

6th GECCO Workshop on Blackbox Optimization Benchmarking (BBOB): Welcome and Introduction to COCO/BBOB

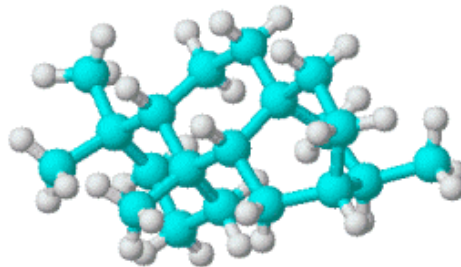
The BBOBies

<https://github.com/numbbo/coco>

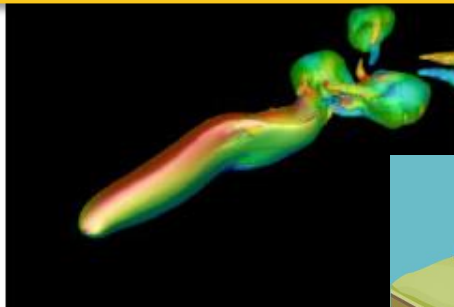
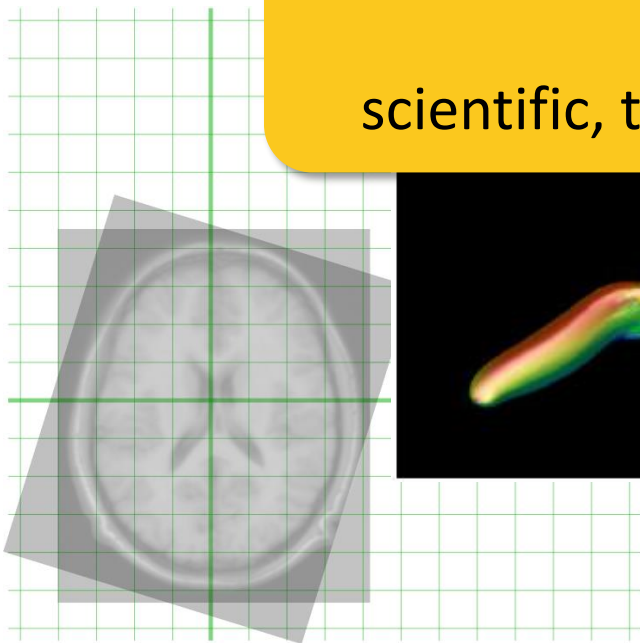
The logo for Inria, featuring the word "Inria" in a red, cursive script font.

INVENTORS FOR THE DIGITAL WORLD

slides based on previous ones by A. Auger, N. Hansen, and D. Brockhoff

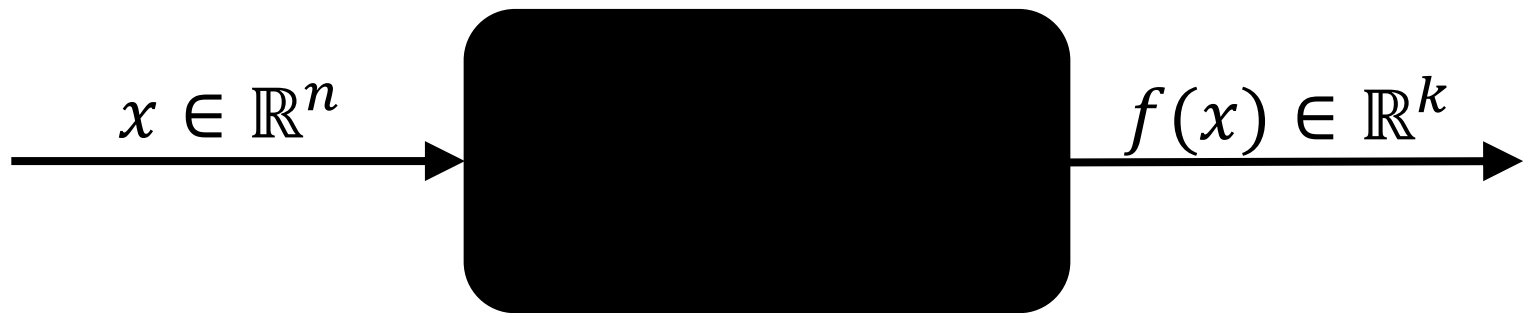


challenging optimization problems
appear in many
scientific, technological and industrial domains



Numerical Blackbox Optimization

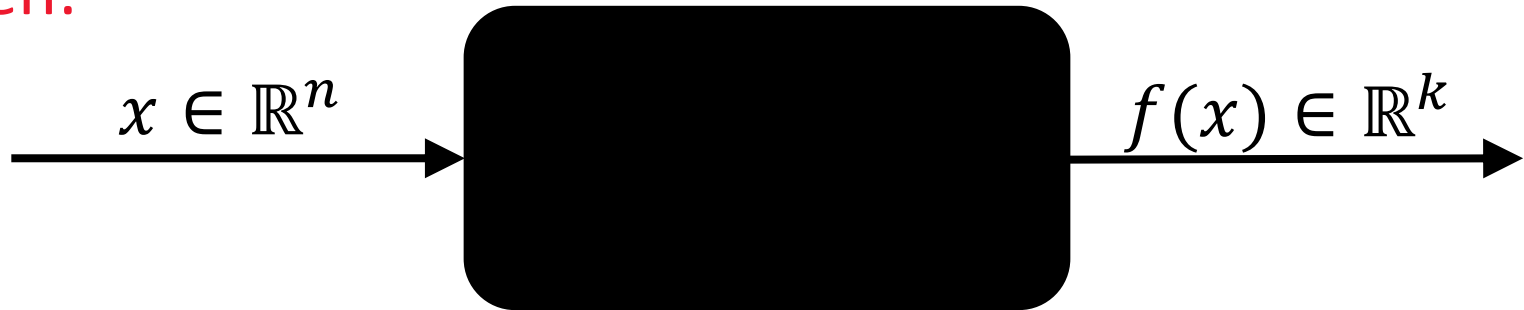
Optimize $f: \Omega \subset \mathbb{R}^n \mapsto \mathbb{R}^k$



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 which difficulties their problem has (e.g. multi-modality, non-separability, ...)

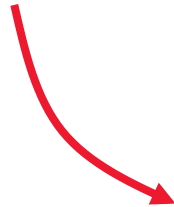
Need: Benchmarking

- 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** and **BBOB** come 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 ... x +

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

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

This repository Search Pull requests Issues Gist

numbbo / coco Unwatch 10 Unstar 9 Fork 12

Code Issues 111 Pull requests 1 Pulse Graphs Settings

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

6,931 commits 11 branches 15 releases 13 contributors

Branch: master New pull request New file Upload files Find file HTTPS https://github.com/numbt Download ZIP

nikohansen Merge pull request #720 from numbbo/development Latest commit bcea0b2 5 days ago

code-experiments	modified: code-experiments/build/python/cython/interface.c	5 days ago
code-postprocessing	Stop condition fixed.	6 days ago
docs	docs/coco-doc edit	7 days ago
howtos	Update release-howto.md	20 days ago
.clang-format	raising an error in bbob2009_logger.c when best_value is NULL. Plus s...	a year ago
.hignore	raising an error in bbob2009_logger.c when best_value is NULL. Plus s...	a year ago
AUTHORS	minor	a month ago
LICENSE	Create LICENSE	2 months ago
README.md	Update README.md	10 days ago
do.py	Added other paths to jdk on mac.	6 days ago
devxgon.ini	moved all files into code-experiments/ folder besides the do.py scrip	4 months ago

https://github.com/numbbo/coco

numbbo/coco: Numerical ...

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

This repository Search Pull requests Issues Gist

numbbo / coco Unwatch 10 Unstar 9 Fork 12

Code Issues 111 Pull requests 1 Pulse Graphs Settings

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

6,931 commits 11 branches 15 releases 13 contributors

Branch: master New pull request New file Upload files Find file HTTPS https://github.com/numbt Download ZIP

nikohansen Merge pull request #720 from numbbo/development Latest commit bcea0b2 5 days ago

code-experiments	modified: code-experiments/build/python/cython/interface.c	5 days ago
code-postprocessing	Stop condition fixed.	6 days ago
docs	docs/coco-doc edit	7 days ago
howtos	Update release-howto.md	20 days ago
.clang-format	raising an error in bbob2009_logger.c when best_value is NULL. Plus s...	a year ago
.hignore	raising an error in bbob2009_logger.c when best_value is NULL. Plus s...	a year ago
AUTHORS	minor	a month ago
LICENSE	Create LICENSE	2 months ago
README.md	Update README.md	10 days ago
do.py	Added other paths to jdk on mac.	6 days ago
devxgon.ini	moved all files into code-experiments/ folder besides the do.py scrip	4 months ago

https://github.com/numbbo/coco

The screenshot shows the GitHub repository page for 'numbbo / coco'. The repository is described as a 'Numerical Black-Box Optimization Benchmarking Framework' with a link to 'http://coco.gforge.inria.fr/'. It has 6,931 commits, 11 branches, 15 releases, and 13 contributors. The 'Download ZIP' button is highlighted with a red box. Below the repository information, there is a list of recent commits by 'nikohansen', including updates to 'code-experiments', 'code-postprocessing', 'docs', 'howtos', and various source files like '.clang-format', '.hgignore', 'AUTHORS', 'LICENSE', 'README.md', 'do.py', and 'doxygen.ini'.

numbbo / coco

Unwatch 10 Unstar 9 Fork 12

Code Issues 111 Pull requests 1 Pulse Graphs Settings

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

6,931 commits 11 branches 15 releases 13 contributors

Branch: master New pull request New file Upload files Find file HTTPS https://github.com/numbt Download ZIP

nikohansen Merge pull request #720 from numbbo/development Latest commit b5ea002 5 days ago

code-experiments	modified: code-experiments/build/python/cython/interface.c	5 days ago
code-postprocessing	Stop condition fixed.	6 days ago
docs	docs/coco-doc edit	7 days ago
howtos	Update release-howto.md	20 days ago
.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	minor	a month ago
LICENSE	Create LICENSE	2 months ago
README.md	Update README.md	10 days ago
do.py	Added other paths to jdk on mac.	6 days ago
doxygen.ini	moved all files into code-experiments/ folder besides the do.py scrip...	4 months ago

README.md

https://github.com/numbbo/coco

numbbo/coco: Numerical ... x +

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

Code Issues 111 Pull requests 1 Pulse Graphs Settings

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

6,931 commits 11 branches 15 releases 13 contributors

Branch: master New pull request New file Upload files Find file HTTPS https://github.com/numbt Download ZIP

nikohansen Merge pull request #720 from numbbo/development Latest commit bcea0b2 3 days ago

code-experiments	modified: code-experiments/build/python/cython/interface.c	5 days ago
code-postprocessing	Stop condition fixed.	6 days ago
docs	docs/coco-doc edit	7 days ago
howtos	Update release-howto.md	20 days ago
.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	minor	a month ago
LICENSE	Create LICENSE	2 months ago
README.md	Update README.md	10 days ago
do.py	Added other paths to jdk on mac.	6 days ago
doxygen.ini	moved all files into code-experiments/ folder besides the do.py scrip...	4 months ago

