# Benchmarking Blackbox Optimization Algorithms

July 5, 2017

CEA/EDF/Inria summer school "Numerical Analysis"

Université Pierre-et-Marie-Curie, Paris, France

Dimo Brockhoff

Inria Saclay – Ile-de-France

CMAP, Ecole Polytechnique

# Overview of the Remaining Lectures & Exercises

**Introduction to (Evolutionary) Multiobjective Optimization (now)**

- difference to single-objective optimization, the basics
- algorithms and their design principles; MO-CMA-ES

**Benchmarking Optimization Algorithms (this morning)**

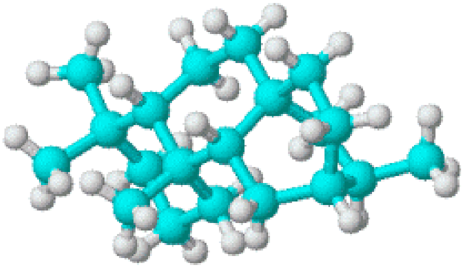
- performance assessment
- automated benchmarking with the COCO platform

**Exercise around COCO (this afternoon)**

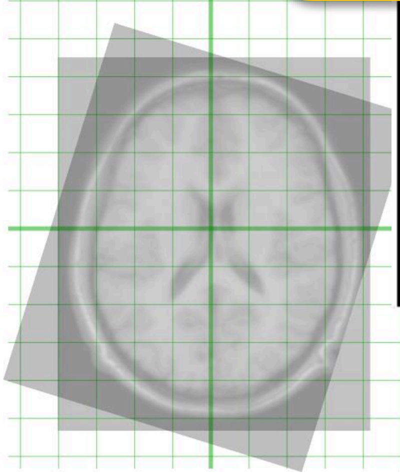
- interpreting available COCO data
- if time allows: looking critically at published results

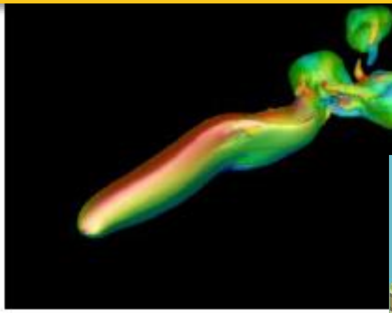
**Exercise on Anne's part (tomorrow afternoon)**

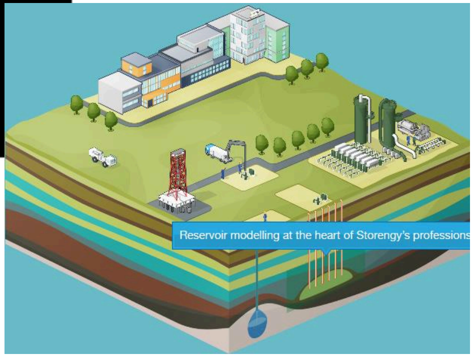
- The (1+1)-ES, running CMA-ES and interpreting its output, ...



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

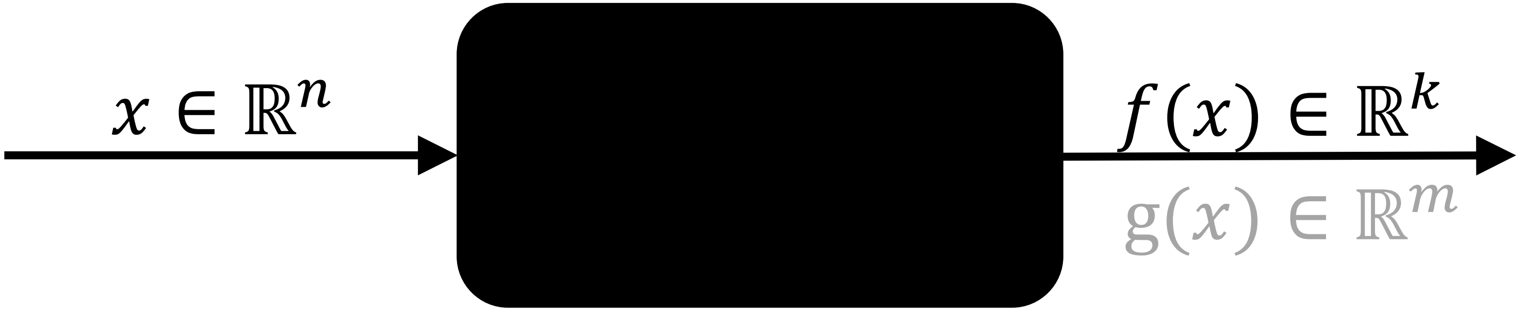






# Numerical Blackbox Optimization

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



$x \in \mathbb{R}^n$

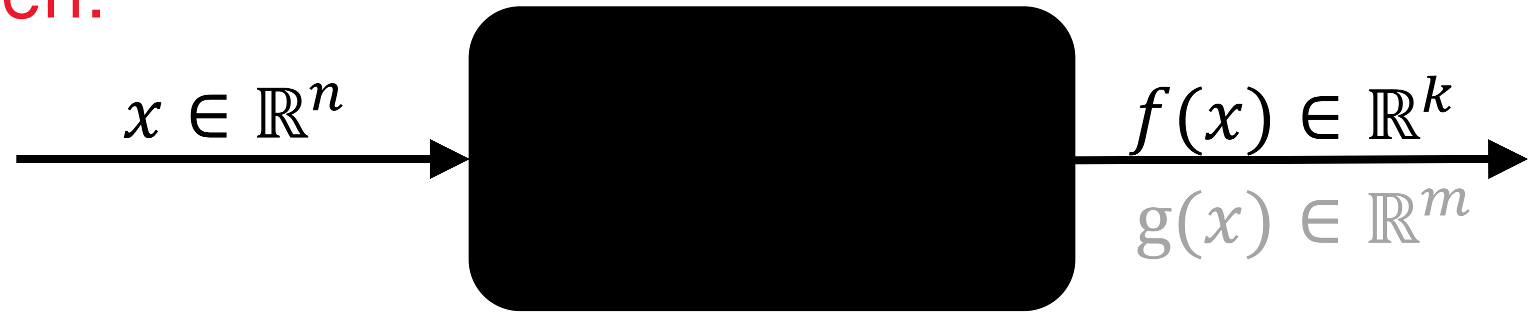
$f(x) \in \mathbb{R}^k$

$g(x) \in \mathbb{R}^m$

*derivatives not available or not useful*

# Practical Blackbox Optimization

Given:



$x \in \mathbb{R}^n$

$f(x) \in \mathbb{R}^k$

$g(x) \in \mathbb{R}^m$

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 2001]
- Cross Entropy Method (same as EDA) [Rubinstein 1999]
- 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]

**best choice typically not immediately clear**
although practitioners often have knowledge about which difficulties a problem has (e.g. multi-modality or non-separability)

- Simultaneous perturbation stochastic approx. (SPSA) [Spall 2000]

# 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

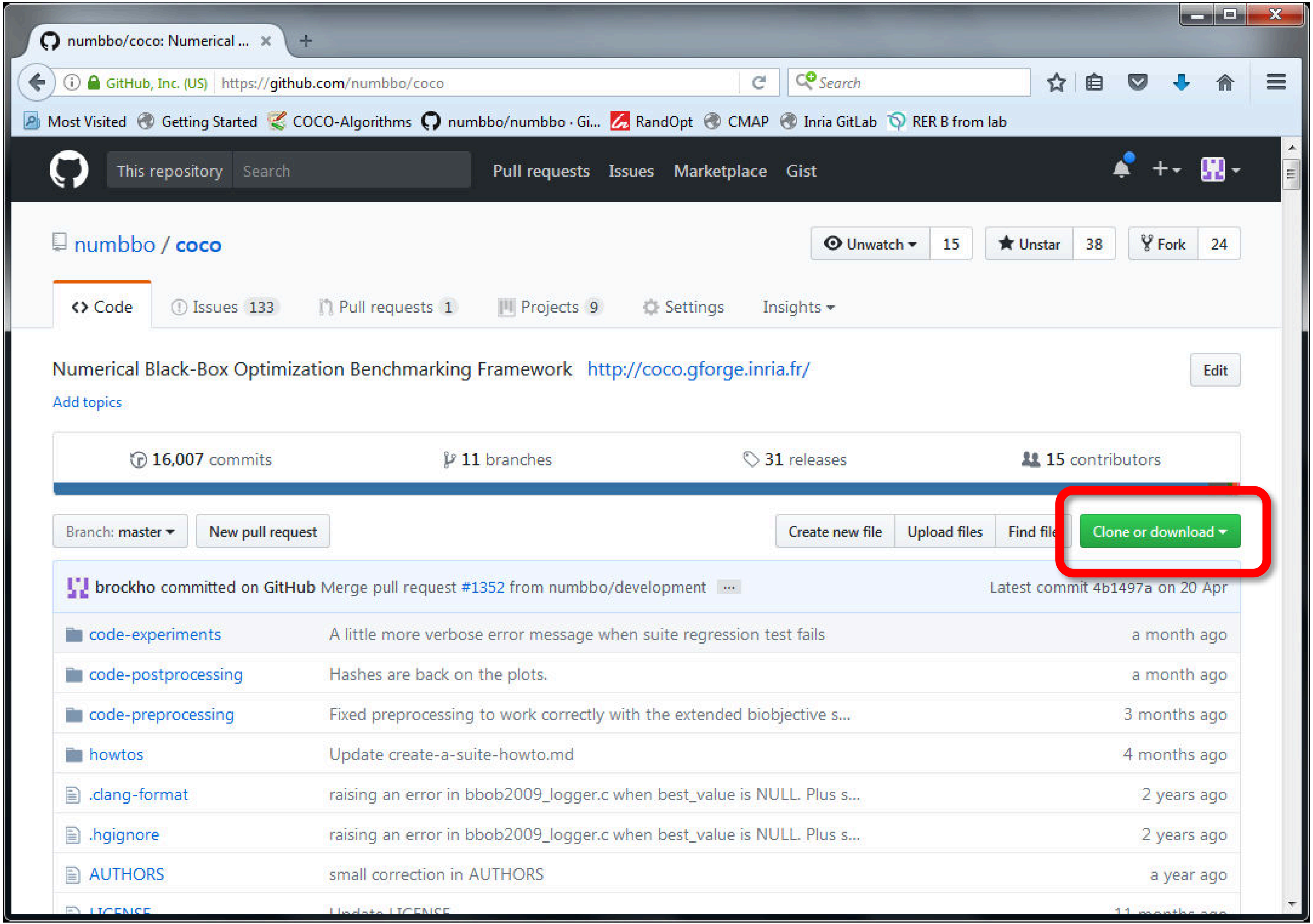
**Comparing Continuous Optimizers Platform**

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

# automatized benchmarking

# How to benchmark algorithms with COCO?

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



numbbo / coco

Numerical Black-Box Optimization Benchmarking Framework http://coco.gforge.inria.fr/

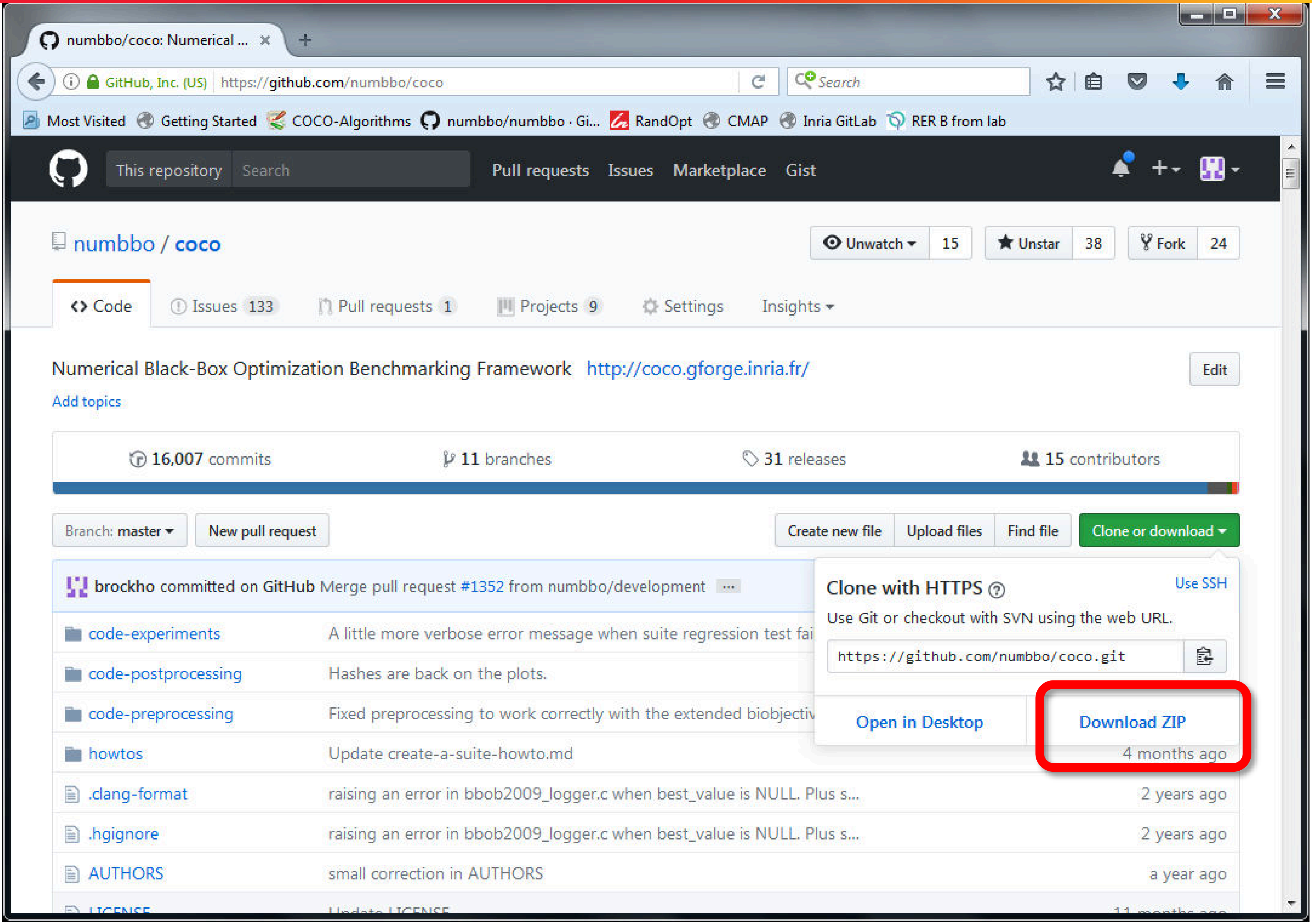
Add topics

16,007 commits | 11 branches | 31 releases | 15 contributors

brockho committed on GitHub Merge pull request #1352 from numbbo/development ... Latest commit 4b1497a on 20 Apr

| | | |
|---|---|---|
| code-experiments | A little more verbose error message when suite regression test fails | a month ago |
| code-postprocessing | Hashes are back on the plots. | a month ago |
| code-preprocessing | Fixed preprocessing to work correctly with the extended biobjective s... | 3 months ago |
| 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 |

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



numbbo / coco

Unwatch 15 | Unstar 38 | Fork 24

<> Code | Issues 133 | Pull requests 1 | Projects 9 | Settings | Insights

Numerical Black-Box Optimization Benchmarking Framework http://coco.gforge.inria.fr/

Edit

Add topics

16,007 commits | 11 branches | 31 releases | 15 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

brockho committed on GitHub Merge pull request #1352 from numbbo/development ...

**Clone with HTTPS** Use SSH

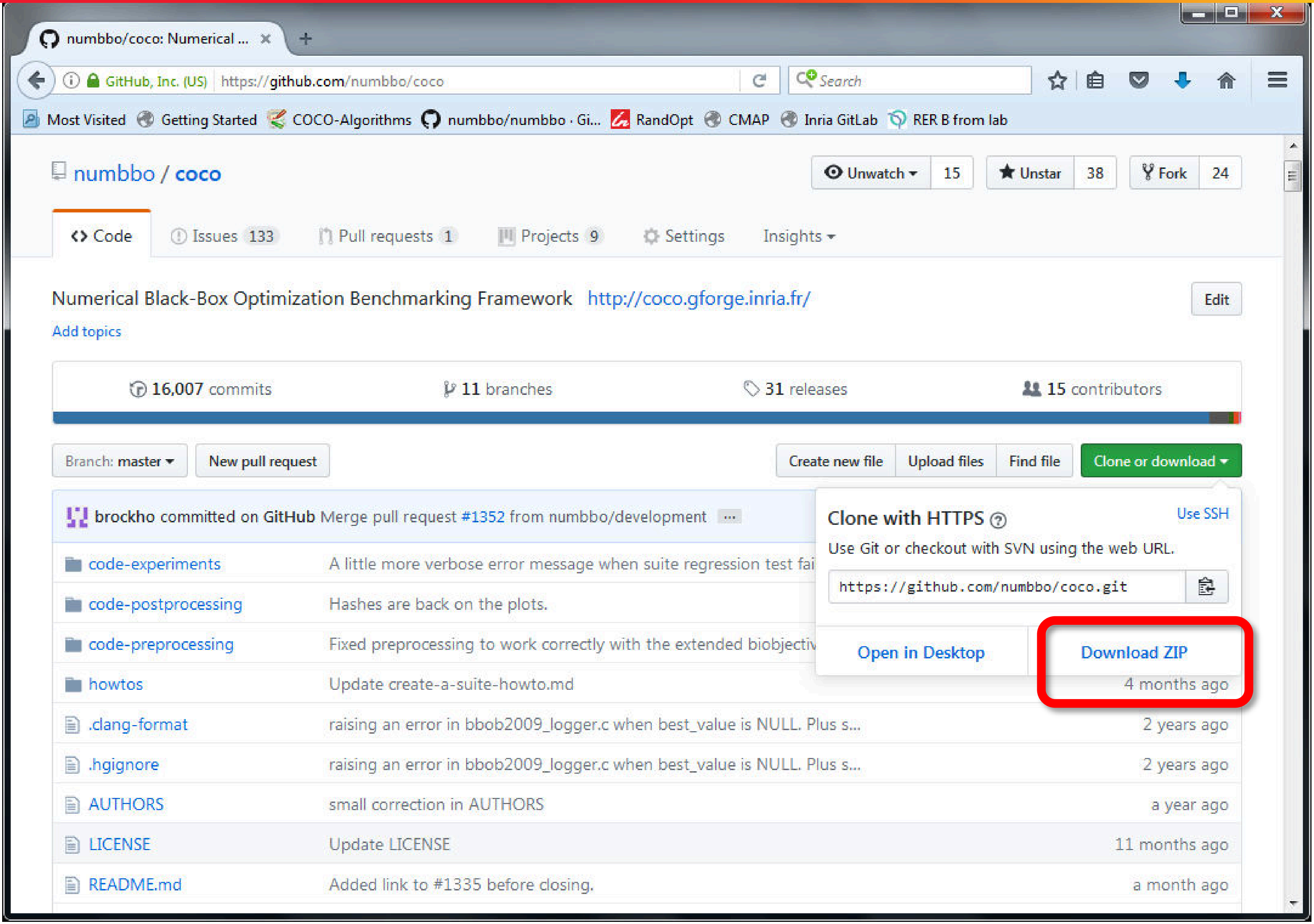
Use Git or checkout with SVN using the web URL.

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

Open in Desktop | Download ZIP

| | | |
|---|---|---|
| code-experiments | A little more verbose error message when suite regression test fai | |
| code-postprocessing | Hashes are back on the plots. | |
| 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 |

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



numbbo / coco

Unwatch 15 | Unstar 38 | Fork 24

Code | Issues 133 | Pull requests 1 | Projects 9 | Settings | Insights

Numerical Black-Box Optimization Benchmarking Framework http://coco.gforge.inria.fr/ Edit

Add topics

16,007 commits | 11 branches | 31 releases | 15 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

brockho committed on GitHub Merge pull request #1352 from numbbo/development ...

Clone with HTTPS ? Use SSH

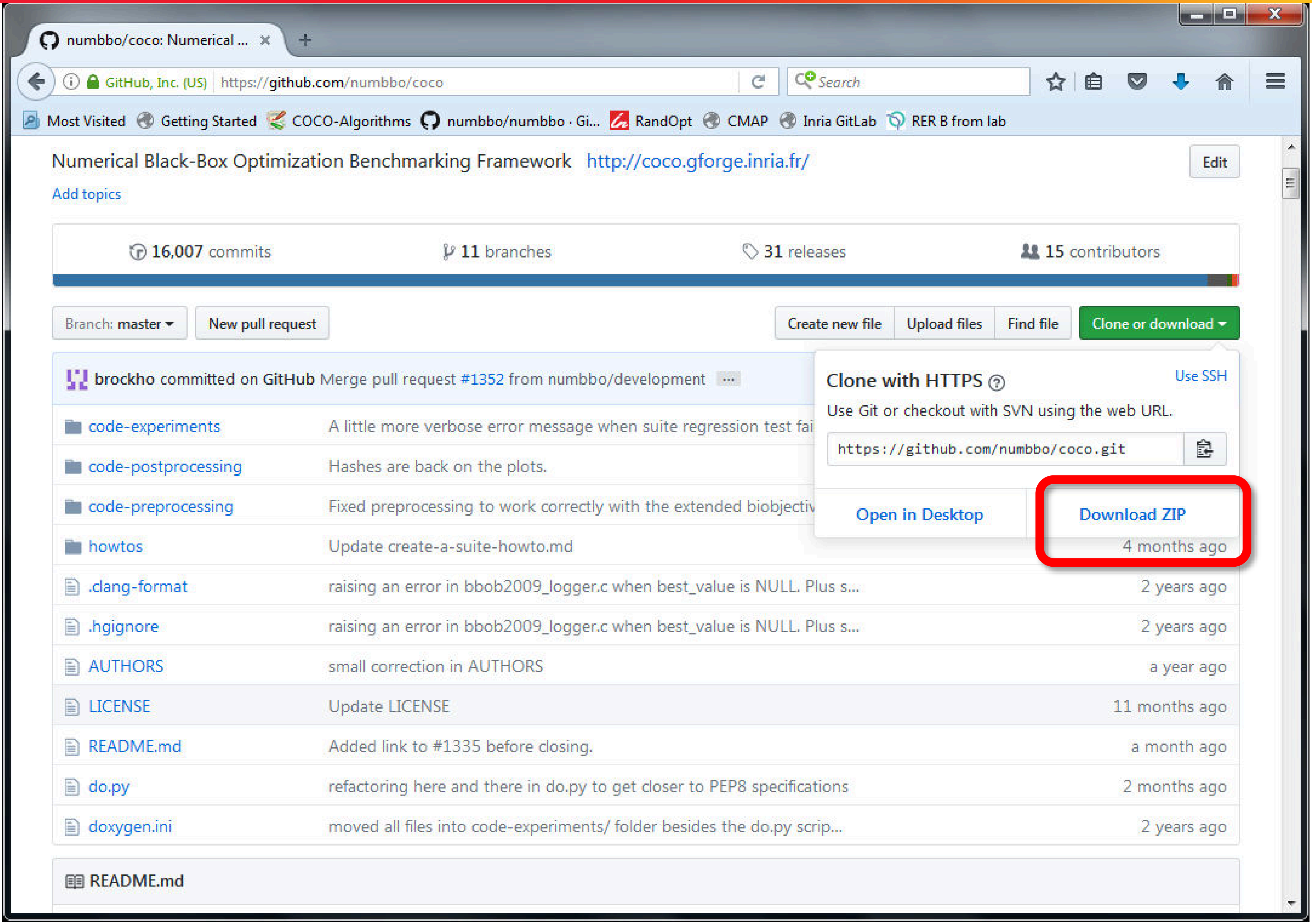
Use Git or checkout with SVN using the web URL.

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

Open in Desktop | Download ZIP

| | | |
|---|---|---|
| code-experiments | A little more verbose error message when suite regression test fai | |
| code-postprocessing | Hashes are back on the plots. | |
| 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 |

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



Numerical Black-Box Optimization Benchmarking Framework http://coco.gforge.inria.fr/

brockho committed on GitHub Merge pull request #1352 from numbbo/development ...

Clone with HTTPS

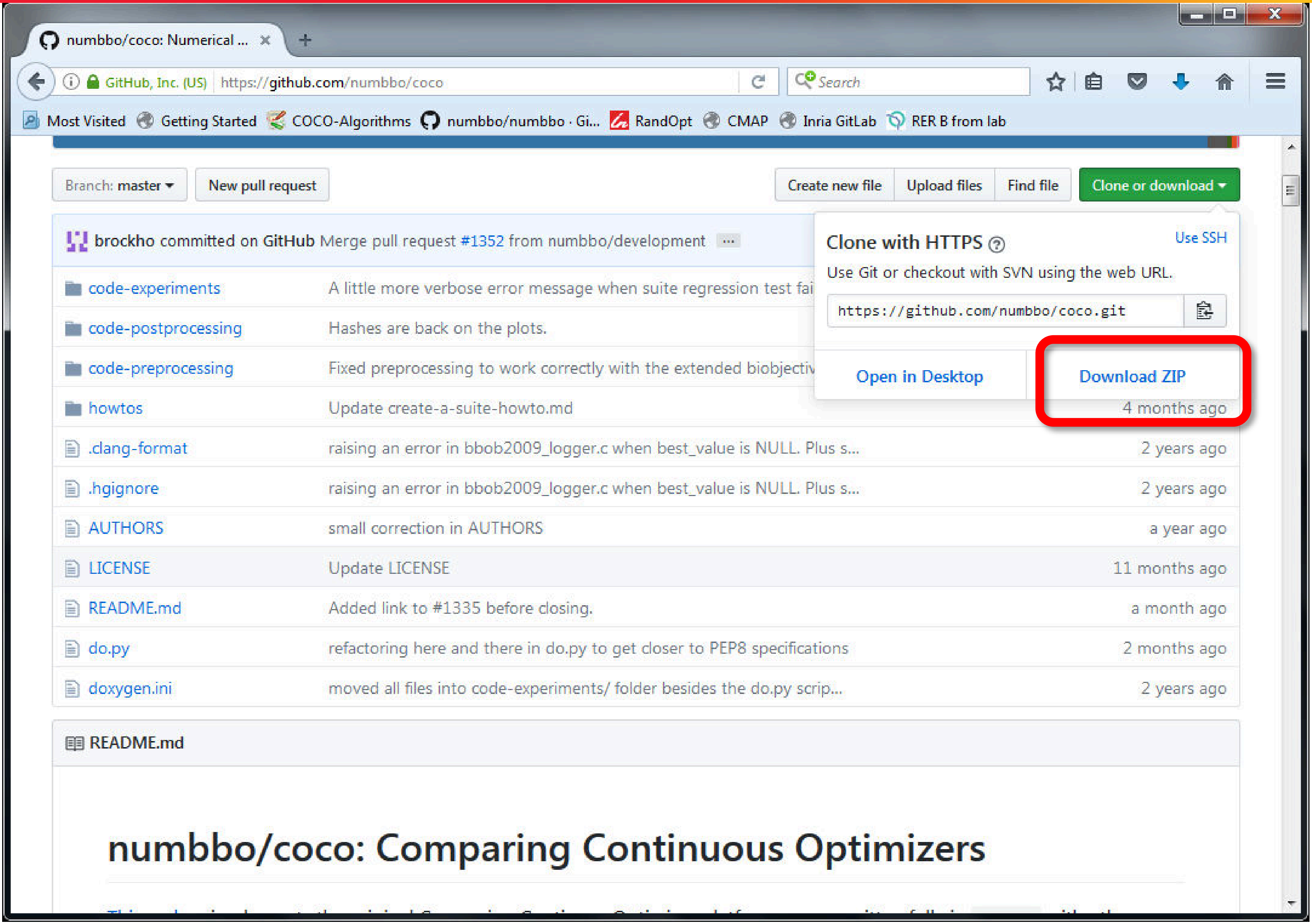
Use Git or checkout with SVN using the web URL.

