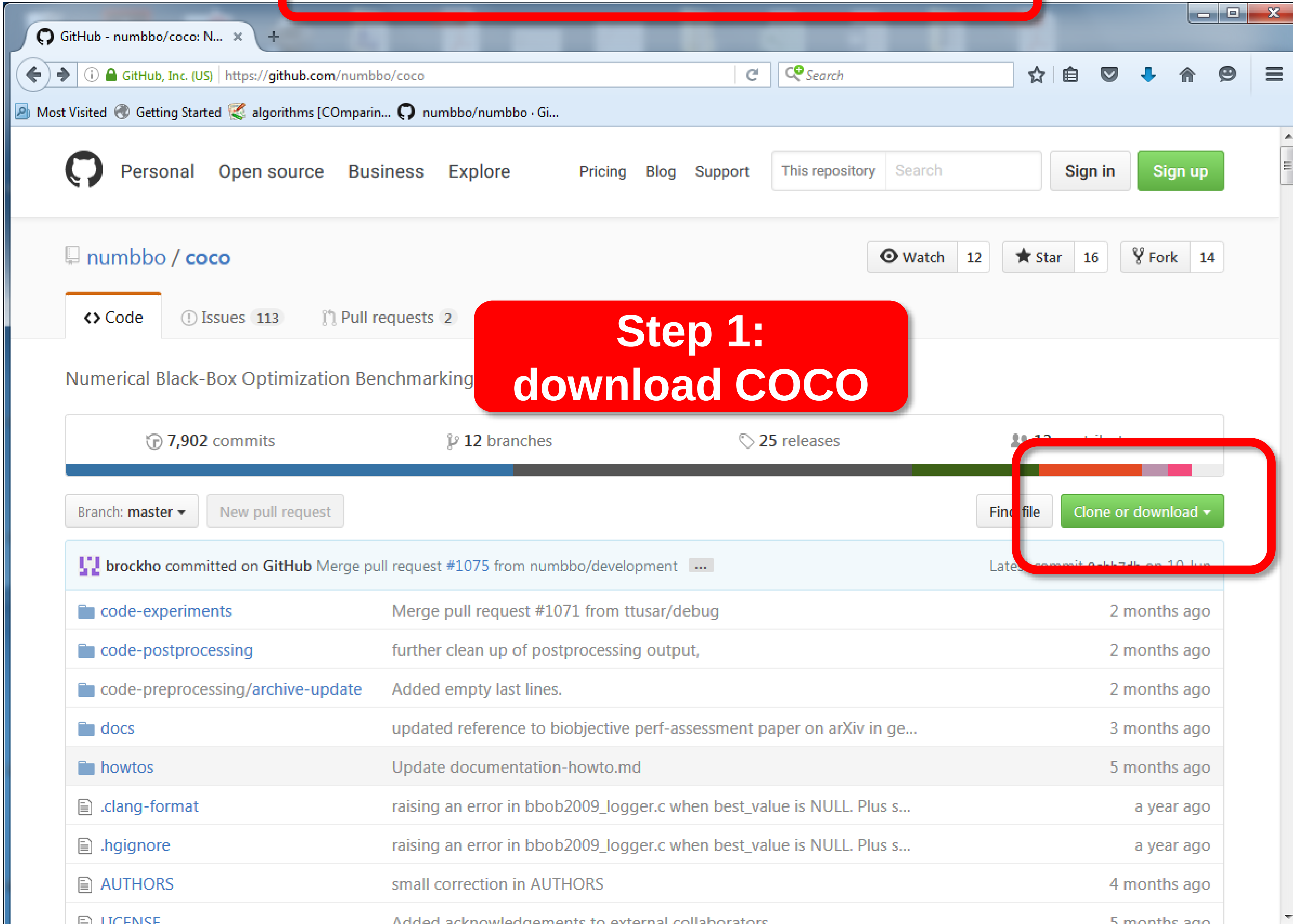
[in total on single- and multiobjective problems]

118 workshop papers

by 79 authors from 25 countries

Exercise (Part 1): Comparing Numerical Optimization Algorithms with COCO

https://github.com/numbbo/coco



Step 1: download COCO

https://github.com/numbbo/coco

GitHub - numbbo/coco: N...

https://github.com/numbbo/coco

corresponds to the master branch as linked above.

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

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

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

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

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

Step 2:
installation of post-processing

to (user-locally) install the post-processing. From

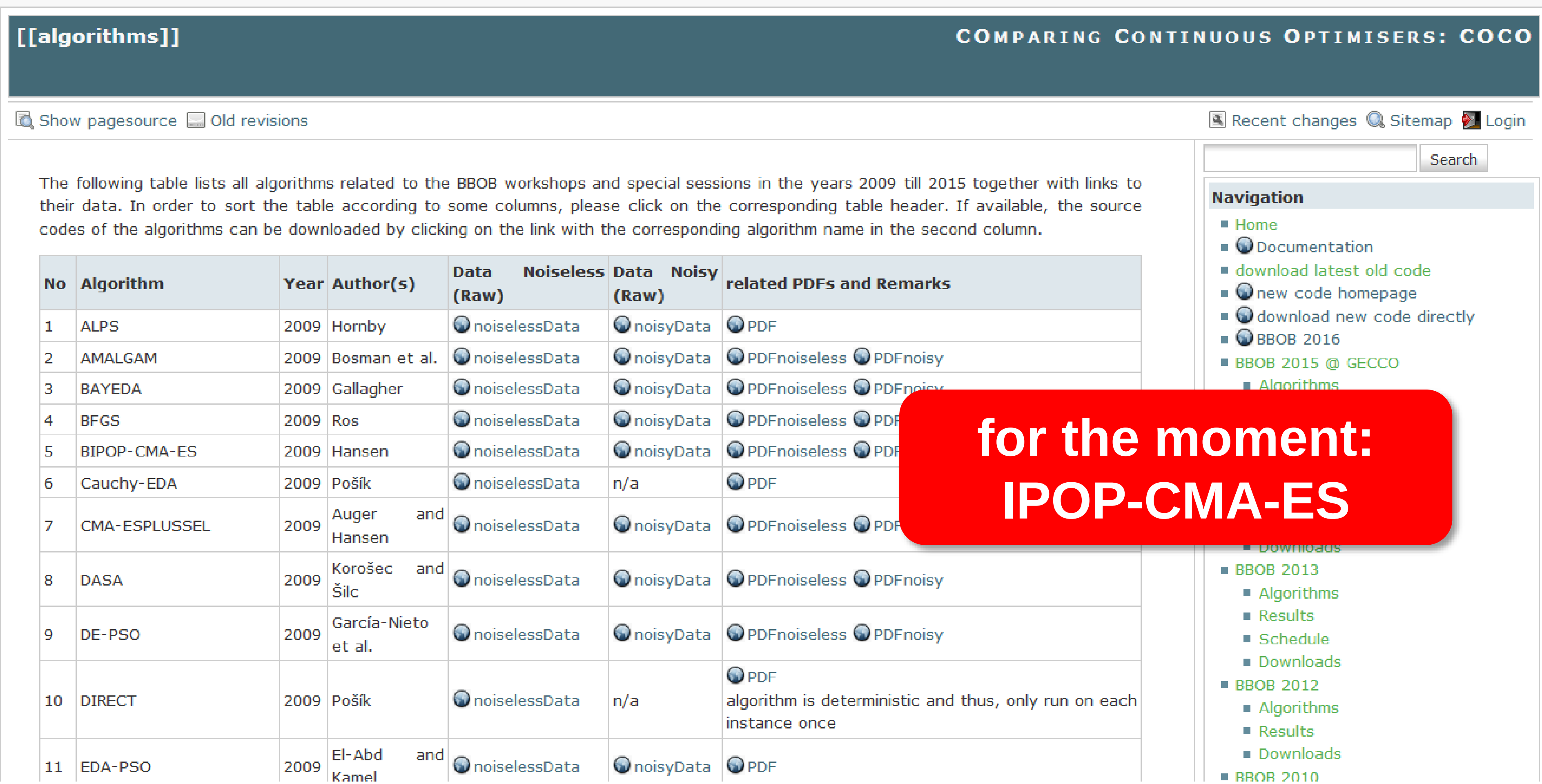
the builds to a new release.

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

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

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

Step 3: downloading data



[[algorithms]] — COMPARING CONTINUOUS OPTIMISERS: COCO

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

Search

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

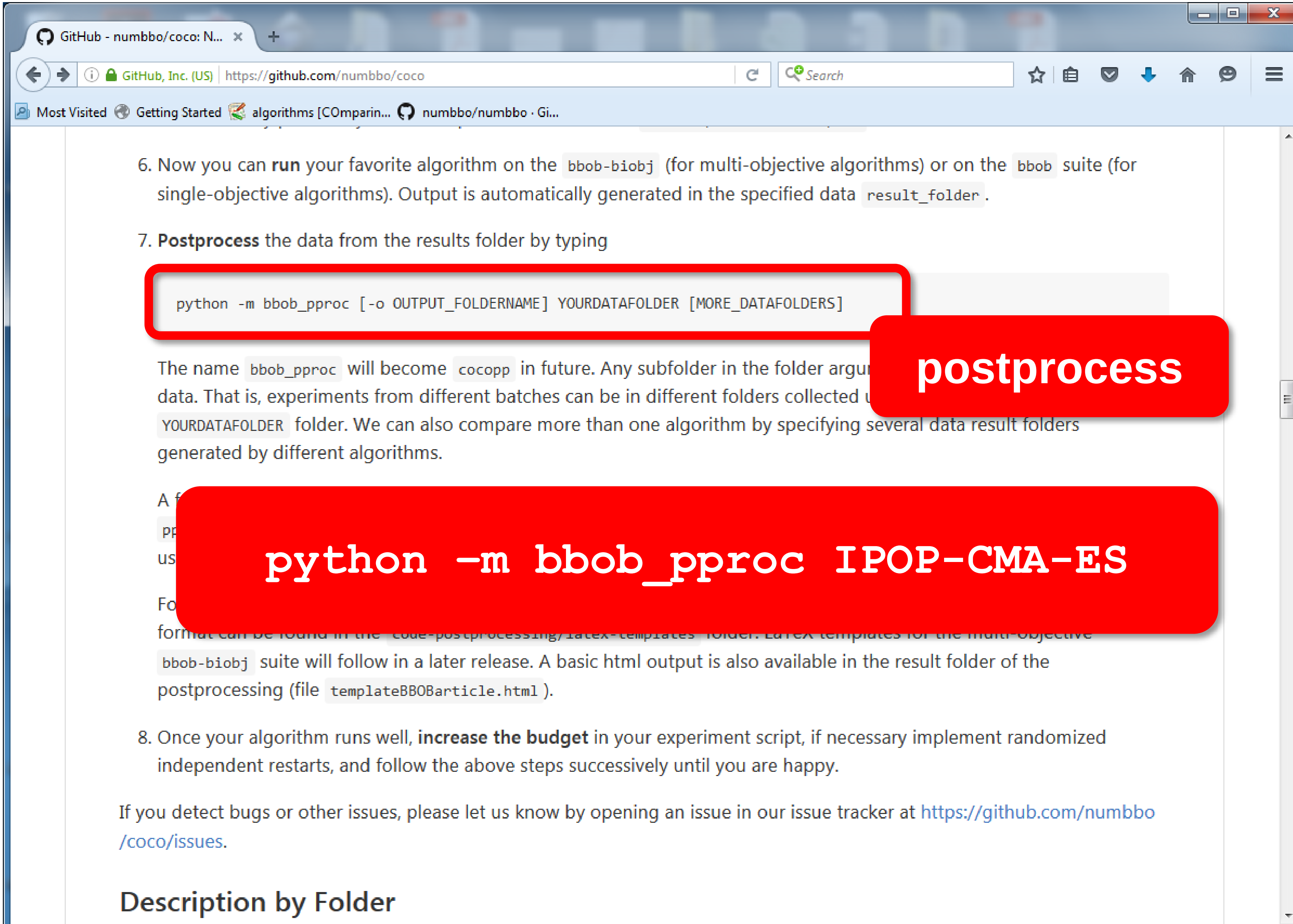
No	Algorithm	Year	Author(s)	Data Noiseless (Raw)	Data Noisy (Raw)	related PDFs and Remarks
1	ALPS	2009	Hornby	noiselessData	noisyData	PDF
2	AMALGAM	2009	Bosman et al.	noiselessData	noisyData	PDFnoiseless PDFnoisy
3	BAYEDA	2009	Gallagher	noiselessData	noisyData	PDFnoiseless PDFnoisy
4	BFGS	2009	Ros	noiselessData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

Navigation

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

for the moment: IPOP-CMA-ES

https://github.com/numbbo/coco



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

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

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

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

A f...
pp...
us...

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

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

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

Description by Folder

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

Available Test Suites in COCO

bbob	24 noiseless fcts	140+ algo data sets
bbob-noisy	30 noisy fcts	40+ algo data sets
bbob-biobj	55 bi-objective fcts	**new** in 2016 15 algo data sets

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

runtime or first hitting time is the prime candidate
(we don't have many choices anyway)

How Do We Measure Performance?

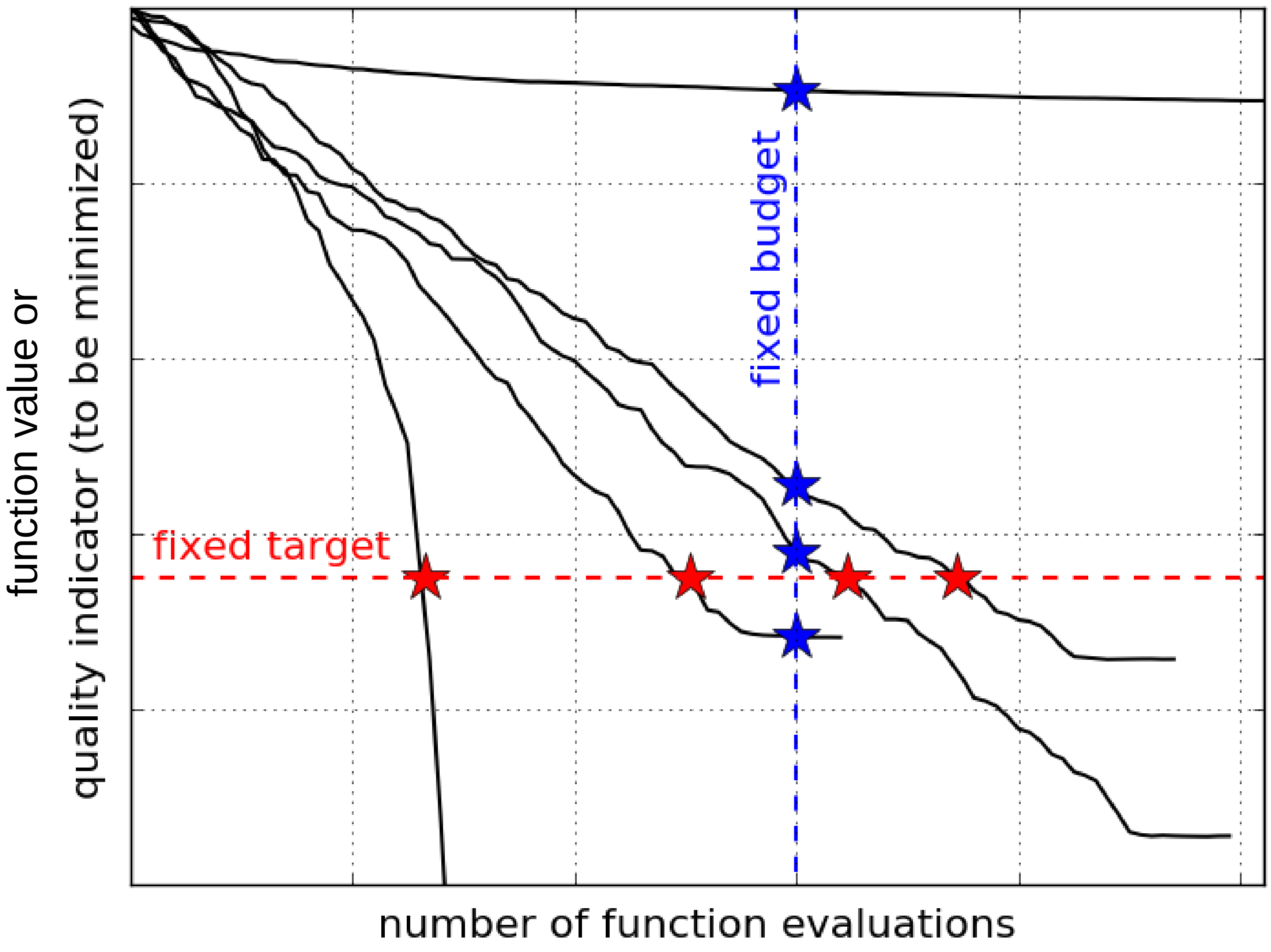
Two objectives:

- 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...