README.md

https://github.com/numbbo/coco

numbbo/coco: Numerical ... x +

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

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

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

6,931 commits 11 branches 15 releases 13 contributors

Branch: master New pull request New file Upload files Find file HTTPS https://github.com/numbt Download ZIP

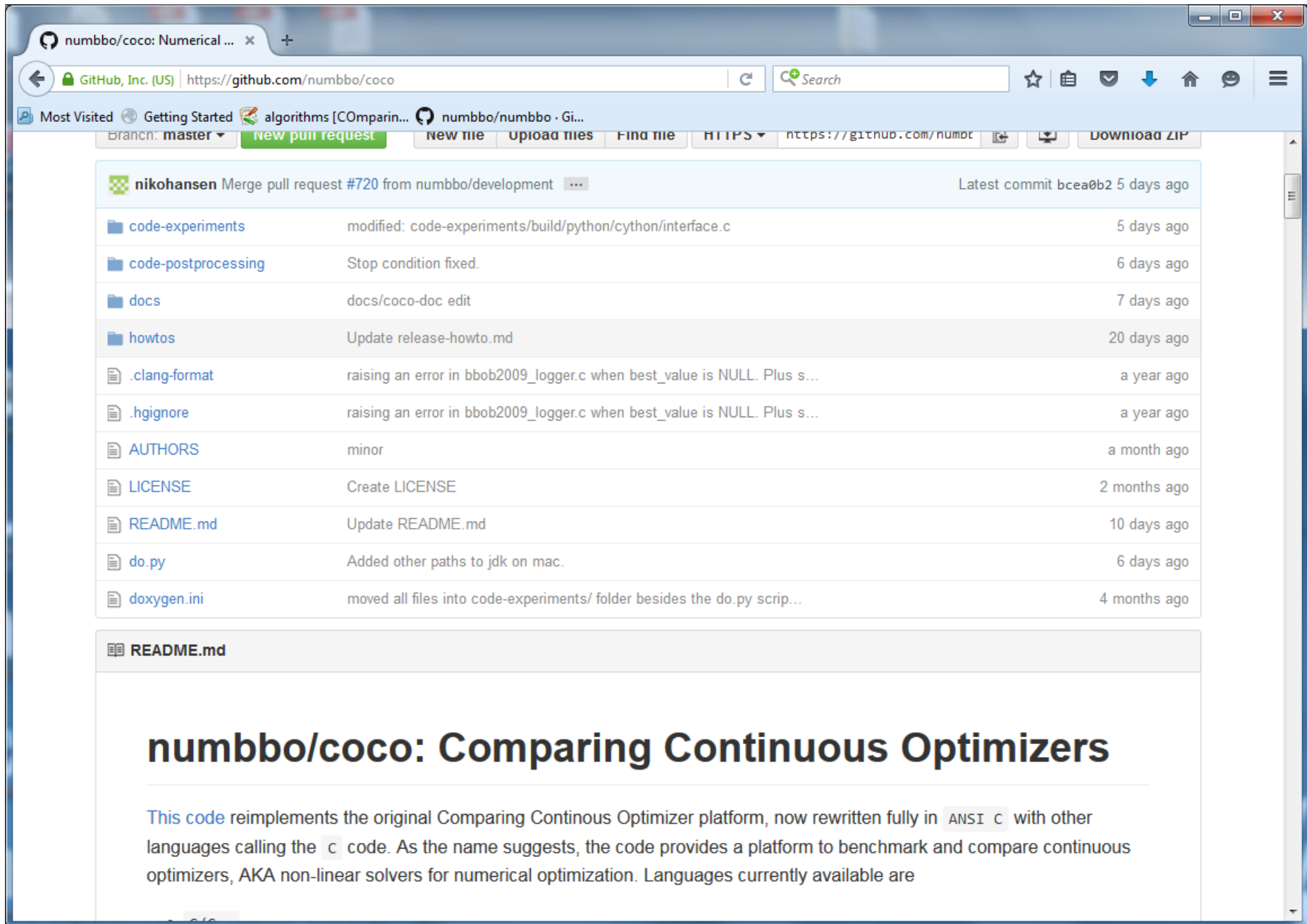
nikohansen Merge pull request #720 from numbbo/development Latest commit bcea0b2 5 days ago

code-experiments	modified: code-experiments/build/python/cython/interface.c	5 days ago
code-postprocessing	Stop condition fixed.	6 days ago
docs	docs/coco-doc edit	7 days ago
howtos	Update release-howto.md	20 days ago
.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	minor	a month ago
LICENSE	Create LICENSE	2 months ago
README.md	Update README.md	10 days ago
do.py	Added other paths to jdk on mac.	6 days ago
doxygen.ini	moved all files into code-experiments/ folder besides the do.py scrip...	4 months ago

README.md

numbbo/coco: Comparing Continuous Optimizers

https://github.com/numbbo/coco



numbbo/coco: Numerical ... x +

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

Branch: master | new pull request | new file | upload files | find file | HTTPS | https://github.com/numbbo/coco | Download ZIP

nikohansen Merge pull request #720 from numbbo/development | Latest commit bcea0b2 5 days ago

code-experiments	modified: code-experiments/build/python/cython/interface.c	5 days ago
code-postprocessing	Stop condition fixed.	6 days ago
docs	docs/coco-doc edit	7 days ago
howtos	Update release-howto.md	20 days ago
.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	minor	a month ago
LICENSE	Create LICENSE	2 months ago
README.md	Update README.md	10 days ago
do.py	Added other paths to jdk on mac.	6 days ago
doxygen.ini	moved all files into code-experiments/ folder besides the do.py scrip...	4 months ago

README.md

numbbo/coco: Comparing Continuous Optimizers

This code reimplements the original Comparing Continuous 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

https://github.com/numbbo/coco

The screenshot shows a web browser window with the URL `https://github.com/numbbo/coco`. The browser's address bar and tabs are visible at the top. Below the browser window, a table lists recent commits:

File	Commit Message	Time Ago
howtos	Update release-howto.md	20 days ago
.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	minor	a month ago
LICENSE	Create LICENSE	2 months ago
README.md	Update README.md	10 days ago
do.py	Added other paths to jdk.py on mac.	6 days ago
doxygen.ini	moved all files into code-experiments/ folder besides the do.py scrip...	4 months ago

Below the commit list, the `README.md` file is selected and its content is displayed:

numbbo/coco: Comparing Continuous Optimizers

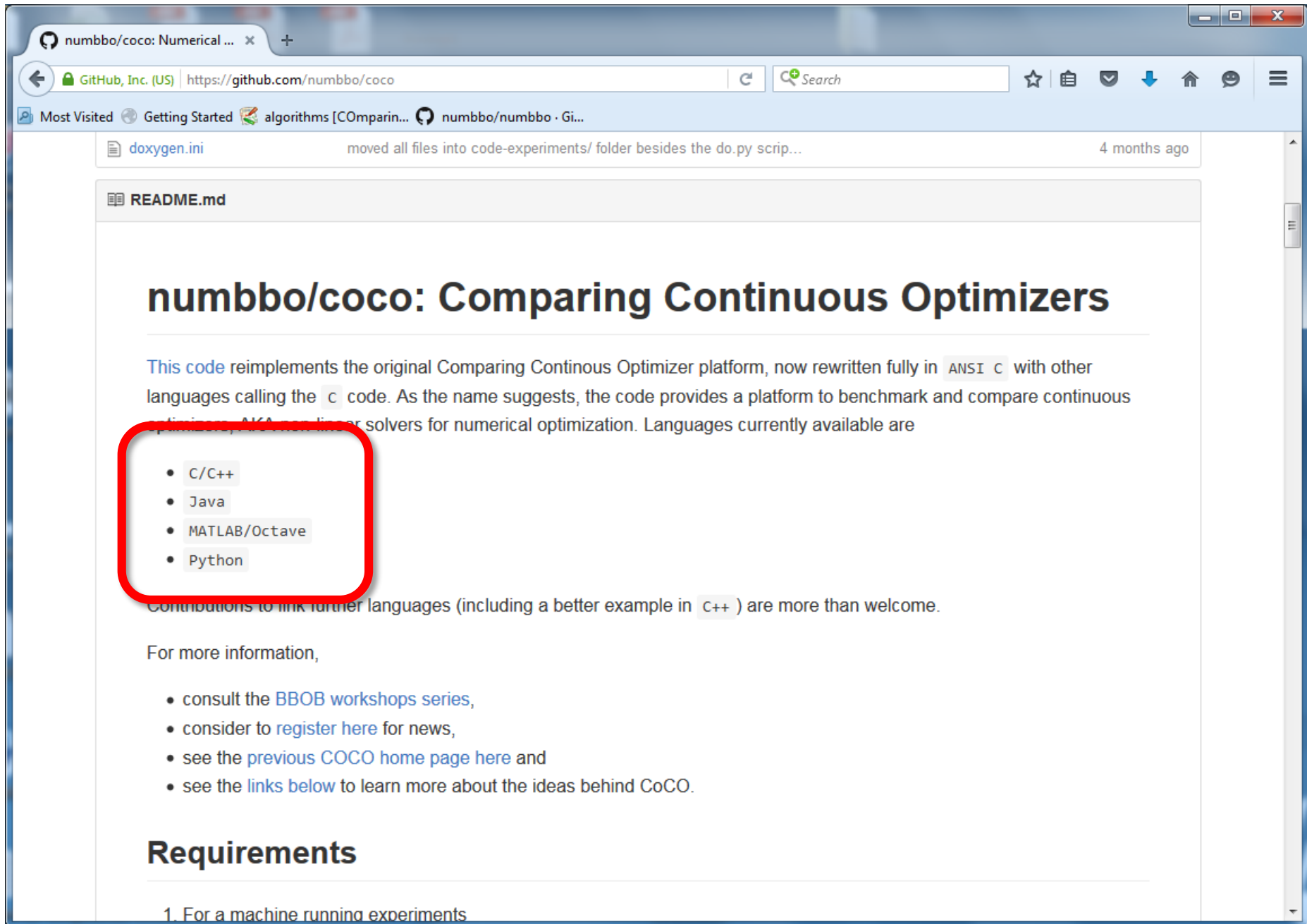
This code reimplements the original Comparing Continuous Optimizer platform, now rewritten fully in `ANSI C` with other languages calling the `C` code. As the name suggests, the code provides a platform to benchmark and compare continuous optimizers, AKA non-linear solvers for numerical optimization.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



numbbo/coco: Numerical ... x +

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

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

doxygen.ini moved all files into code-experiments/ folder besides the do.py scrip... 4 months ago

README.md

numbbo/coco: Comparing Continuous Optimizers

This code reimplements the original Comparing Continuous 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* 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,

- 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.

Requirements

1. For a machine running experiments

https://github.com/numbbo/coco

Getting Started

1. Check out the [Requirements](#) above.
2. **Download** the [COCO framework code](#) from [github](#),
 - o either by clicking [here](#) and unzip the `zip` file,
 - o or (preferred) 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.

CAVEAT: this code is still under heavy development. The record of official releases can be found [here](#). The latest release 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
```

https://github.com/numbbo/coco

Getting Started

1. Check out the [Requirements](#) above.
2. **Download** the [COCO framework code](#) from [github](#),
 - o either by clicking [here](#) and unzip the `zip` file,
 - o or (preferred) 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.

CAVEAT: this code is still under heavy development. The record of official releases can be found [here](#). The latest release 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
```

https://github.com/numbbo/coco

4. 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.