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

| | | |
|---|---|---|
| code-experiments | A little more verbose error message when suite regression test fai | |
| code-postprocessing | Hashes are back on the plots. | |
| 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

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



brockho committed on GitHub Merge pull request #1352 from numbbo/development

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

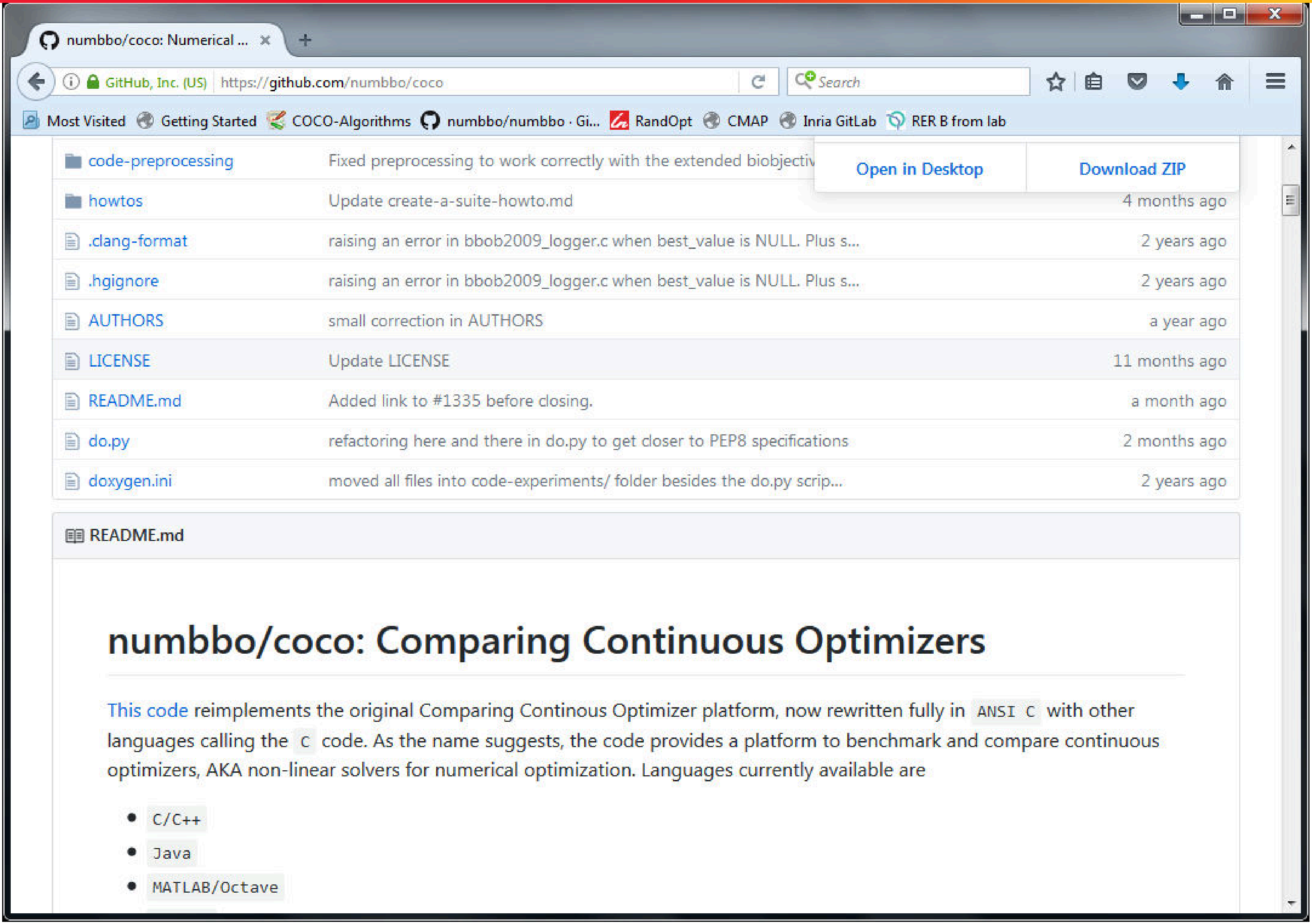
```
https://github.com/numbbo/coco.git
```

| | | |
|---|---|---|
| code-experiments | A little more verbose error message when suite regression test fai | |
| code-postprocessing | Hashes are back on the plots. | |
| 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

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



| | | |
|---|---|---|
| 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 |

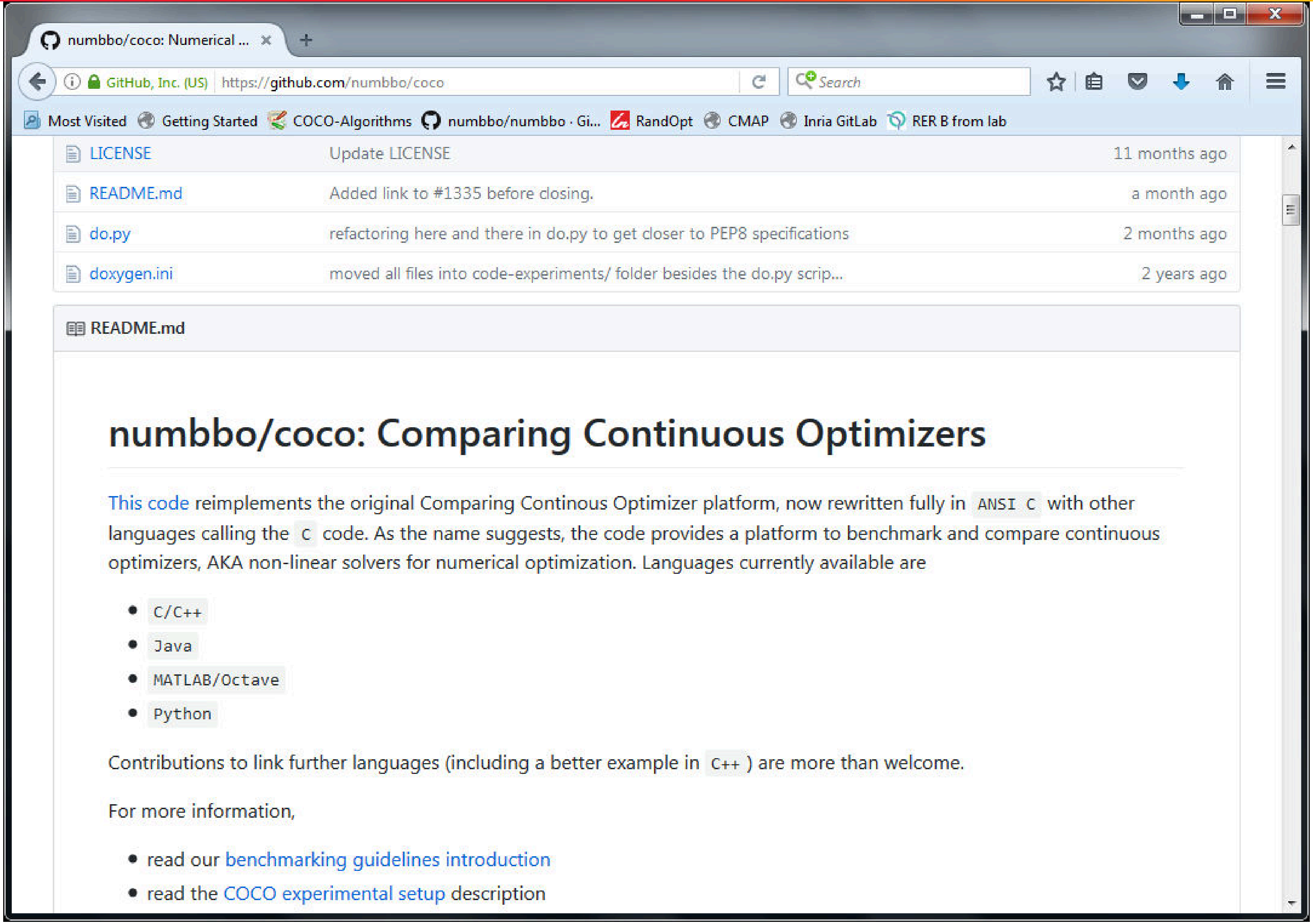
README.md

## 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



| | | |
|---|---|---|
| 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

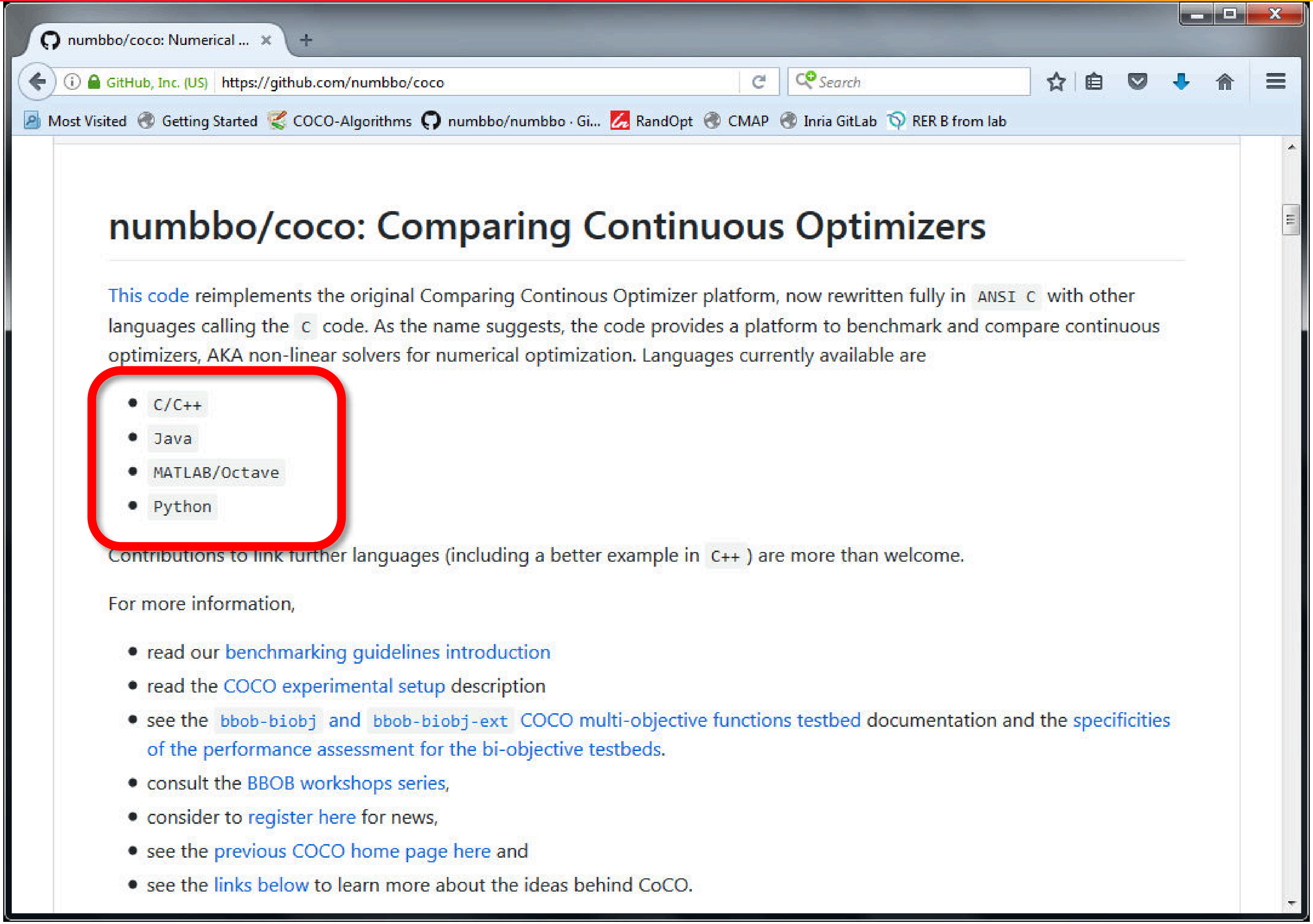
- `C/C++`
- `Java`
- `MATLAB/Octave`
- `Python`

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: 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

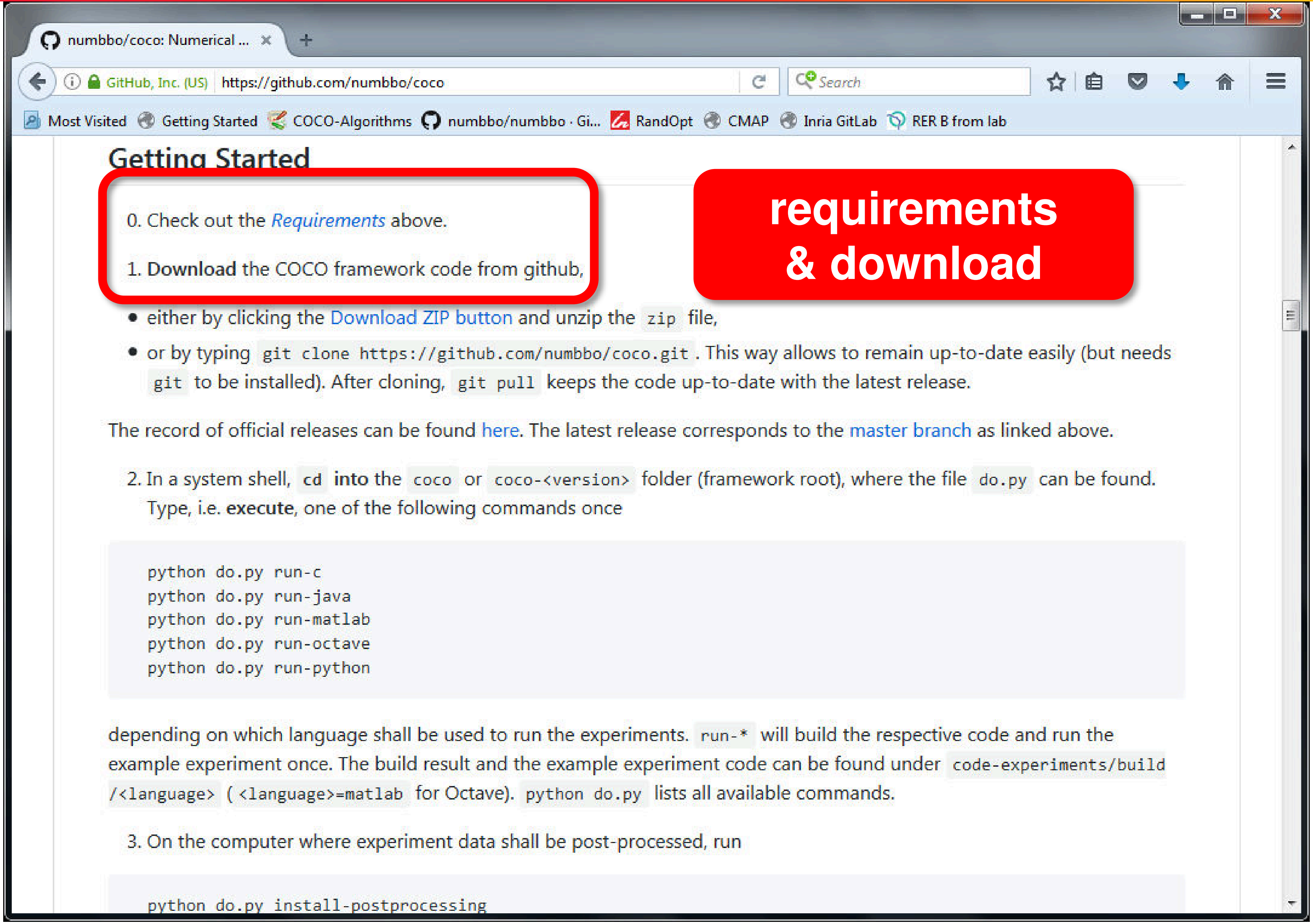
- `C/C++`
- `Java`
- `MATLAB/Octave`
- `Python`

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



## Getting Started

0. Check out the *Requirements* above.

1. **Download** the COCO framework code from github,

**requirements & download**

- either by clicking the Download ZIP button and unzip the `zip` file,
- or by typing `git clone https://github.com/numbbo/coco.git`. This way allows to remain up-to-date easily (but needs `git` to be installed). After cloning, `git pull` keeps the code up-to-date with the latest release.

The record of official releases can be found here. The latest release corresponds to the master branch as linked above.

2. 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

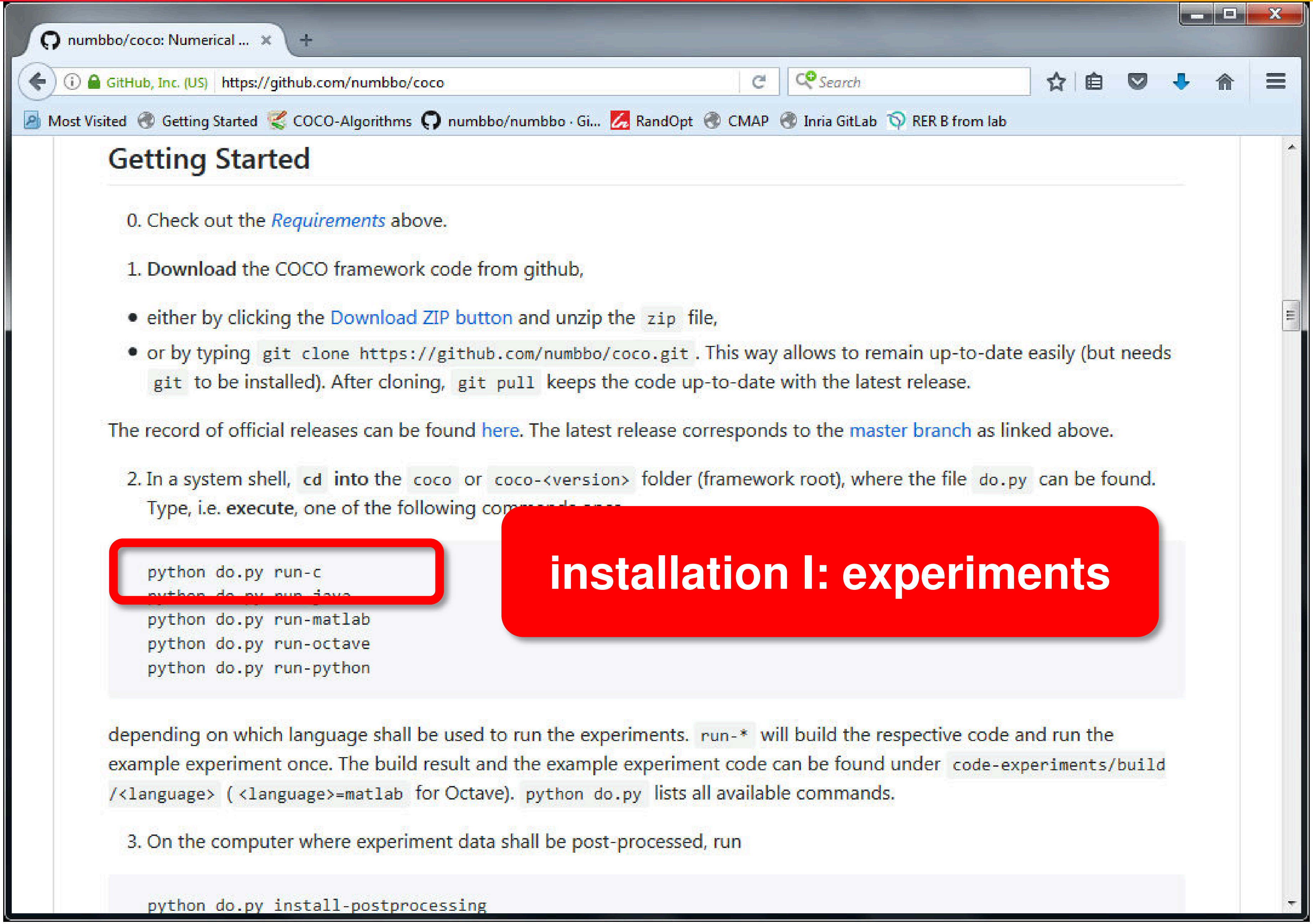
```
python do.py run-c
python do.py run-java
python do.py run-matlab
python do.py run-octave
python do.py run-python
```

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.

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

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

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



## Getting Started

0. Check out the *Requirements* above.

1. **Download** the COCO framework code from github,

- either by clicking the Download ZIP button and unzip the `zip` file,
- or by typing `git clone https://github.com/numbbo/coco.git`. This way allows to remain up-to-date easily (but needs `git` to be installed). After cloning, `git pull` keeps the code up-to-date with the latest release.

The record of official releases can be found here. The latest release corresponds to the master branch as linked above.

2. 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 com

```
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
```

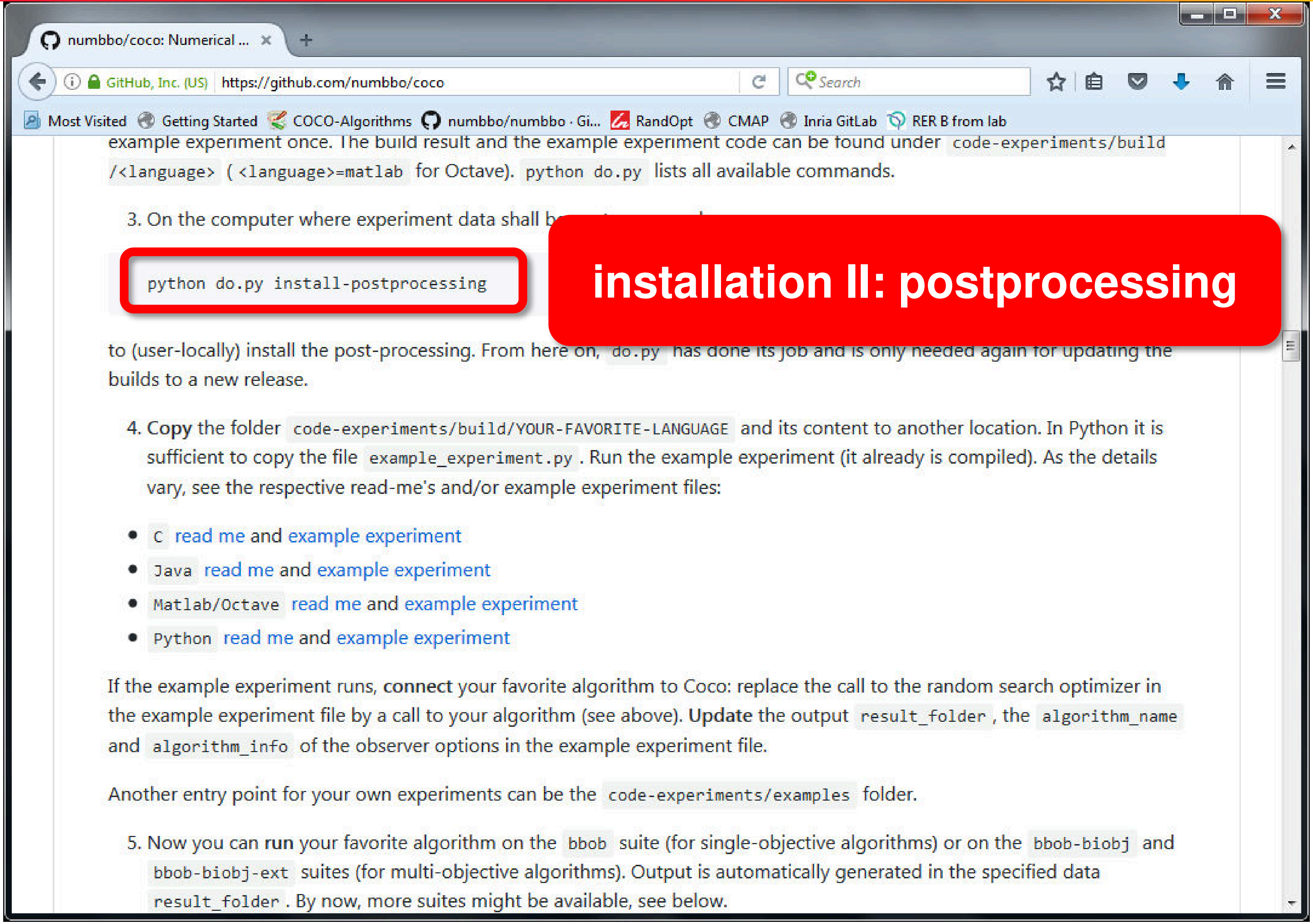
**installation I: experiments**

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.

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

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

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



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 b

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

**installation II: 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:

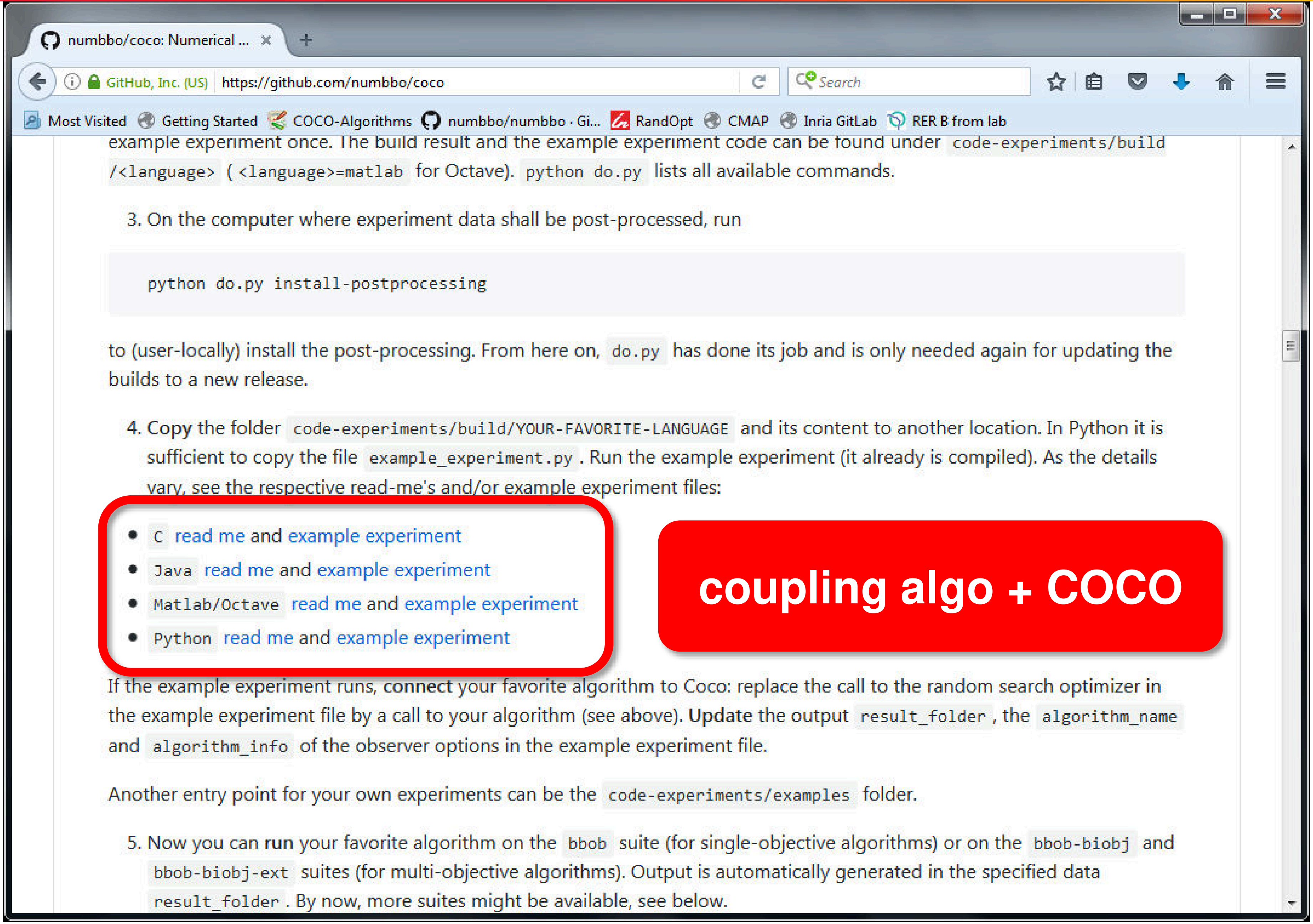
- `C` read me and example experiment
- `Java` read me and example experiment
- `Matlab/Octave` read me and example experiment
- `Python` read me and example experiment

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.

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



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.

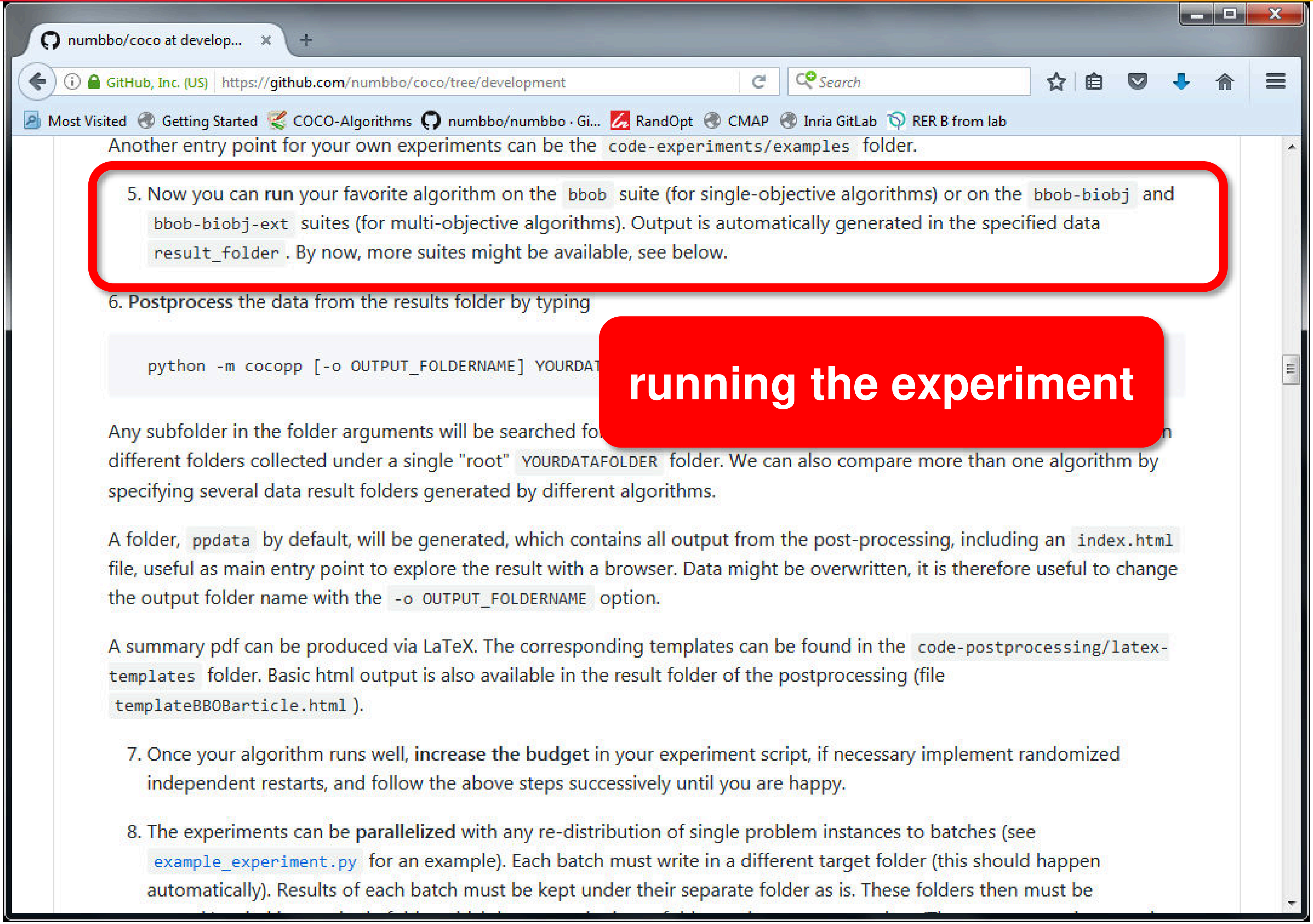
Another entry point for your own experiments can be the `code-experiments/examples` folder.

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.

# exampleexperiment.m (slightly simplified)

```
while true
    problem = cocoSuiteGetNextProblem(suite, observer);
    dimension = cocoProblemGetDimension(problem);
    i = -1; % count number of independent restarts
    while (BUDGET_MULTIPLIER*dimension > (cocoProblemGetEvaluations(problem) + ...
                                 cocoProblemGetEvaluationsConstraints(problem)))
        i = i+1;
        doneEvalsBefore = cocoProblemGetEvaluations(problem) + ...
                          cocoProblemGetEvaluationsConstraints(problem);
        % start algorithm with remaining number of function evaluations:
        my_optimizer(problem,...
            cocoProblemGetSmallestValuesOfInterest(problem), ...
            cocoProblemGetLargestValuesOfInterest(problem), ...
            BUDGET_MULTIPLIER*dimension - doneEvalsBefore);
        % check whether experiment is over:
        doneEvalsAfter = cocoProblemGetEvaluations(problem) + ...
                         cocoProblemGetEvaluationsConstraints(problem);
        if cocoProblemFinalTargetHit(problem) == 1 || ...
                doneEvalsAfter >= BUDGET_MULTIPLIER * dimension
            break;
        end
        if (i >= NUM_OF_INDEPENDENT_RESTARTS)
            break;
        end
    end
    doneEvalsTotal = doneEvalsTotal + doneEvalsAfter;
end
```

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



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.

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

```
python -m cocopp [-o OUTPUT_FOLDERNAME] YOURDAT
```

Any subfolder in the folder arguments will be searched fo n 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 an `index.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.