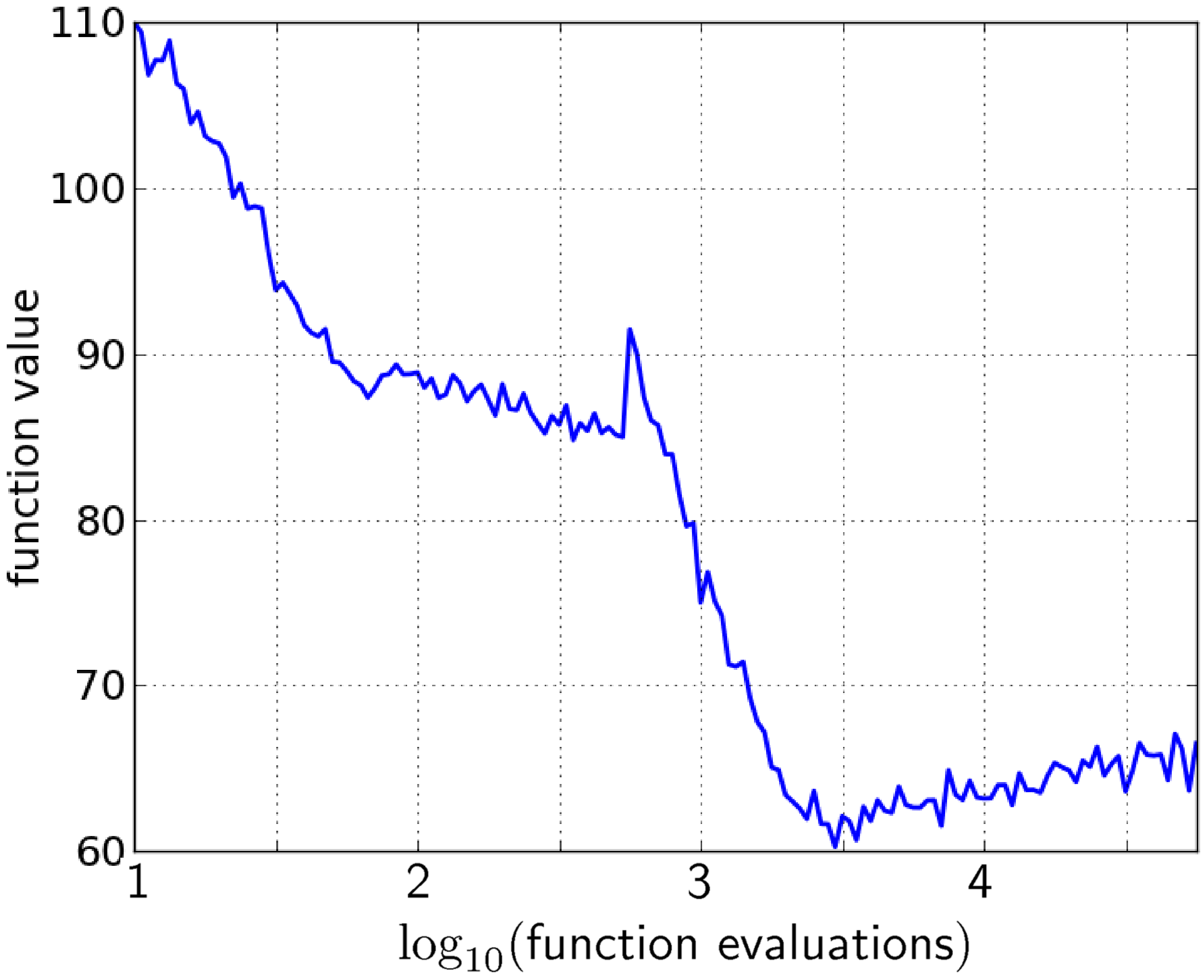


ECDF:

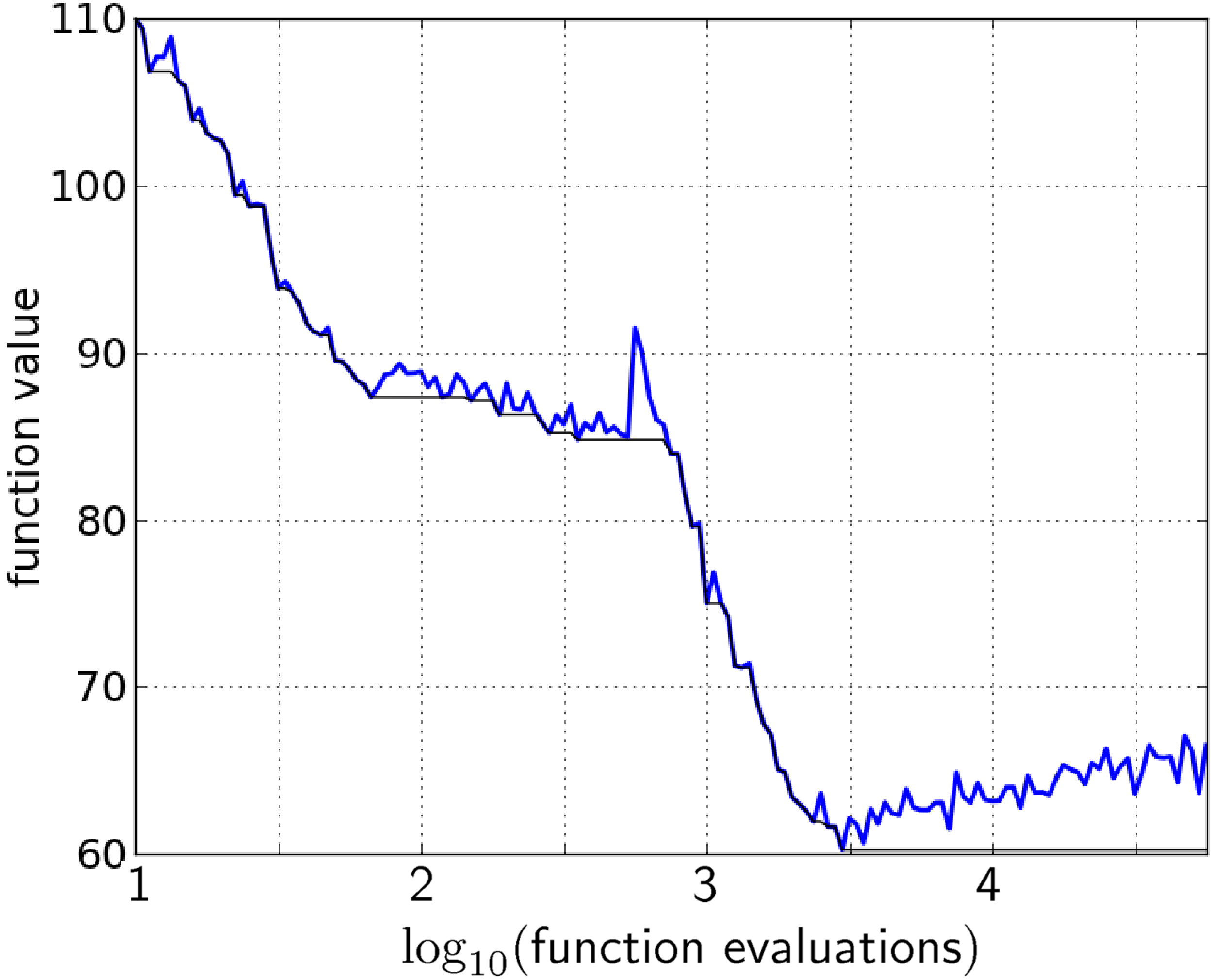
Empirical Cumulative Distribution Function of the Runtime

[aka data profile]

A Convergence Graph

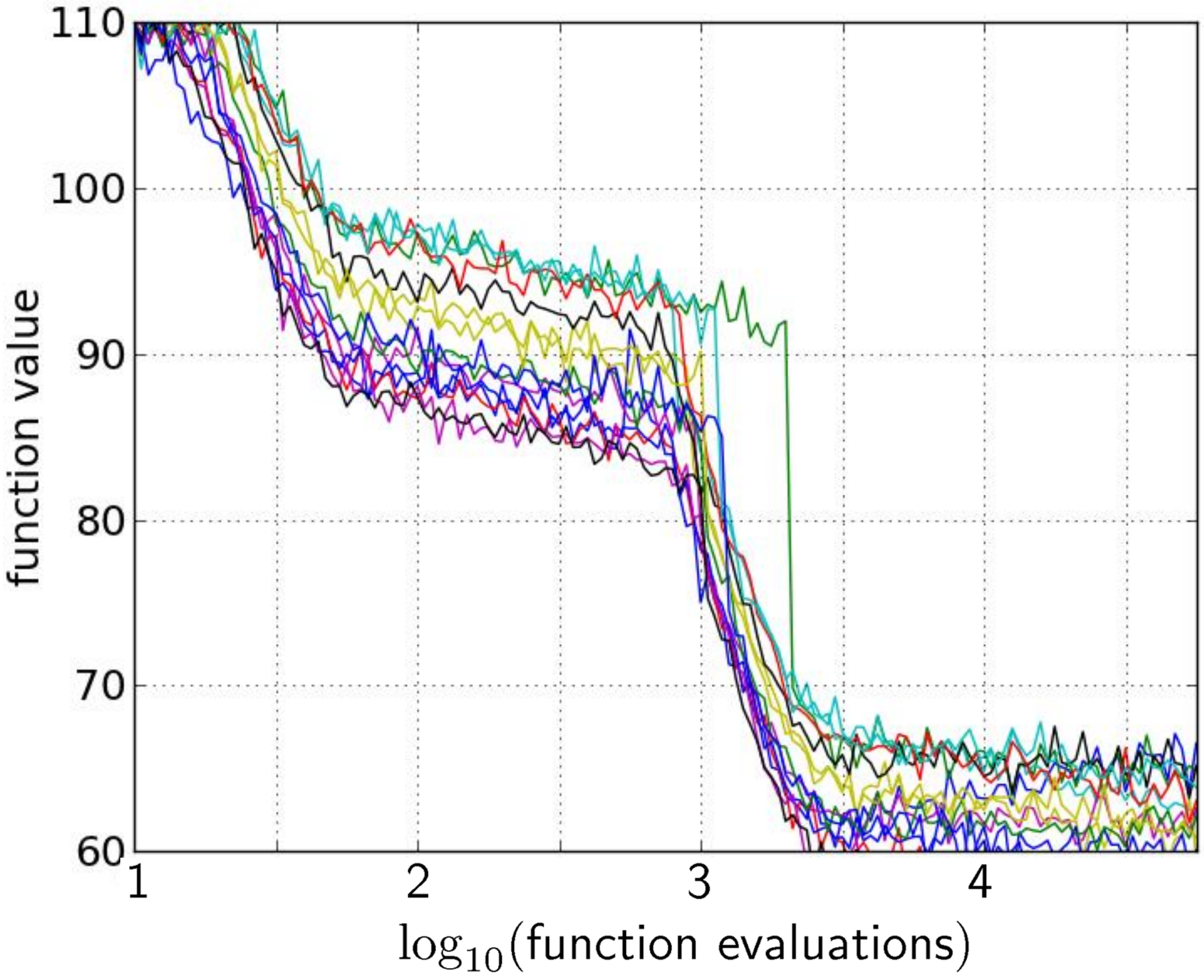


First Hitting Time is Monotonous

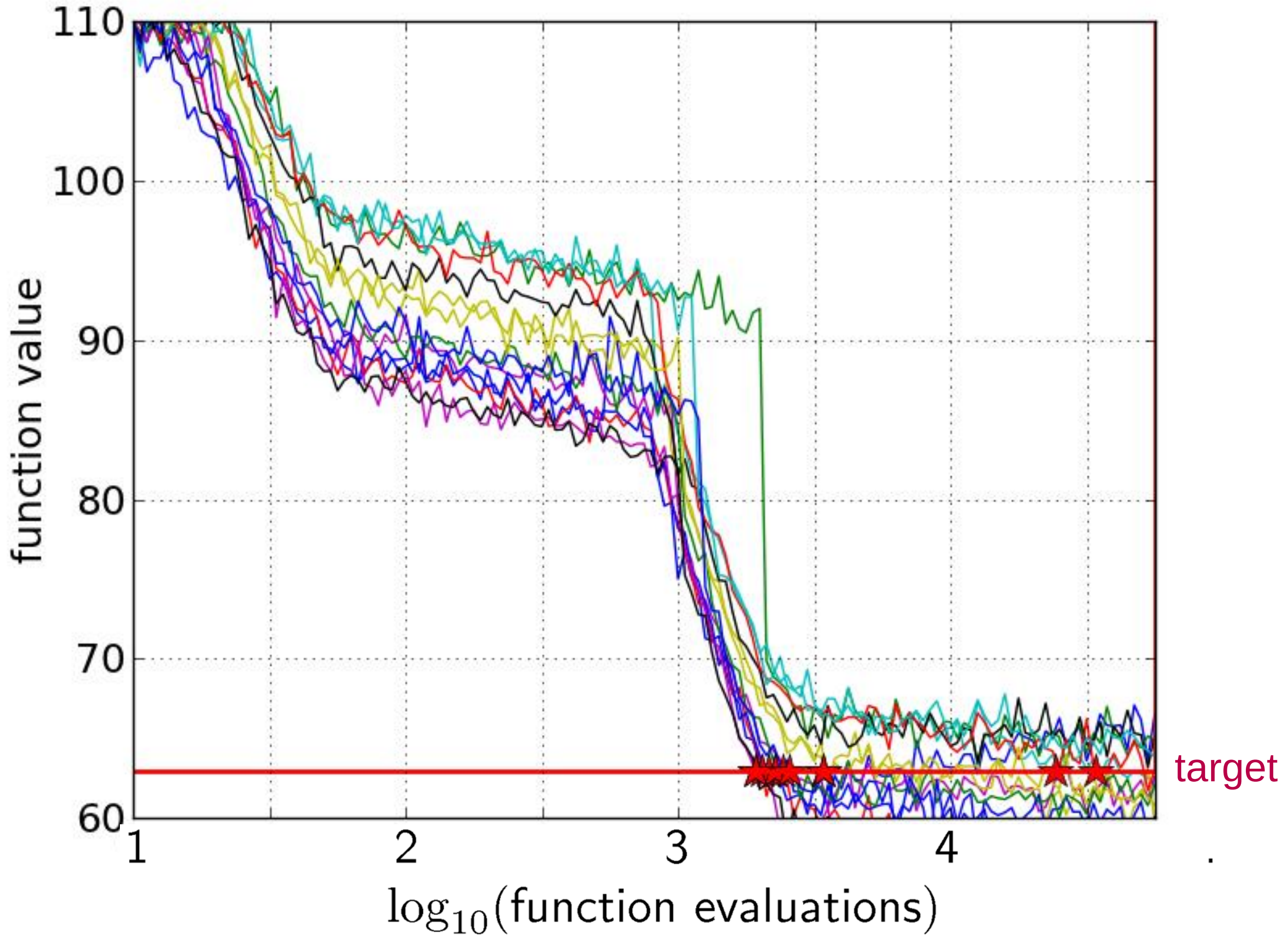


.