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:

- o `C` [read me and example experiment](#)
- o `Java` [read me and example experiment](#)
- o `Matlab/Octave` [read me and example experiment](#)
- o `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.

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]
```

https://github.com/numbbo/coco

4. 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.

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:

- o C [read me and example experiment](#)
- o Java [read me and example experiment](#)
- o Matlab/Octave [read me and example experiment](#)
- o 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.

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]
```

example_experiment.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);
    }
}
```

example_experiment.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);

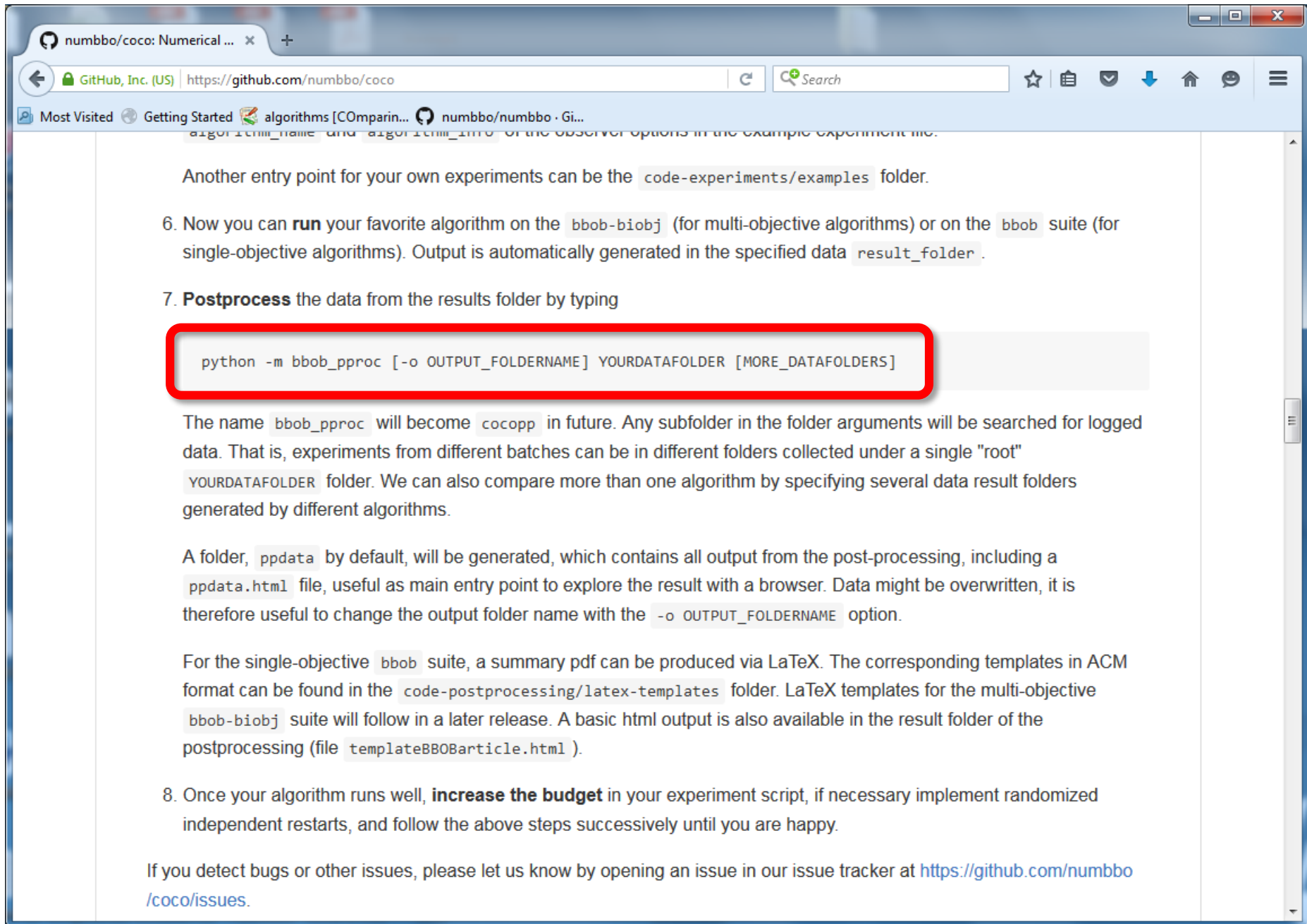
    /* Run the algorithm at least once */
    for (run = 1; run <= 1 + INDEPENDENT_RESTARTS; run++) {

        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);
    }
}
```


https://github.com/numbbo/coco



numbbo/coco: Numerical ... x +

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

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

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.

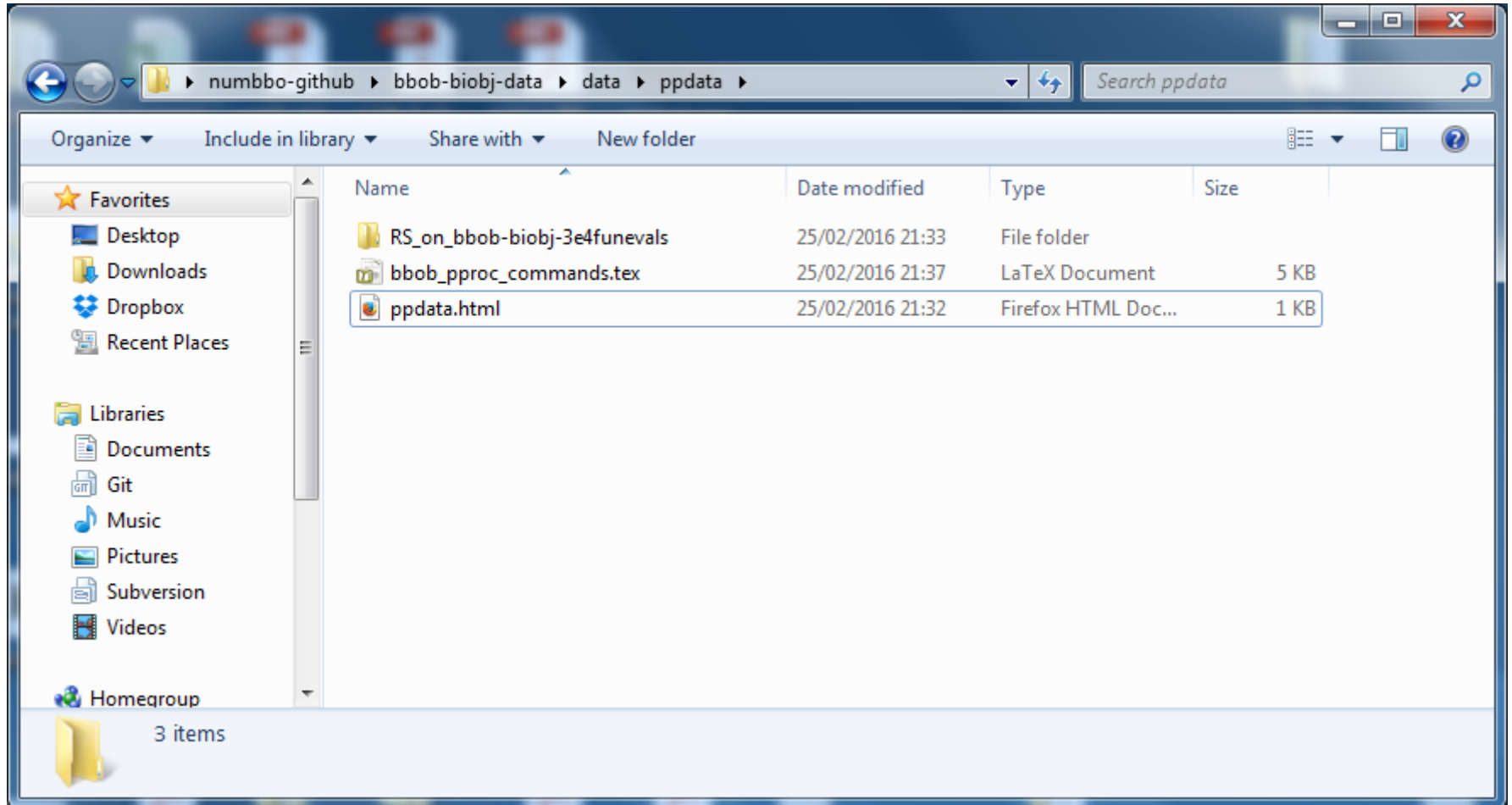
A folder, `ppdata` by default, will be generated, which contains all output from the post-processing, including a `ppdata.html` file, useful as main entry point to explore the result with a browser. Data might be overwritten, it is therefore useful to change the output folder name with the `-o OUTPUT_FOLDERNAME`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

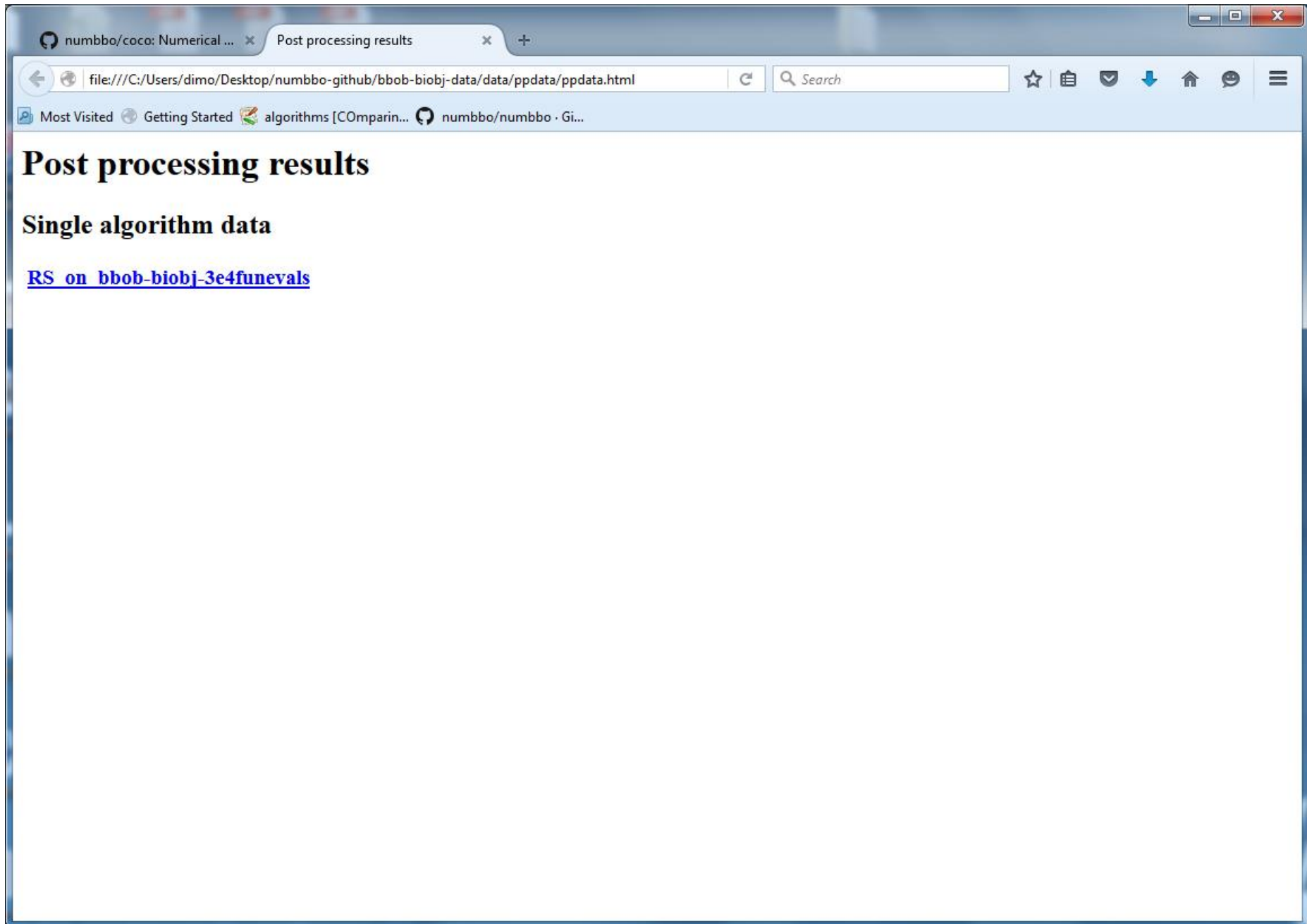
8. Once your algorithm runs well, **increase the budget** in your experiment script, if necessary implement randomized independent restarts, and follow the above steps successively until you are happy.