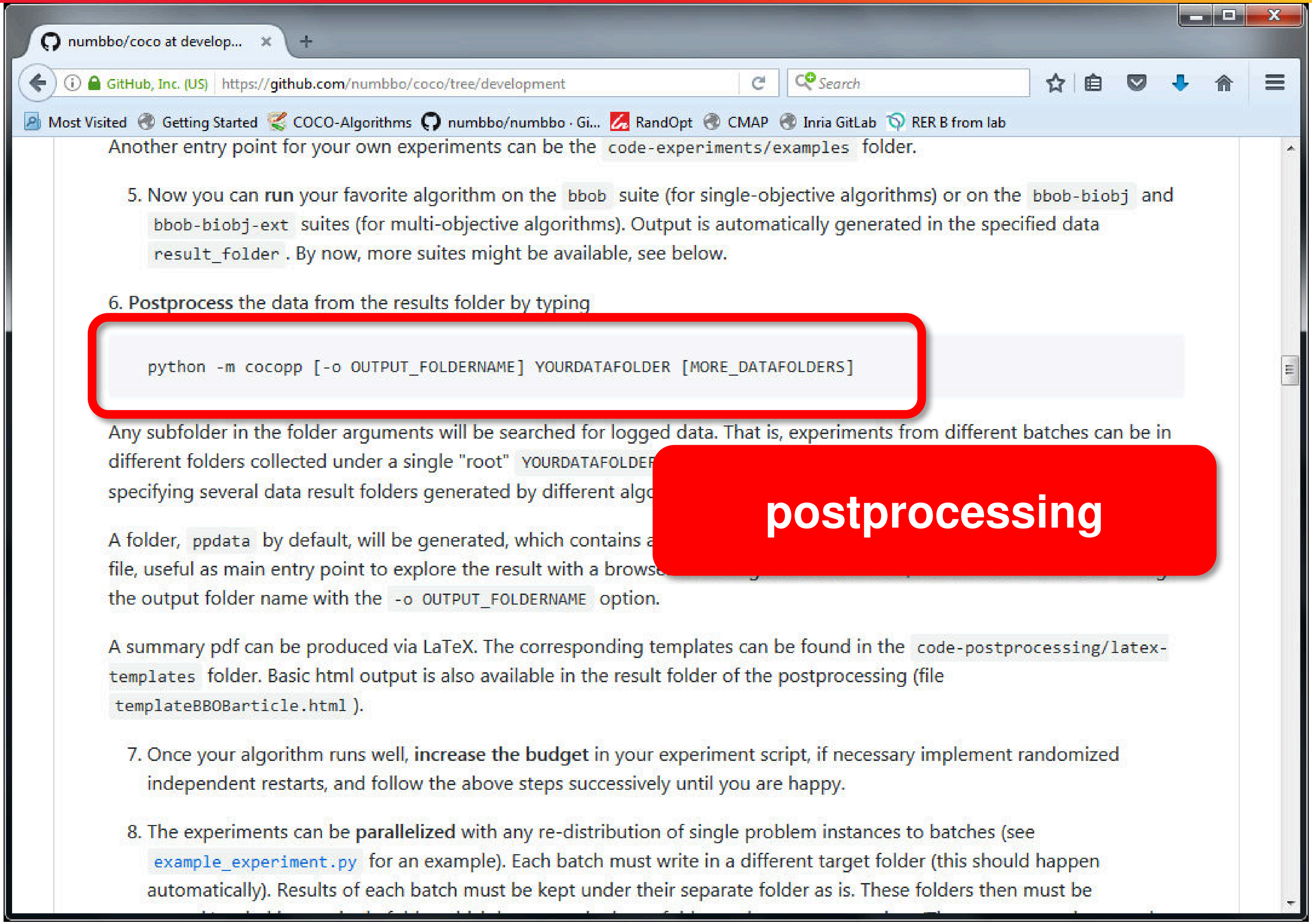
A summary pdf can be produced via LaTeX. The corresponding templates can be found in the `code-postprocessing/latex-templates` folder. Basic html output is also available in the result folder of the postprocessing (file `templateBBOBarticle.html`).

7. 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.

8. The experiments can be **parallelized** with any re-distribution of single problem instances to batches (see `example_experiment.py` for an example). Each batch must write in a different target folder (this should happen automatically). Results of each batch must be kept under their separate folder as is. These folders then must be

running the experiment

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



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.

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

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

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" `YOURDATAFOLDE`

specifying several data result folders generated by different algo

**postprocessing**

A folder, `ppdata` by default, will be generated, which contains a

file, useful as main entry point to explore the result with a browse

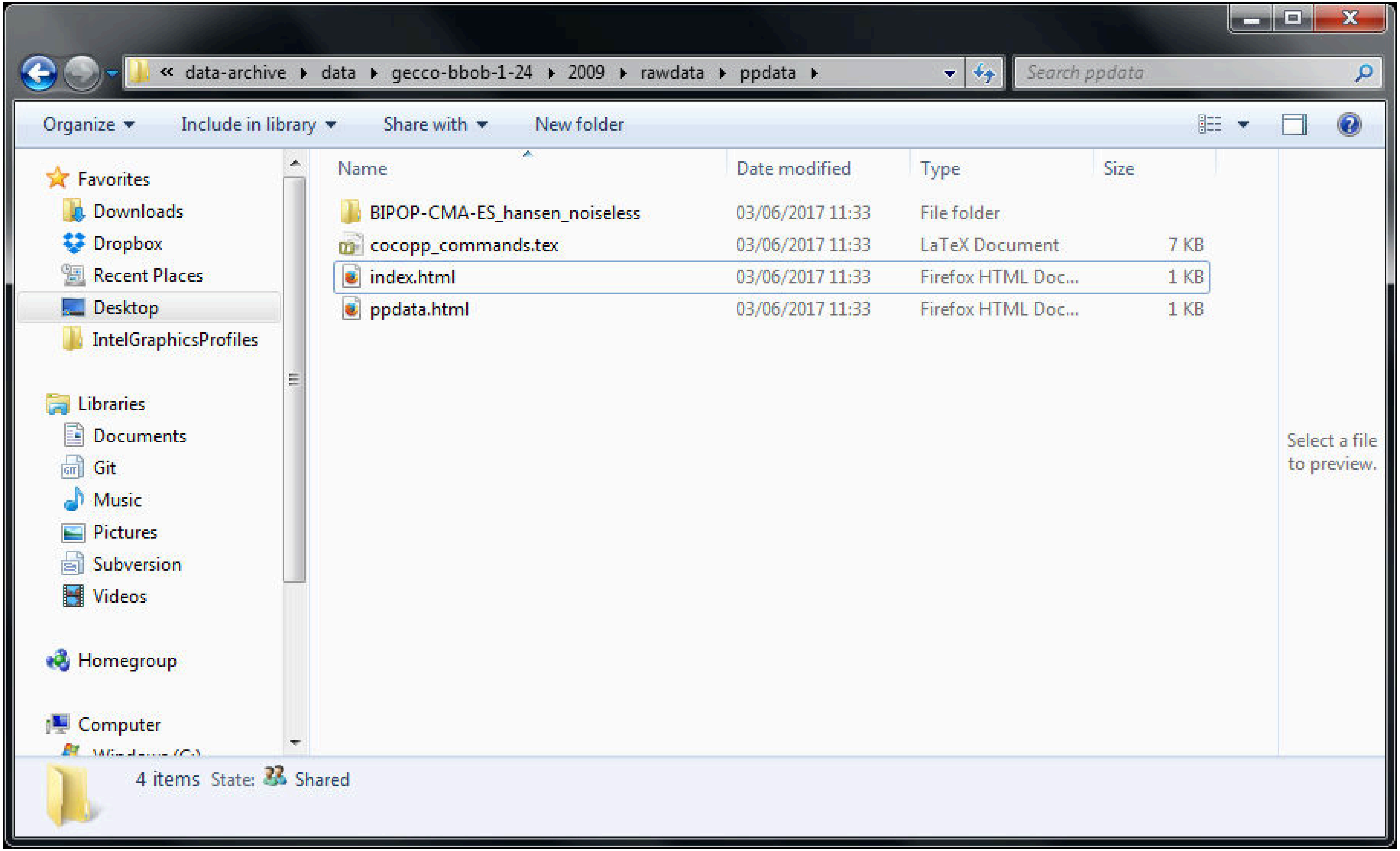
the output folder name with the `-o OUTPUT_FOLDERNAME` option.

A summary pdf can be produced via LaTeX. The corresponding templates can be found in the `code-postprocessing/latex-templates` folder. Basic html output is also available in the result folder of the postprocessing (file `templateBBOBarticle.html`).

7. 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.

8. The experiments can be **parallelized** with any re-distribution of single problem instances to batches (see `example_experiment.py` for an example). Each batch must write in a different target folder (this should happen automatically). Results of each batch must be kept under their separate folder as is. These folders then must be

# Result Folder

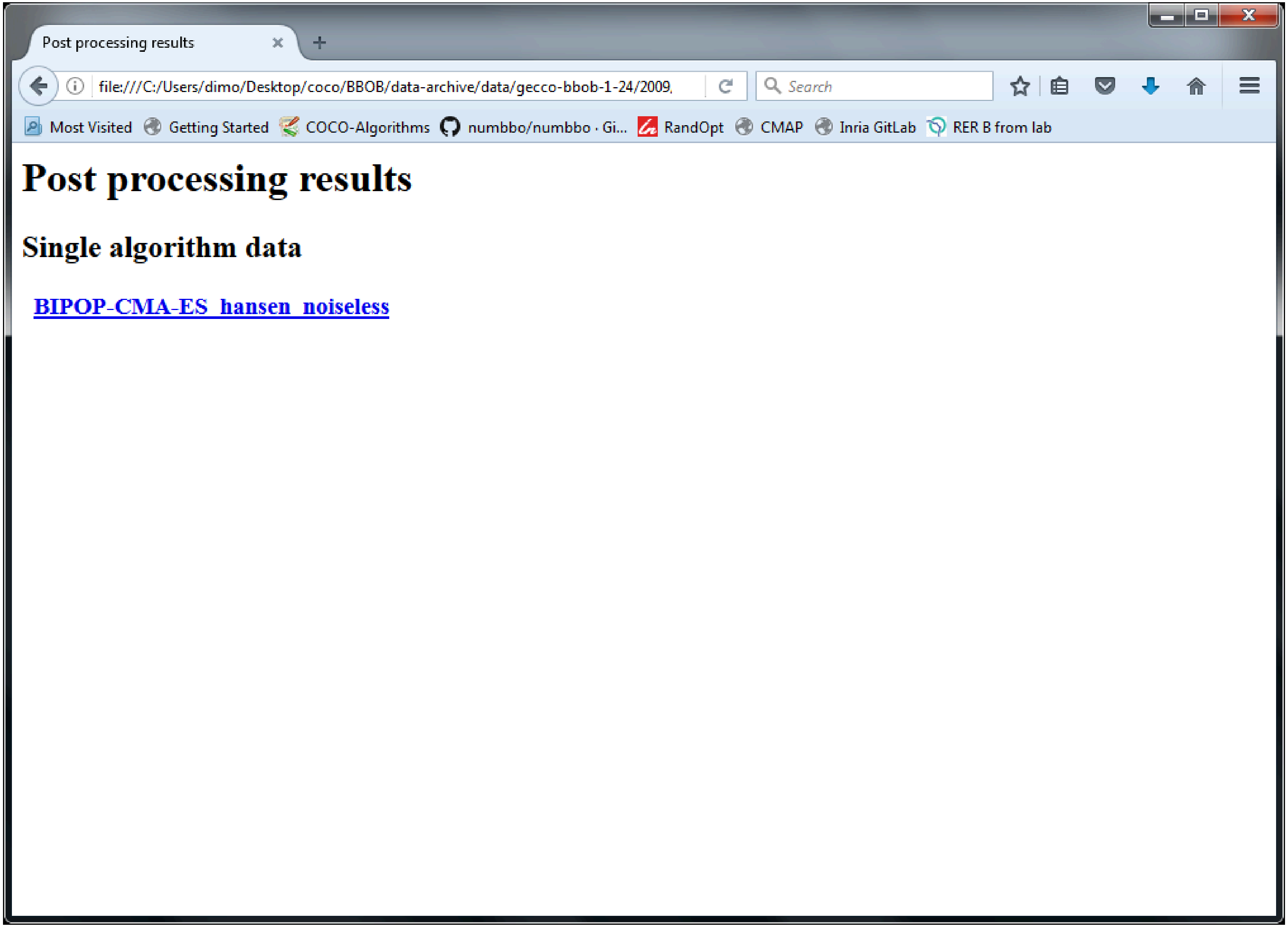


« data-archive ▸ data ▸ gecco-bbob-1-24 ▸ 2009 ▸ rawdata ▸ ppdata ▸

| 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 |

4 items State: Shared

# Automatically Generated Results

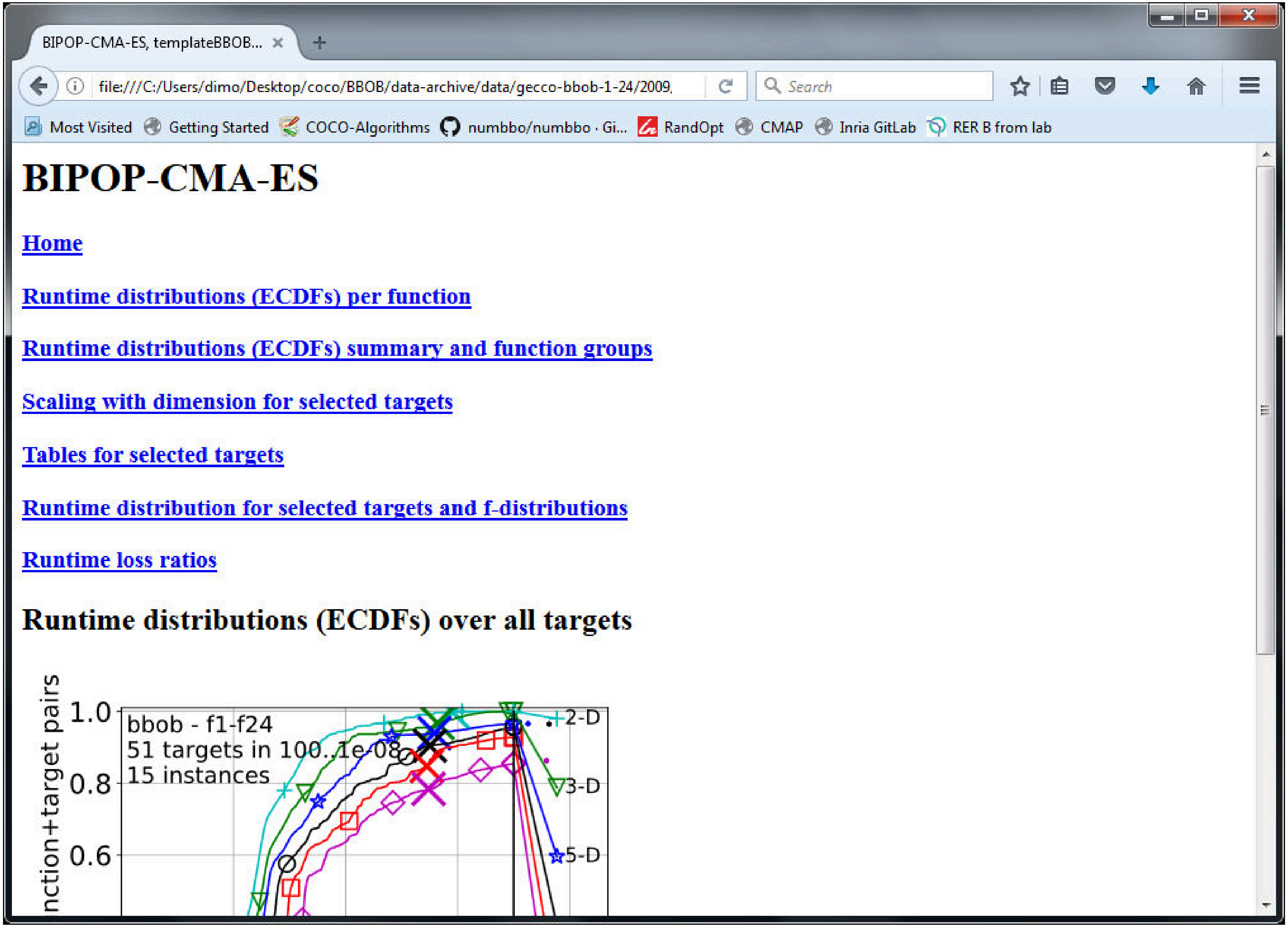


# Post processing results

## Single algorithm data

BIPOP-CMA-ES_hansen_noiseless

# Automatically Generated Results



BIPOP-CMA-ES, templateBBOB...
file:///C:/Users/dimo/Desktop/coco/BBOB/data-archive/data/gecco-bbob-1-24/2009,
Search
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

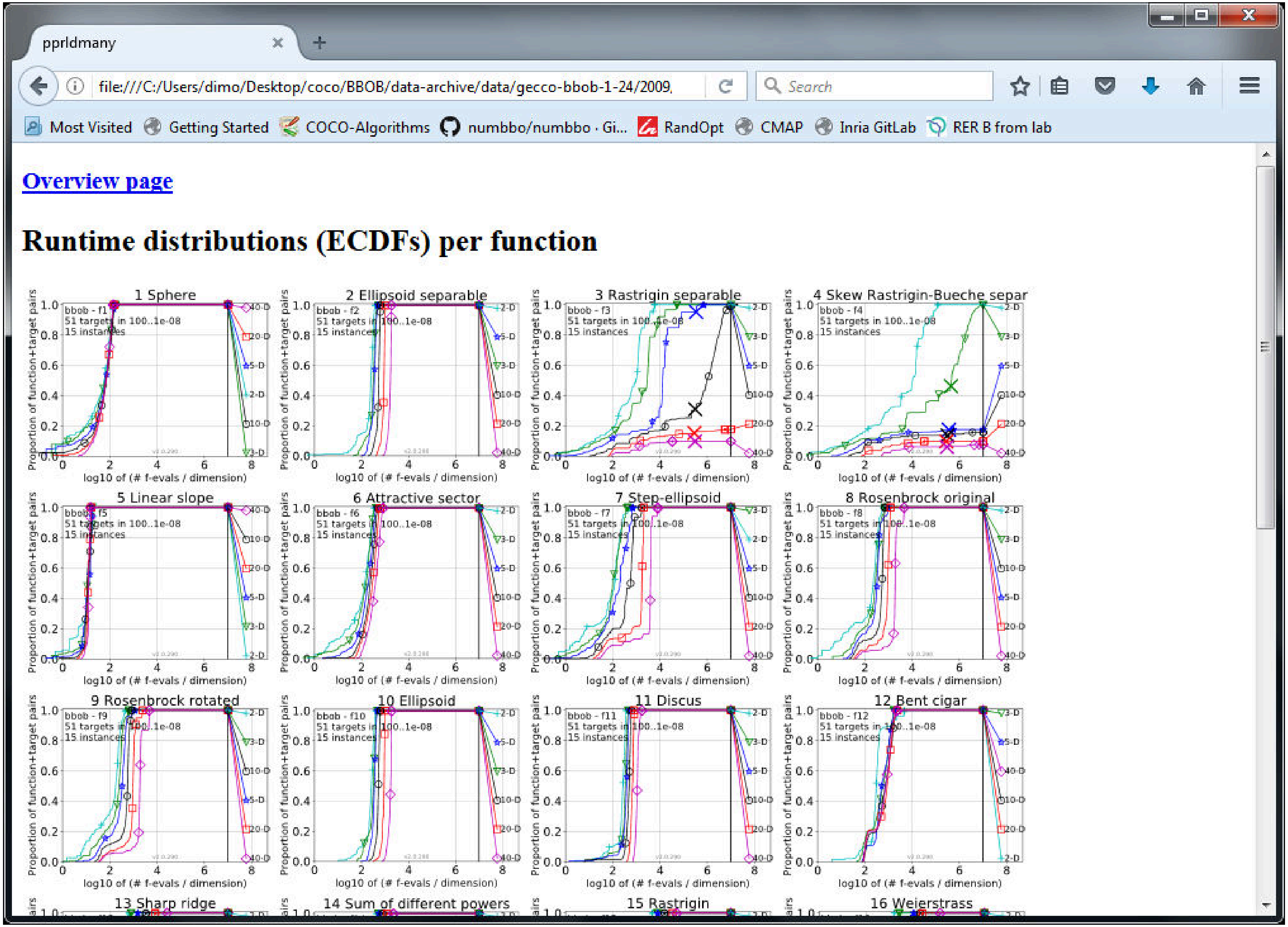
Tables for selected targets

Runtime distribution for selected targets and f-distributions

Runtime loss ratios

## Runtime distributions (ECDFs) over all targets

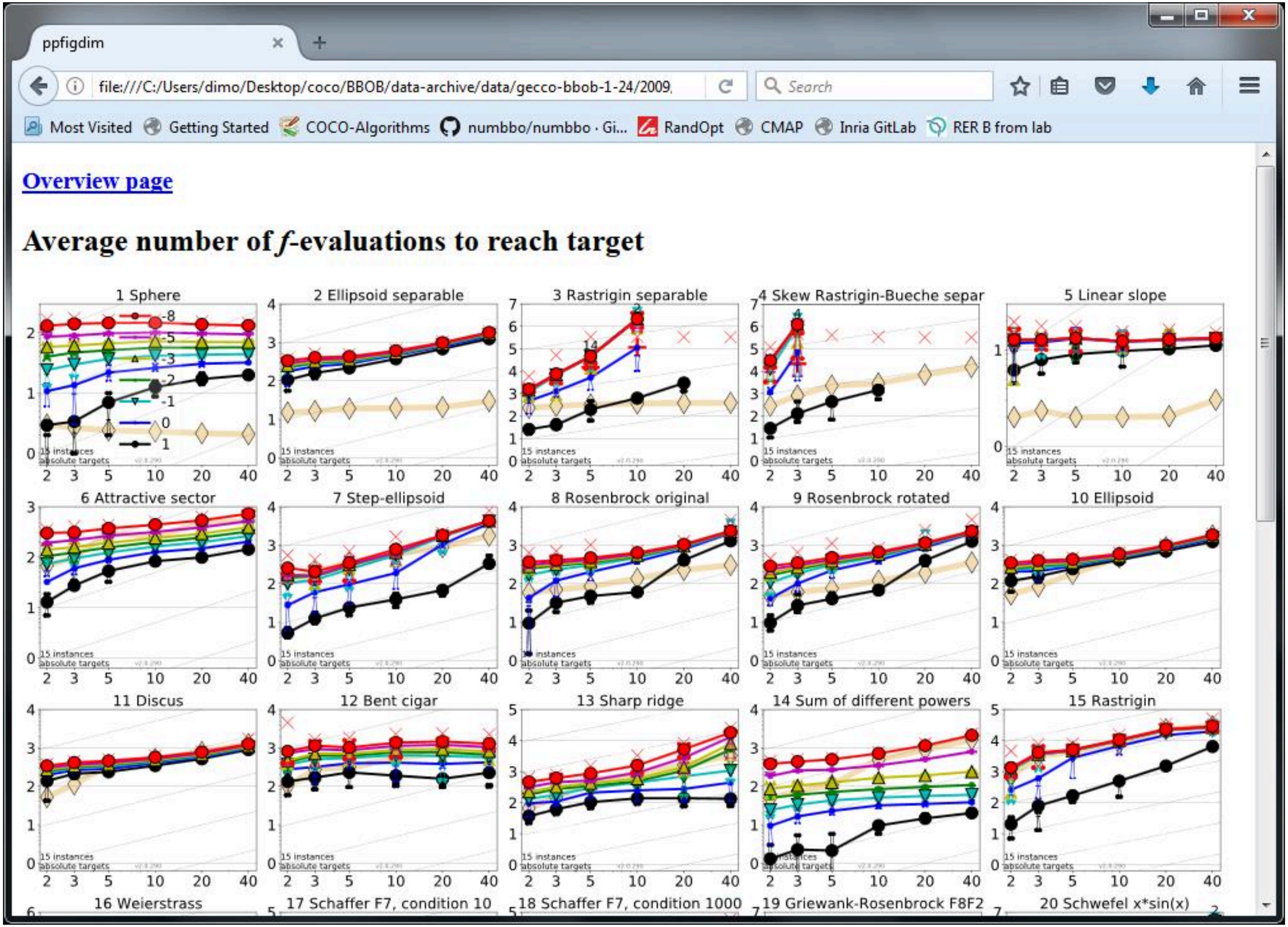
# Automatically Generated Results



Overview page

## Runtime distributions (ECDFs) per function

# Automatically Generated Results



Overview page

## Average number of *f*-evaluations to reach target

# doesn't look too complicated, does it?

[the devil is in the details ☺]

# Available Data Sets

**so far:**

data for about 170 algorithm variants
(some of which on noisy or multiobjective test functions)
145 workshop papers
by 93 authors from 25 countries

# 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 | 16 algo data sets |

## Under development:

- an extended biobjective suite
- a large-scale version of the bbob suite
- a constrained test suite

## Long-term goals:

- combining difficulties
- almost real-world problems
- real-world problems

# 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?

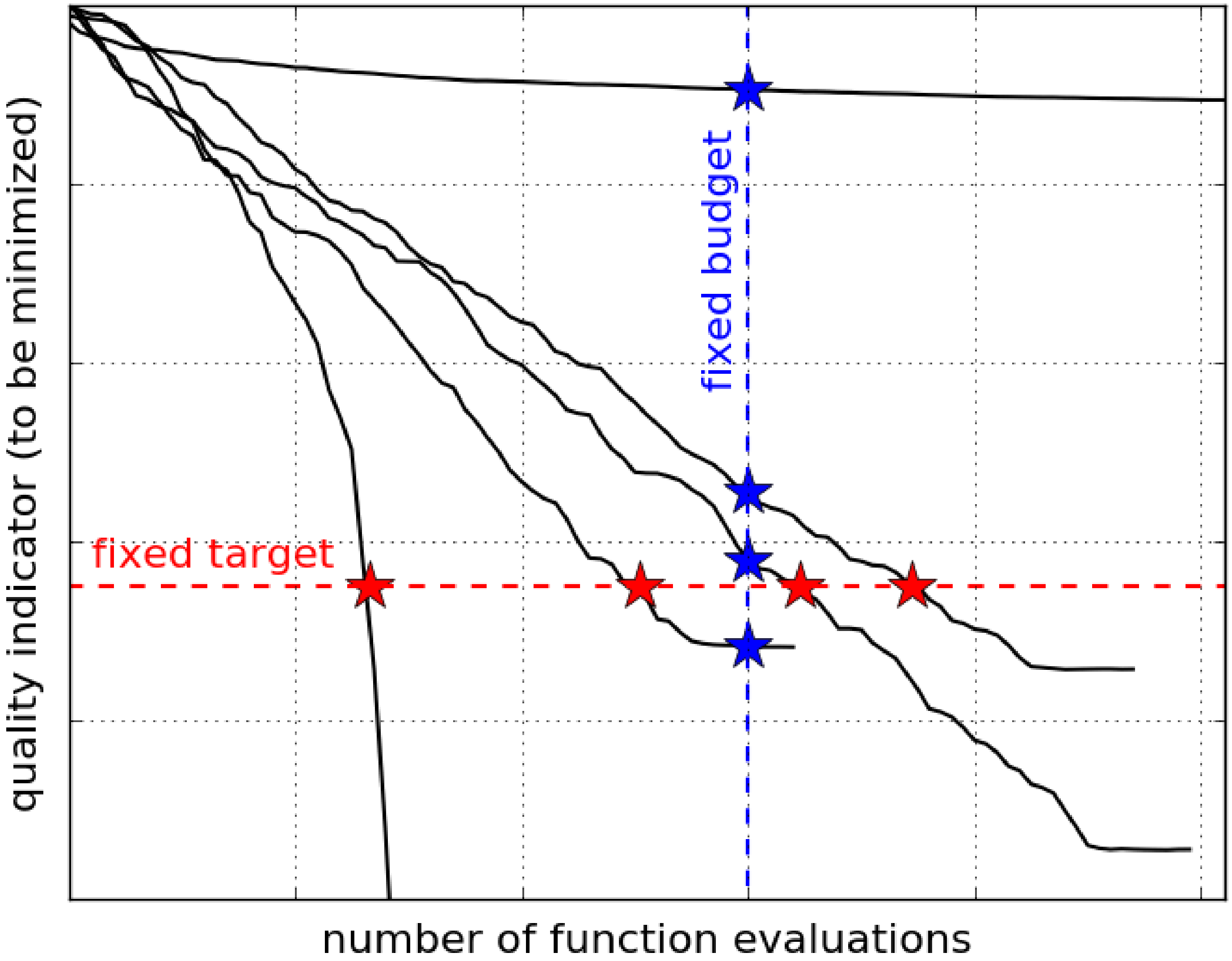
## Two objectives:

- 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...