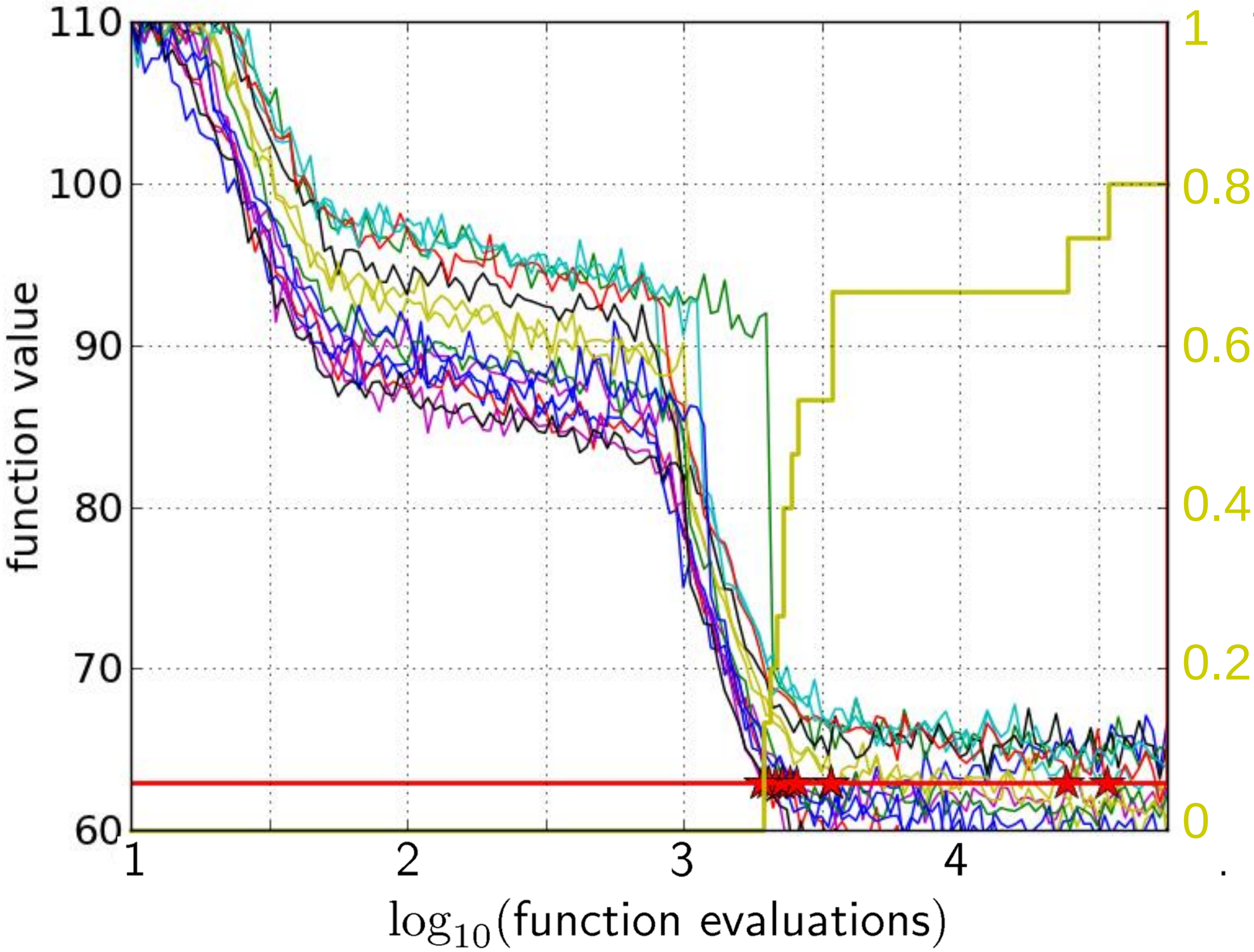
15 Runs


110
100
90
80
70
60
function value
1
2
3
4
log10(function evaluations)

15 Runs ≤ 15 Runtime Data Points



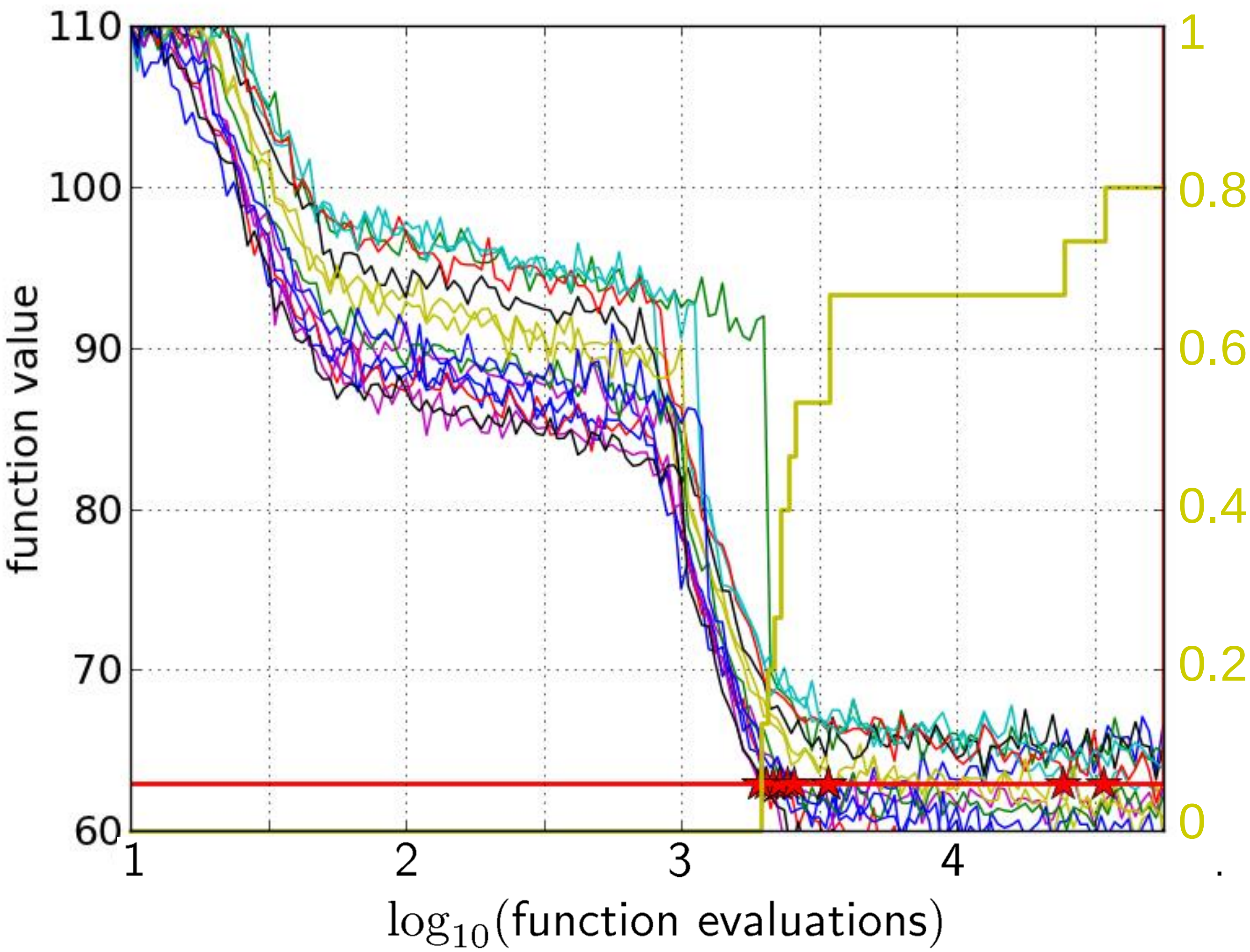
Empirical Cumulative Distribution



the ECDF of run lengths to reach the target

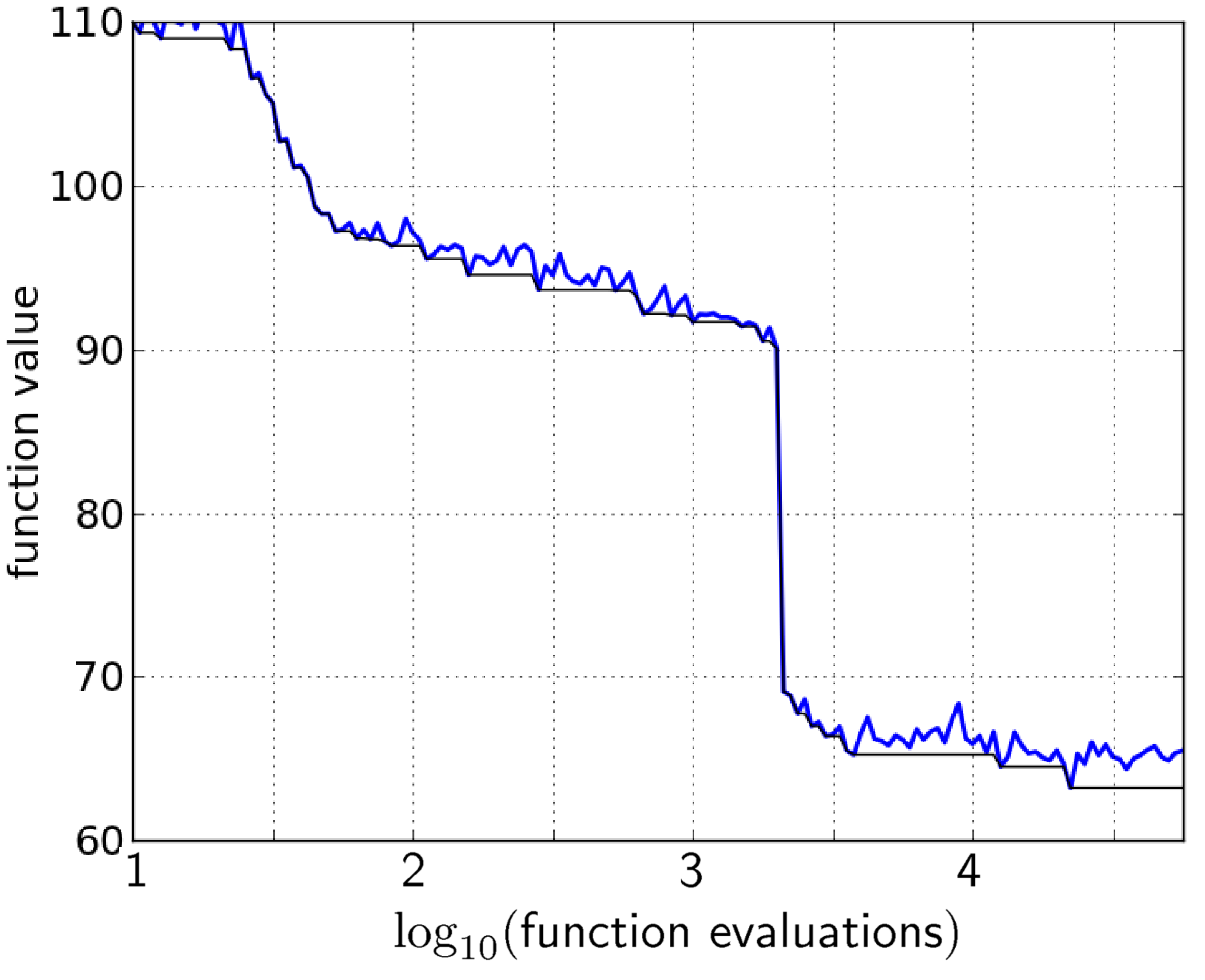
- has for each data point a vertical step of constant size
- displays for each x-value (budget) the count of observations to the left (first hitting times)

Empirical Cumulative Distribution



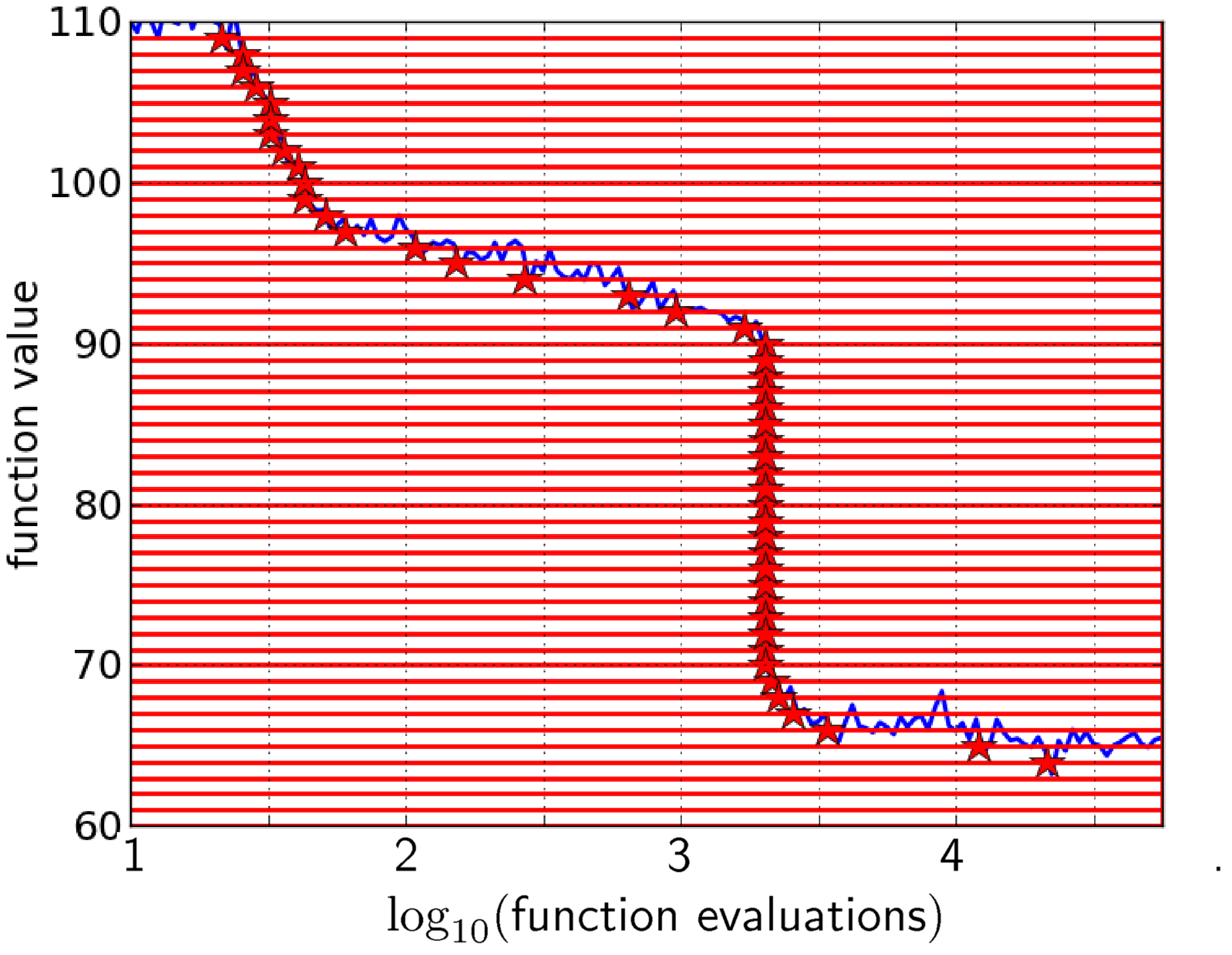
- interpretations possible:
- 80% of the runs reached the target
- e.g. 60% of the runs need between 2000 and 4000 evaluations