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>.

result folder



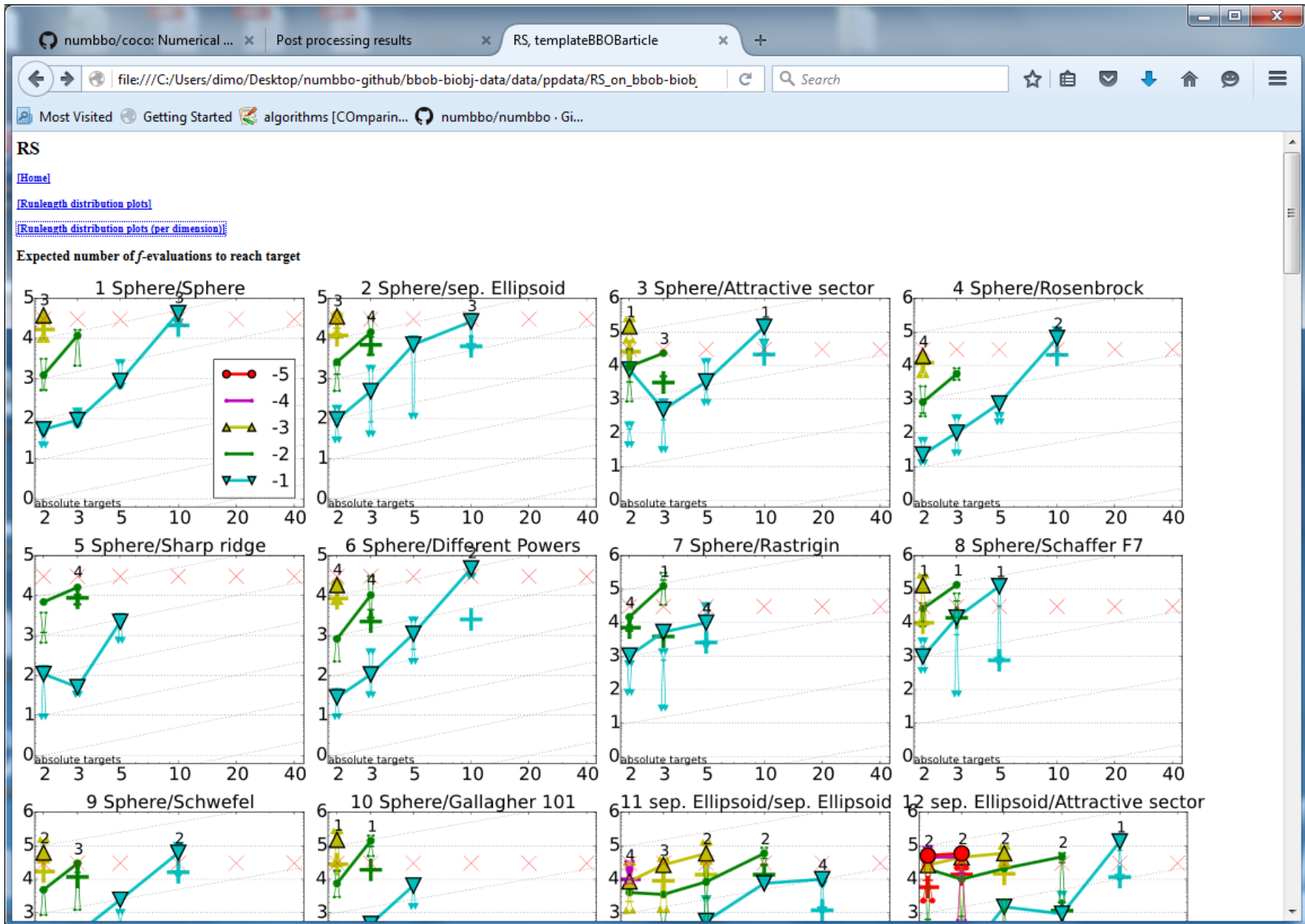
automatically generated results



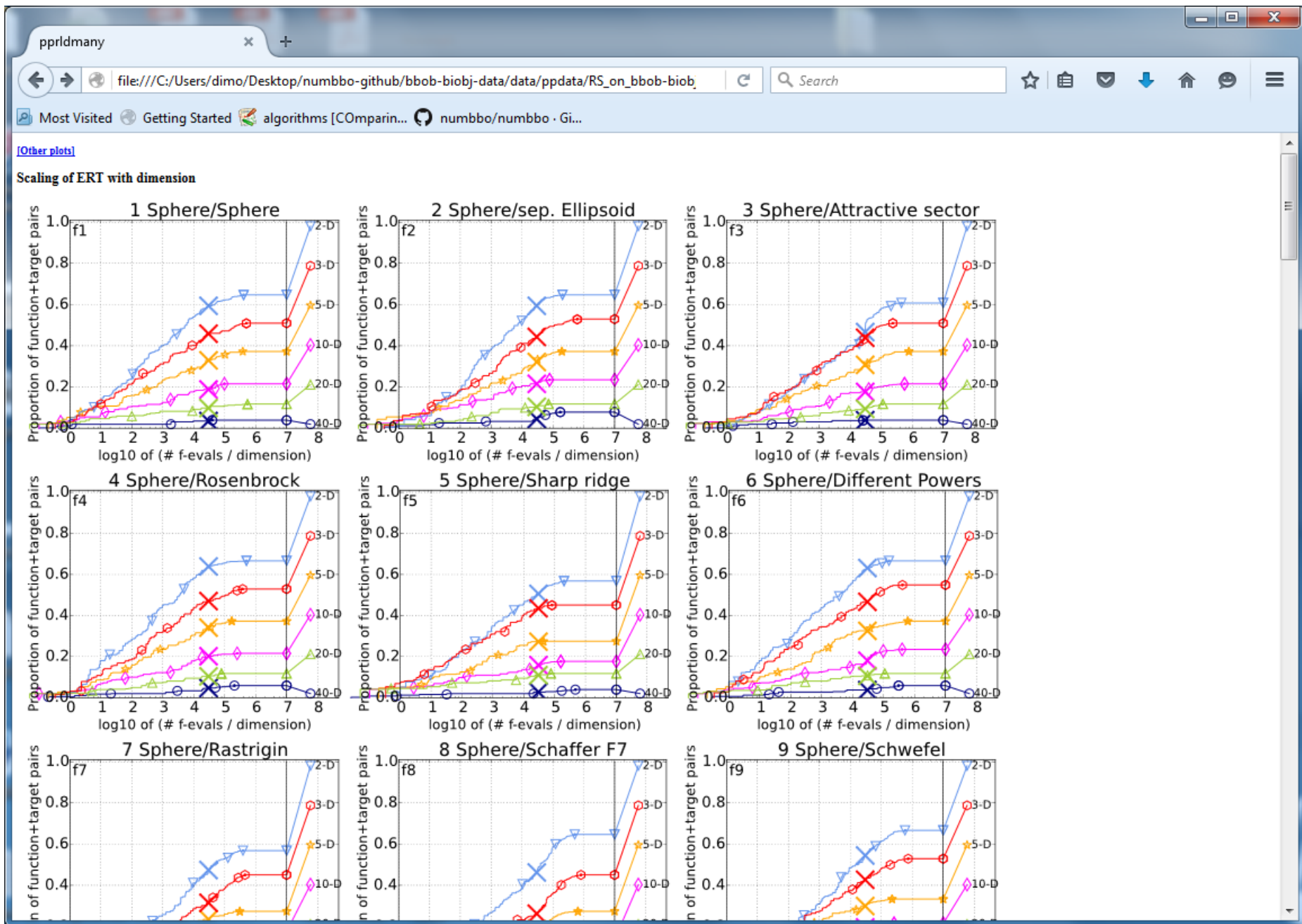
The image shows a web browser window with the following content:

- Address bar: `file:///C:/Users/dimo/Desktop/numbbo-github/bbob-biobj-data/data/ppdata/ppdata.html`
- Page title: **Post processing results**
- Section header: **Single algorithm data**
- Link: [RS on bbob-biobj-3e4funevals](#)

automatically generated results



automatically generated results



doesn't look too complicated, does it?

[the devil is in the details 😊]

so far (i.e. before 2016):

data for about 150 algorithm variants

118 workshop papers

by 79 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 |  new in 2016
15 algo data sets |

Under development:

- large-scale versions
- 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?