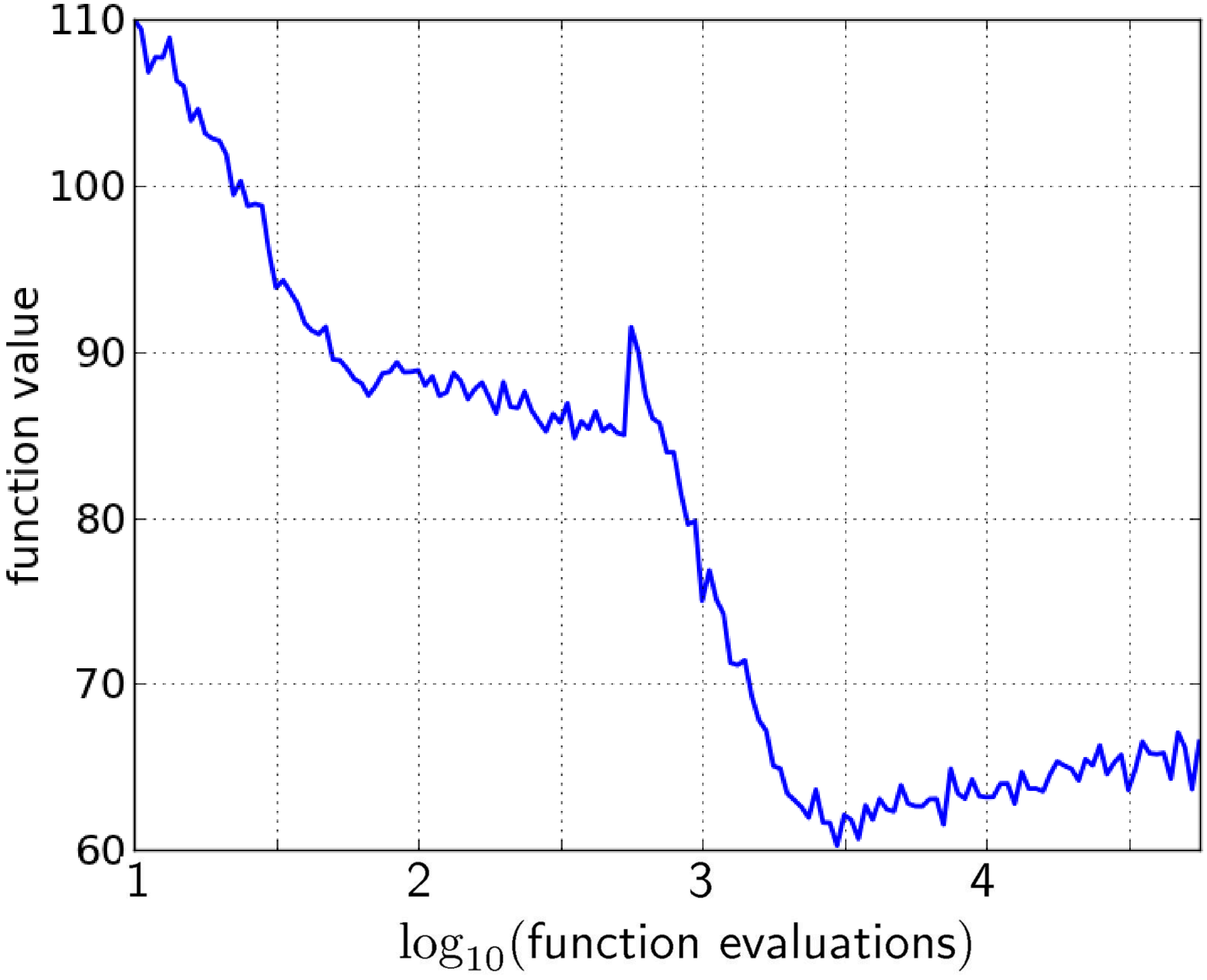


# ECDF:

Empirical Cumulative Distribution Function of the Runtime

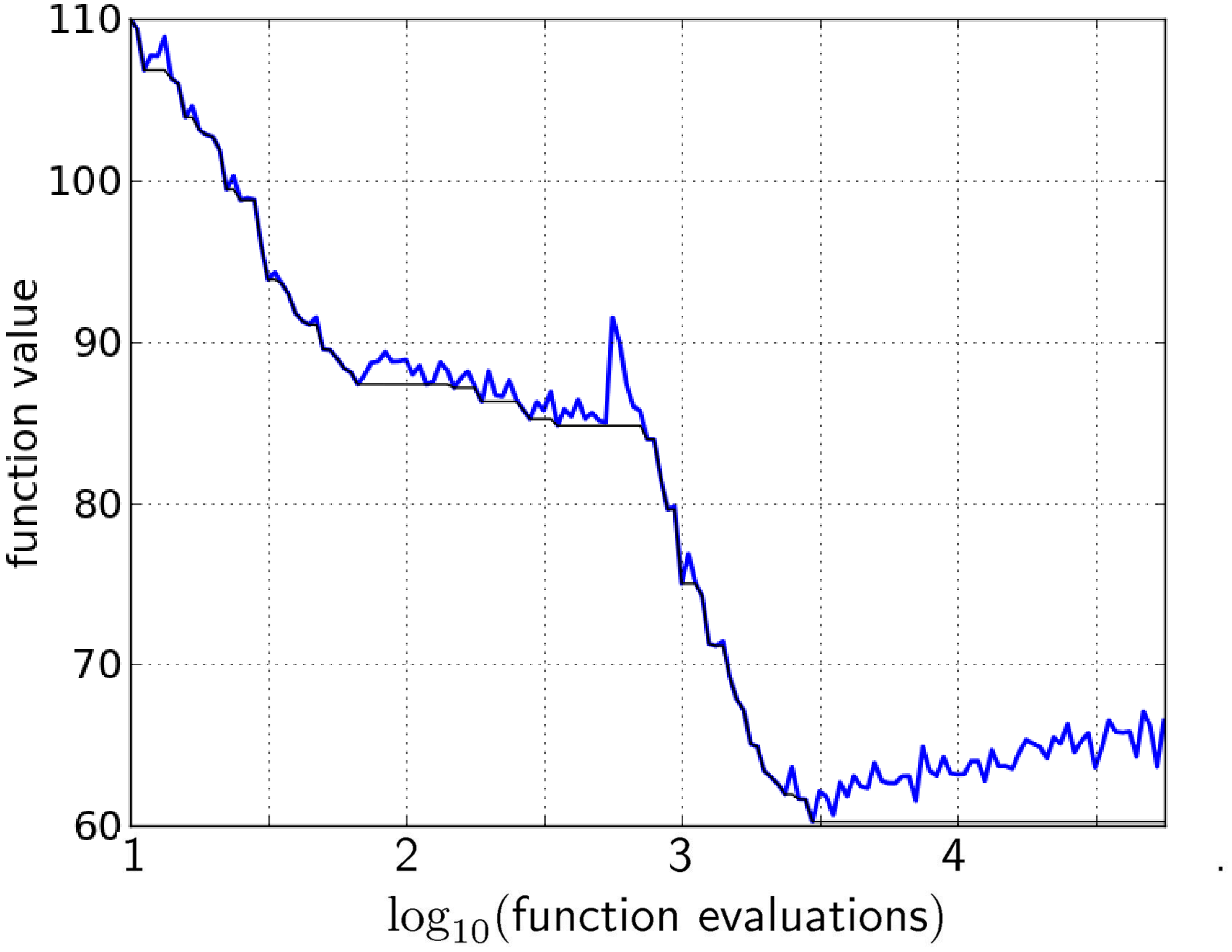
[aka data profile]

# A Convergence Graph

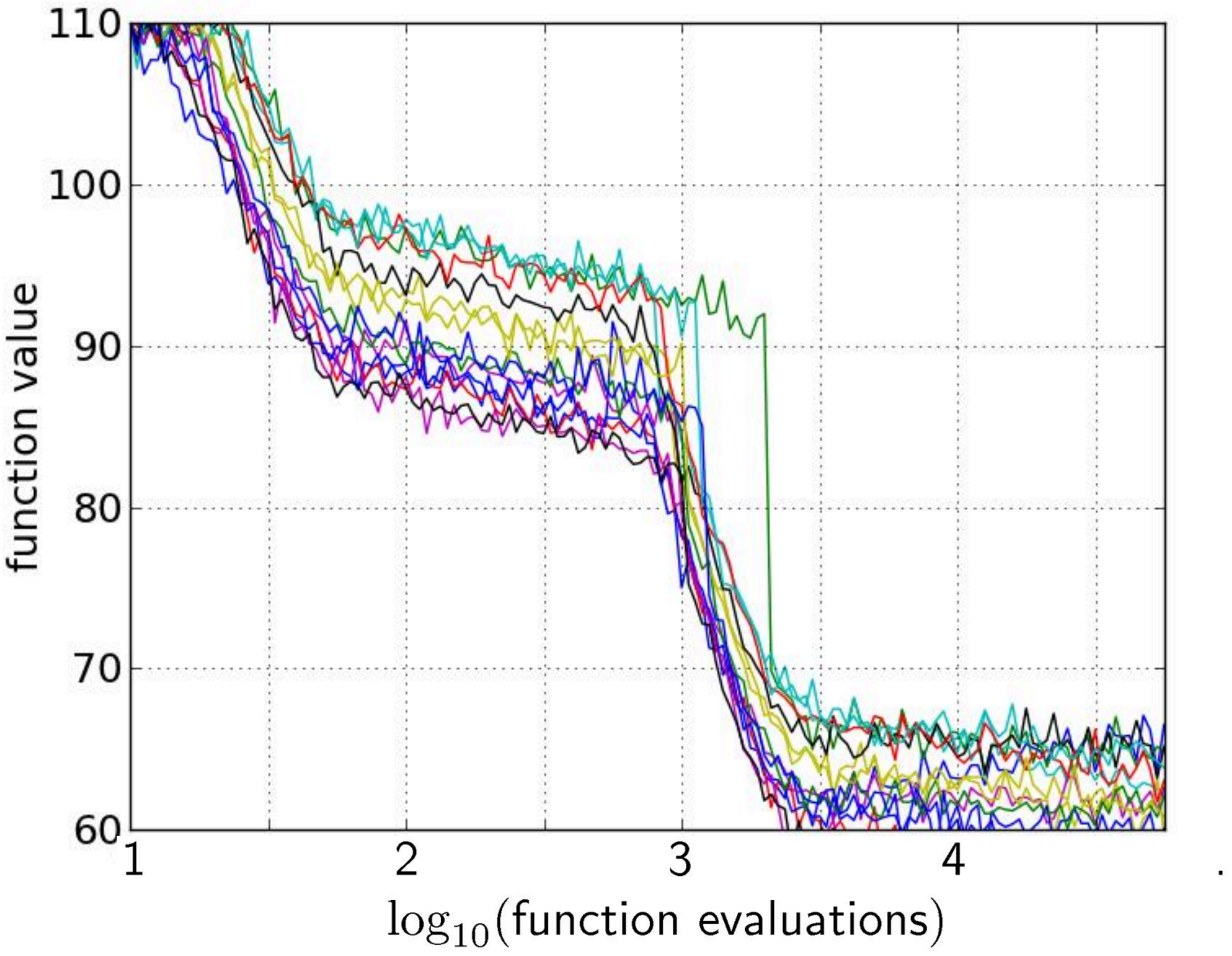


.

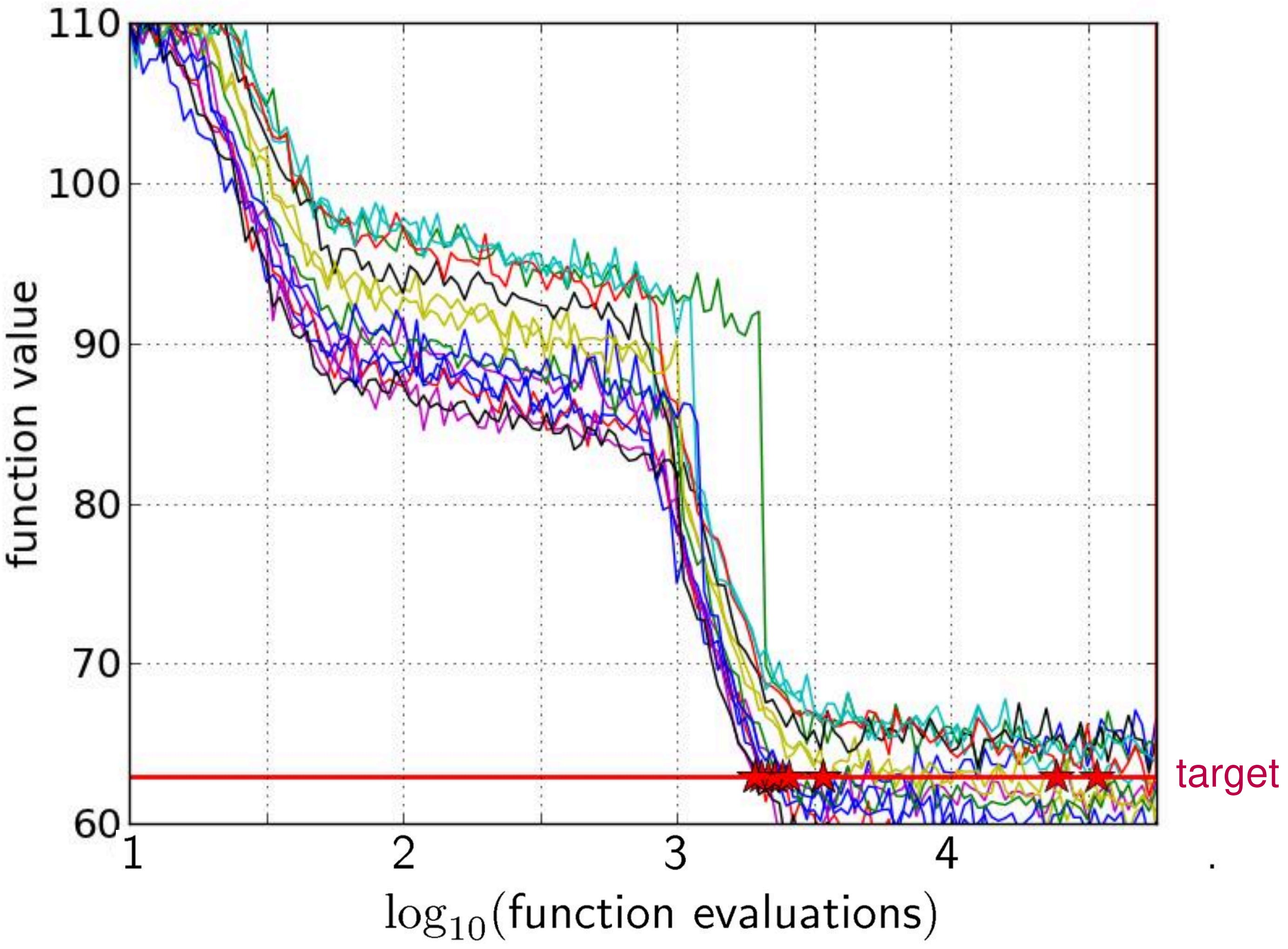
# First Hitting Time is Monotonous



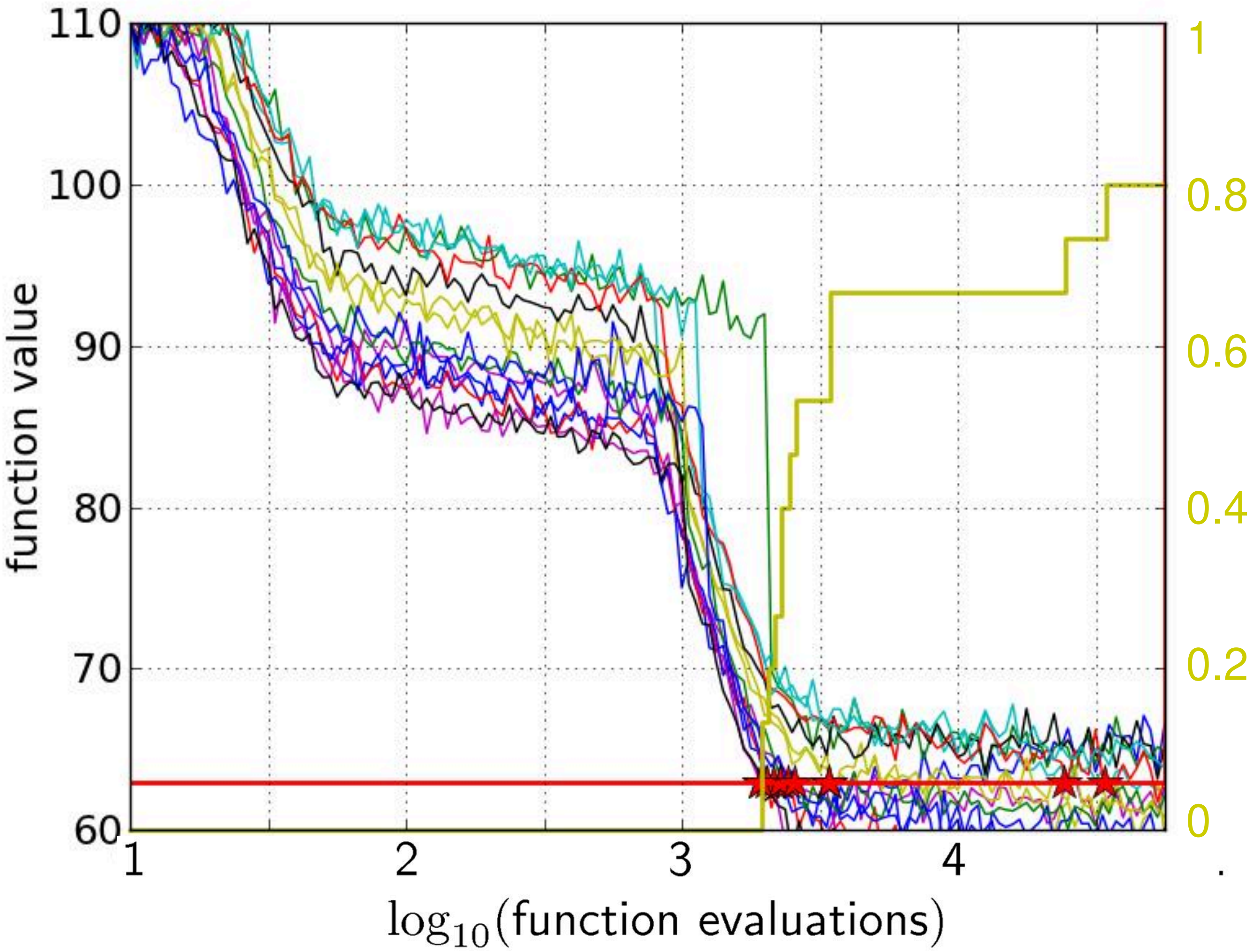
# 15 Runs


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

# 15 Runs ≤ 15 Runtime Data Points



# Empirical Cumulative Distribution Function (ECDF)

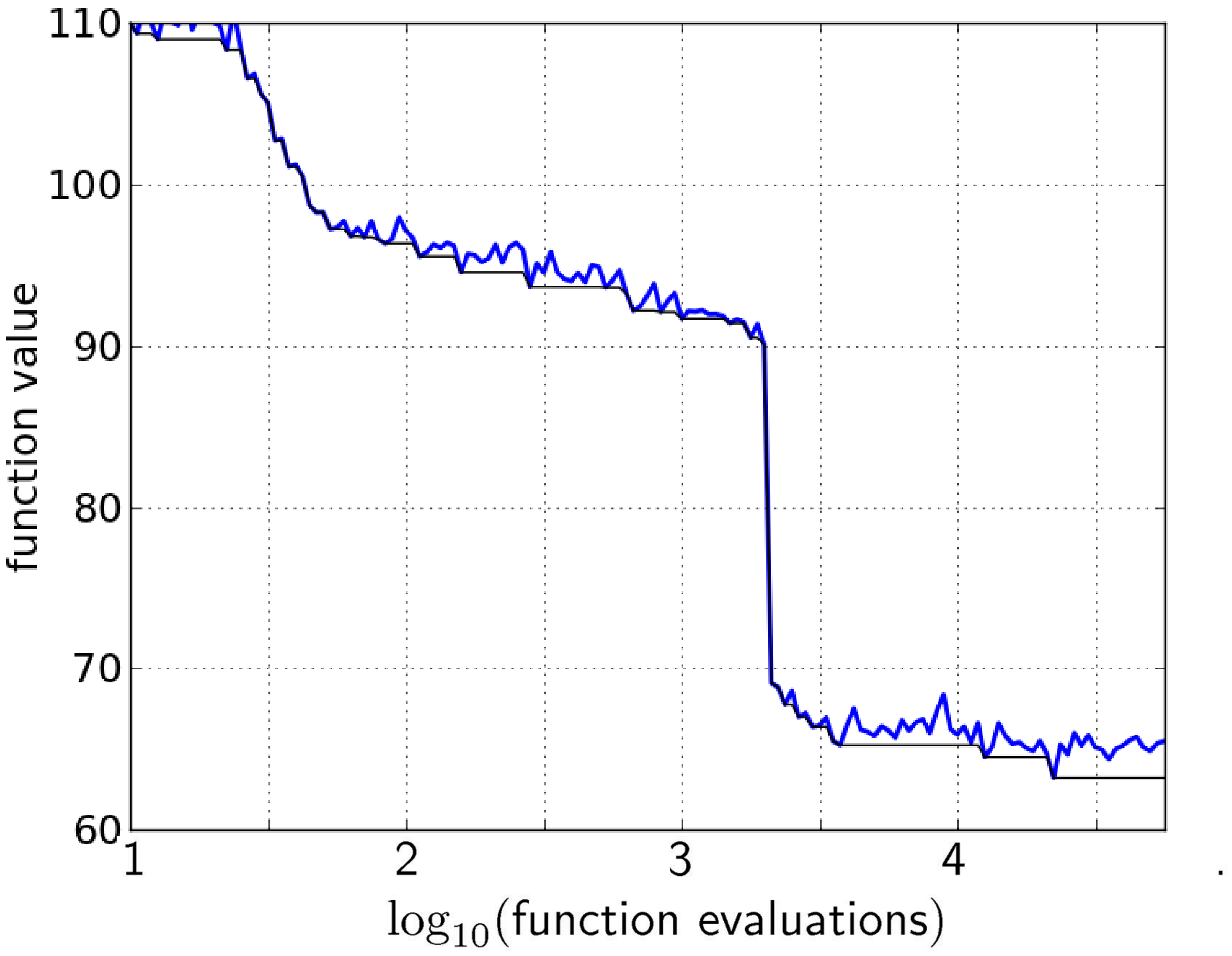


the ECDF of run lengths to reach the target

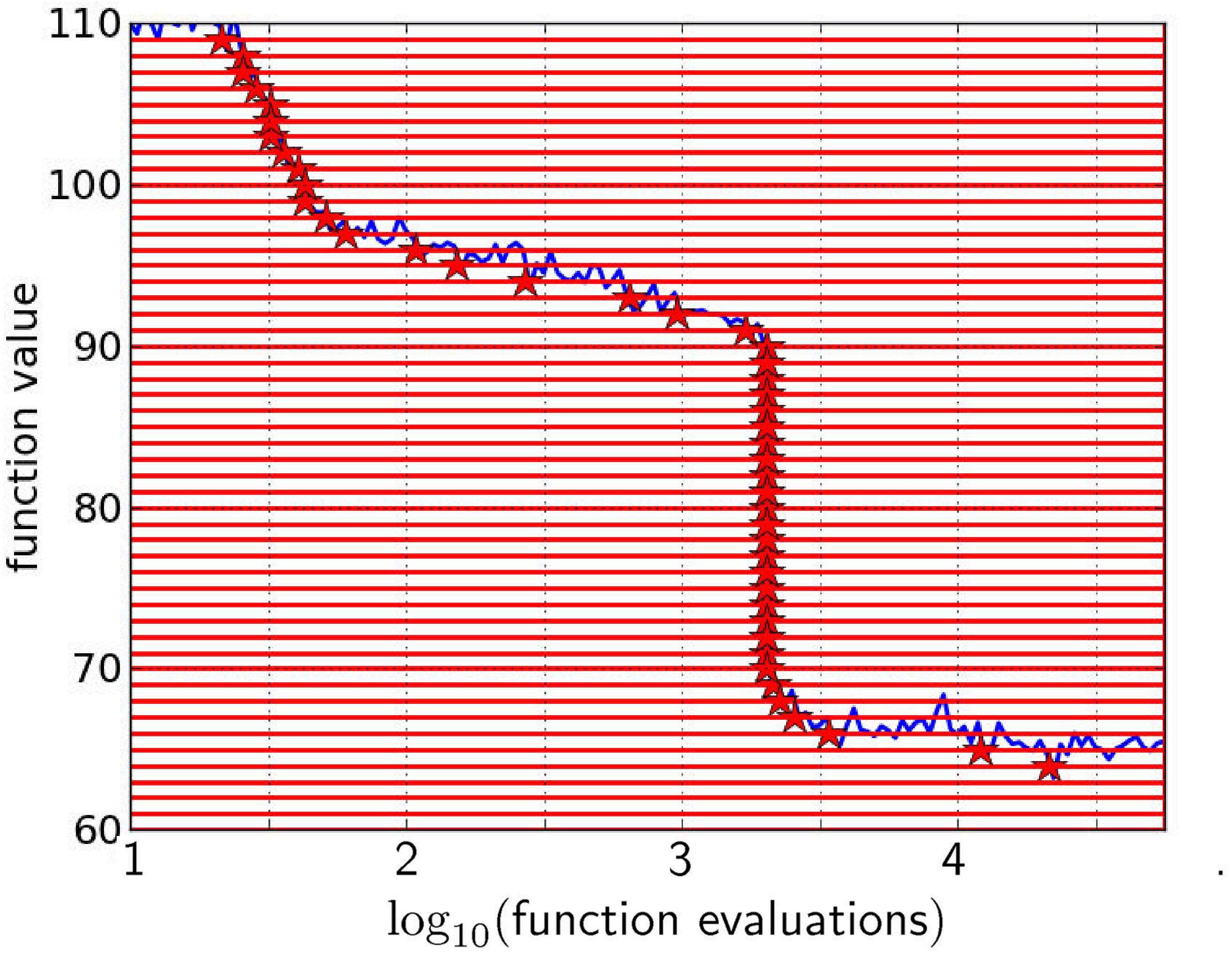
- has for each data point a vertical step of constant size
- displays for each x-value (budget) the count of observations to the left (first hitting times)

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

# Reconstructing A Single Run

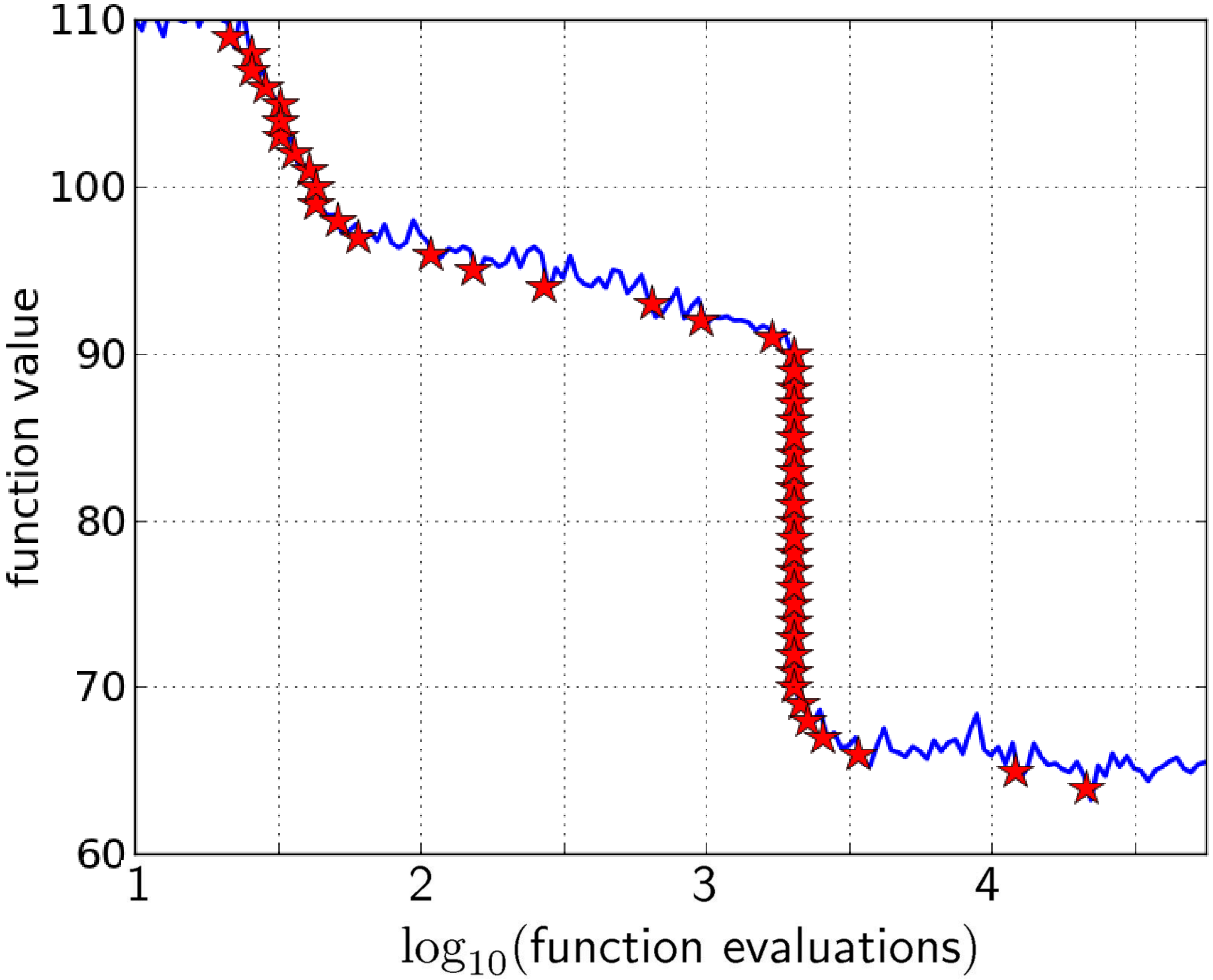


# Reconstructing A Single Run



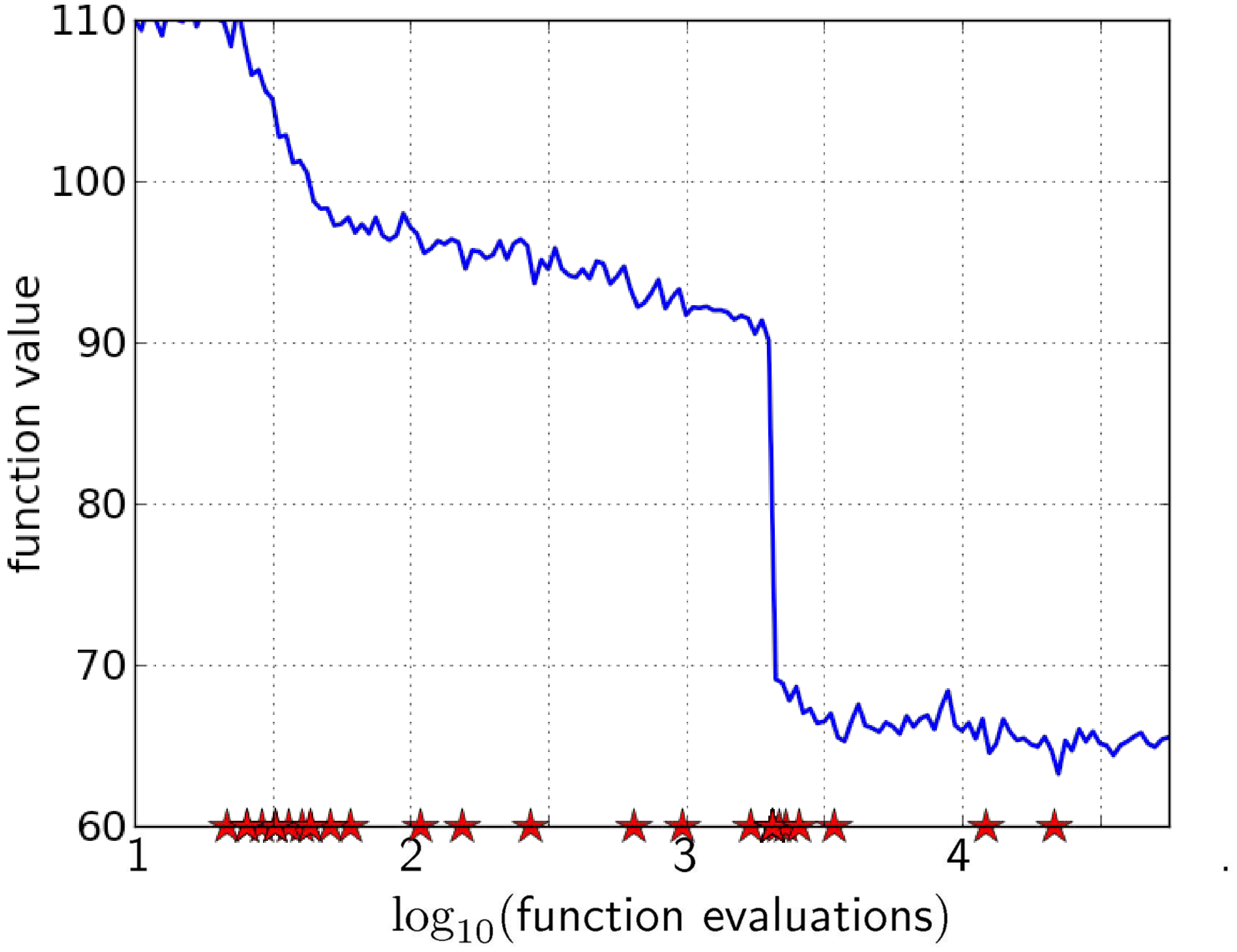
50 equally spaced targets

# Reconstructing A Single Run

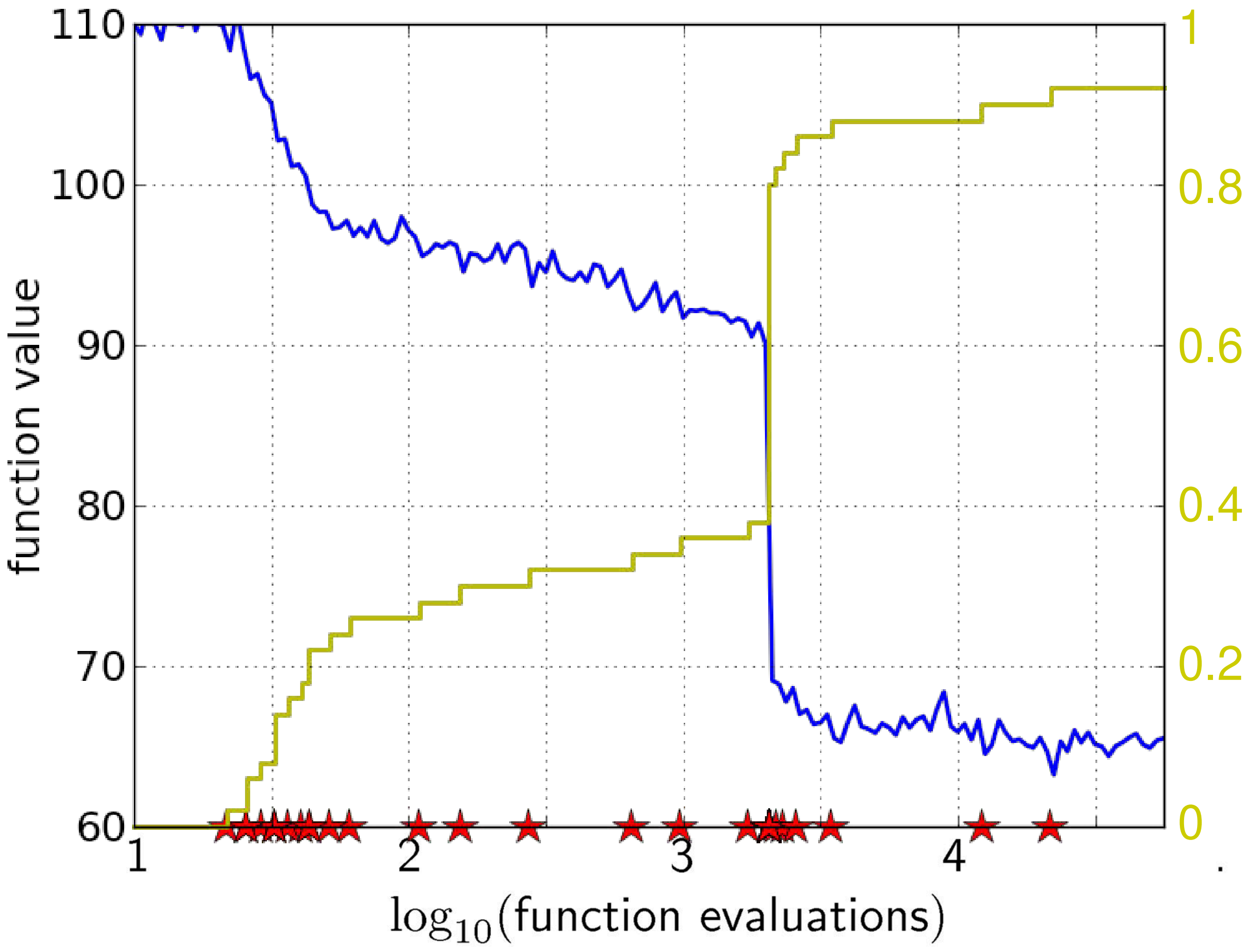


.