Reconstructing A Single Run



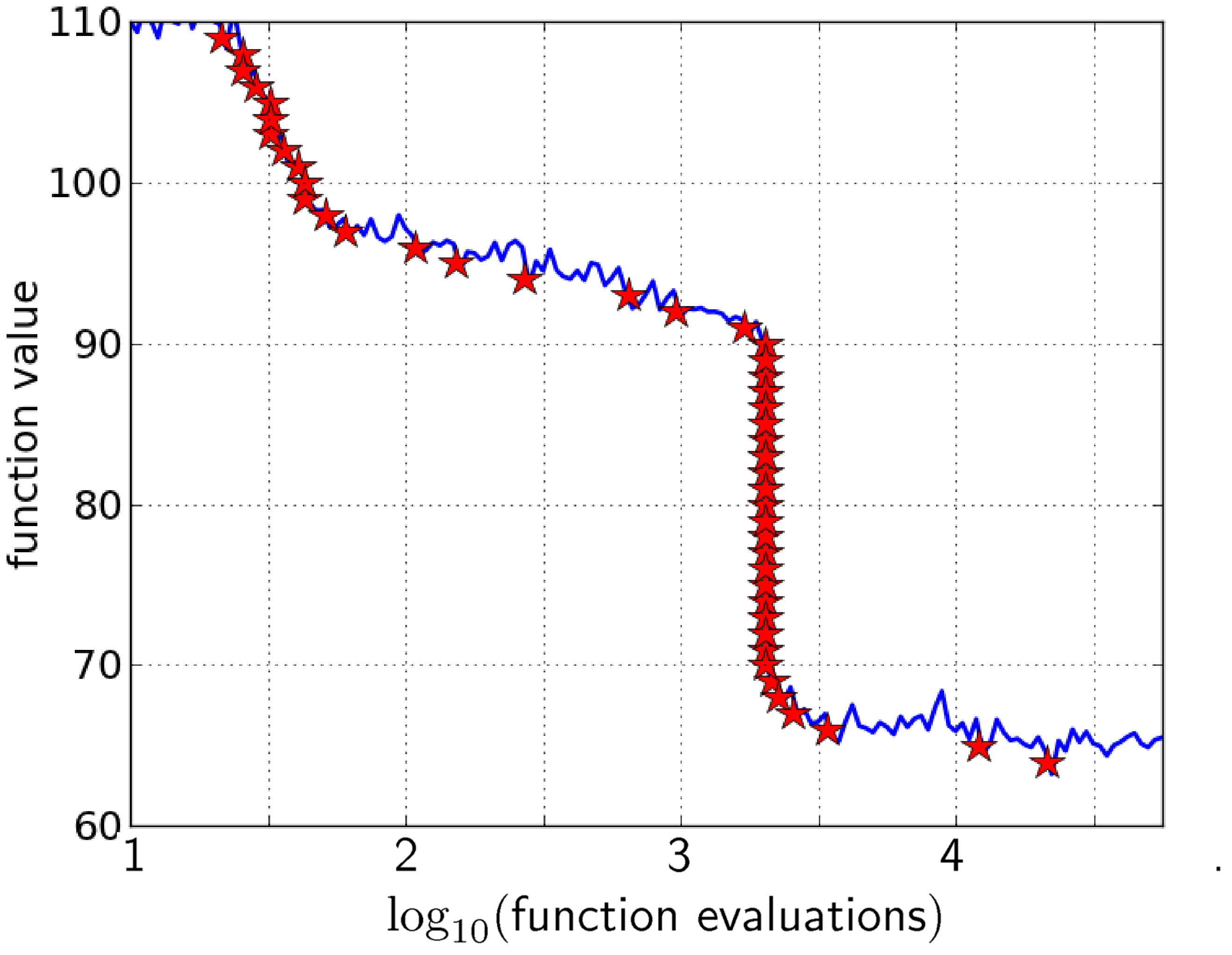
.

Reconstructing A Single Run

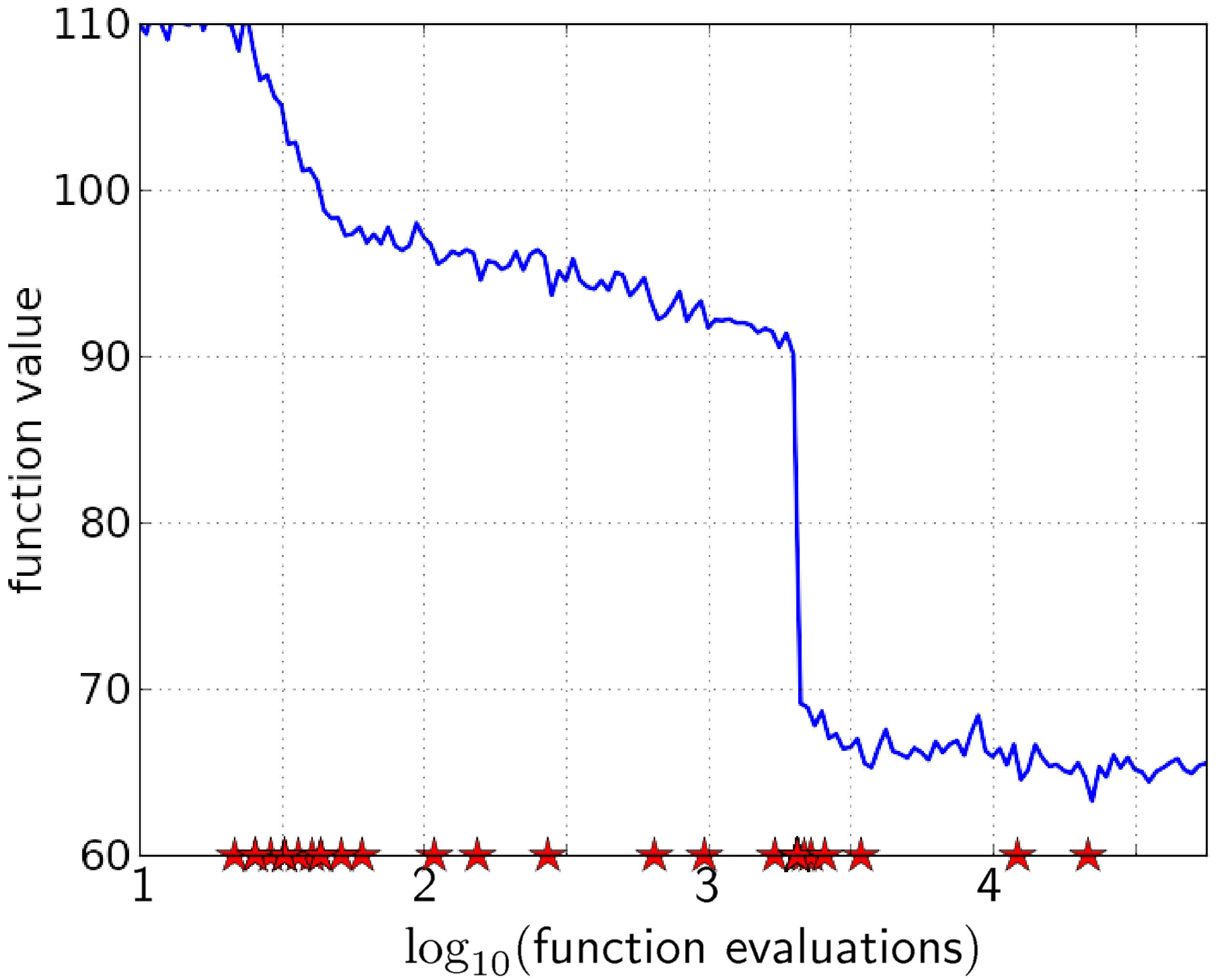


50 equally spaced targets

Reconstructing A Single Run

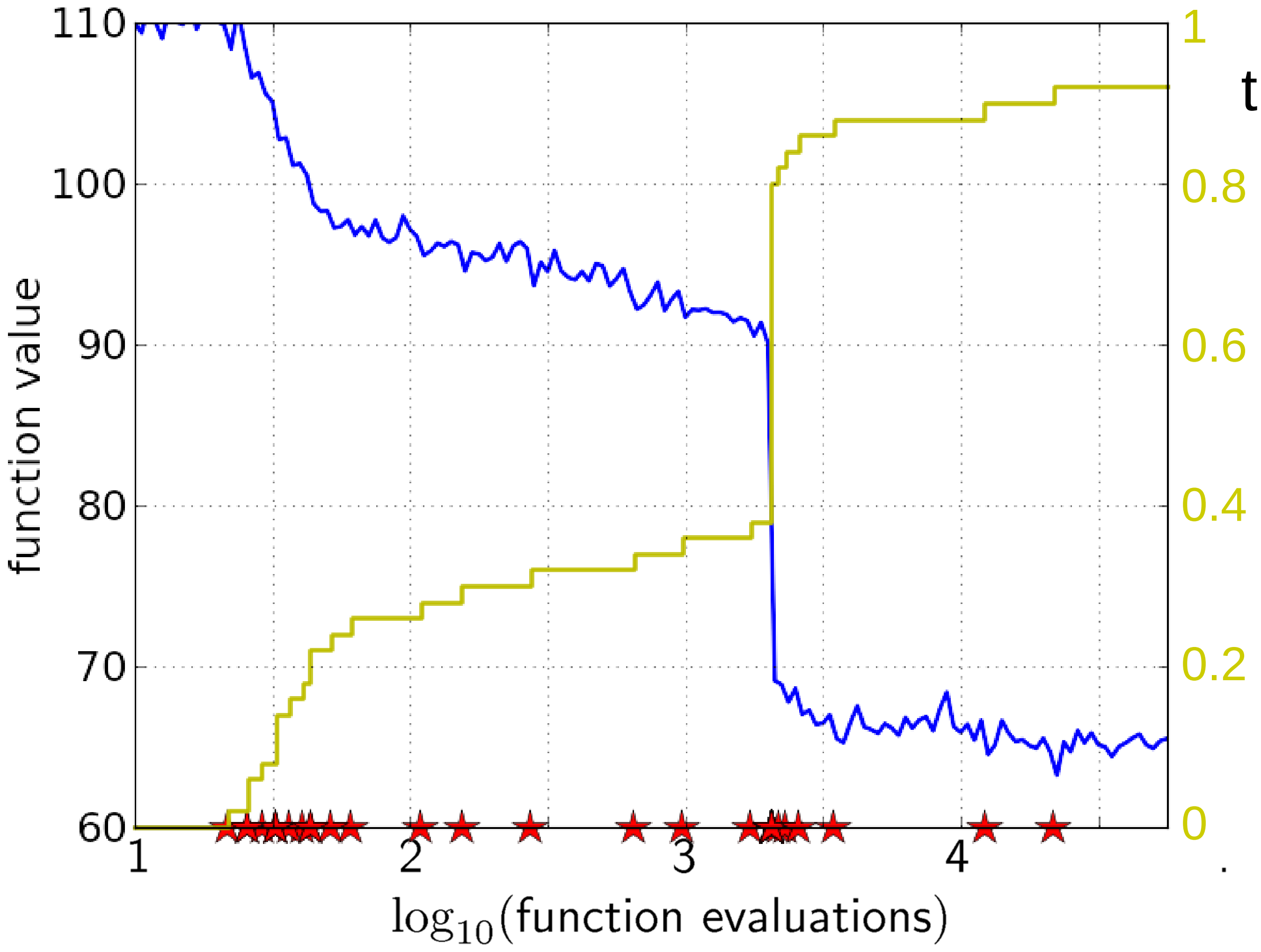


Reconstructing A Single Run



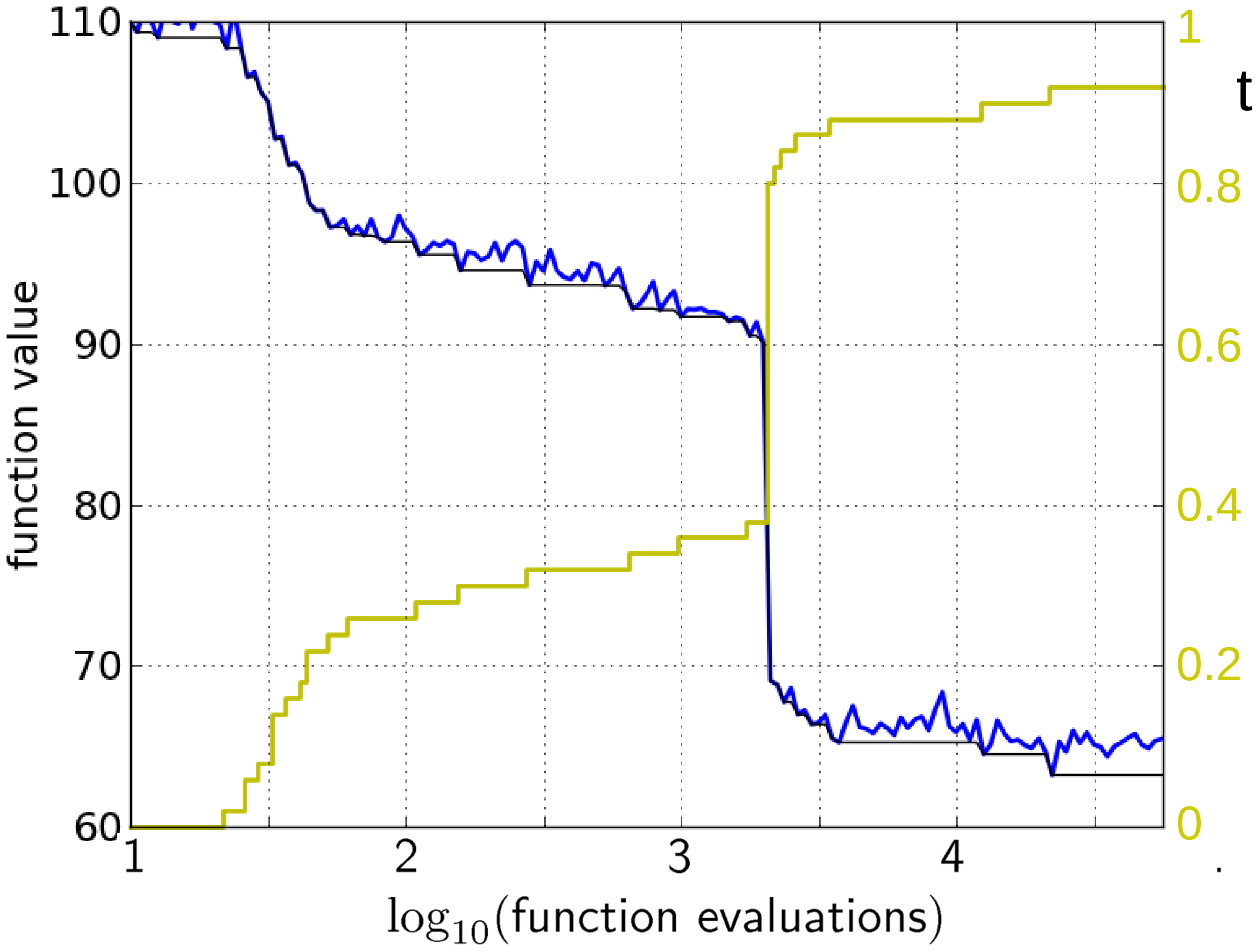
.