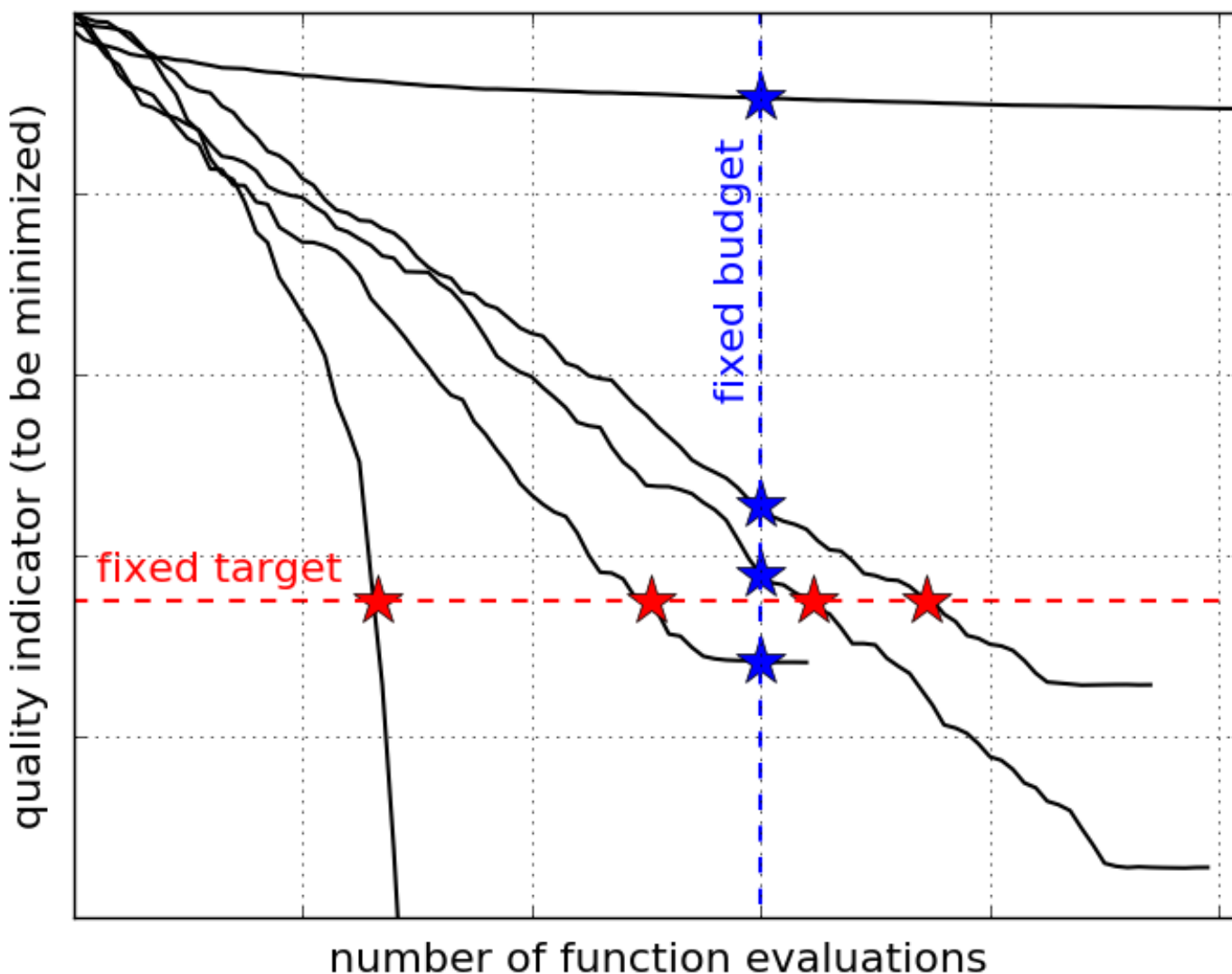
Two objectives:

- 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

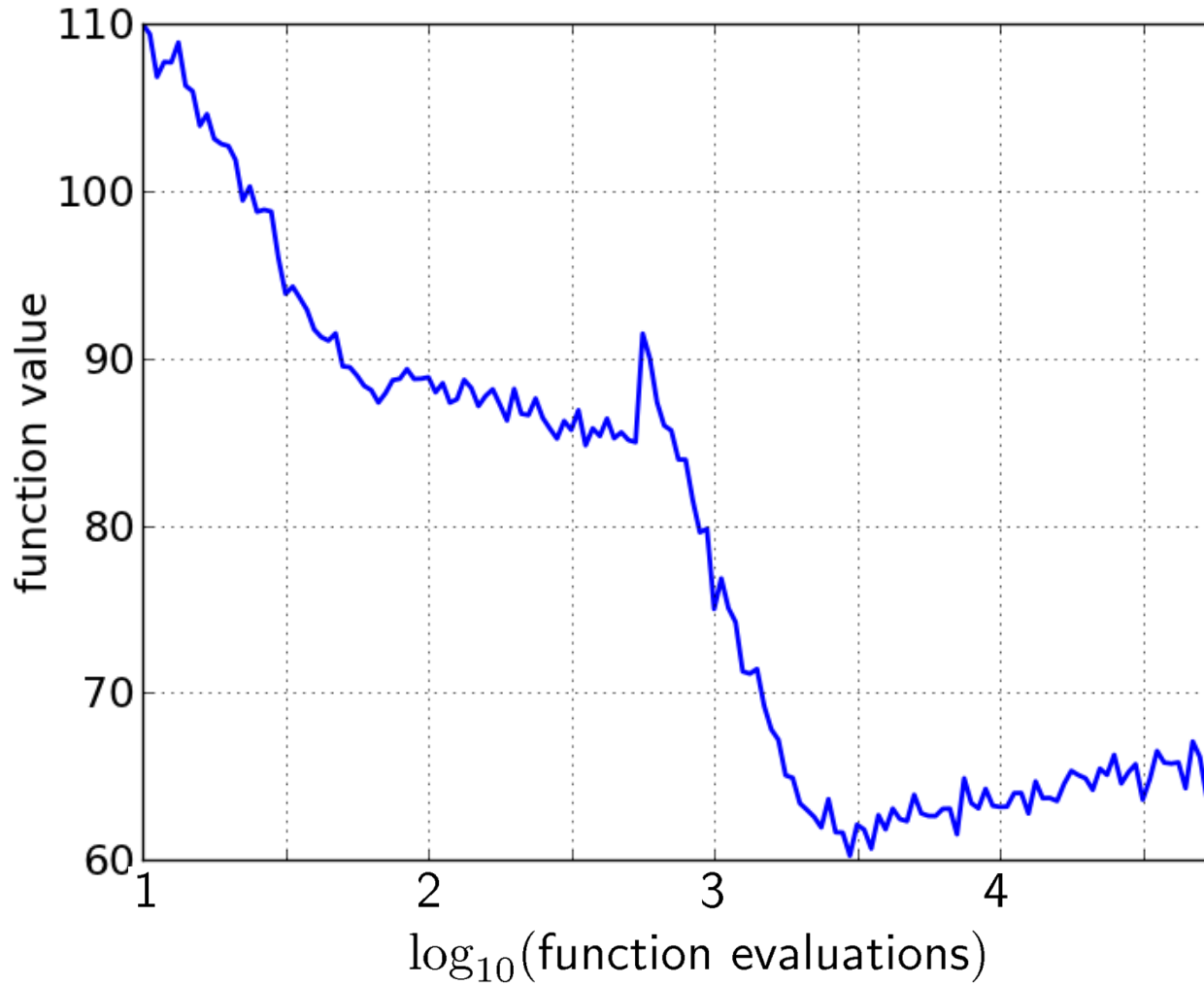
convergence graphs is all we have to start with...



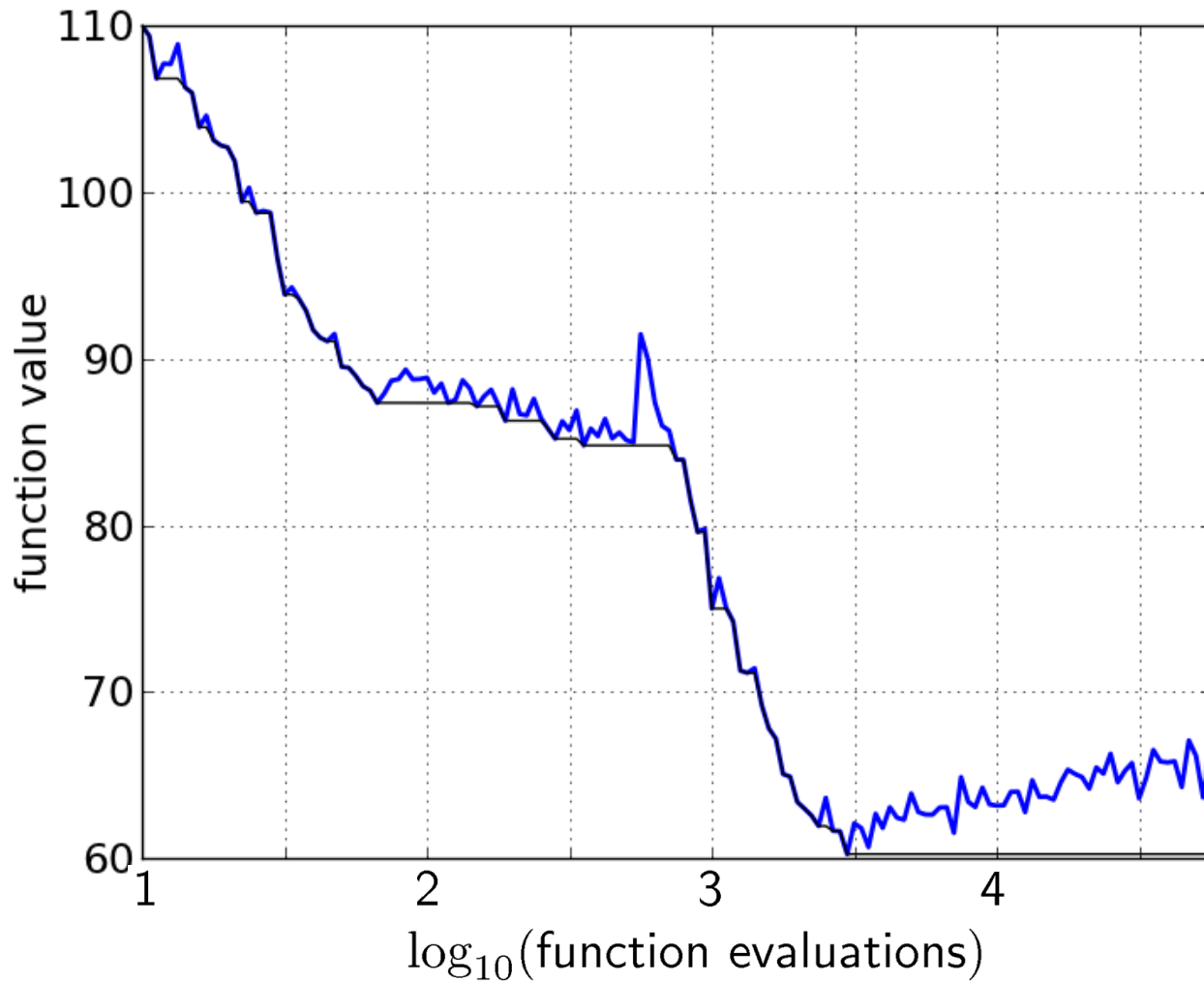
ECDF:

Empirical Cumulative Distribution Function of the Runtime
[aka data profile]