# Reconstructing A Single Run

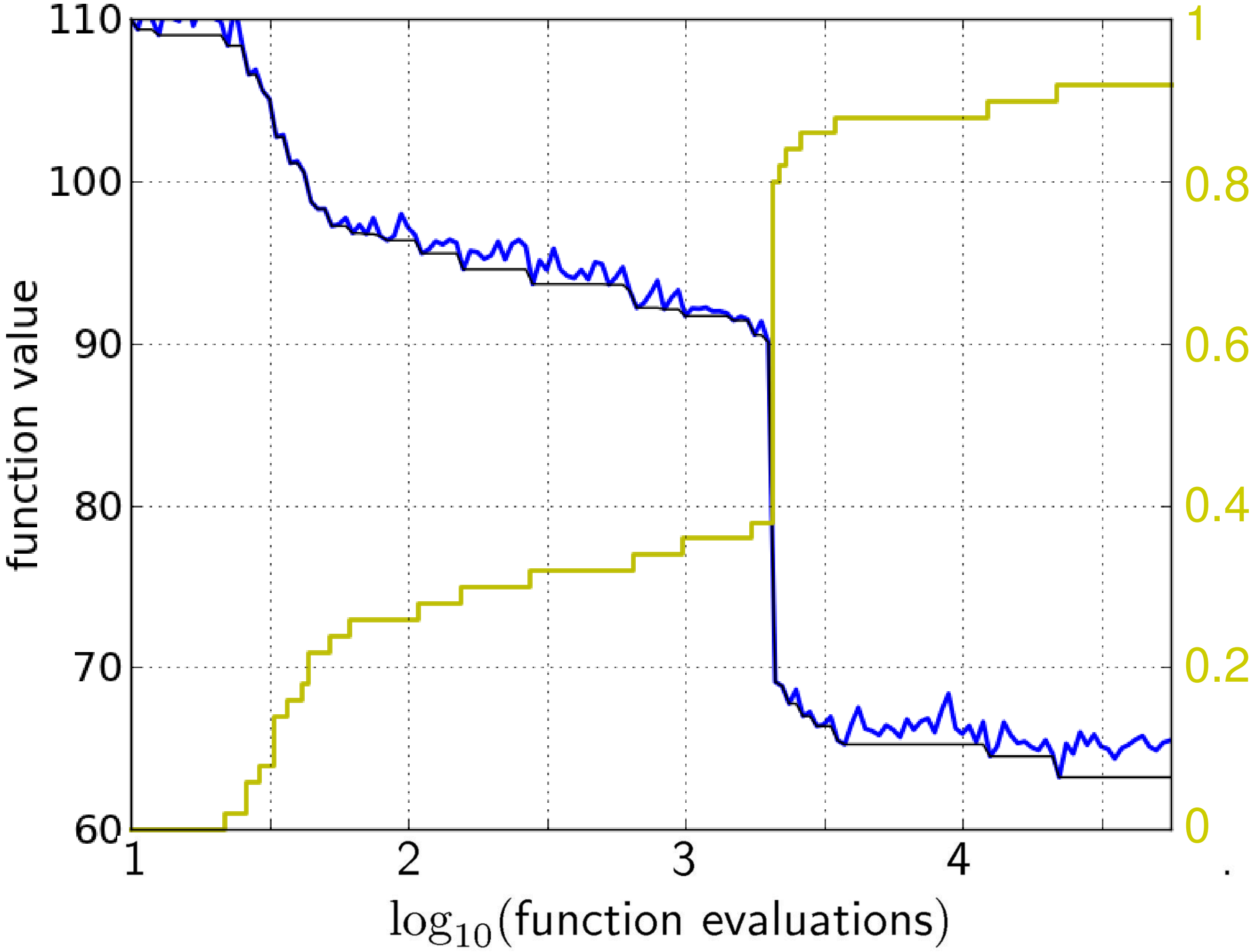


# Reconstructing A Single Run



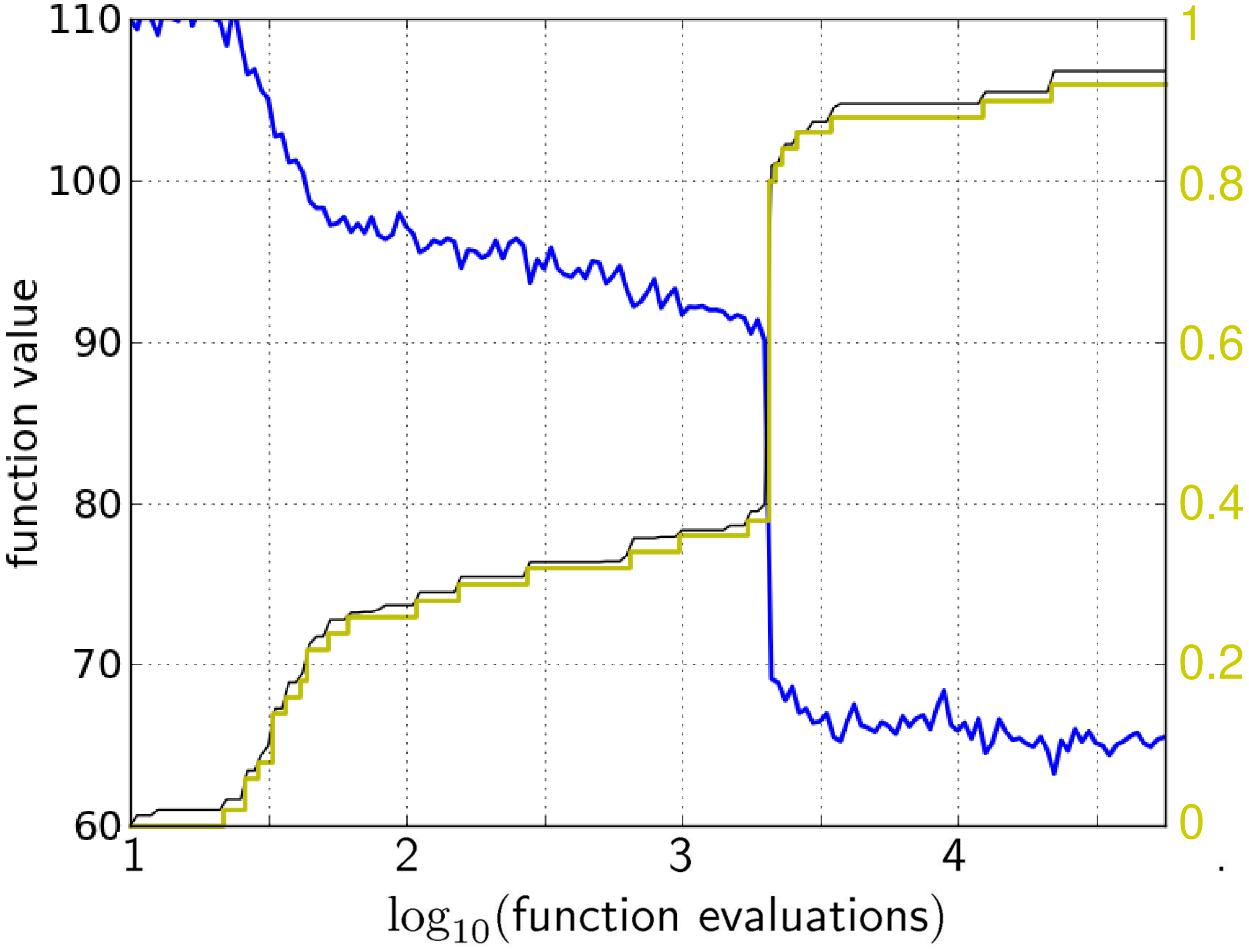
the empirical CDF makes a step for each star, is monotonous and displays for each budget the fraction of targets achieved within the budget

# Reconstructing A Single Run



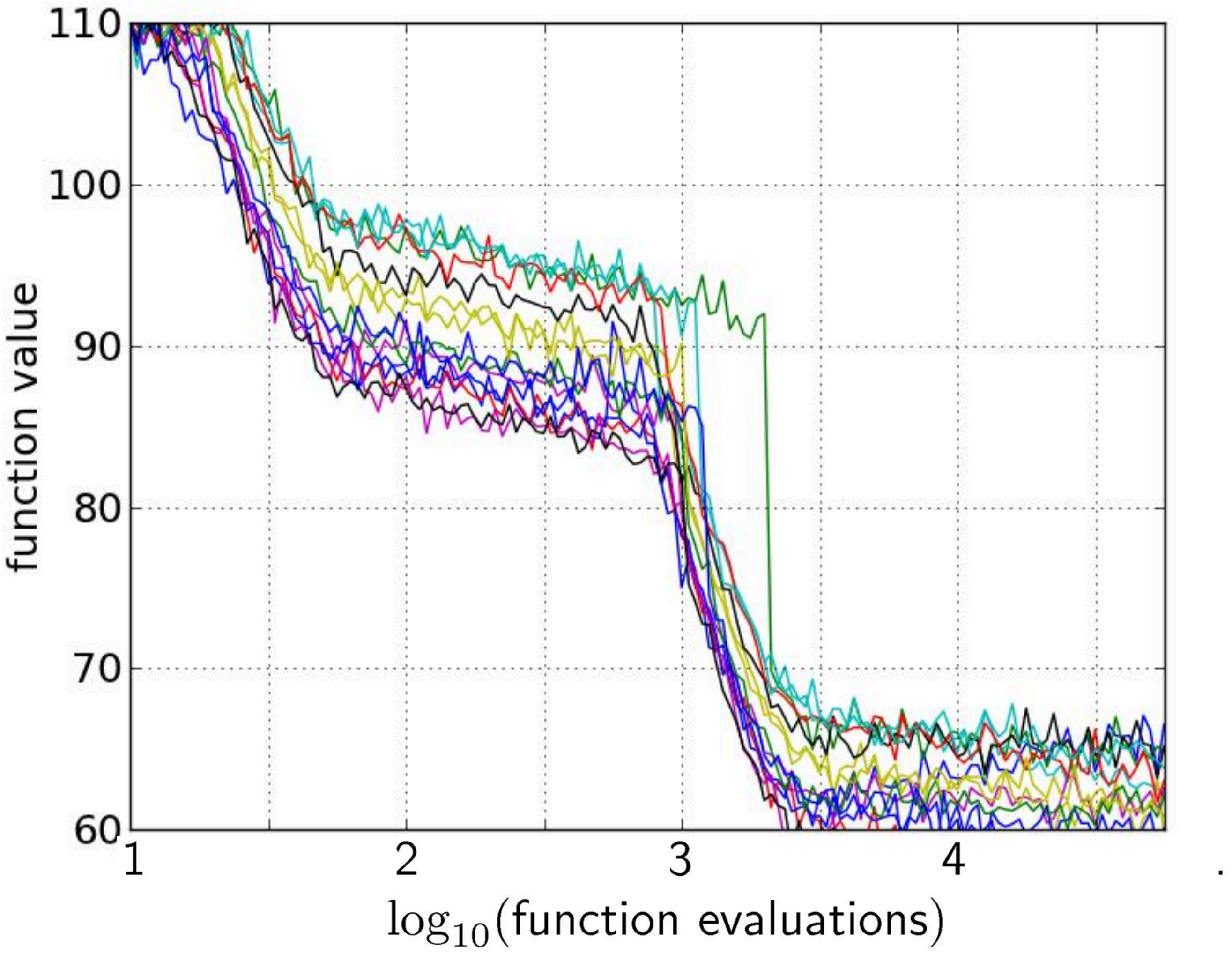
the ECDF recovers the monotonous graph, discretised and flipped