Reconstructing A Single Run



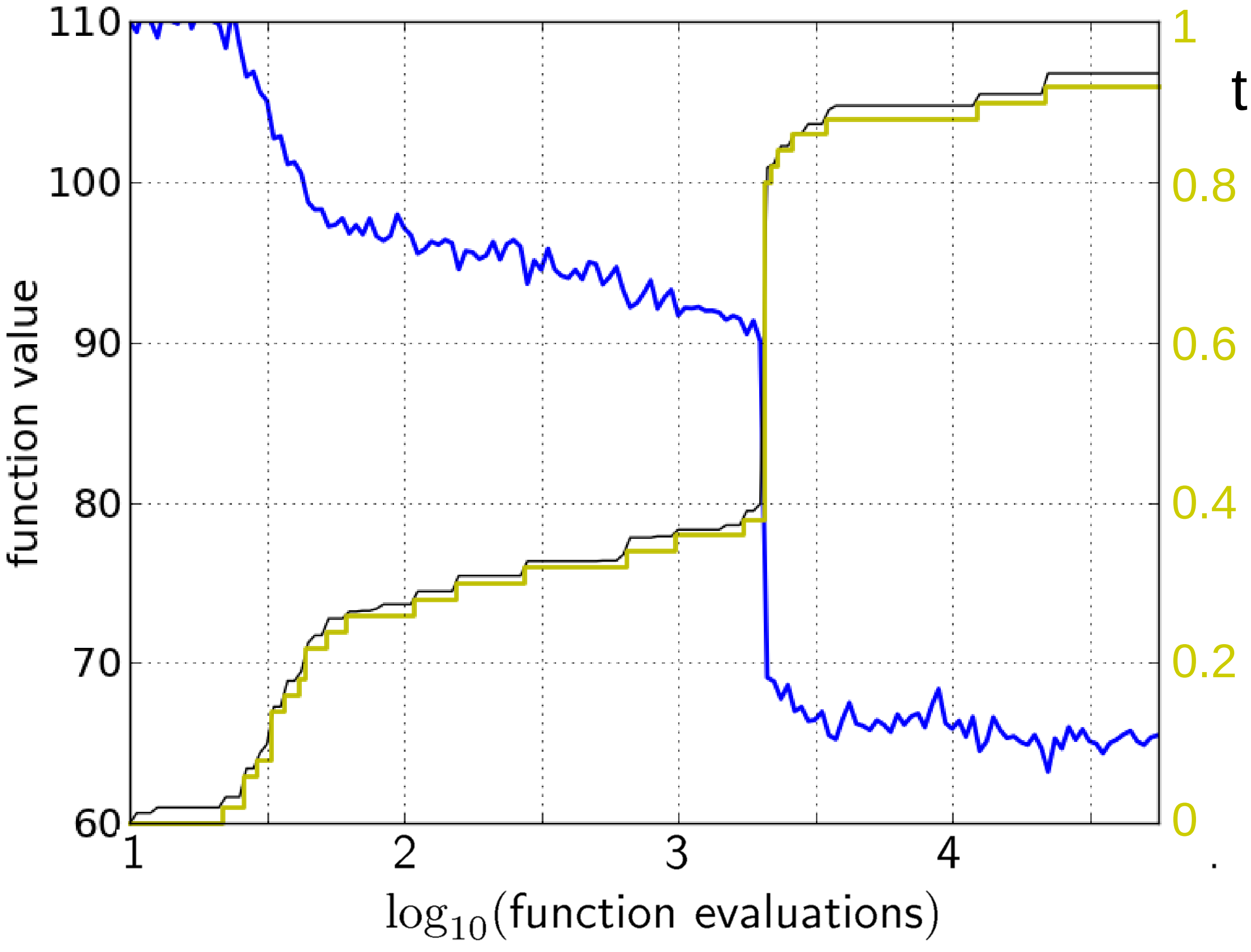
the empirical CDF makes a step for each star, is monotonous and displays for each budget the fraction of targets achieved within the budget

Reconstructing A Single Run



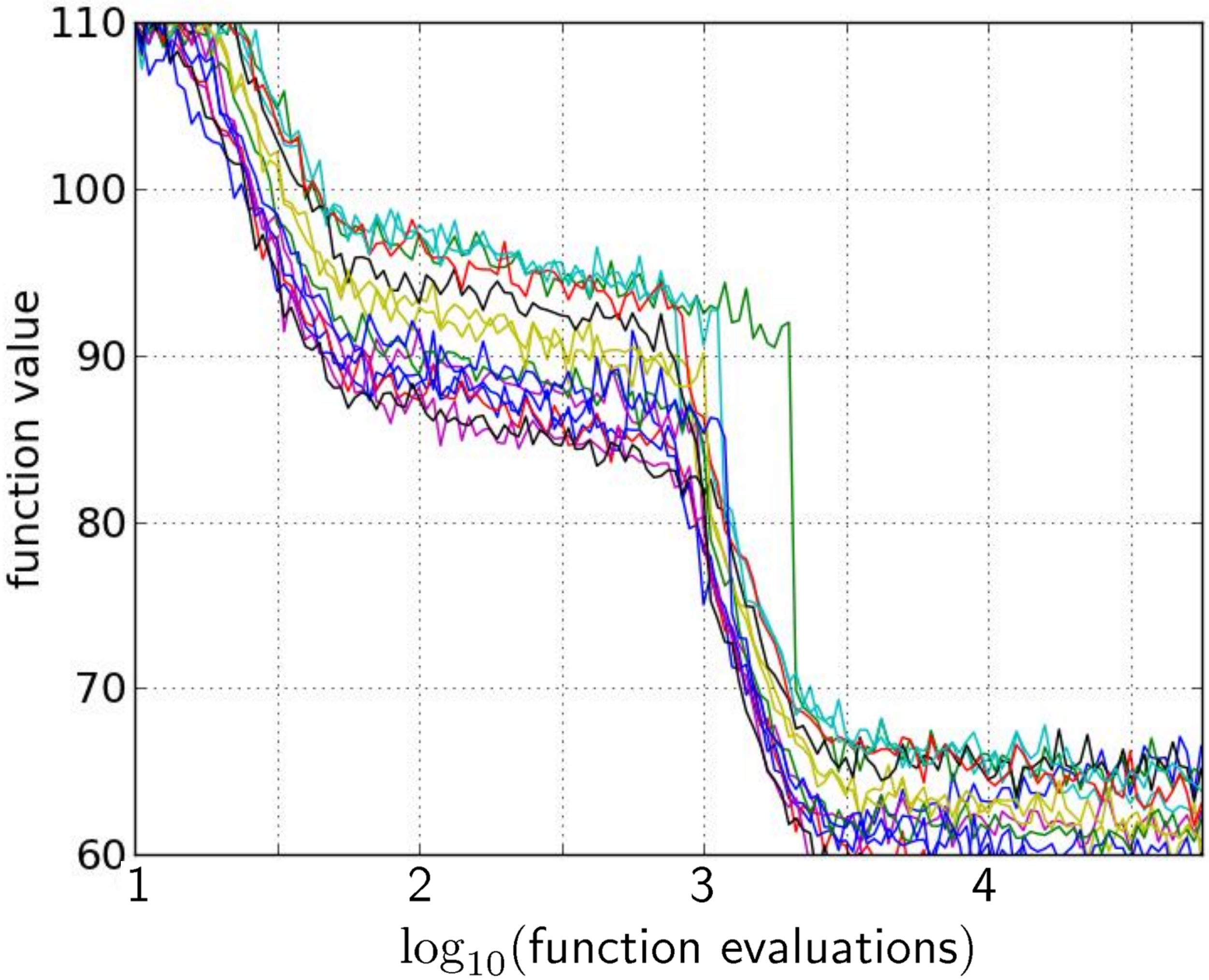
the ECDF recovers the monotonous graph, discretized and flipped

Reconstructing A Single Run



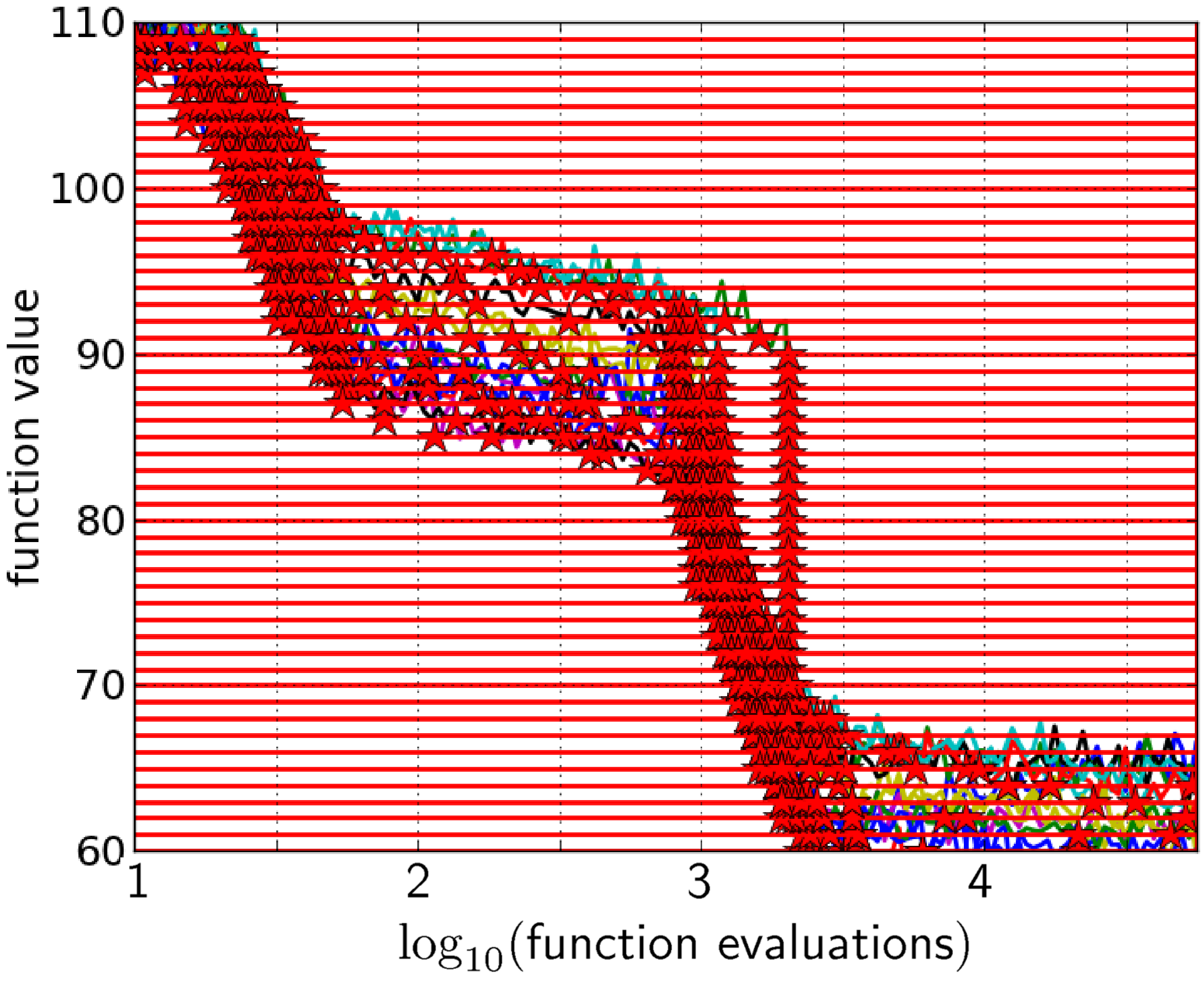
the ECDF recovers the monotonous graph, discretized and flipped