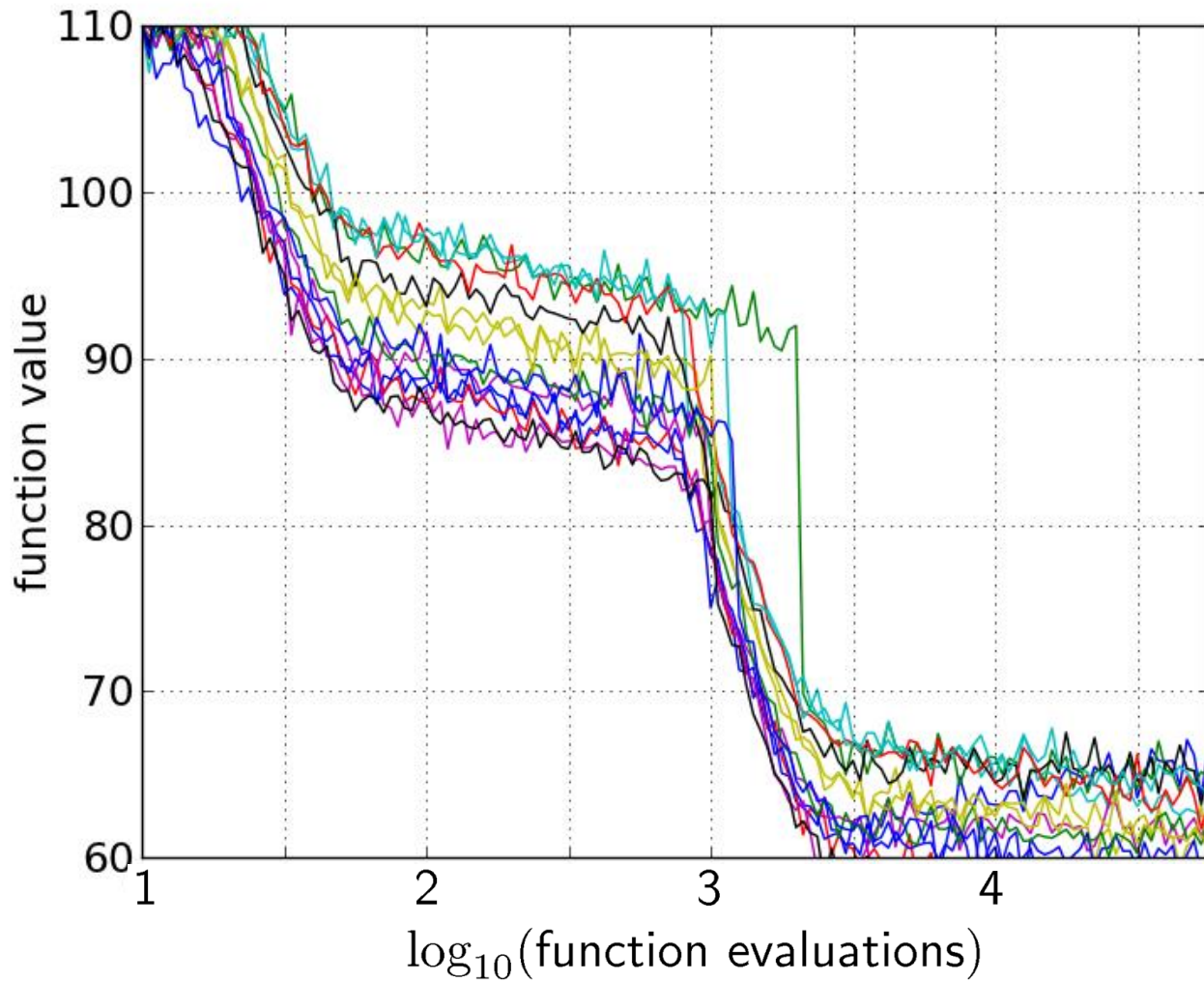
A Convergence Graph



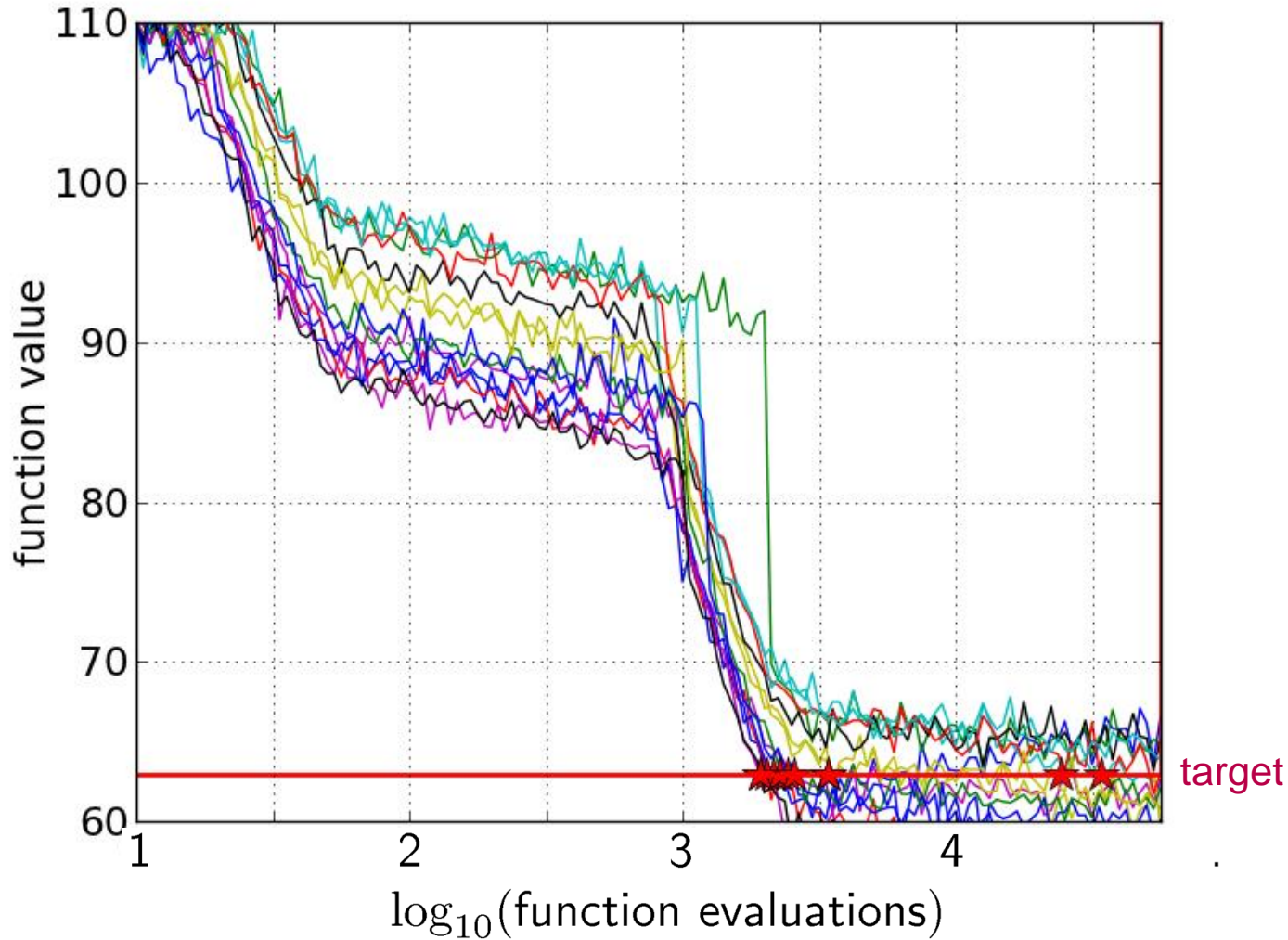
First Hitting Time is Monotonous



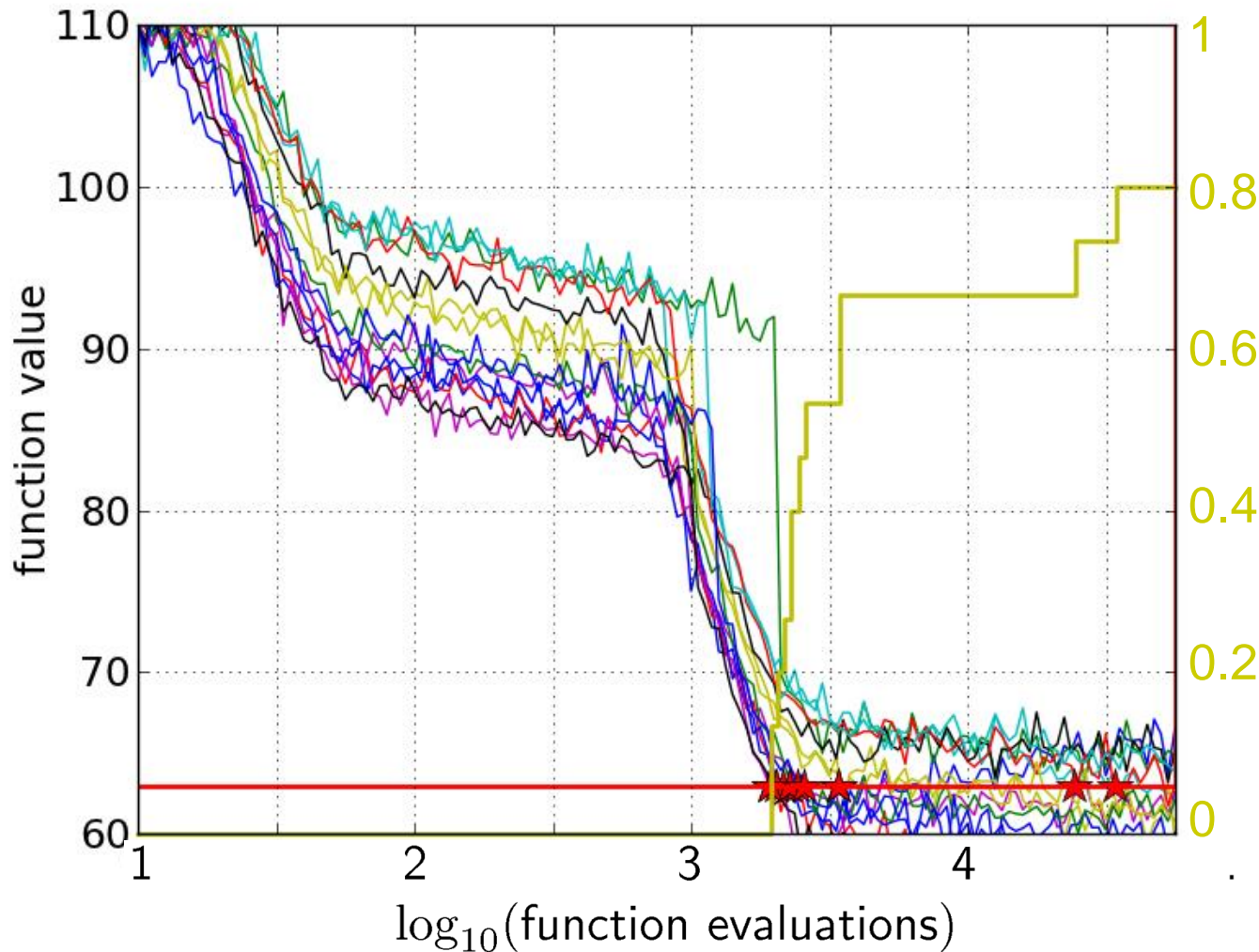
15 Runs



15 Runs \leq 15 Runtime Data Points



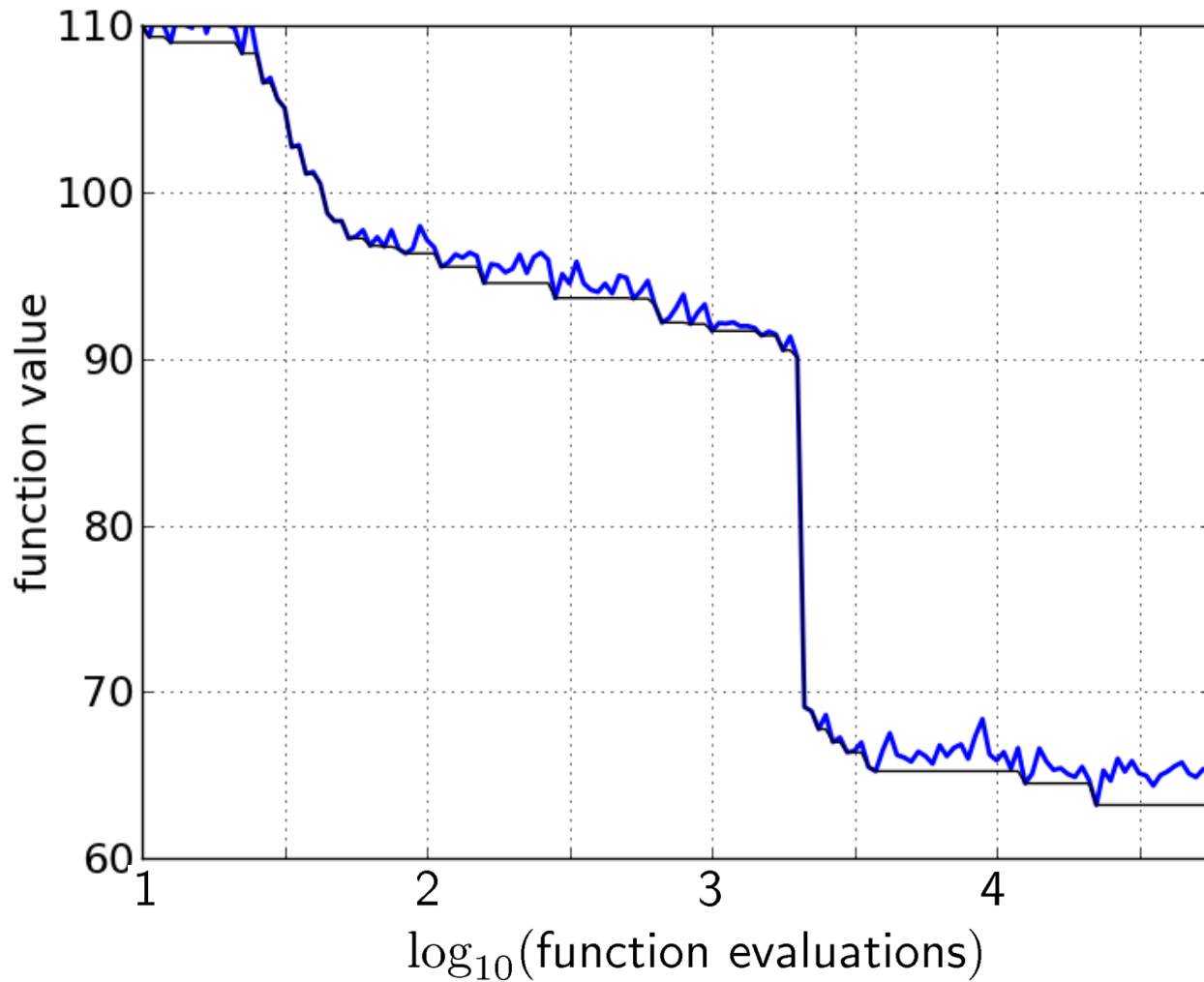