# Reconstructing A Single Run

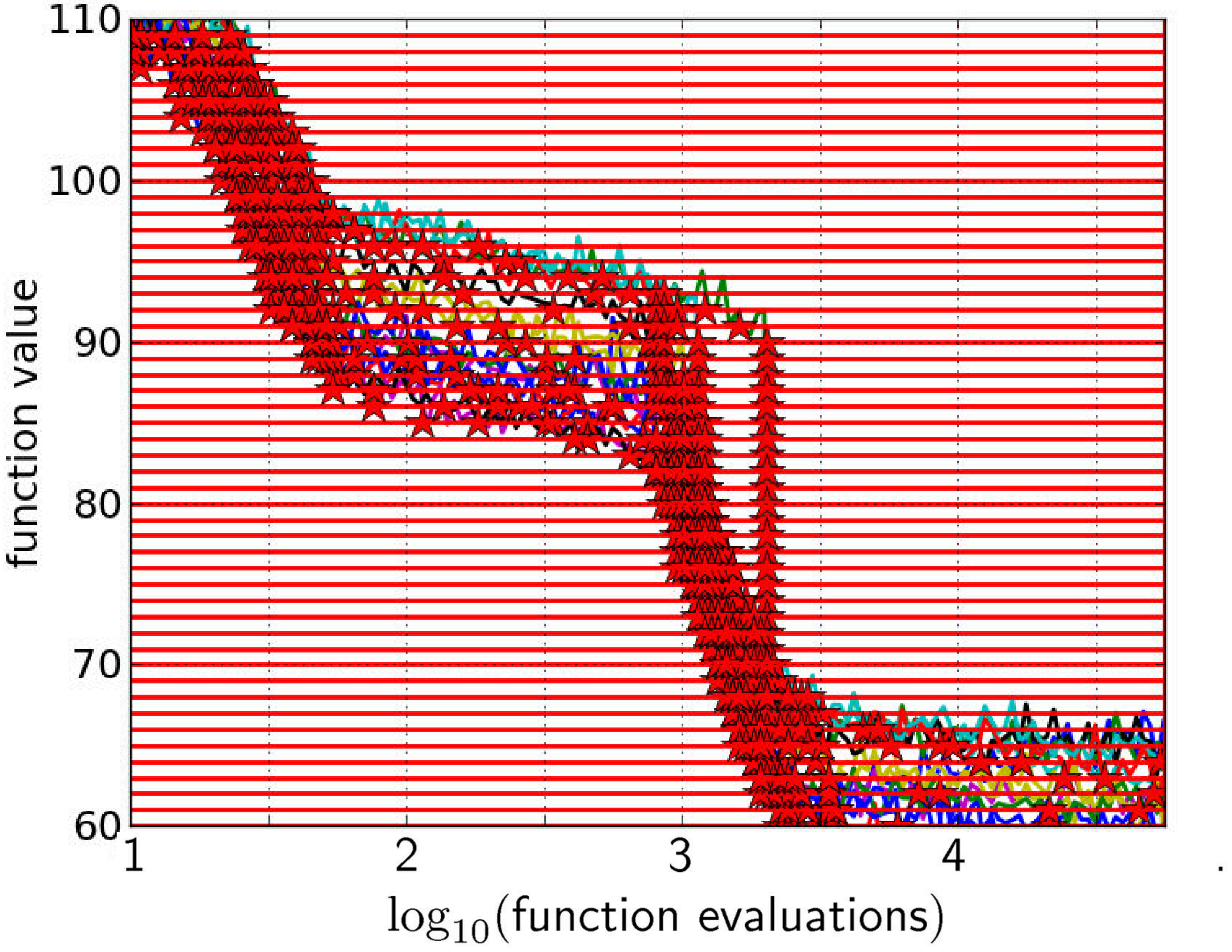


the ECDF recovers the monotonous graph, discretised and flipped

# Aggregation



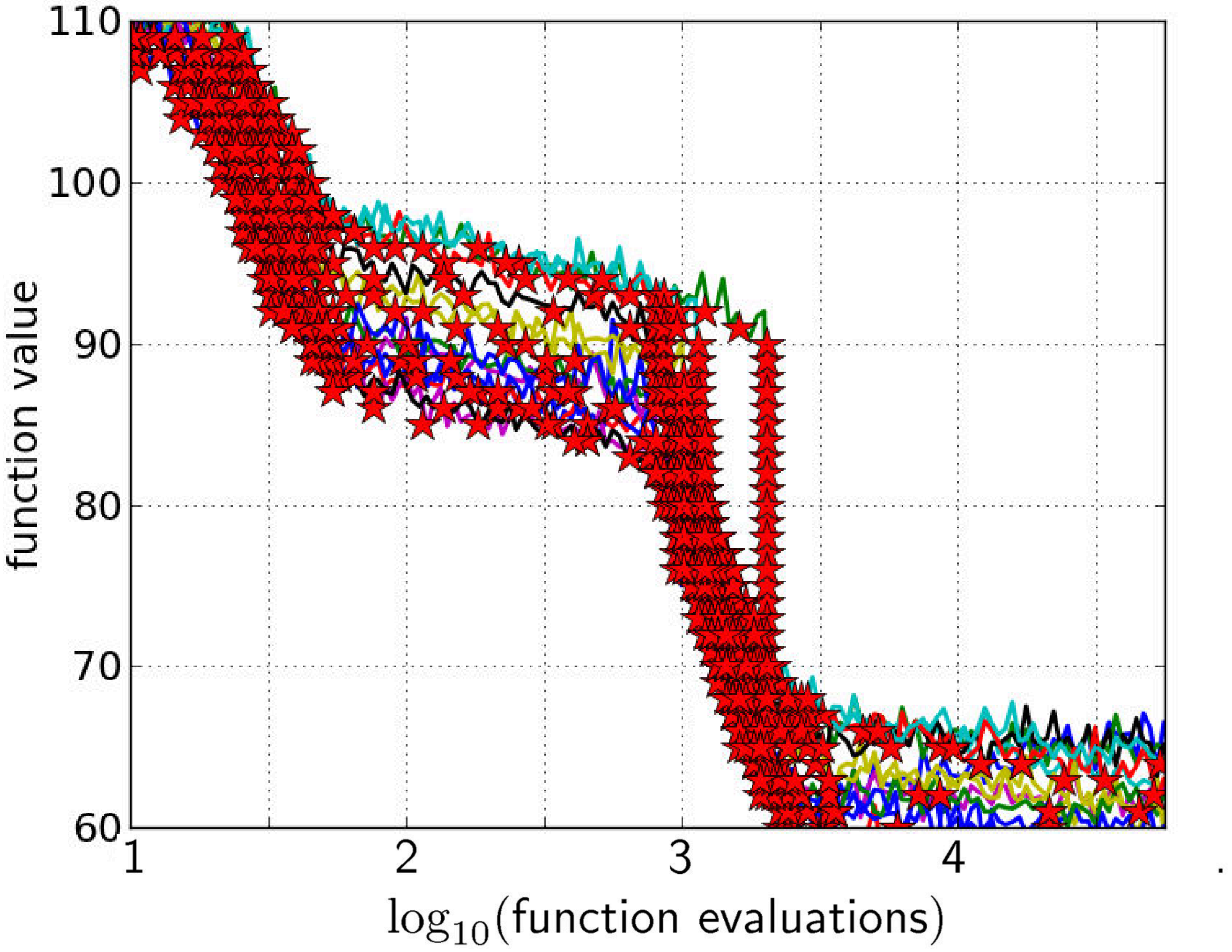
# Aggregation



15 runs

50 targets

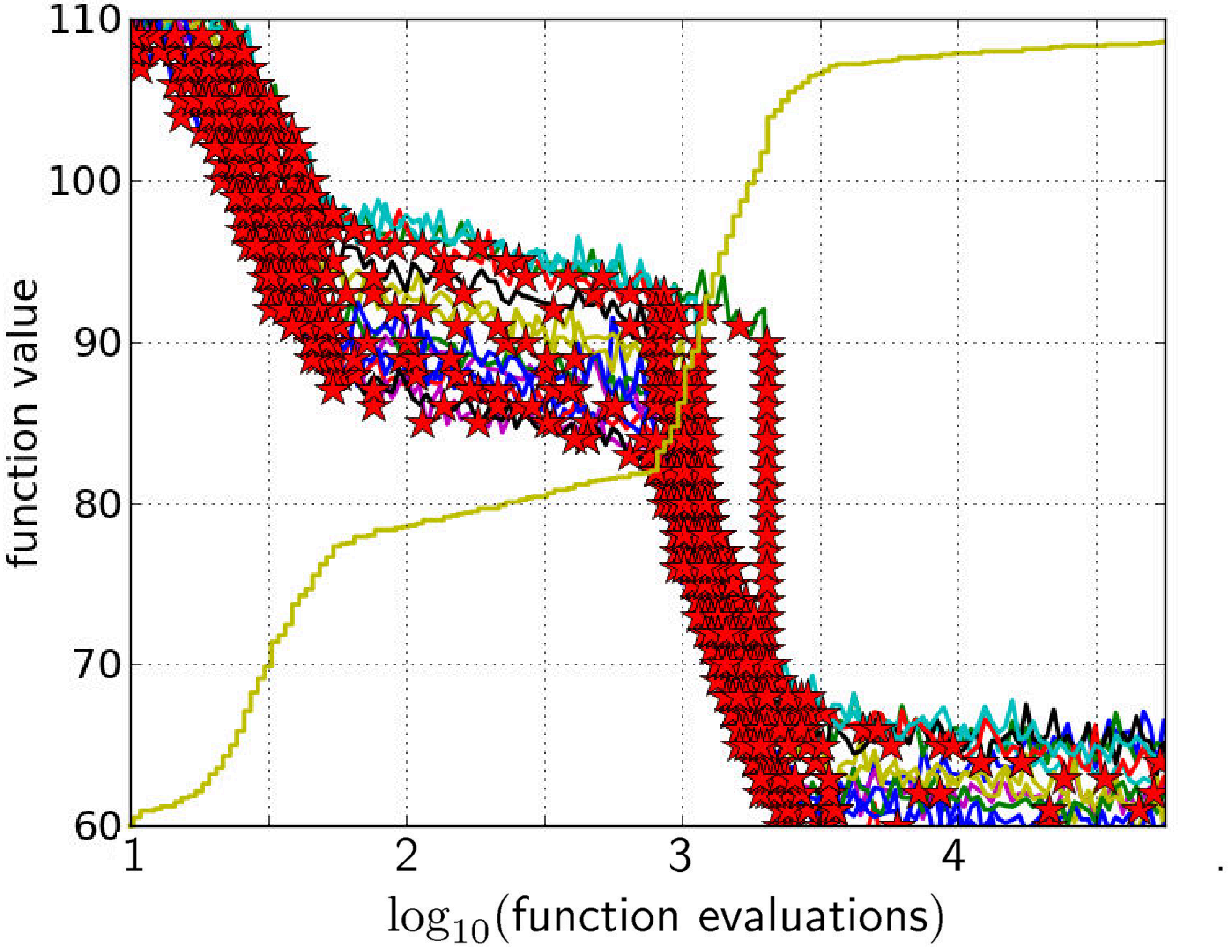
# Aggregation



15 runs

50 targets

# Aggregation

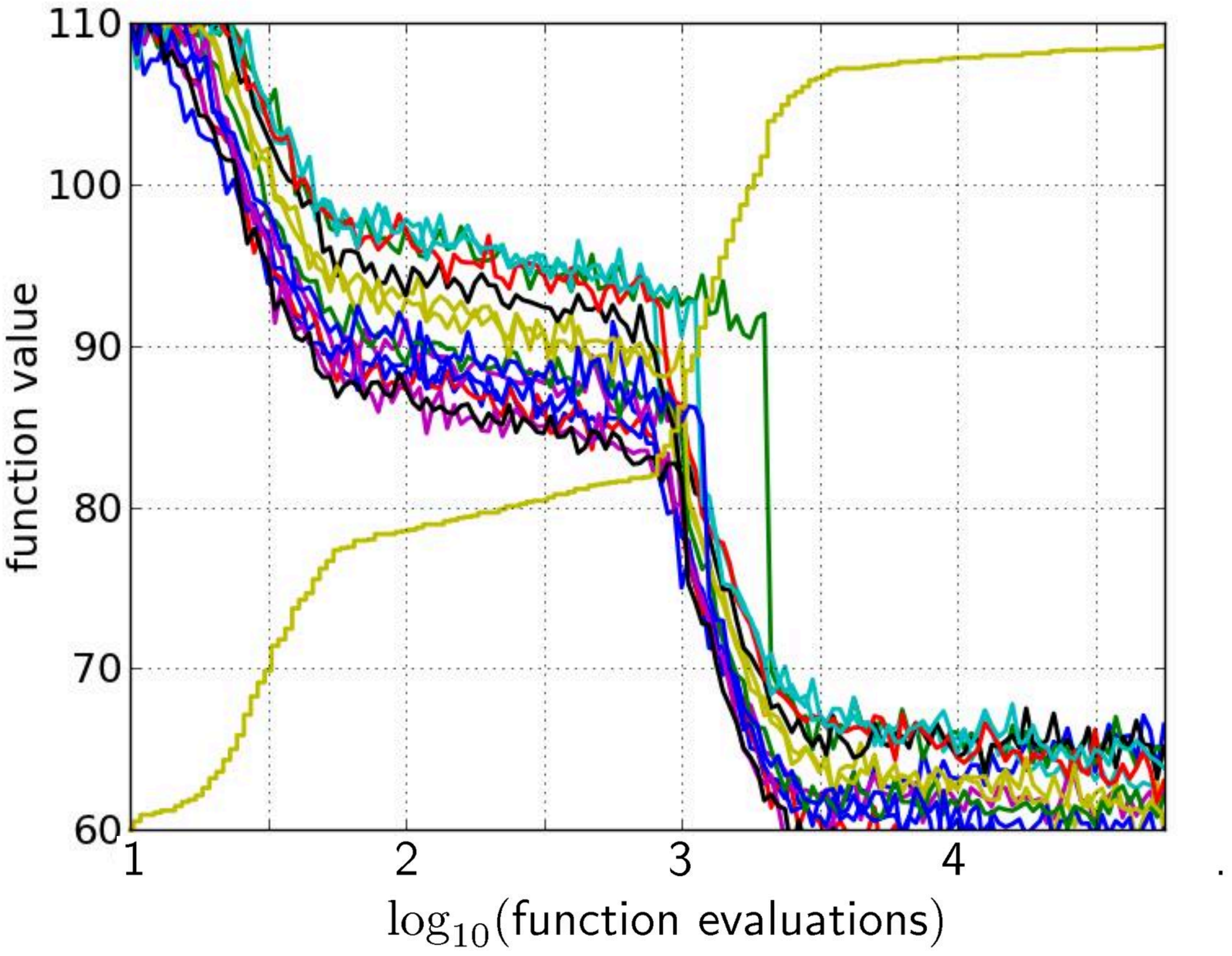


15 runs

50 targets

ECDF with
750 steps

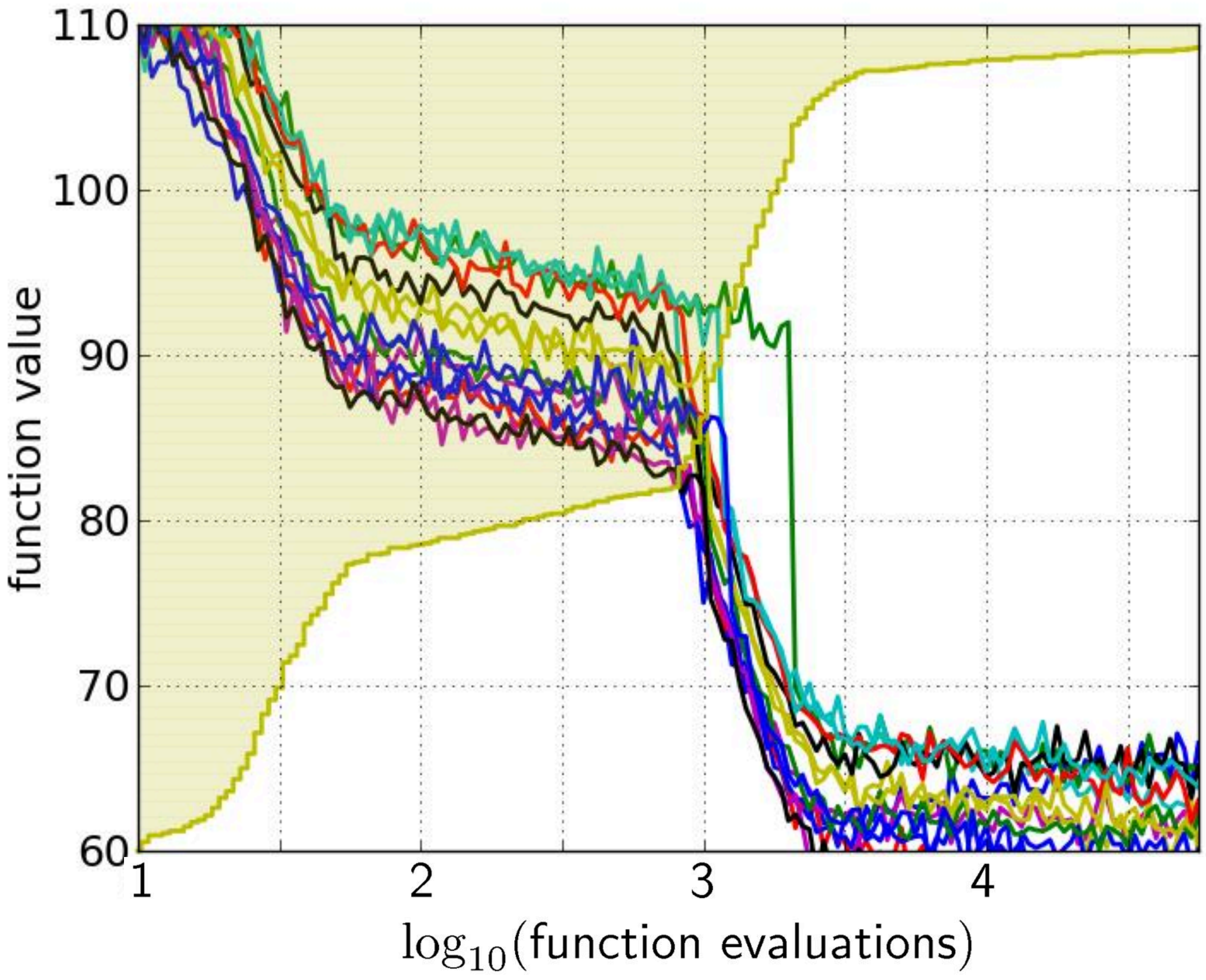
# Aggregation



50 targets from 15 runs

...integrated in a single graph

# Interpretation



50 targets from 15 runs

...integrated in a single graph

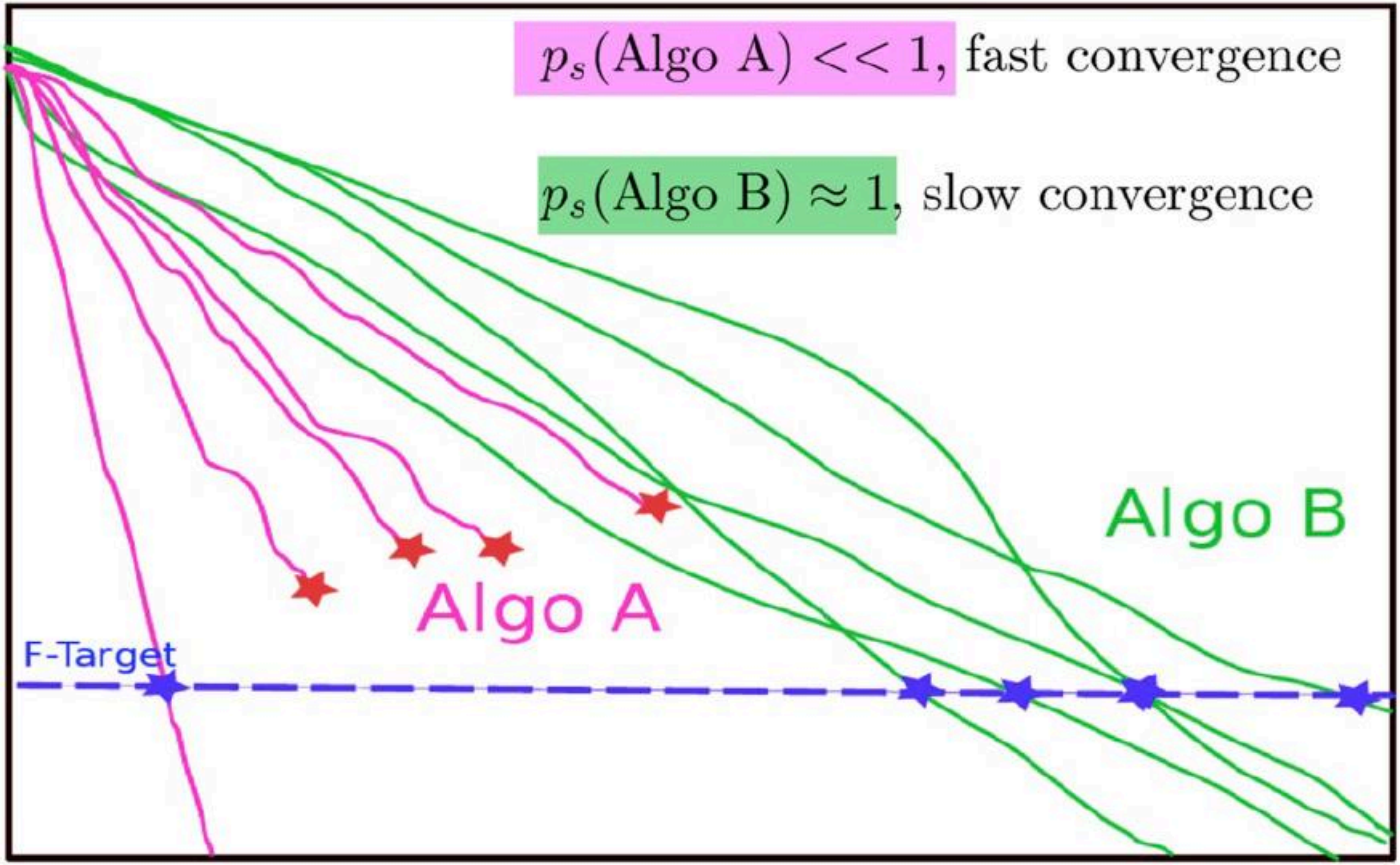
area over the ECDF curve

=

average log runtime

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

# Fixed-target: Measuring Runtime

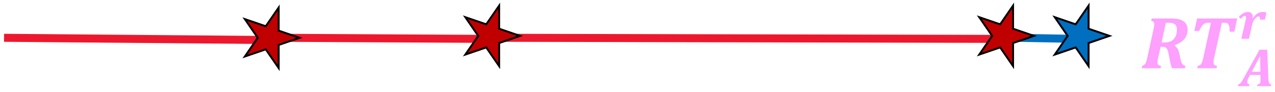


$p_s$(Algo A) $<< 1$, fast convergence

$p_s$(Algo B) $\approx 1$, slow convergence

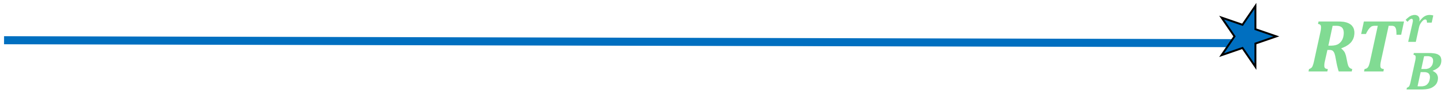
# Fixed-target: Measuring Runtime of Restarted Algo

- Algo Restart A:



$p_s$(Algo Restart A) = 1

- Algo Restart B:



$p_s$(Algo Restart A) = 1

# Fixed-target: Measuring Runtime of Restarted Algo

- 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):

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

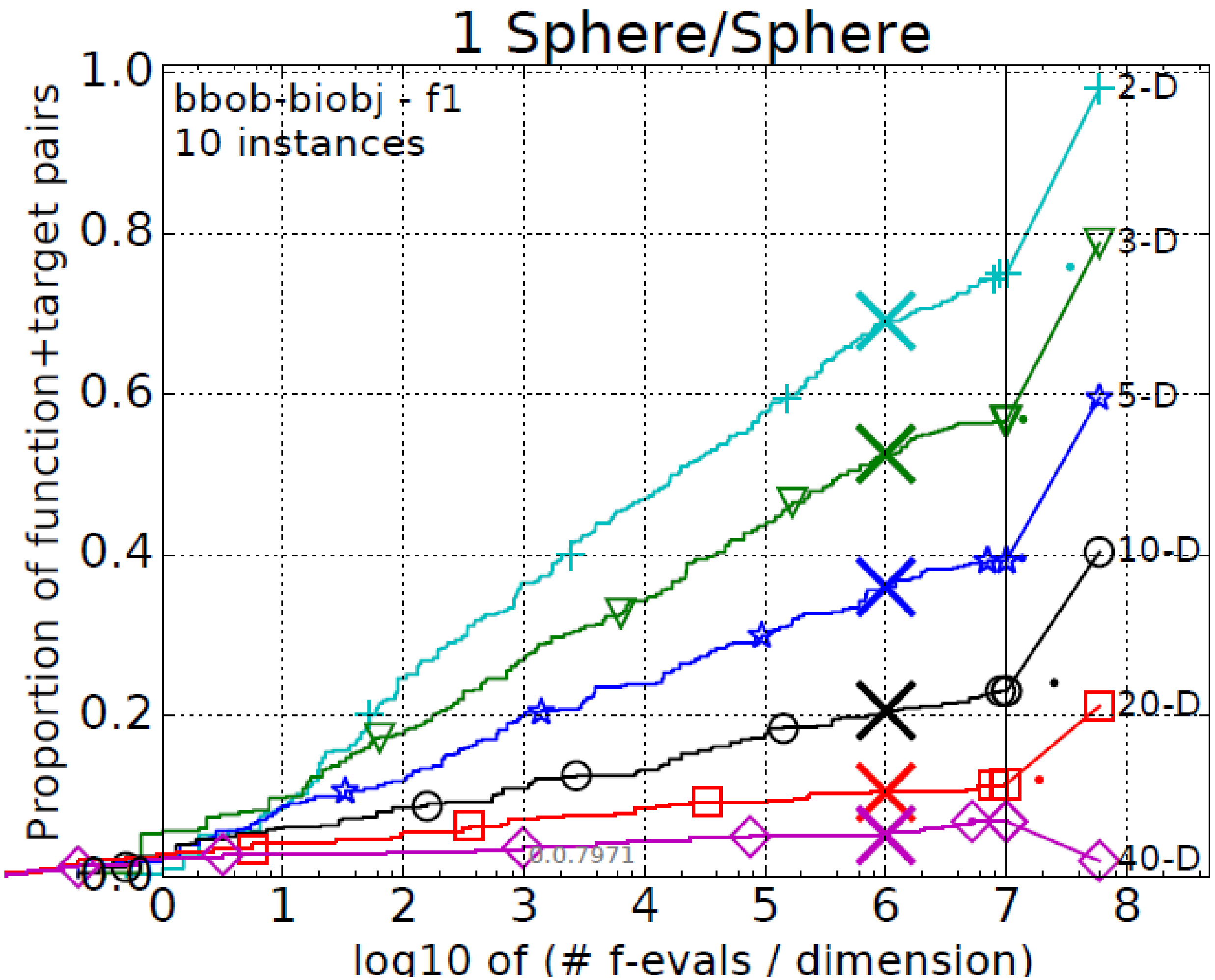
$$\widehat{RT_{unsucc}} = \text{Average evals of unsuccessful runs}$$

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

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

# ECDFs with Simulated Restarts

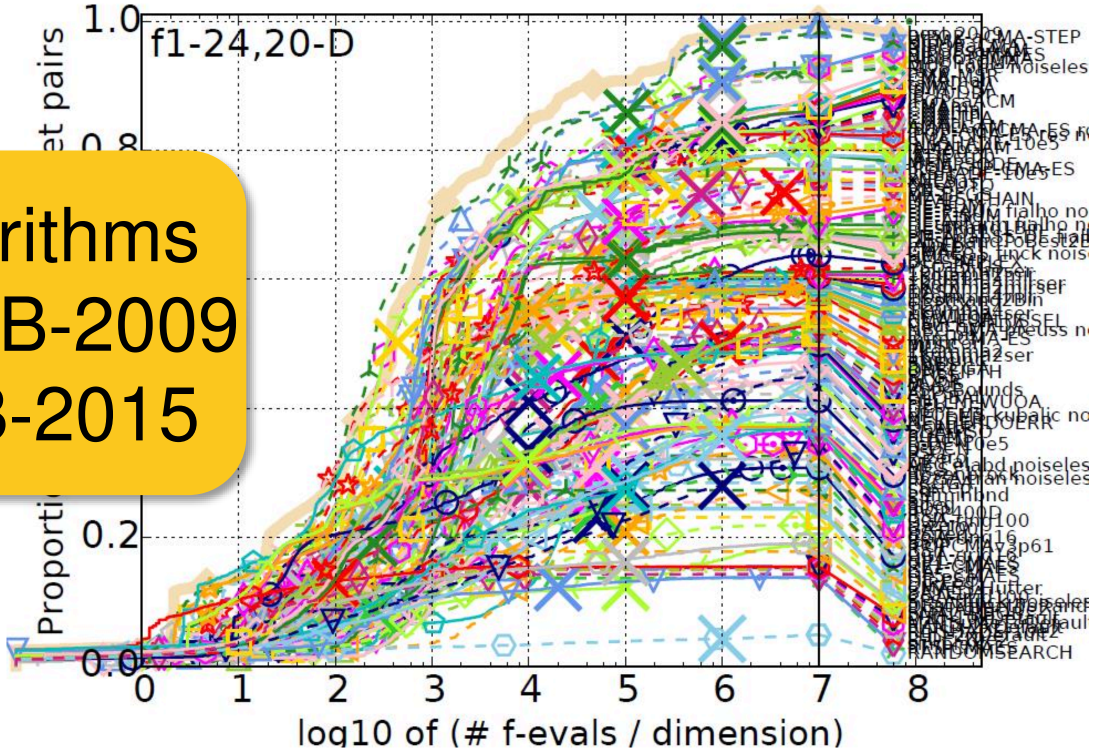
What we typically plot are ECDFs of the simulated restarted algorithms:



# Worth to Note: ECDFs in COCO

In COCO, ECDF graphs

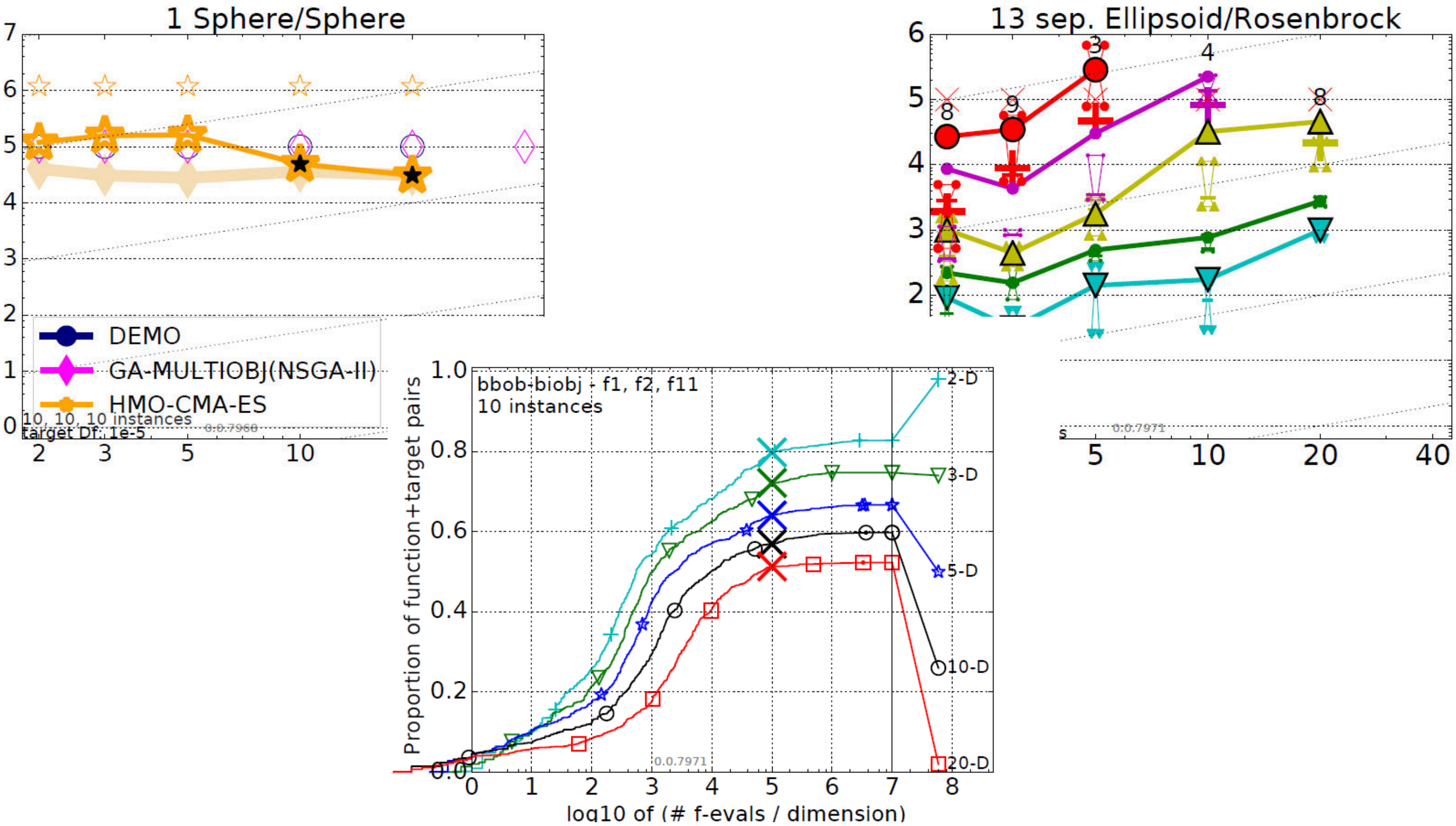
- never aggregate over dimension
  - but often over targets and functions
- can show data of more than 1 algorithm at a time



150 algorithms from BBOB-2009 till BBOB-2015

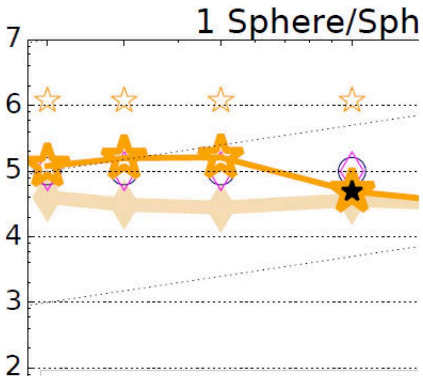
# More Automated Plots...

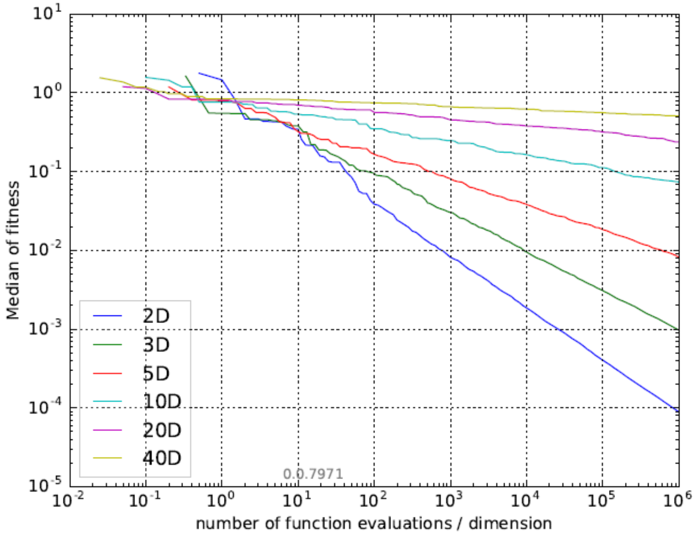
...might come back to it in detail later today



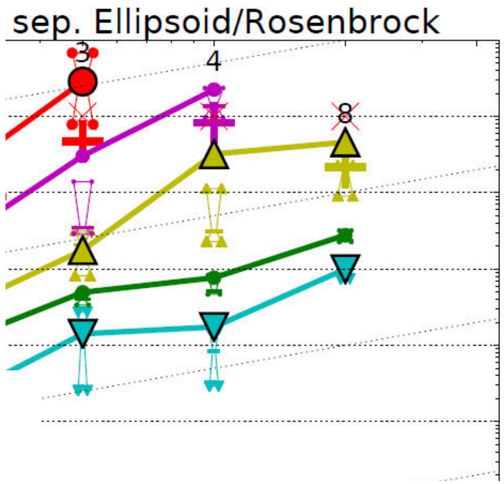
# More Automated Plots...

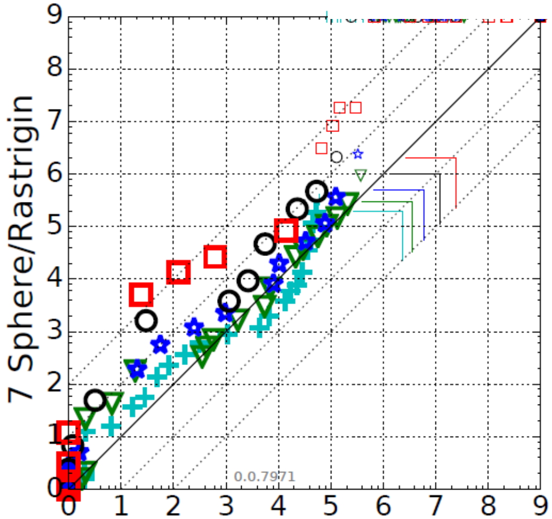
...might come

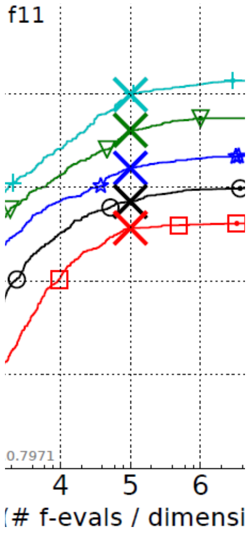


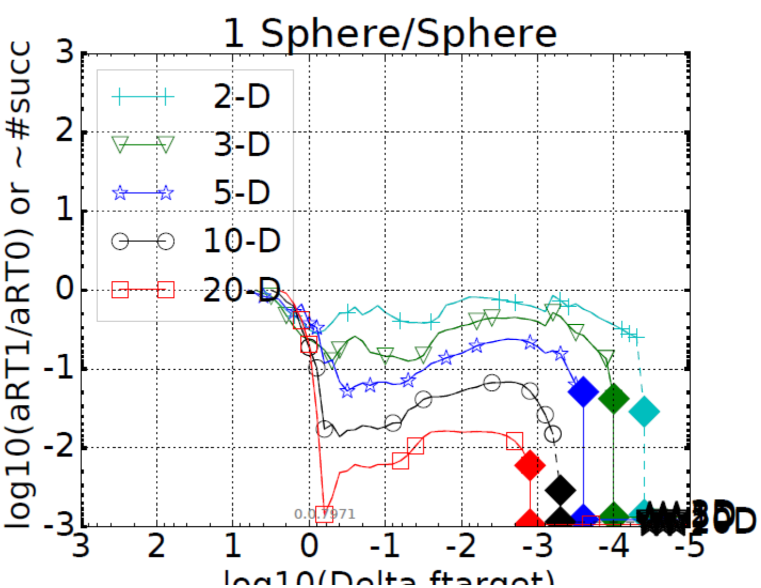


today





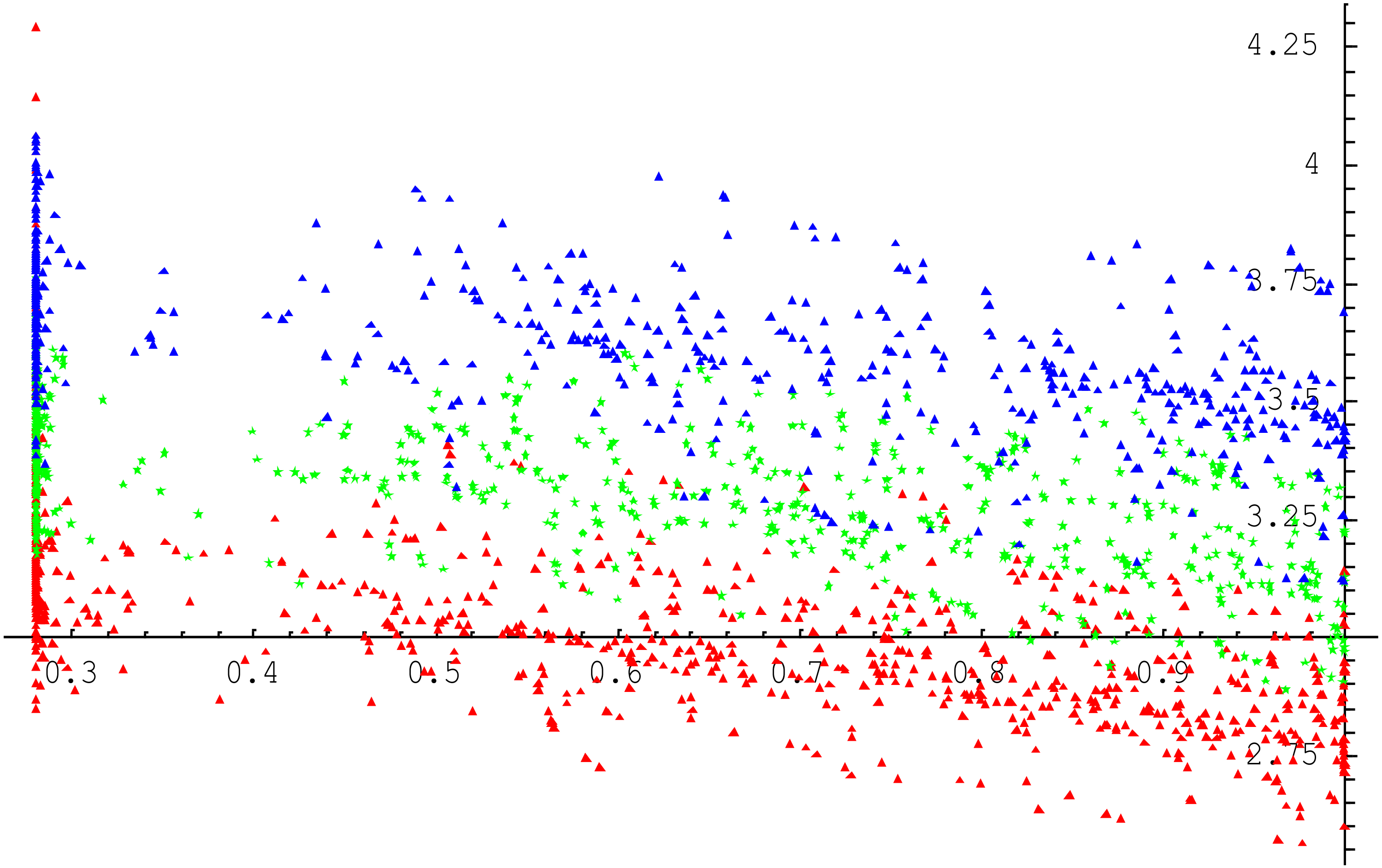




how to do benchmarking in the multiobjective case?