Aggregation



15 runs

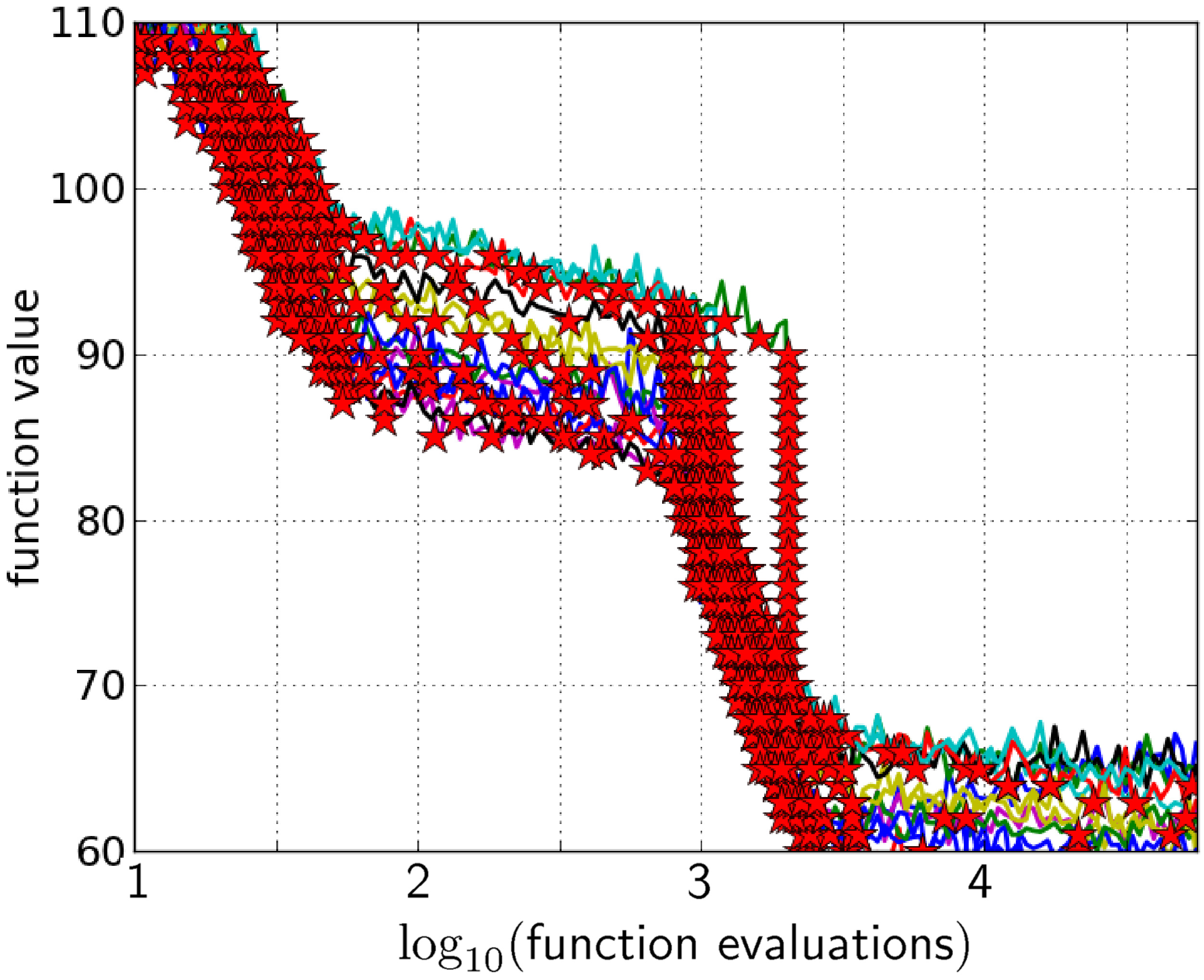
Aggregation



15 runs

50 targets

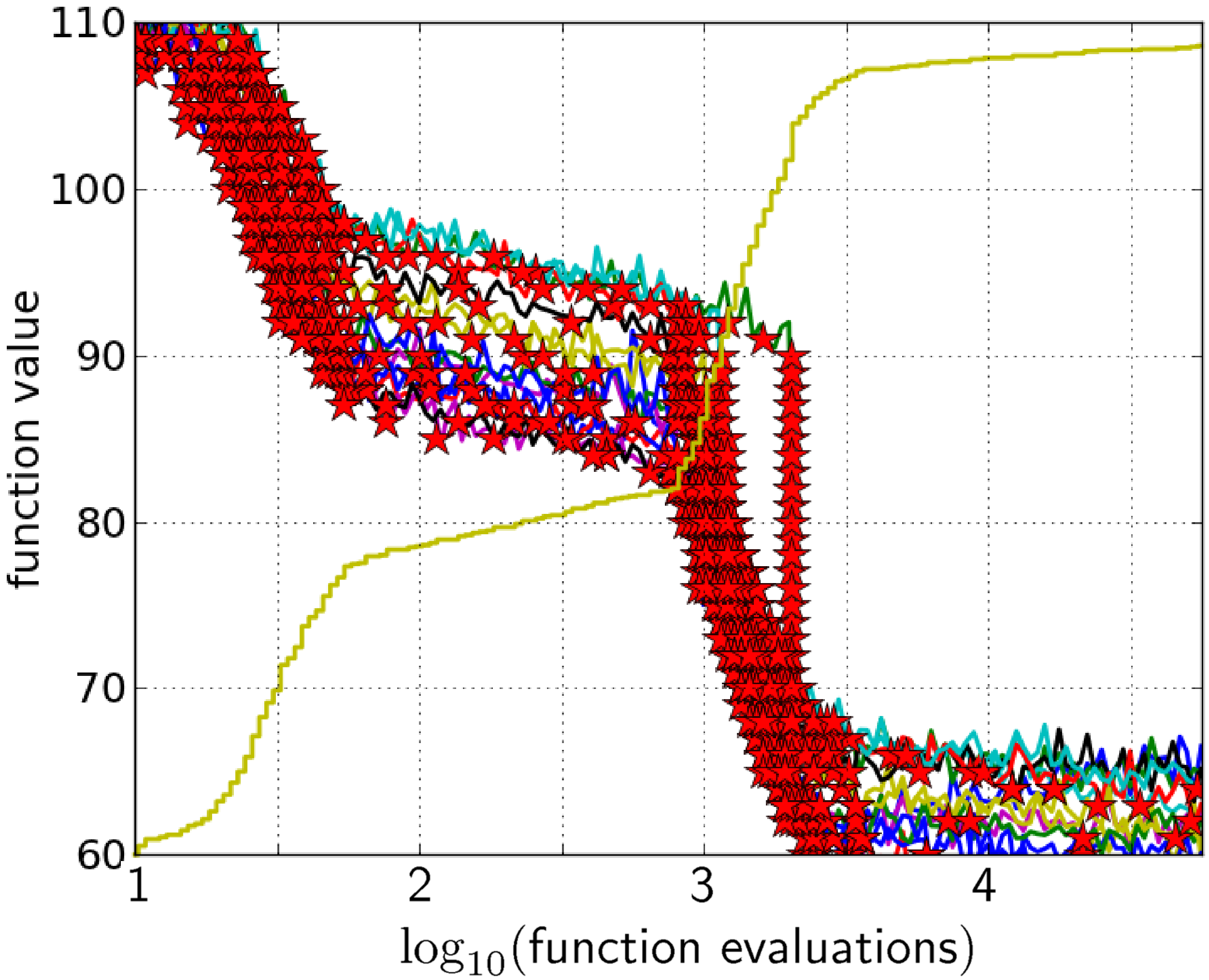
Aggregation



15 runs

50 targets

Aggregation

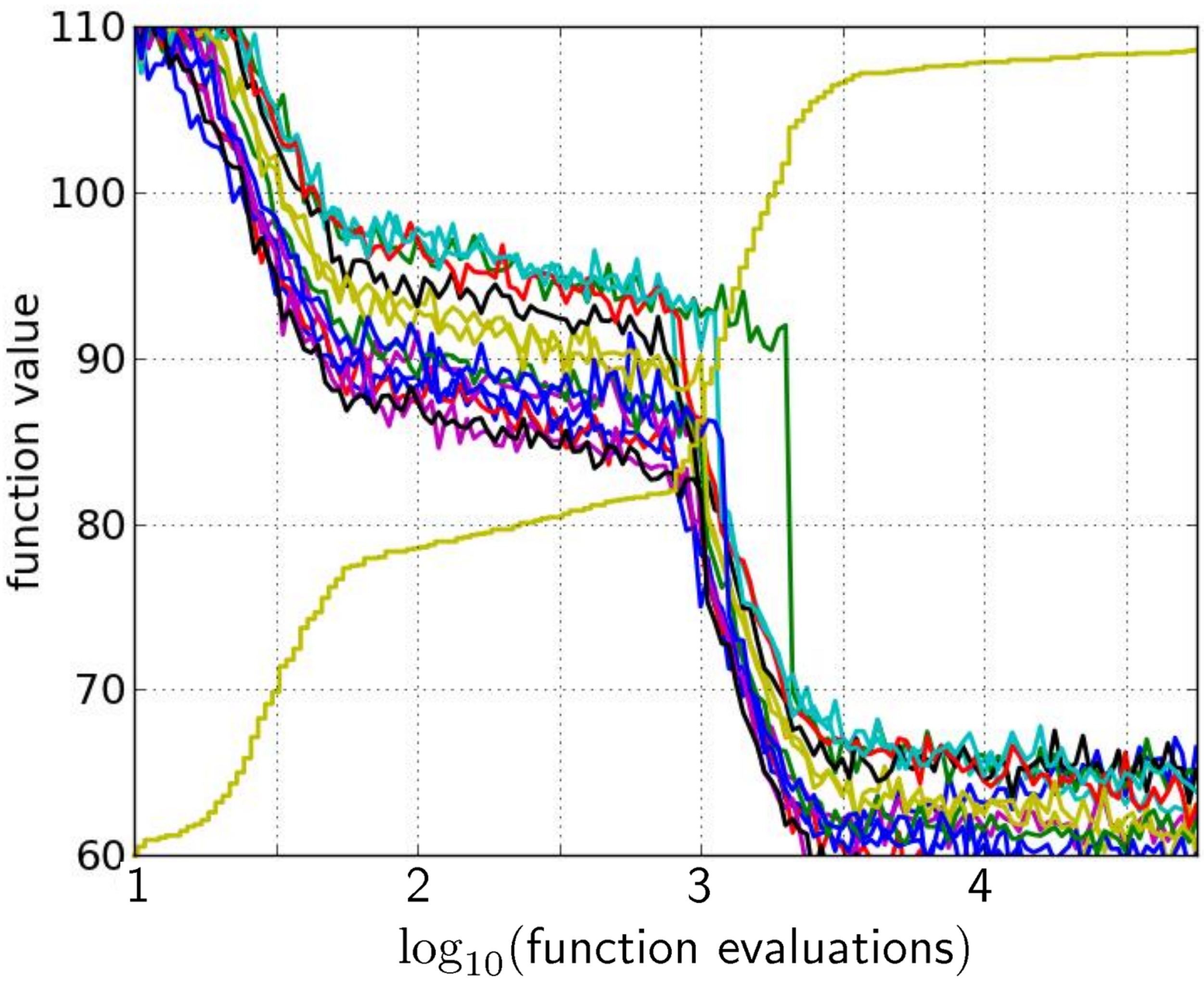


15 runs

50 targets

ECDF with 750 steps

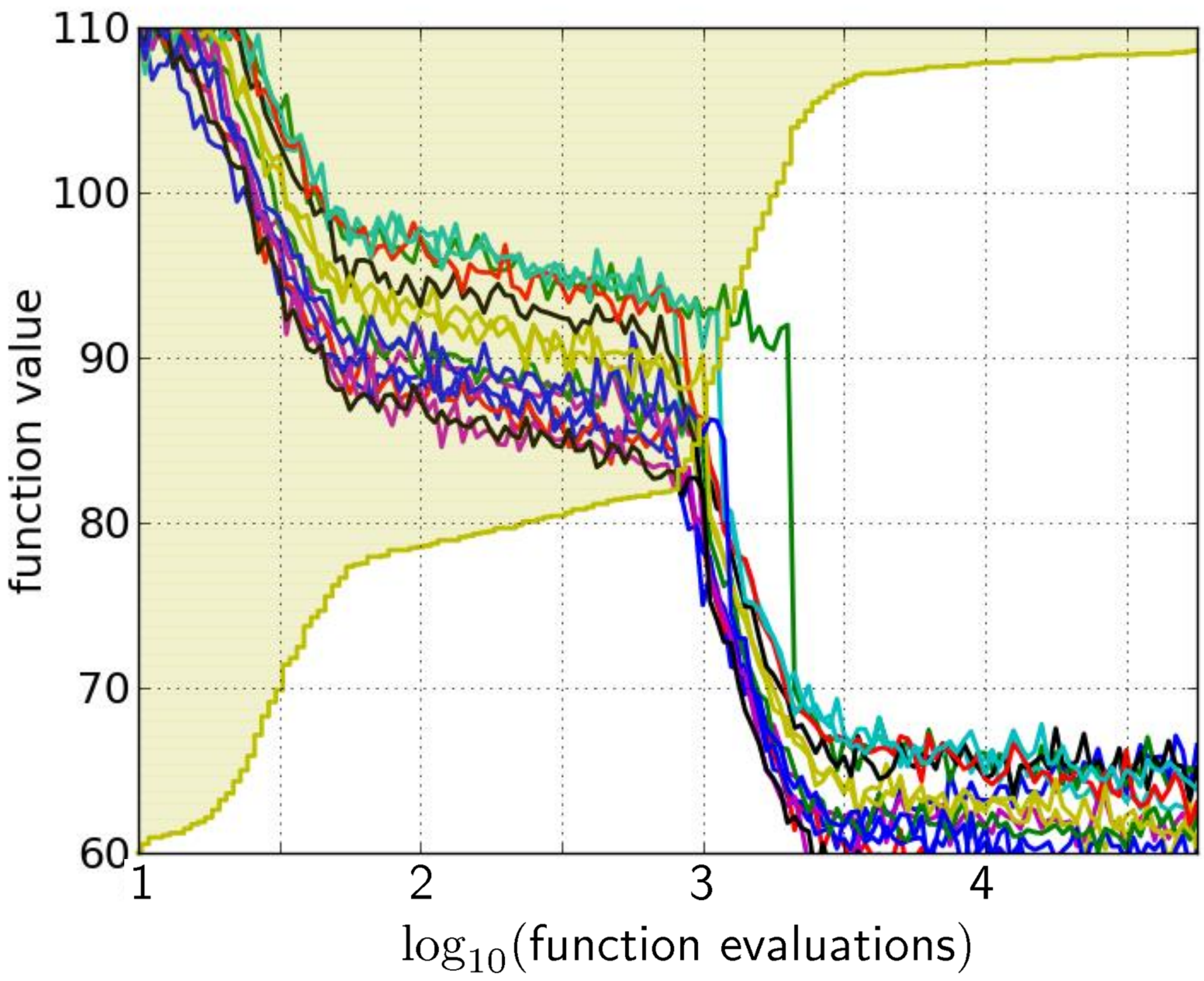
Aggregation



50 targets from 15 runs

...integrated in a single graph

Interpretation



50 targets from 15 runs integrated in a single graph

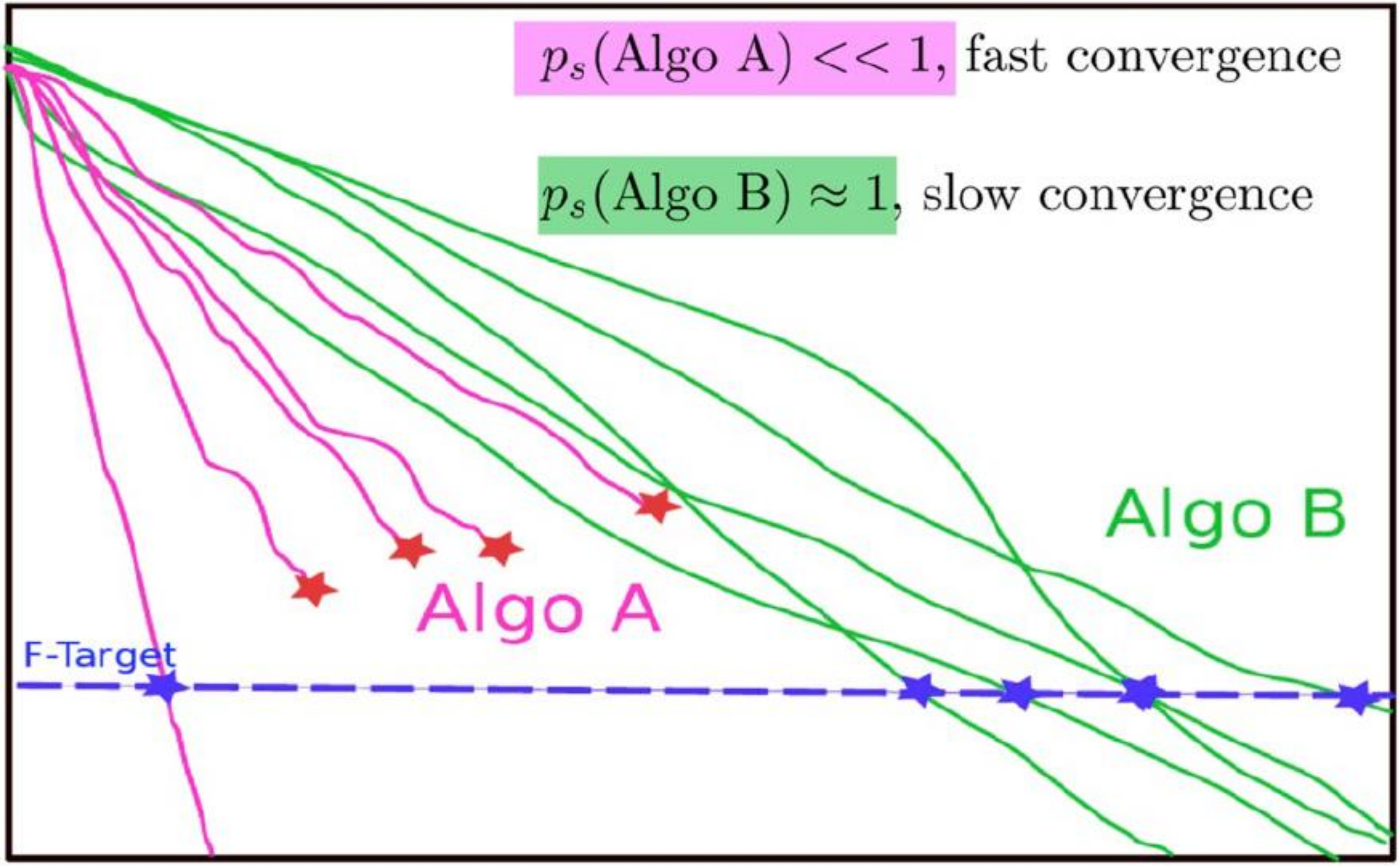
area over the ECDF curve

=

average log runtime

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

Fixed-target: Measuring Runtime

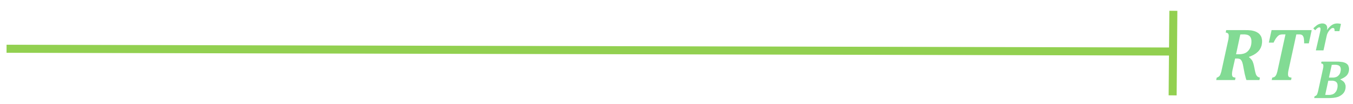


Fixed-target: Measuring Runtime

- Algo Restart A:

p_s(Algo Restart A) = 1

- Algo Restart B:



p_s(Algo Restart B) = 1

Fixed-target: Measuring Runtime

- Expected running time of the restarted algorithm:

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

- Estimator average running time (aRT):

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

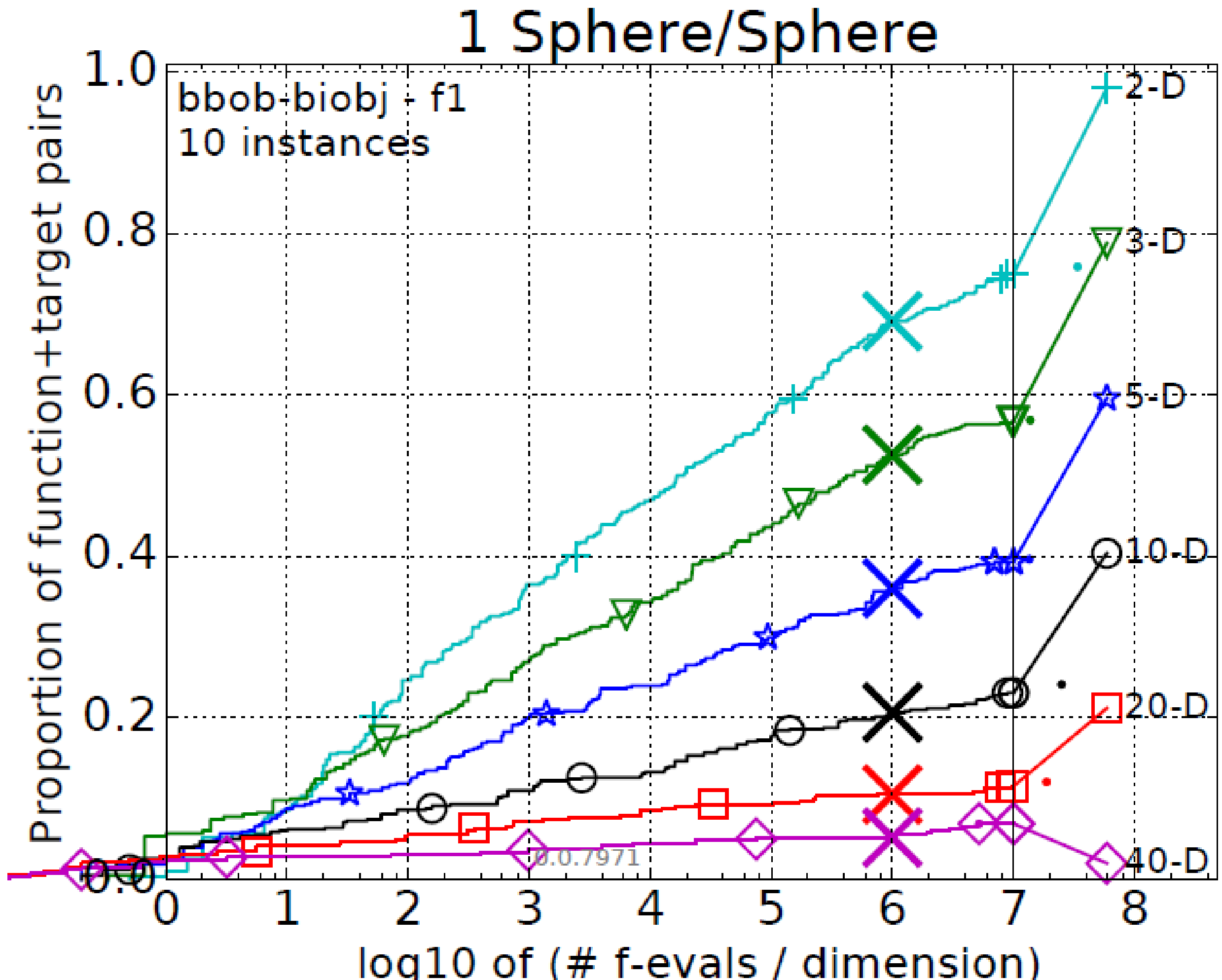
$$\widehat{RT_{unsucc}} = \text{Average evals of unsuccessful runs}$$

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

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

ECDFs with Simulated Restarts

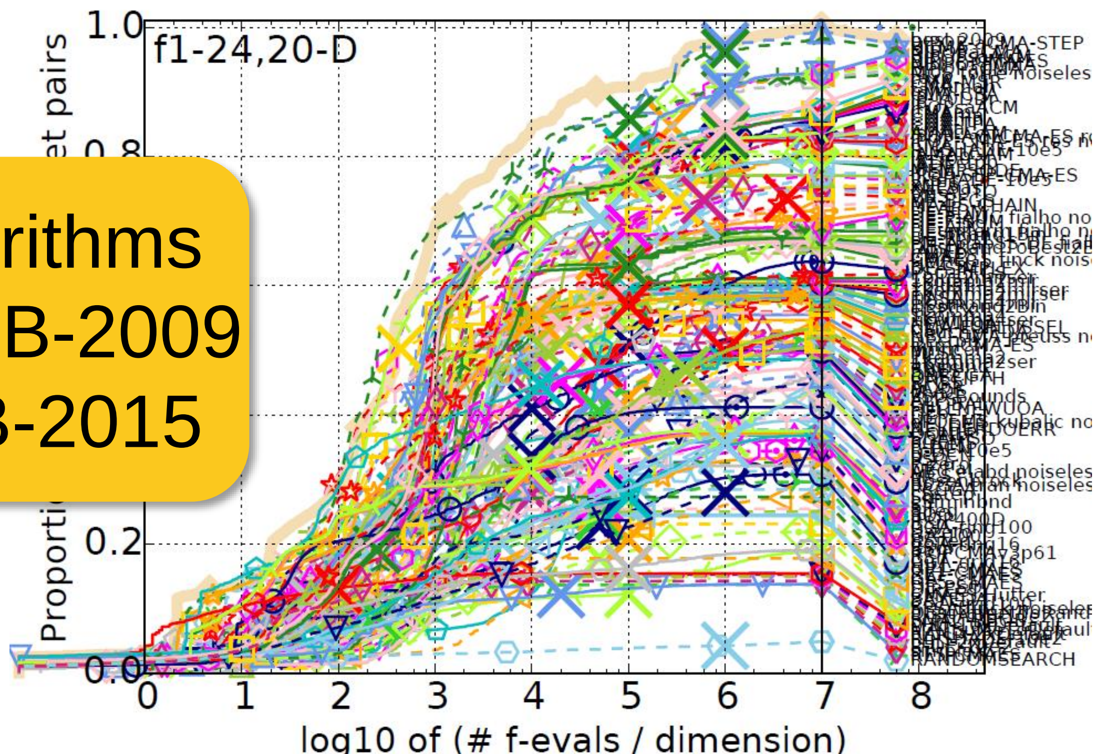
What we typically plot are ECDFs of the simulated restarted algorithms:



Worth to Note: ECDFs in COCO

In COCO, ECDF graphs

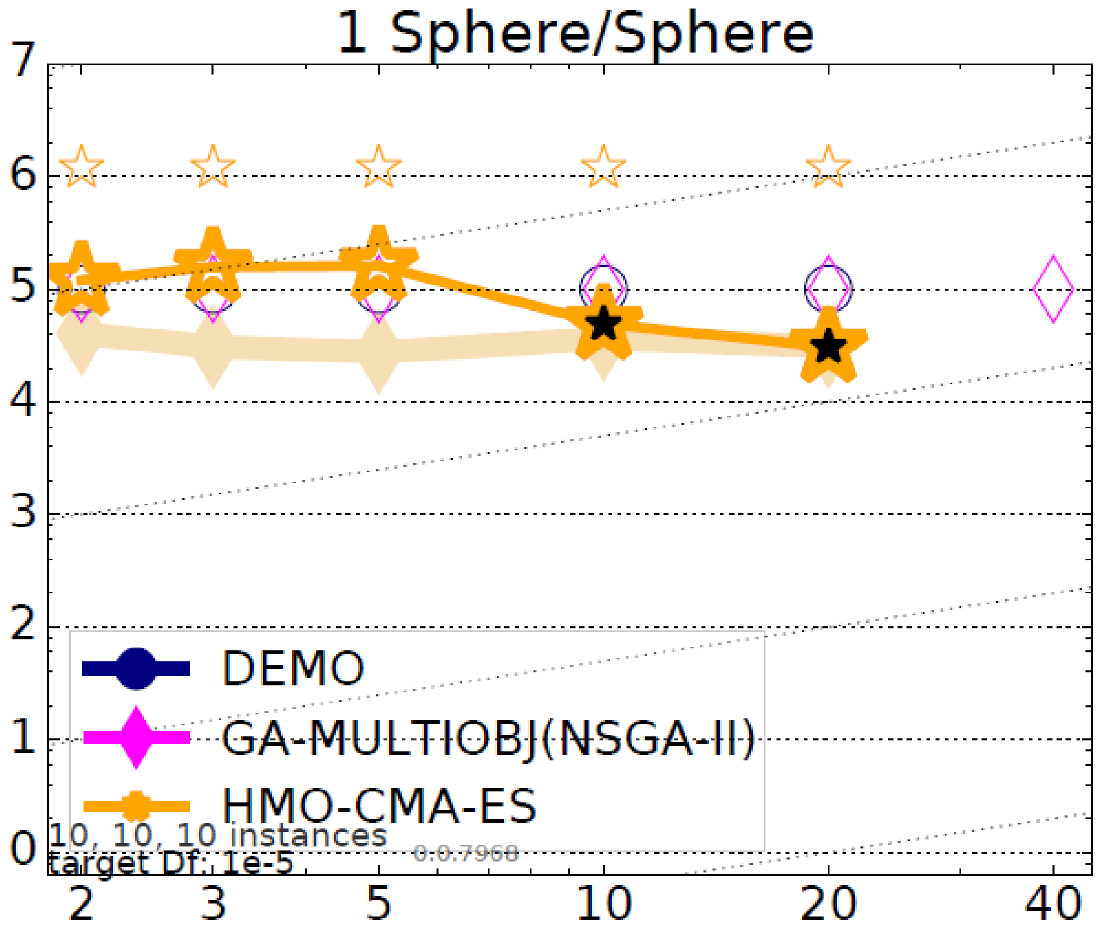
- never aggregate over dimension
  - but often over targets and functions
- can show data of more than 1 algorithm at a time



150 algorithms
from BBOB-2009
till BBOB-2015

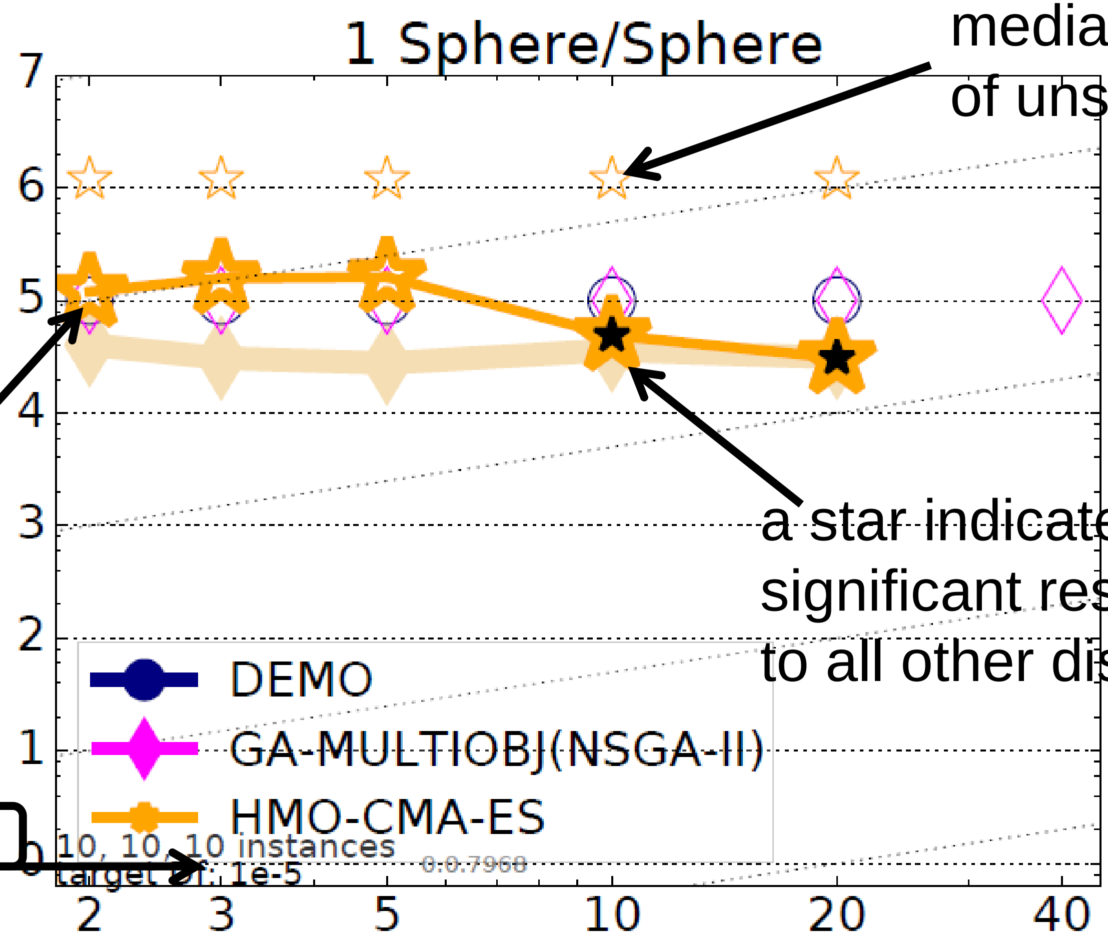
Another Interesting Plot...