Empirical Cumulative Distribution



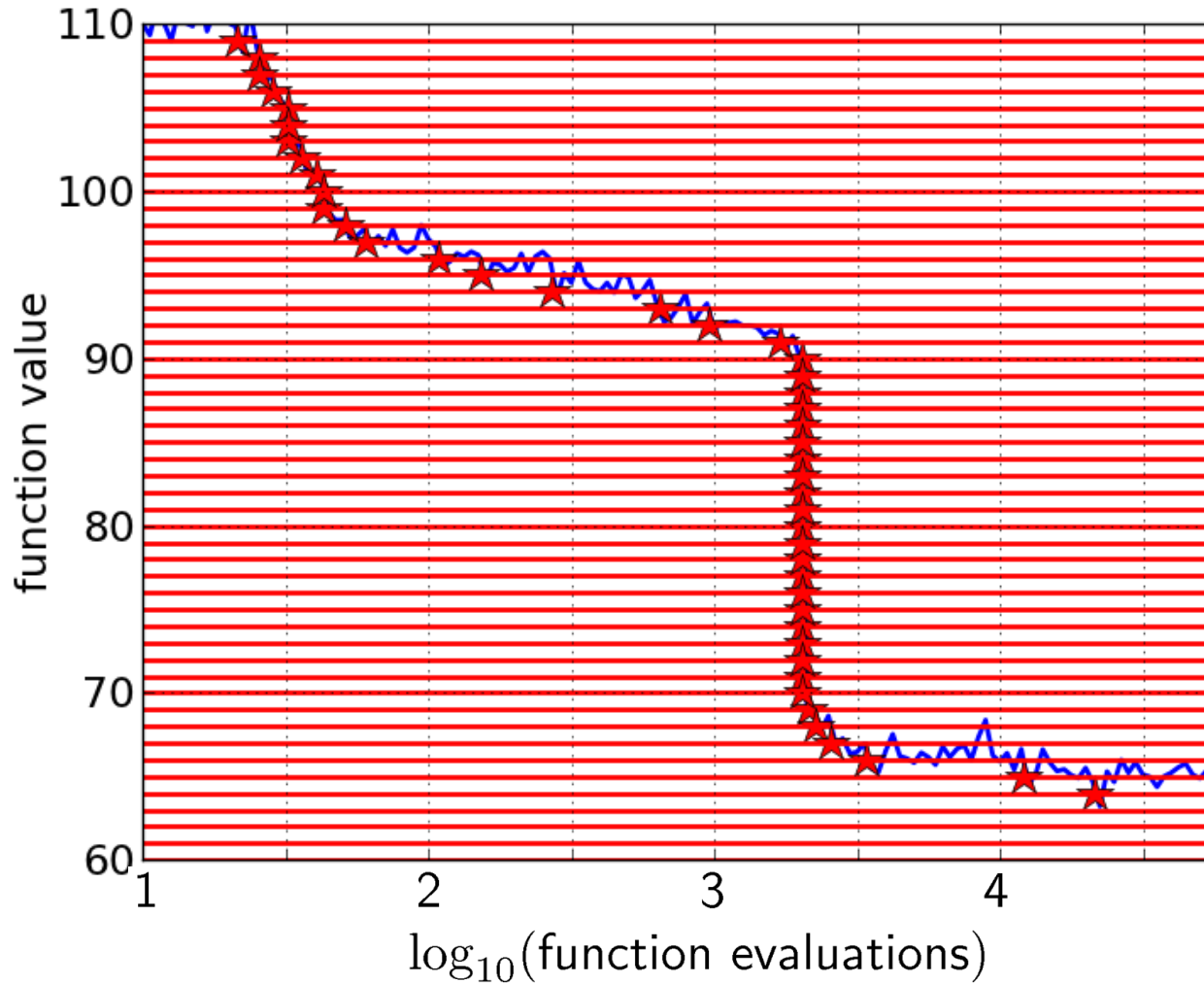
- 1 the **ECDF** of run lengths to reach the target
- has for each data point a **vertical step of constant size**
- displays for each x-value (budget) the count of observations to the left (first hitting times)