# Once Upon a Time...

... multiobjective EAs were mainly compared visually:

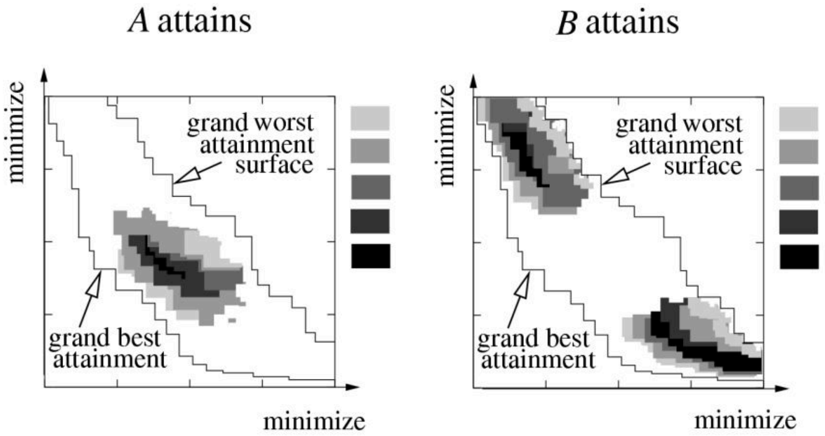


ZDT6 benchmark problem: IBEA, SPEA2, NSGA-II

# Two Main Approaches for Empirical Studies

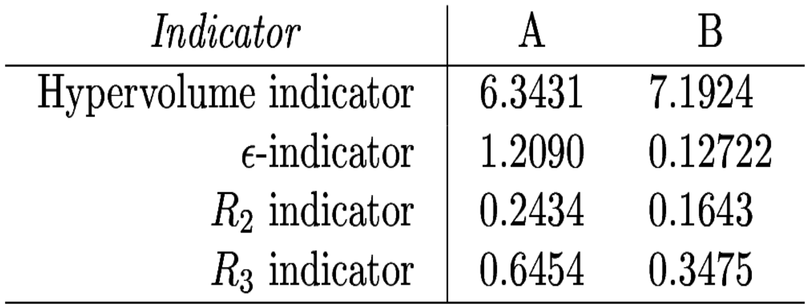
## Attainment function approach

- applies statistical tests directly to the approximation set
- detailed information about how and where performance differences occur



## Quality indicator approach

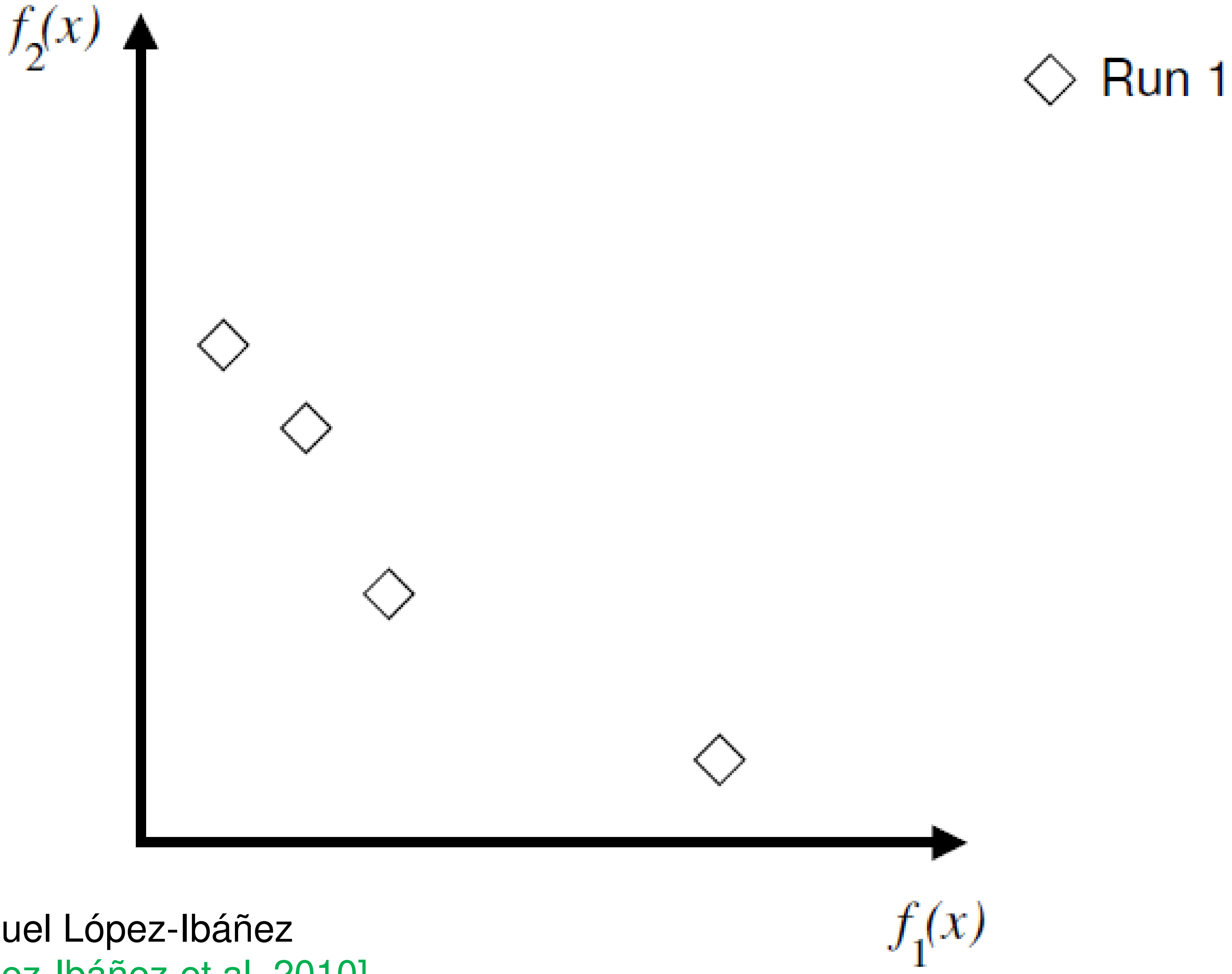
- reduces each approximation set to a single quality value
- applies statistical tests to the quality values



| *Indicator* | A | B |
|---|---|---|
| Hypervolume indicator | 6.3431 | 7.1924 |
| $\epsilon$-indicator | 1.2090 | 0.12722 |
| $R_2$ indicator | 0.2434 | 0.1643 |
| $R_3$ indicator | 0.6454 | 0.3475 |

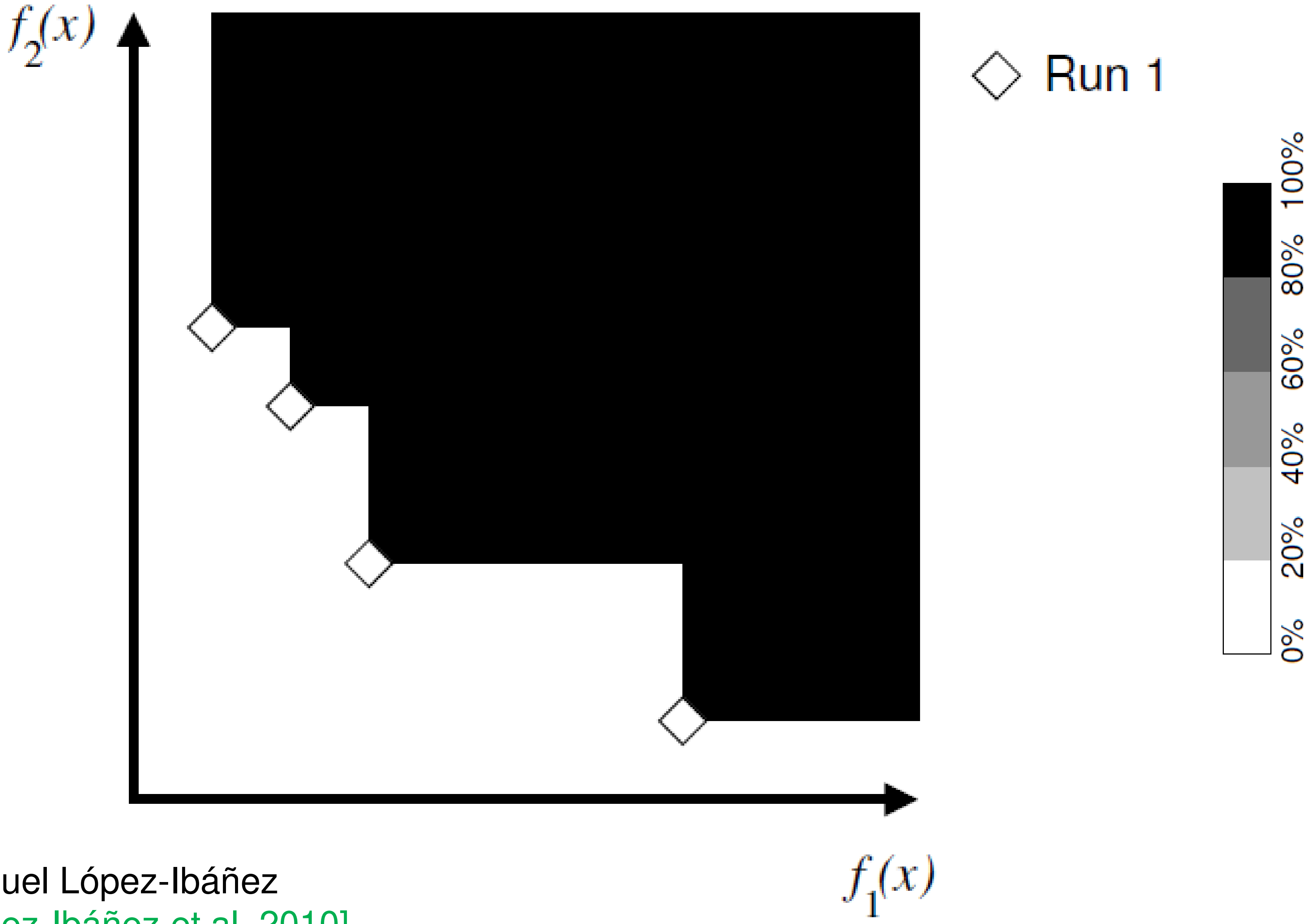
see e.g. [Zitzler et al. 2003]

# Empirical Attainment Functions: Idea



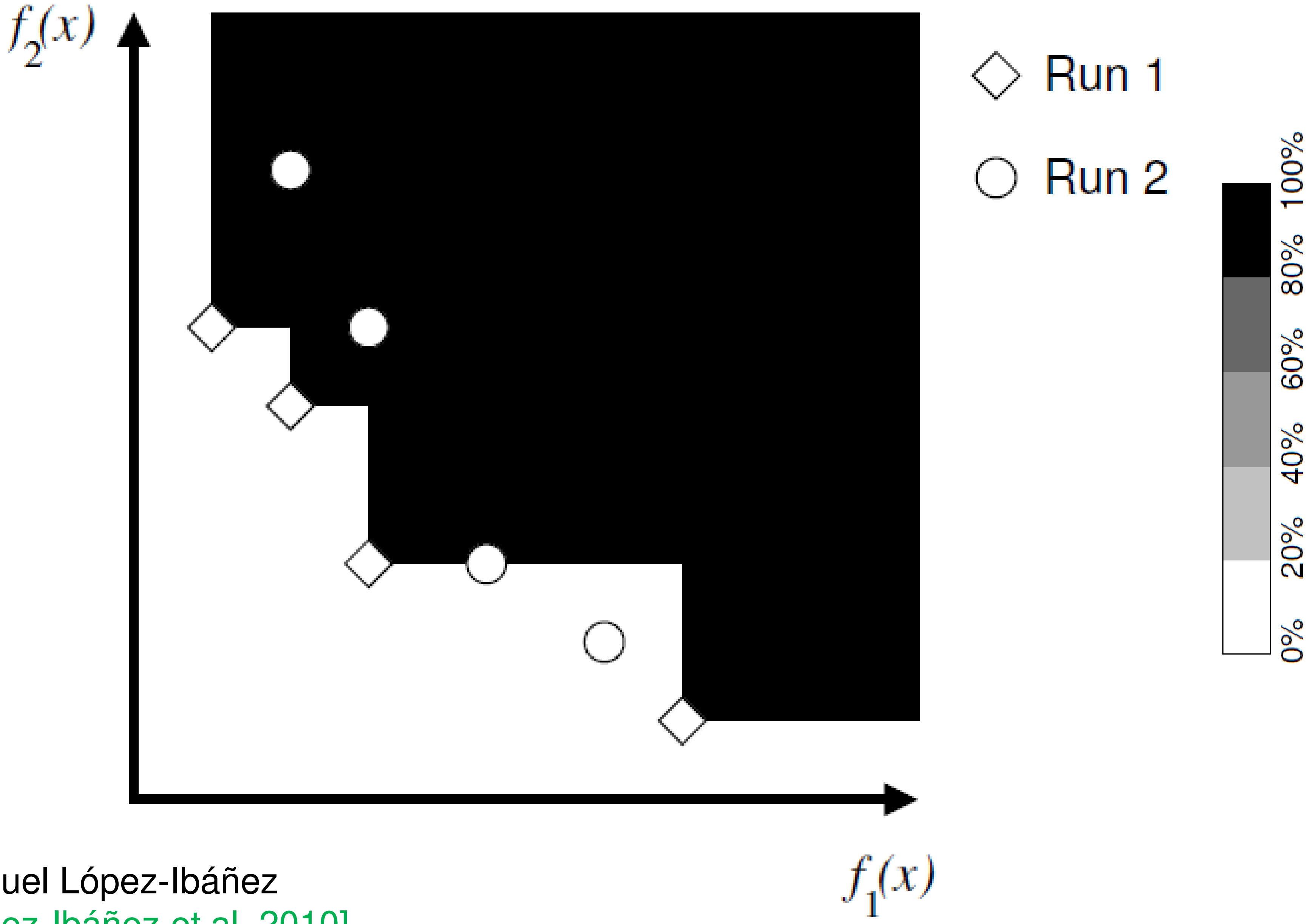
© Manuel López-Ibáñez
[López-Ibáñez et al. 2010]

# Empirical Attainment Functions: Idea



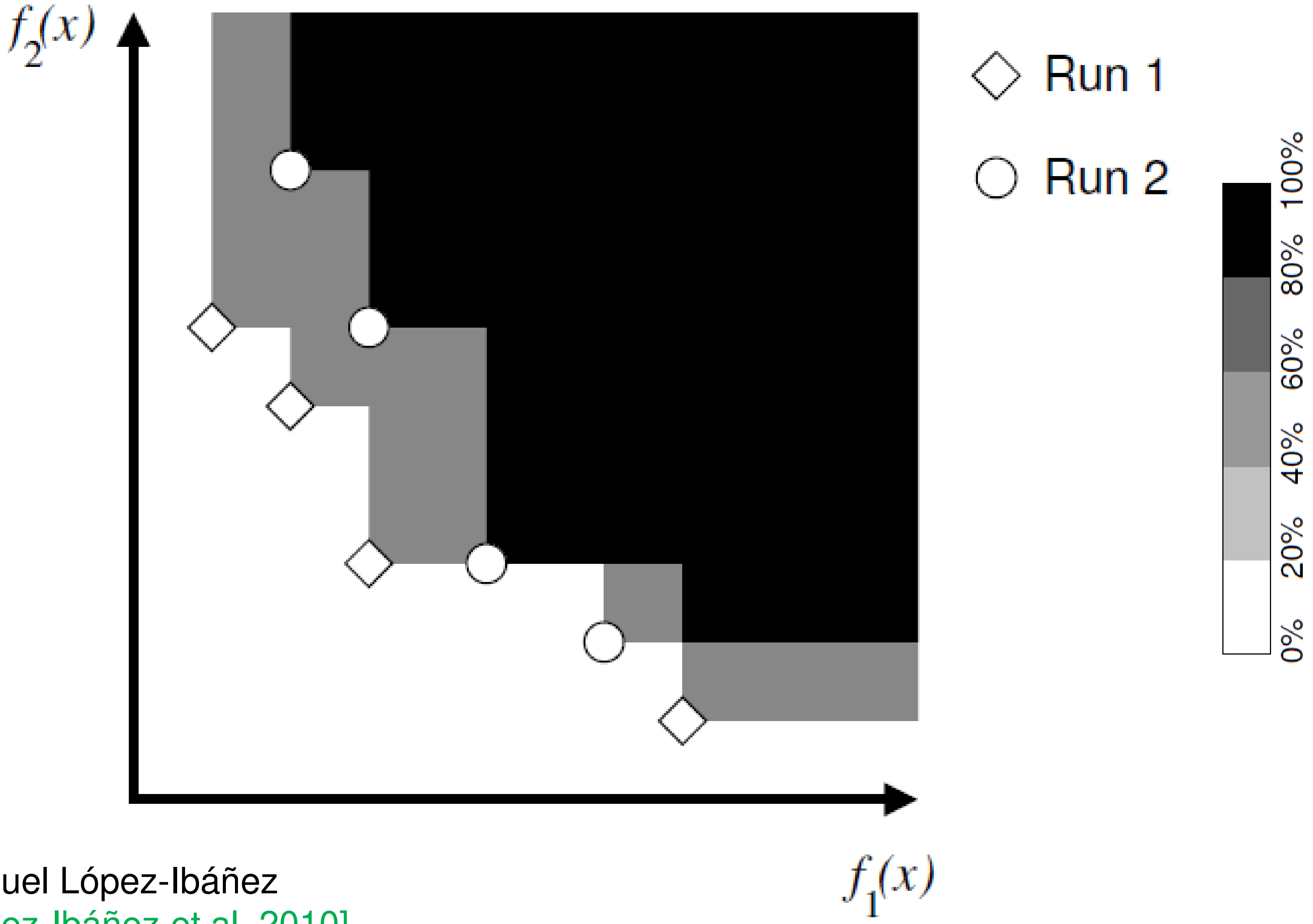
© Manuel López-Ibáñez
[López-Ibáñez et al. 2010]

# Empirical Attainment Functions: Idea



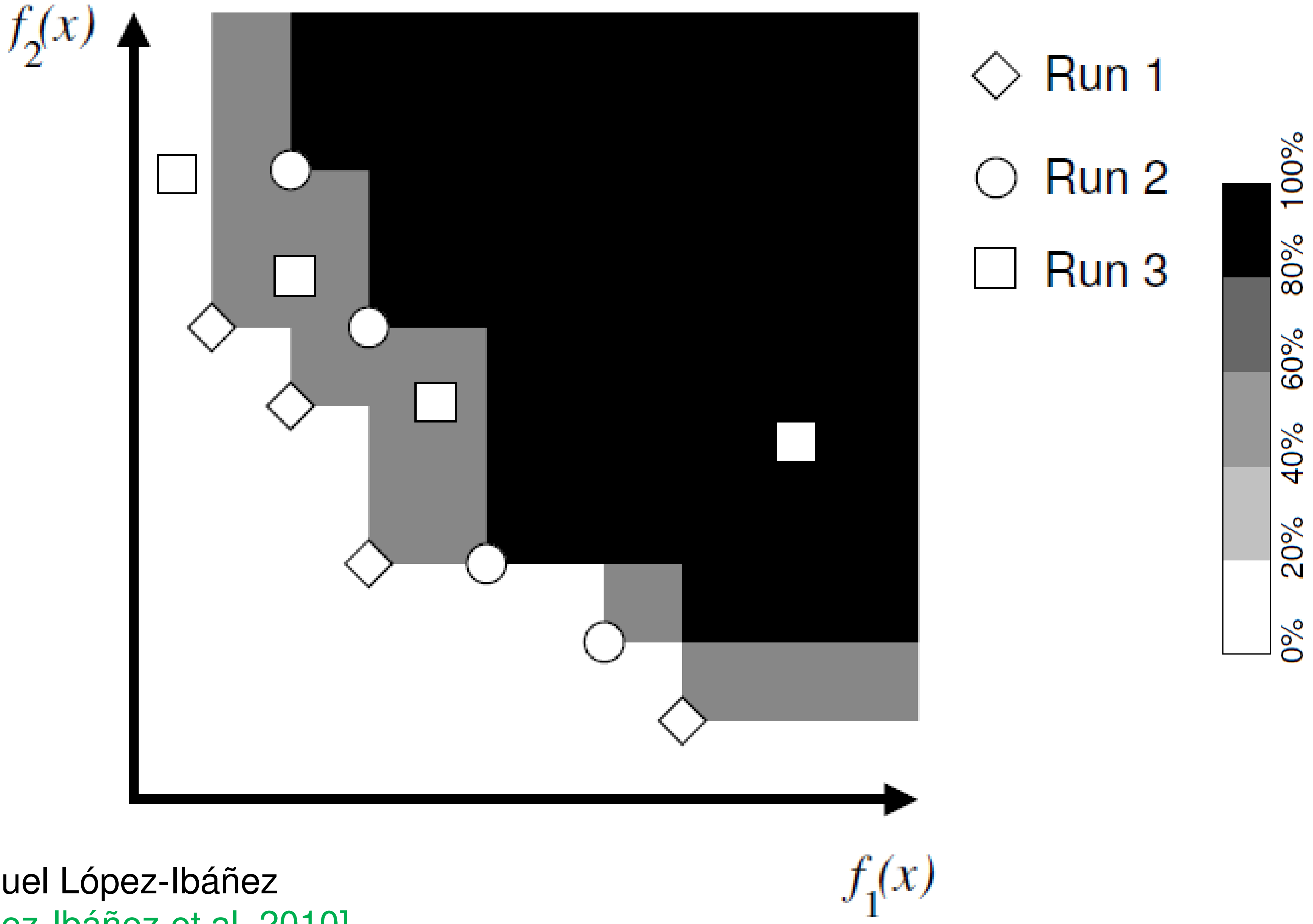
© Manuel López-Ibáñez
[López-Ibáñez et al. 2010]

# Empirical Attainment Functions: Idea



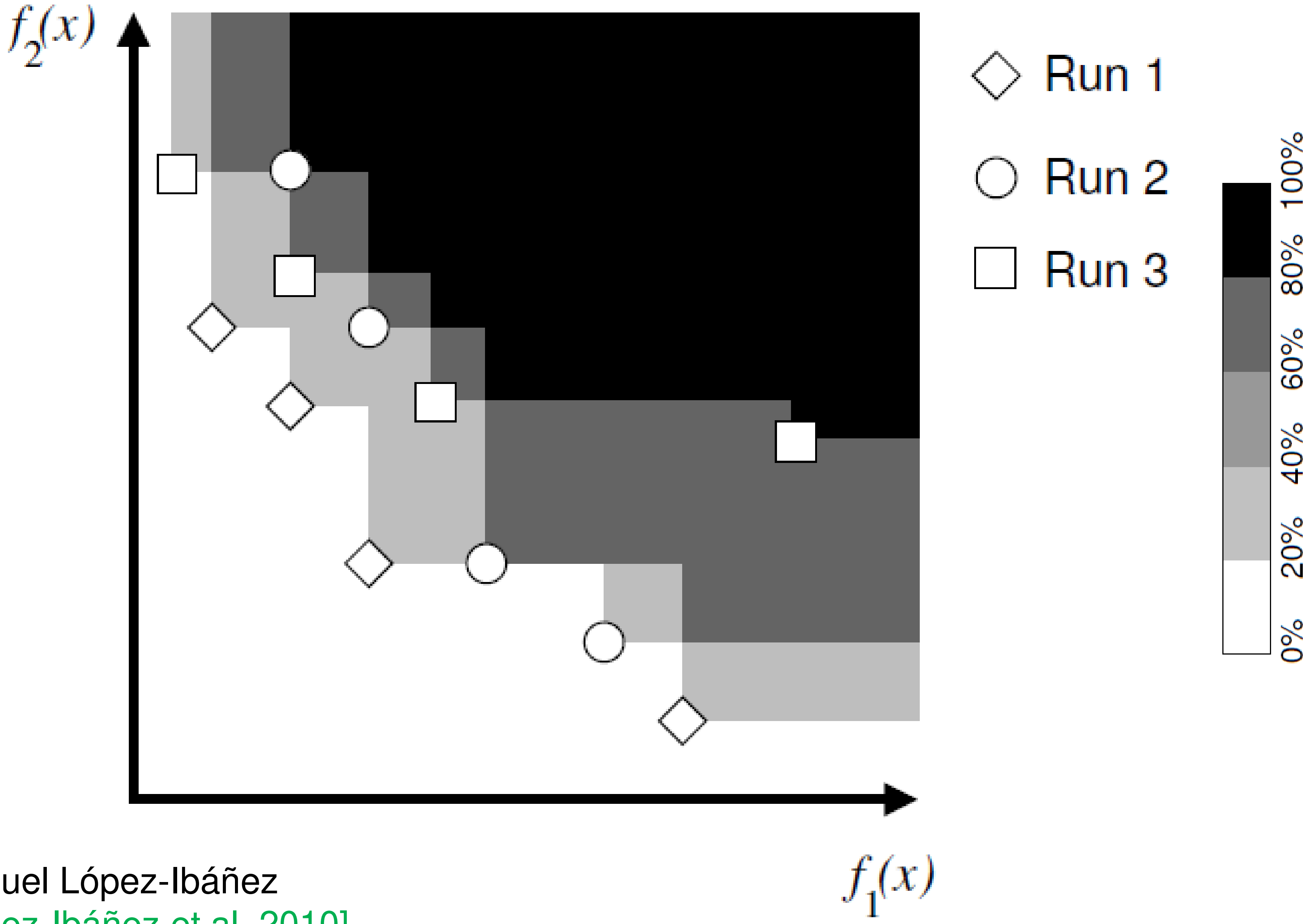
© Manuel López-Ibáñez
[López-Ibáñez et al. 2010]

# Empirical Attainment Functions: Idea



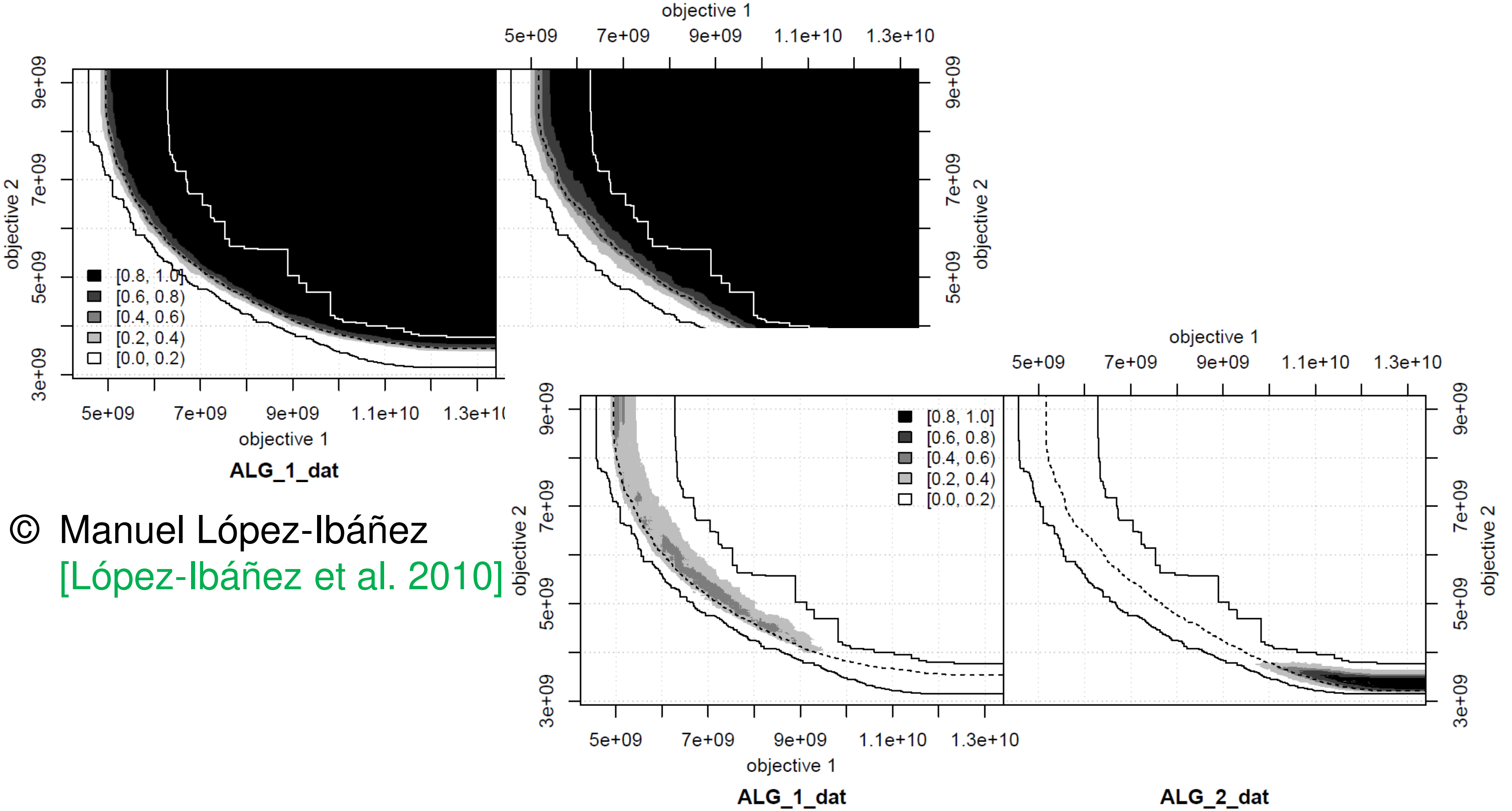
© Manuel López-Ibáñez
[López-Ibáñez et al. 2010]

# Empirical Attainment Functions: Idea



© Manuel López-Ibáñez
[López-Ibáñez et al. 2010]

# Attainment Plots in Practice

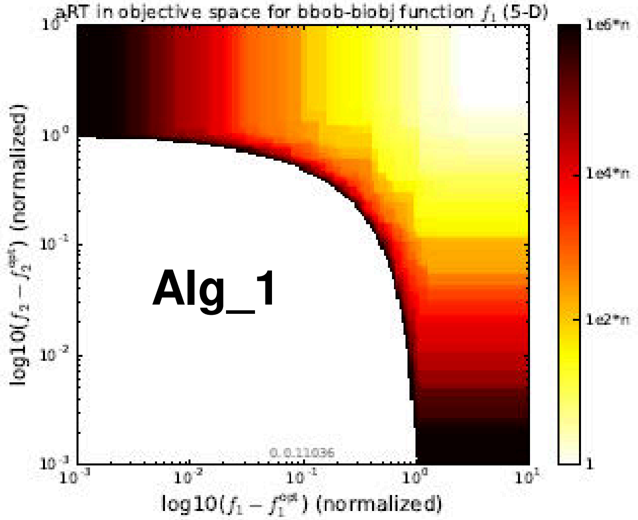


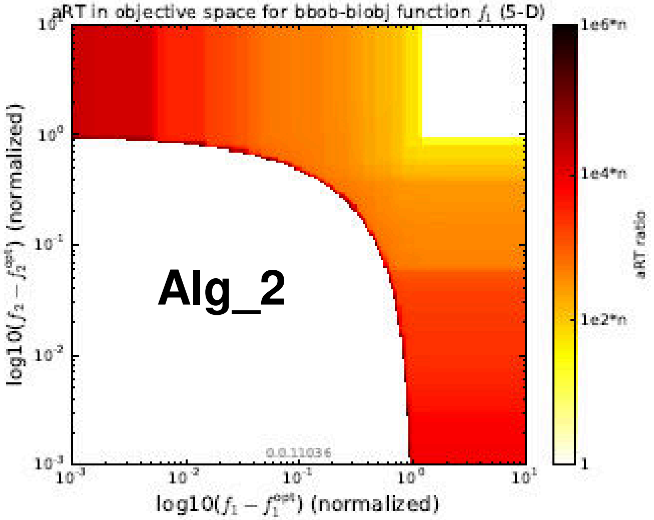
© Manuel López-Ibáñez
[López-Ibáñez et al. 2010]

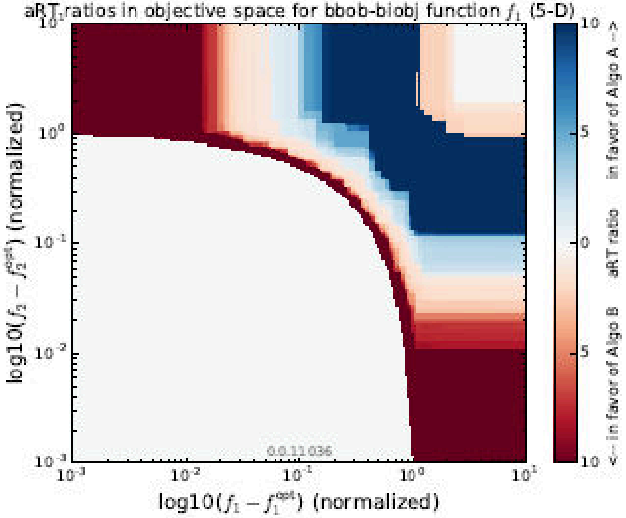
latest implementation online at
`http://eden.dei.uc.pt/~cmfonsec/software.html`
R package: `http://lopez-ibanez.eu/eaftools`
see also [López-Ibáñez et al. 2010, Fonseca et al. 2011]

# Average Runtime Attainment Plots

...display not only the success probabilities, but the average runtime to attain points in objective space:







[Brockhoff et al. 2017]

# Most Used Approach: Quality Indicators

## A quality indicator

- maps a solution set to a real number
- can be used with standard performance assessment
  - report median, variance, ...
  - boxplots
  - statistical tests
- should optimally refine the dominance relation on sets

## Recommendation:

- use hypervolume (refinement, i.e. it does not contradict the dominance relation)
- or epsilon indicator or R2 indicator (are weak refinements)

## Also important:

- interpretation of the results (by knowing theoretical properties of the used indicator)

# Quality Indicator Approach

**Idea:**

- transfer multiobjective problem into a set problem
- define an objective function ("quality indicator") on sets
- use the resulting total (pre-)order (on the quality values)

**Question:**

Can any total (pre-)order be used or are there any requirements concerning the resulting preference relation?