...comparing aRT values over several algorithms



Another Interesting Plot...

...comparing aRT values over several algorithms



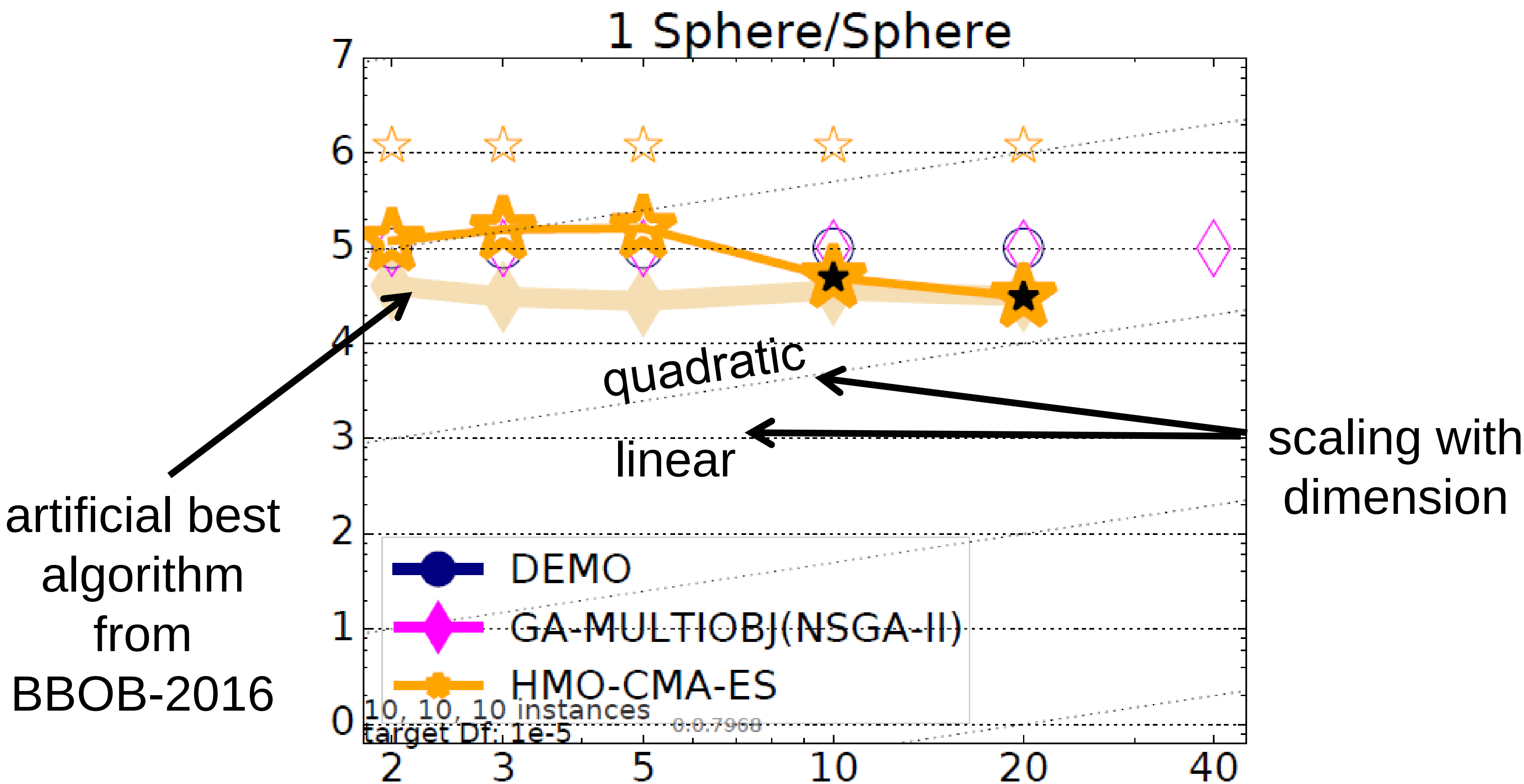
median runlength of unsuccessful runs

a star indicates statistically significant results compared to all other displayed algos

aRT value [if $< \infty$] to reach given target precision

Another Interesting Plot...

...comparing aRT values over several algorithms

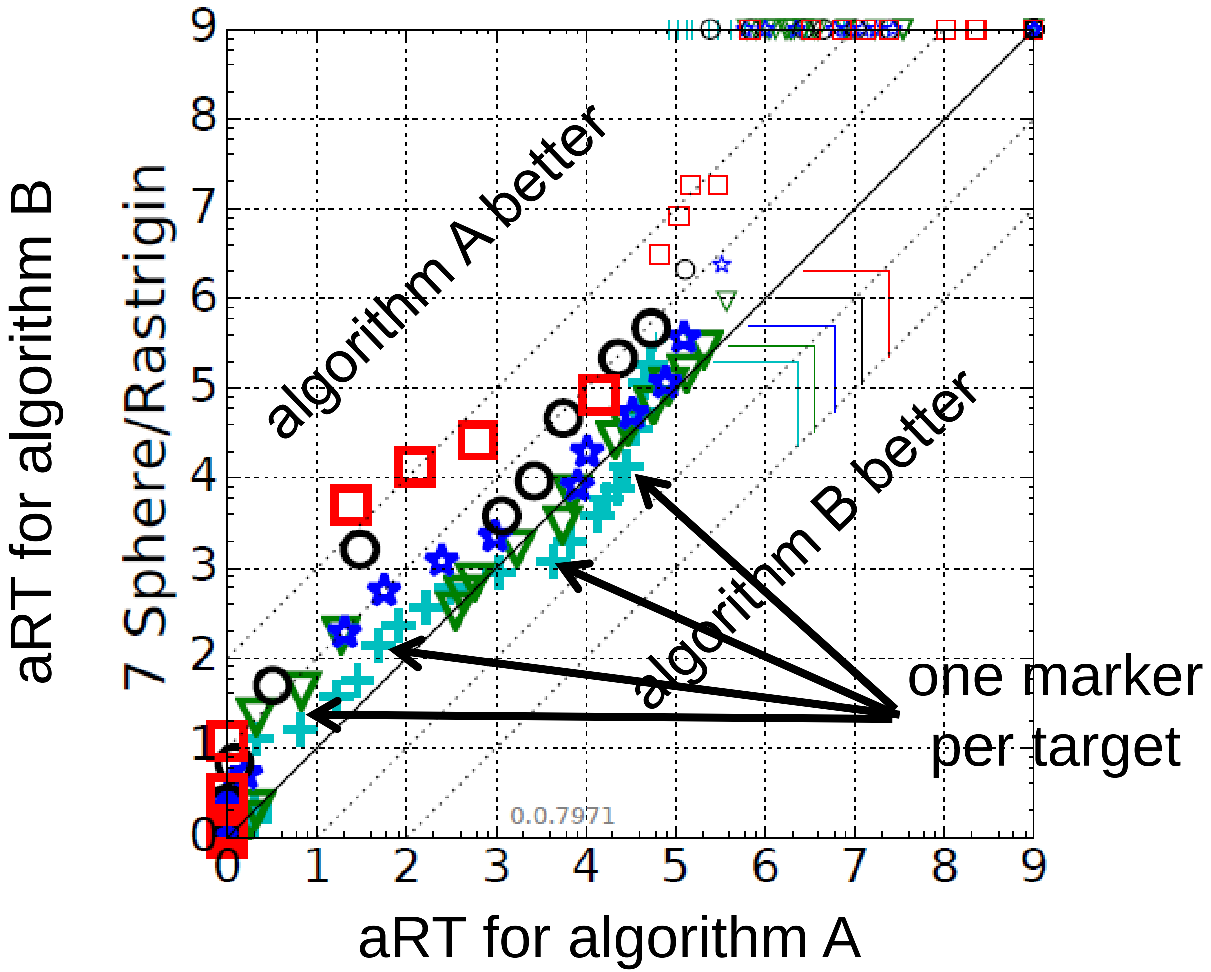


Interesting for 2 Algorithms...

...are scatter plots

dimensions:

2: +, 3: ▽, 5: ⋆, 10: ○, 20: □, 40: ◇.



There are more Plots...

...but they are probably less interesting for us here

Exercise (Part 2): Comparing Numerical Optimization Algorithms with COCO

Exercise (Part 2)

Objectives:

- investigate the performance of algorithms
  - CMA-ES ("IPOP-CMA-ES" version)
  - Nelder-Mead simplex (use "NelderDoerr" version here)
  - BFGS quasi-Newton
  - Genetic Algorithm: discretization of cont. variables ("GA")
  - plus 1-2 algos of your choice from http://coco.gforge.inria.fr
- postprocess (now) and investigate the data (after a few more slides)

tip: use **`--omit-single`** option to save time

The single-objective BBOB functions

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

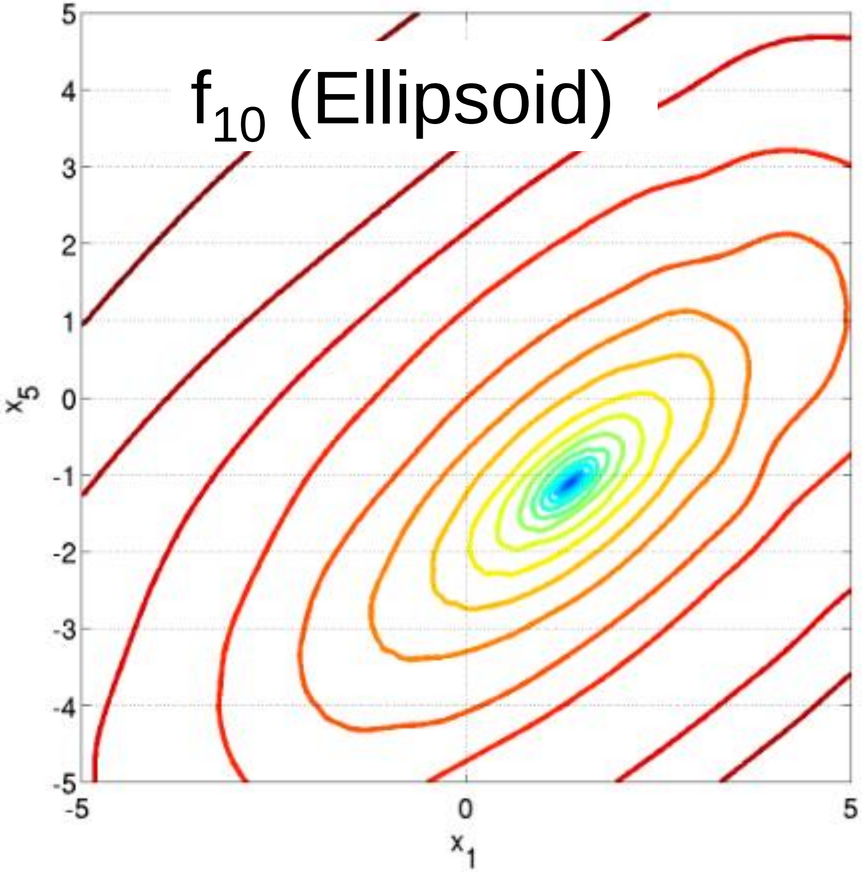
- All COCO problems come in form of instances
  - e.g. as translated/rotated versions of the same function
- Prescribed instances typically change from year to year
  - avoid overfitting
  - 5 instances are always kept the same

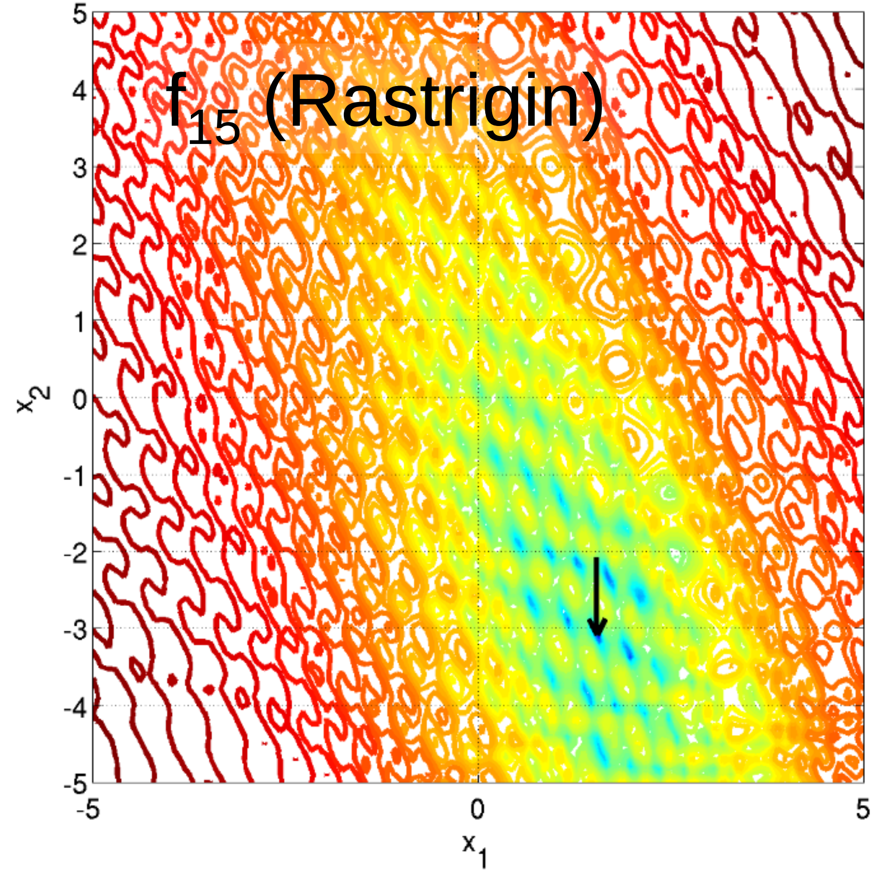
Plus:

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

Notion of Instances

- All COCO problems come in form of instances





linear transformations

Exercise (Part 3): Comparing Numerical Optimization Algorithms with COCO

Exercise (Part 3)

Objective:

investigate the data:

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

reminder: open thesis projects

one is related to this exercise
but automatized & for 150+ data sets ("data science")

Conclusions

I hope it became clear...

...that the fixed-target approach is superior over the budget-based approach (and why)

...that COCO is easy to use and provides a lot of data to explore

...and which algorithms to use/investigate when you have to solve a numerical unconstrained blackbox problem yourself at some point