⇒ Underlying dominance relation should be reflected!

$$A \preceq B :\Leftrightarrow \forall_{y \in B} \exists_{x \in A} x \leq_{par} y$$

# Refinements and Weak Refinements

❶ $\overset{\text{ref}}{\preccurlyeq}$ **refines** a preference relation $\preccurlyeq$ iff

$$A \preccurlyeq B \wedge B \not\preccurlyeq A \Rightarrow A \overset{\text{ref}}{\preccurlyeq} B \wedge B \overset{\text{ref}}{\not\preccurlyeq} A \qquad \textit{(better} \Rightarrow \textit{better)}$$

$\Rightarrow$ fulfills requirement

❷ $\overset{\text{ref}}{\preccurlyeq}$ **weakly refines** a preference relation $\preccurlyeq$ iff

$$A \preccurlyeq B \wedge B \not\preccurlyeq A \Rightarrow A \overset{\text{ref}}{\preccurlyeq} B \qquad \textit{(better} \Rightarrow \textit{weakly better)}$$

$\Rightarrow$ does not fulfill requirement, but $\overset{\text{ref}}{\preccurlyeq}$ does not contradict $\preccurlyeq$

! sought are total refinements… [Zitzler et al. 2010]

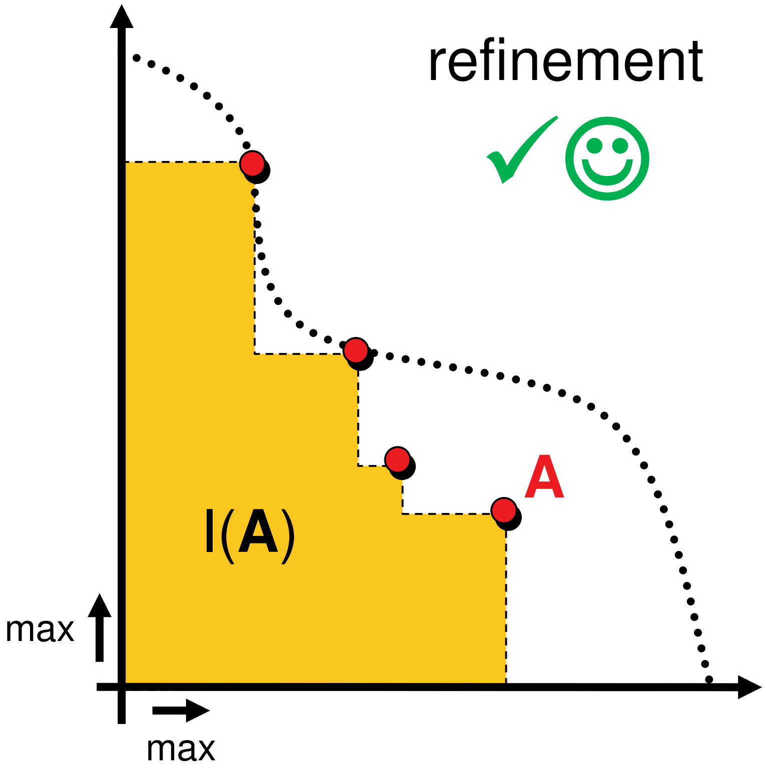
# Example: Refinements Using Indicators

$$A \overset{\text{ref}}{\preccurlyeq} B :\Leftrightarrow I(A) \geq I(B)$$

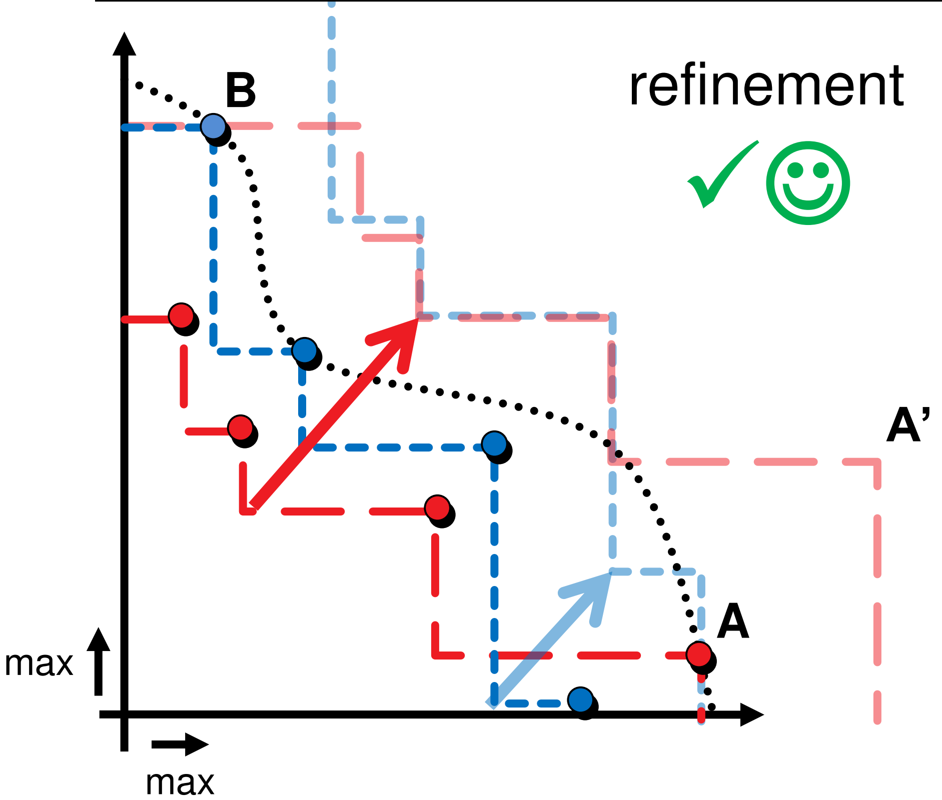
$$A \overset{\text{ref}}{\preccurlyeq} B :\Leftrightarrow I(A,B) \leq I(B,A)$$

I(A) = volume of the weakly dominated area in objective space

I(A,B) = how much needs A to be moved to weakly dominate B



unary hypervolume indicator



binary epsilon indicator

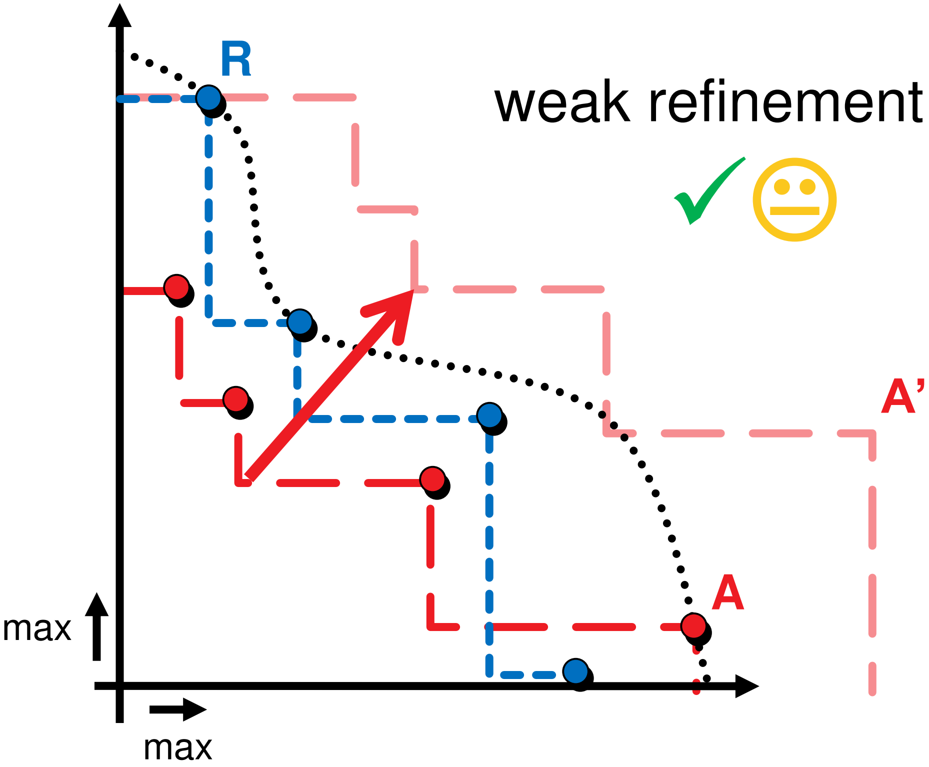
# Example: Weak Refinement / No Refinement

$A \overset{\text{ref}}{\preccurlyeq} B :\Leftrightarrow I(A,R) \leq I(B,R)$

$A \overset{\text{ref}}{\preccurlyeq} B :\Leftrightarrow I(A) \leq I(B)$

I(A,R) = how much needs A to be moved to weakly dominate R

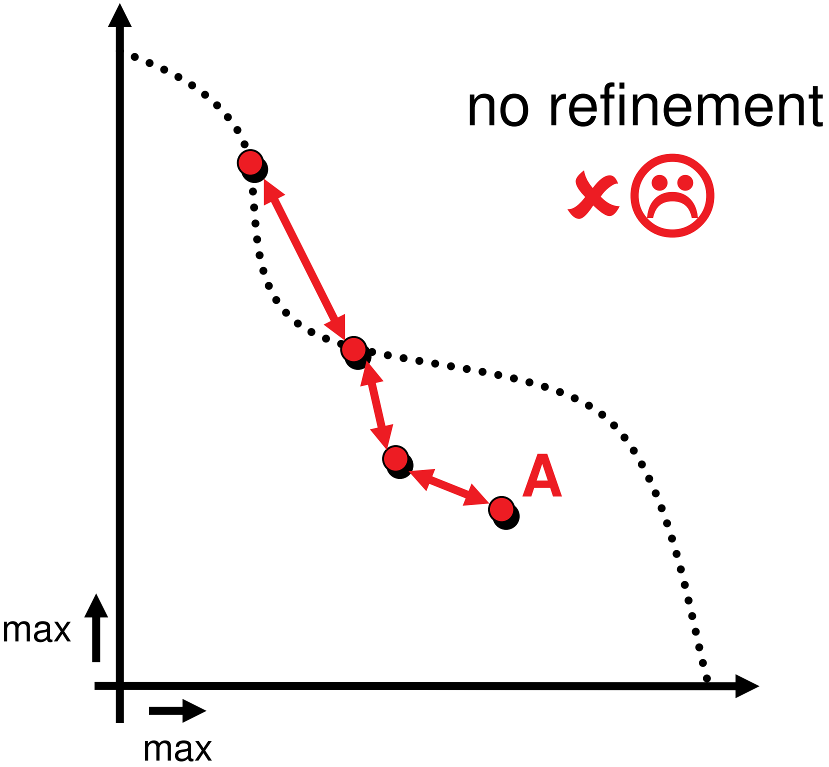
I(A) = variance of pairwise distances



weak refinement

✓ 😐

unary epsilon indicator



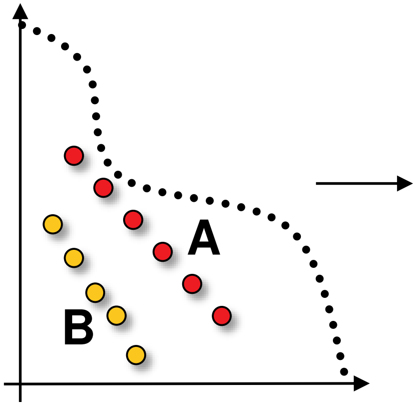
no refinement

✗ ☹

unary diversity indicator

# Quality Indicator Approach

**Goal:** compare two Pareto set approximations A and B



→

| | A | B |
|---|---|---|
| hypervolume | *432.34* | *420.13* |
| distance | *0.3308* | *0.4532* |
| diversity | *0.3637* | *0.3463* |
| spread | *0.3622* | *0.3601* |
| cardinality | *6* | *5* |

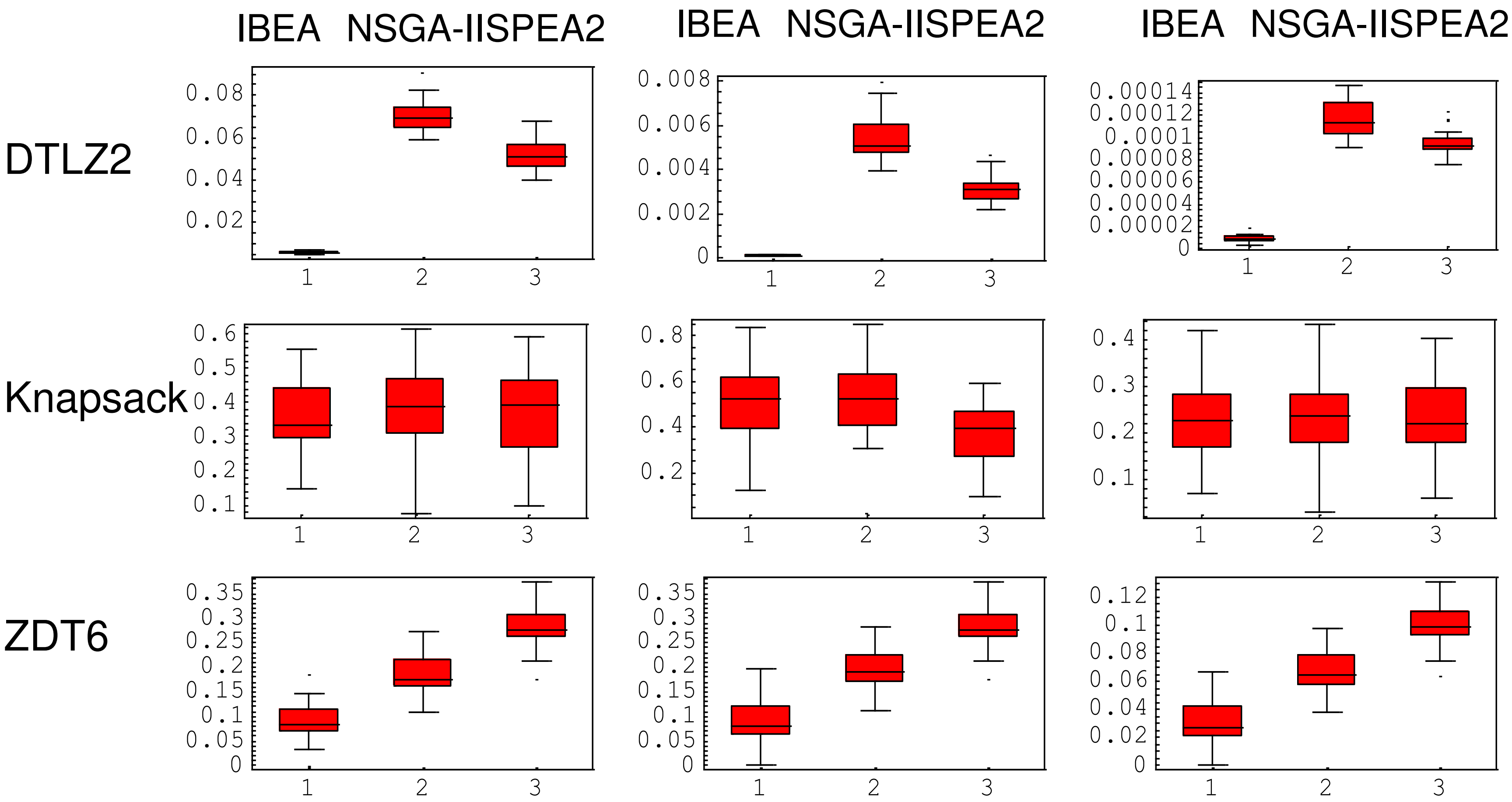
→ "**A** better"

**Comparison method C** = quality measure(s) + Boolean function

$$A, B \; \underset{\textbf{reduction}}{\xrightarrow{\text{quality measure}}} \; \mathbb{R}^n \; \underset{\textbf{interpretation}}{\xrightarrow{\text{Boolean function}}} \; \text{statement}$$

# Example: Box Plots

epsilon indicator hypervolume R indicator



# Statistical Assessment (Kruskal Test)

**ZDT6**
Epsilon

| is better than | IBEA | NSGA2 | SPEA2 |
|---|---|---|---|
| IBEA | | ~0 ☺ | ~0 ☺ |
| NSGA2 | 1 | | ~0 ☺ |
| SPEA2 | 1 | 1 | |

Overall p-value = 6.22079e-17.
Null hypothesis rejected (alpha 0.05)

**DTLZ2**
R

| is better than | IBEA | NSGA2 | SPEA2 |
|---|---|---|---|
| IBEA | | ~0 ☺ | ~0 ☺ |
| NSGA2 | 1 | | 1 |
| SPEA2 | 1 | ~0 ☺ | |

Overall p-value = 7.86834e-17.
Null hypothesis rejected (alpha 0.05)

**Knapsack**/Hypervolume: $H_0$ = No significance of any differences

# so what do we do within COCO?

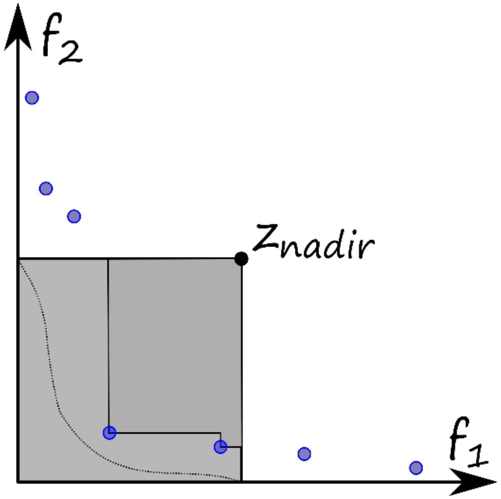
besides the average runtime attainment plots
in objective space

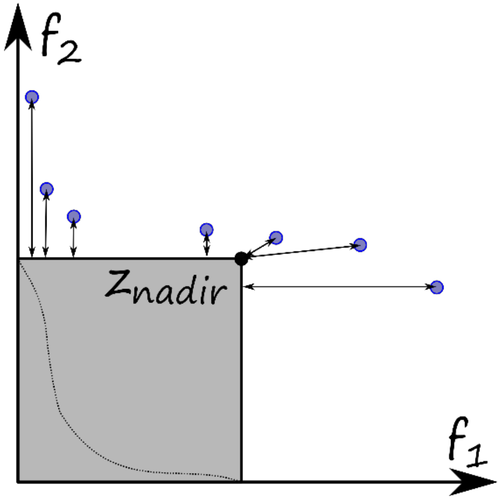
# Bi-objective Performance Assessment

algorithm quality =

normalized* hypervolume (HV)
of all non-dominated solutions
*if a point dominates nadir*

closest normalized* negative
distance to region of interest $[0,1]^2$
*if no point dominates nadir*





* such that ideal=[0,0] and nadir=[1,1]

[Brockhoff et al. 2016]

# Bi-objective Performance Assessment

We measure runtimes to reach (HV indicator) targets:

- relative to a reference set, given as the best Pareto front approximation known (since exact Pareto set not known)

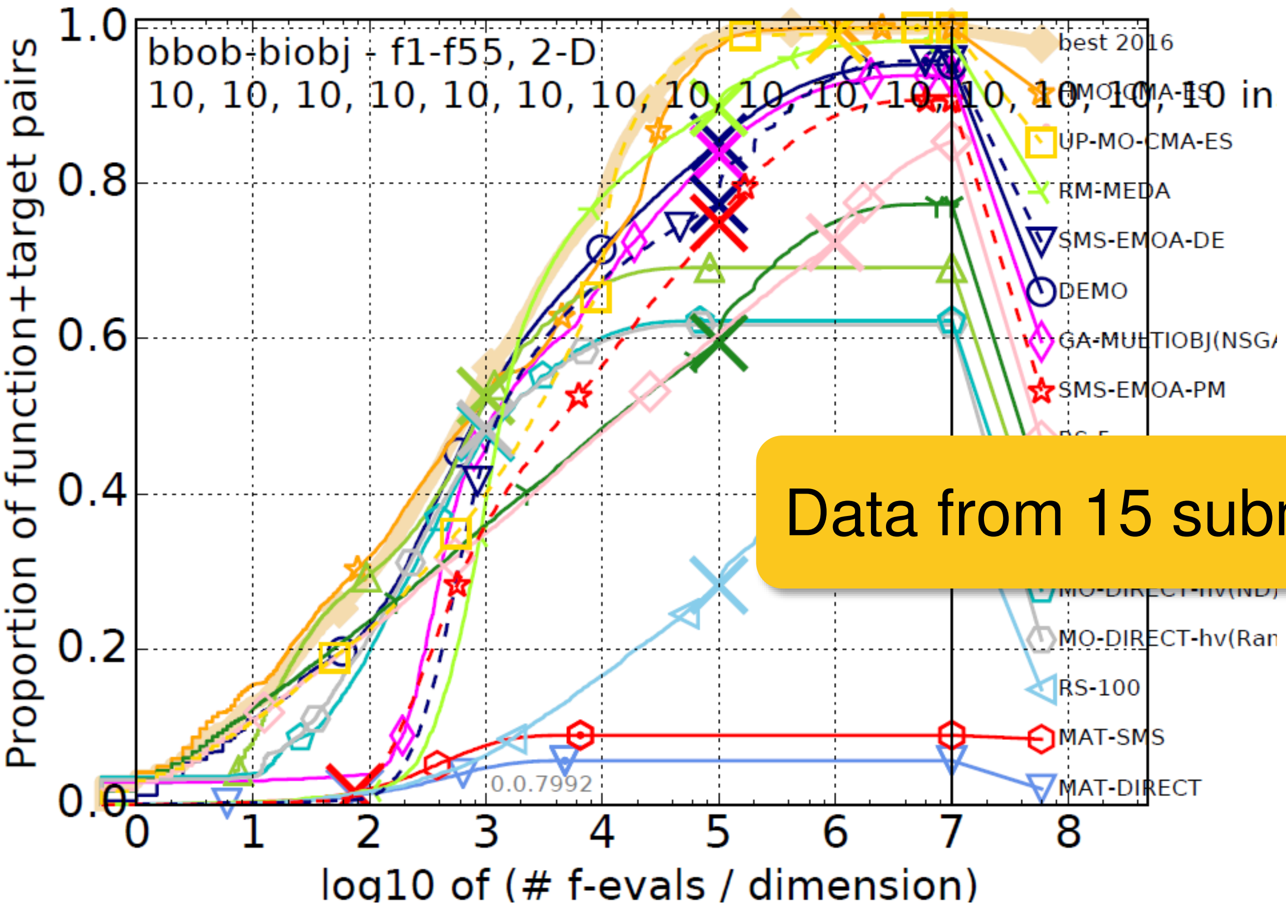
  incl. all non-dominated points found by the 15 algos of BBOB-2016

- actual absolute hypervolume targets used are

HV(refset) – targetprecision

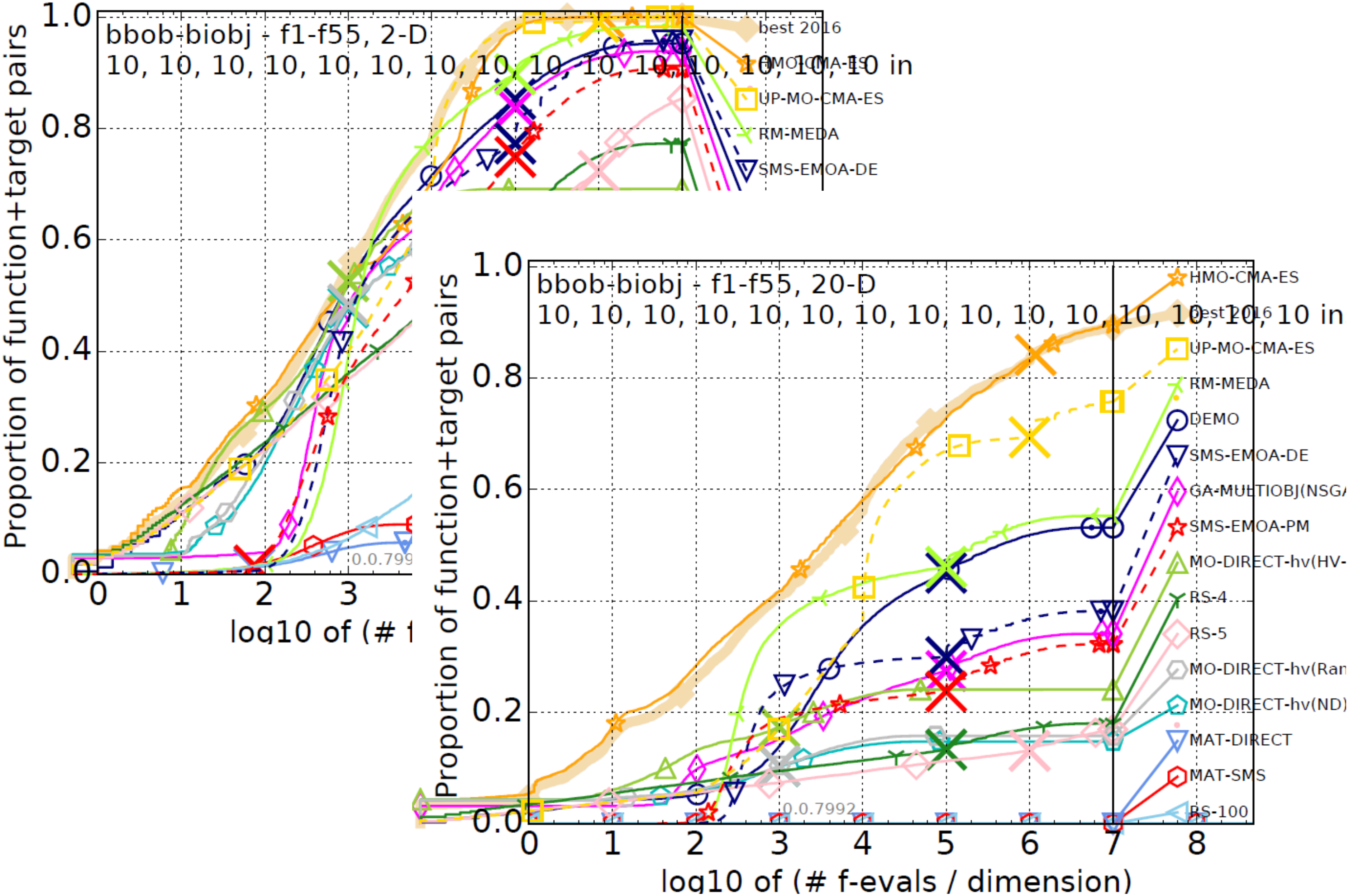
with 58 fixed targetprecisions between 1 and $-10^{-4}$ (same for all functions, dimensions, and instances) in the displays

# Exemplary BBOB-2016 Results



Data from 15 submitted algorithms

# Exemplary BBOB-2016 Results



# Take Home Messages

## State-of-the-art numerical benchmarking

- fixed target view preferred over fixed budget view
- ECDF plot of collected runtimes most important plot
    - allows for aggregation over targets, functions, and instances
    - but should not aggregate over dimension

dimension is "input parameter" to the algorithm

- multiobjective case can be handled the same way by using a quality indicator such as the hypervolume indicator

# References

[Brockhoff et al. 2016] D. Brockhoff, T. Tušar, D. Tušar, T. Wagner, N. Hansen, and A. Auger. Biobjective Performance Assessment with the COCO Platform. arXiv e-print arXiv:1605.01746v1, 2016.

[Brockhoff et al. 2017] D. Brockhoff, A. Auger, N. Hansen, and T. Tusar. Quantitative Performance Assessment of Multiobjective Optimizers: The Average Runtime Attainment Function. In Conference on Evolutionary Multi-Criterion Optimization (EMO 2017), pages 103–119. Springer, 2017

[Broyden et al. 1970] this work actually corresponds to a couple of papers:
C. G. Broyden. J Inst Maths Appl 6 (1970) 76.
R. Fletcher. Comp J 13 (1970) 317.
D. Goldfarb. Math. Comp 24 (1970) 23.
D. F. Shanno. Math Comp. 24 (1970) 647

[Goldberg 1989] D. E. Goldberg. Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA. 1989.

[Fonseca et al. 2011] C. M. Fonseca, A. P. Guerreiro, M. López-Ibáñez, and L. Paquete. On the computation of the empirical attainment function. In Takahashi et al., editors, Proc. EMO, volume 6576 of LNCS, pages 106-120. Springer, 2011

[Hansen & Ostermeier 2001] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. Evolutionary computation, 9(2), 159-195. 2001.

[Holland 1975] J. H. Holland. Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence. Ann Arbor, MI: University of Michigan Press. 1975.I

[Hooke and Jeeves 1961] R. Hooke and T. A. Jeeves. ``Direct Search'' Solution of Numerical and Statistical Problems. Journal of the ACM (JACM), 8(2), 212-229. 1961.

[Kennedy and Price 1997] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on (pp. 39-43). IEEE, 1995.

# References

[Kirkpatrick et al. 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598), 671-680. 1983.

[Larrañaga & Lozano 2001] P. Larrañaga and J. A. Lozano. Estimation of distribution algorithms: A new tool for evolutionary computation (Vol. 2). Springer. 2001.

[López-Ibáñez et al. 2010] M. López-Ibáñez, T. Stützle, and L. Paquete. Graphical tools for the analysis of bi-objective optimization algorithms:[workshop on theoretical aspects of evolutionary multiobjective optimization]. In Proceedings of the 12th annual conference companion on Genetic and evolutionary computation (pp. 1959-1962). 2010, ACM.

[Nelder & Mead 1965] J. A. Nelder and R. Mead. A simplex method for function minimization. The computer journal, 7(4), 308-313. 1965.

[Powell 2006] M. J. Powell. The NEWUOA software for unconstrained optimization without derivatives. In Large-scale nonlinear optimization (pp. 255-297). Springer US. 2006.

[Powell 2009] M. J. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Cambridge NA Report NA2009/06, University of Cambridge, Cambridge. 2009.

[Rechenberg 1965] I. Rechenberg. Cybernetic solution path of an experimental problem. 1965.

[Rubinstein 1999] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. Methodology and Computing in Applied Probability, 2:127–190, 1999.

[Spall 2000] J. C. Spall. Adaptive stochastic approximation by the simultaneous perturbation method. IEEE transactions on automatic control, 45(10), 1839-1853. 2000.

[Storn and Price 1997] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11(4), 341-359. 1997.

[Zitzler et al. 2003] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. IEEE Transactions on Evolutionary Computation, 7(2):117–132, 2003

[Zitzler et al. 2010] E. Zitzler, L. Thiele, and J. Bader. On Set-Based Multiobjective Optimization. IEEE Transactions on Evolutionary Computation, 14(1):58–79, 2010