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

Reconstructing A Single Run

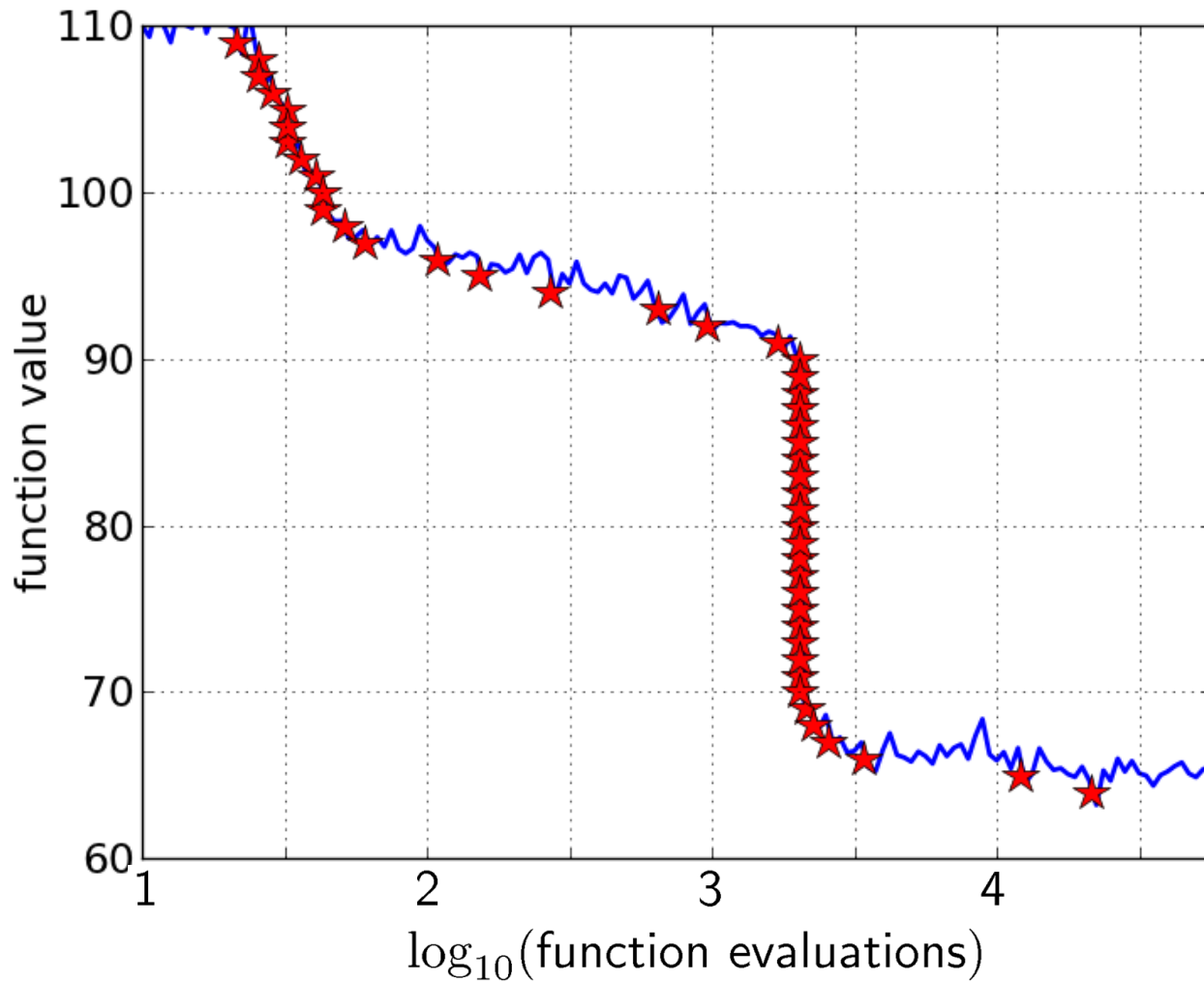


Reconstructing A Single Run

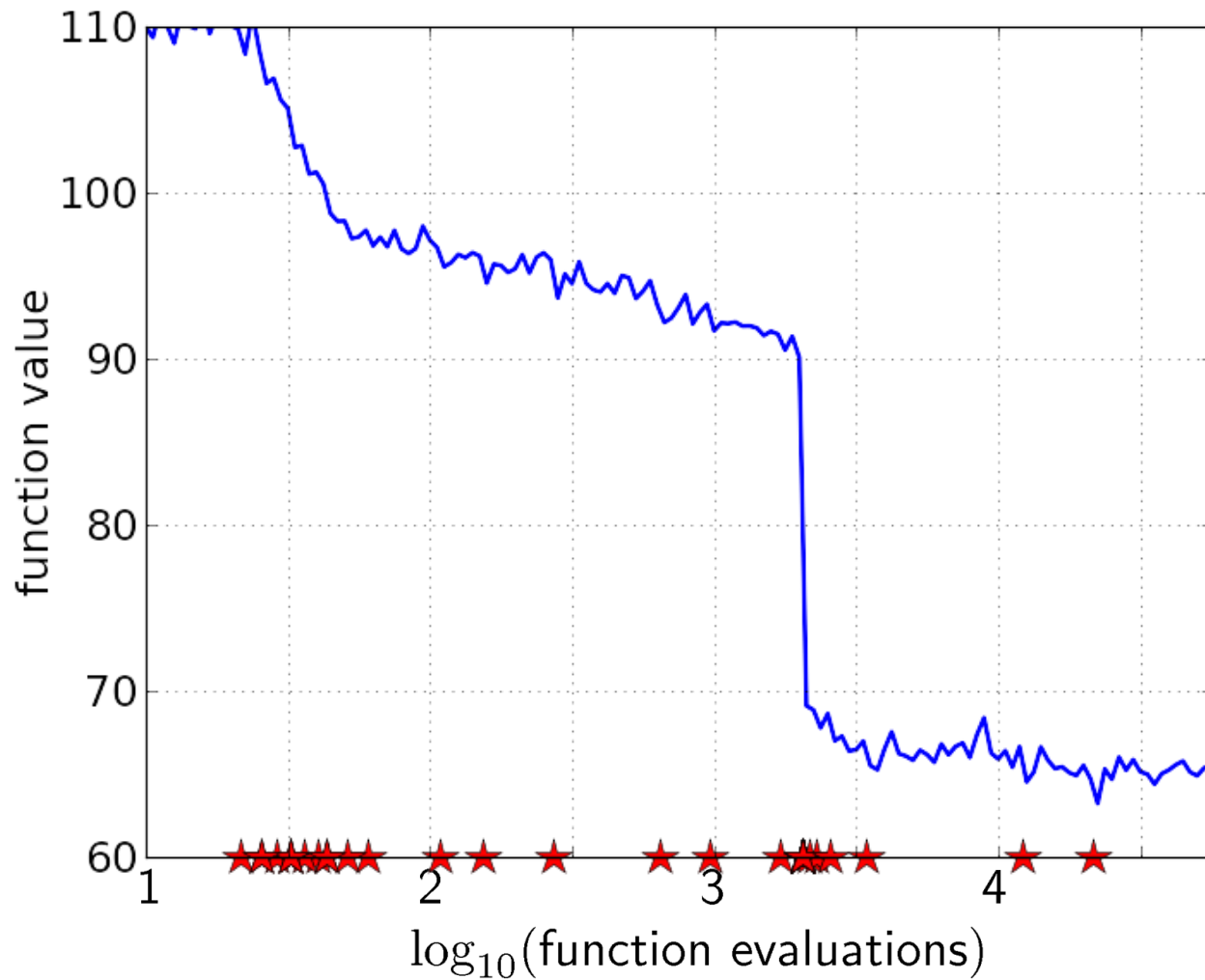


50 equally
spaced targets

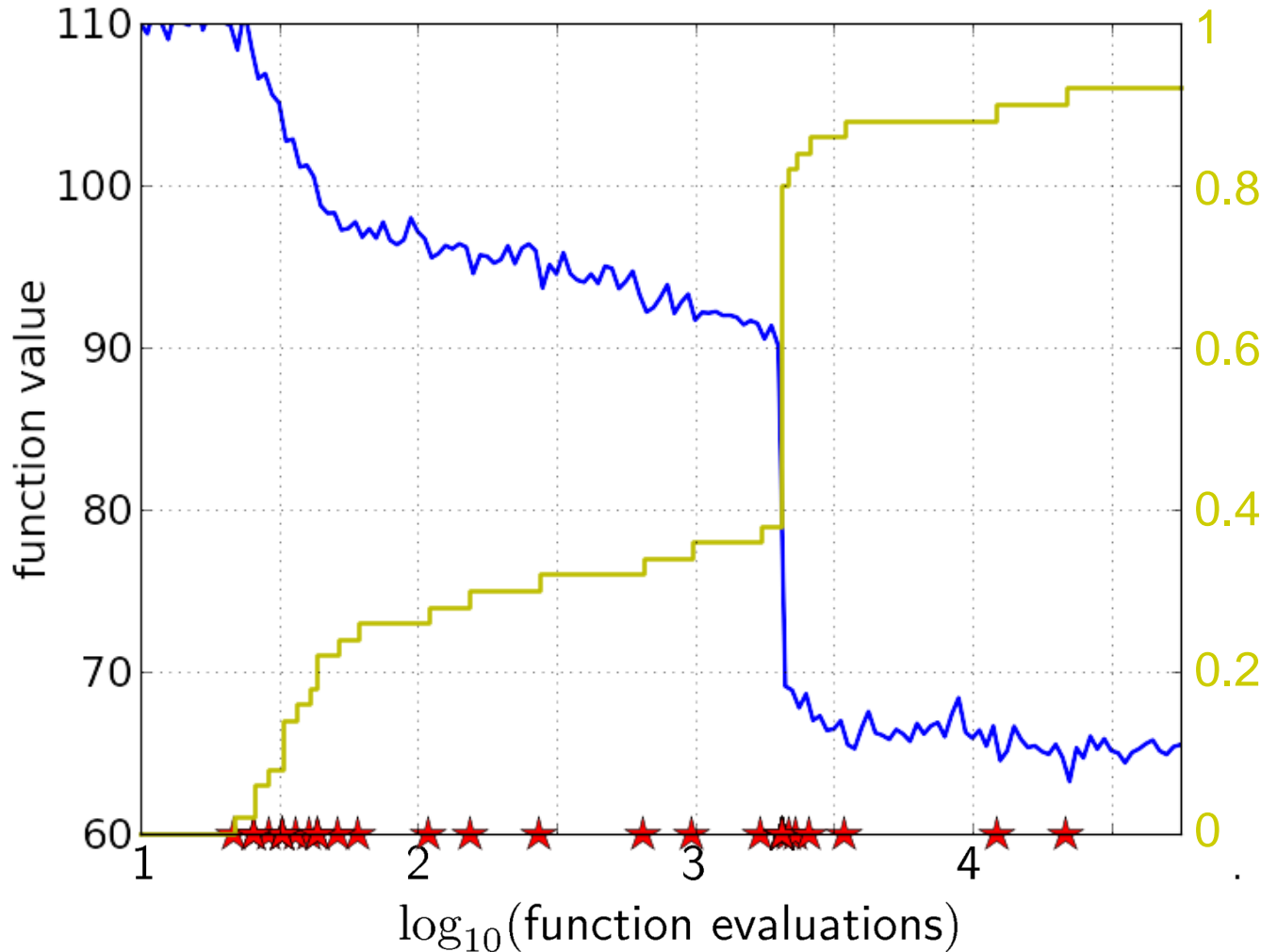
Reconstructing A Single Run



Reconstructing A Single Run

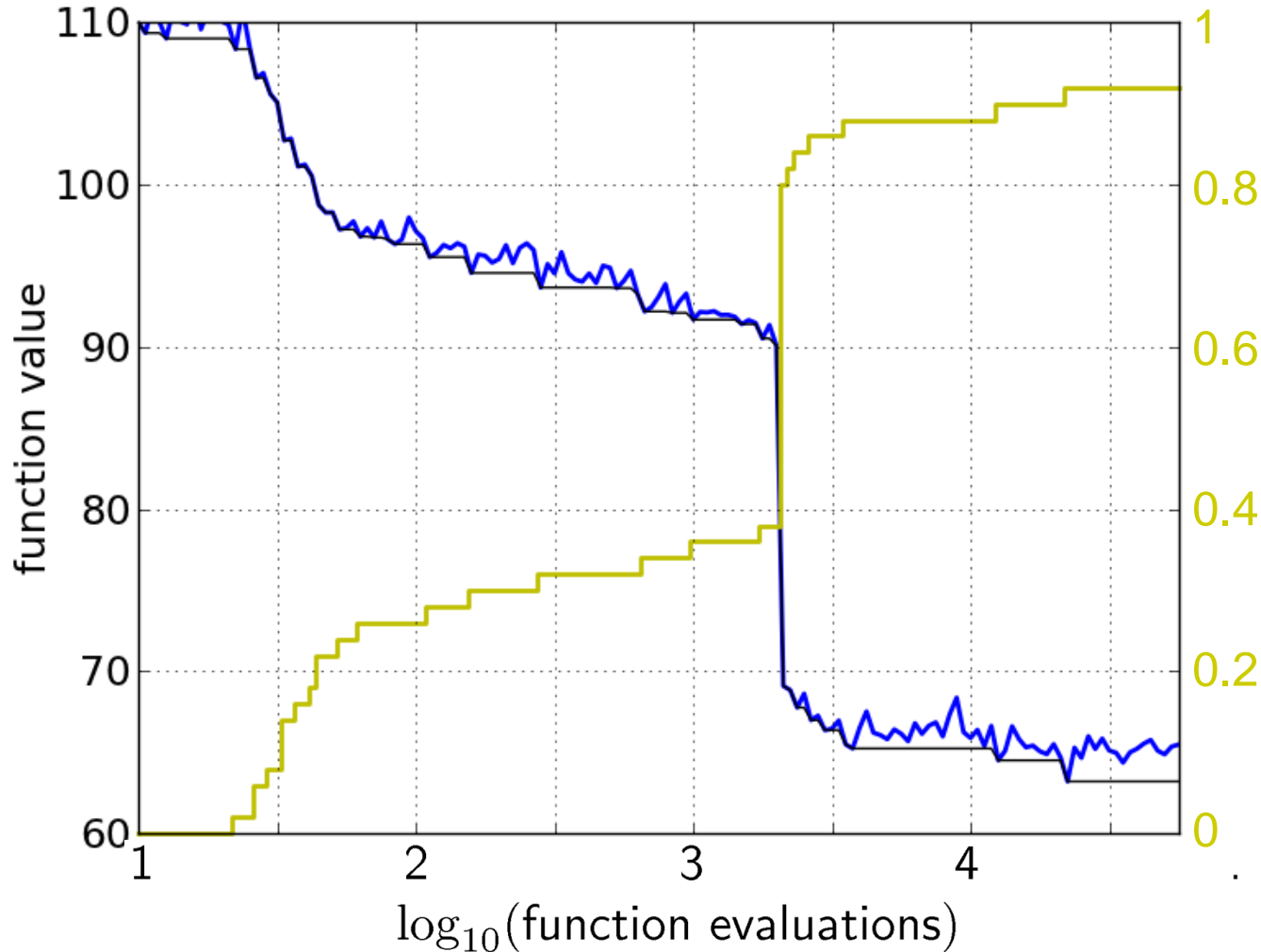


Reconstructing A Single Run



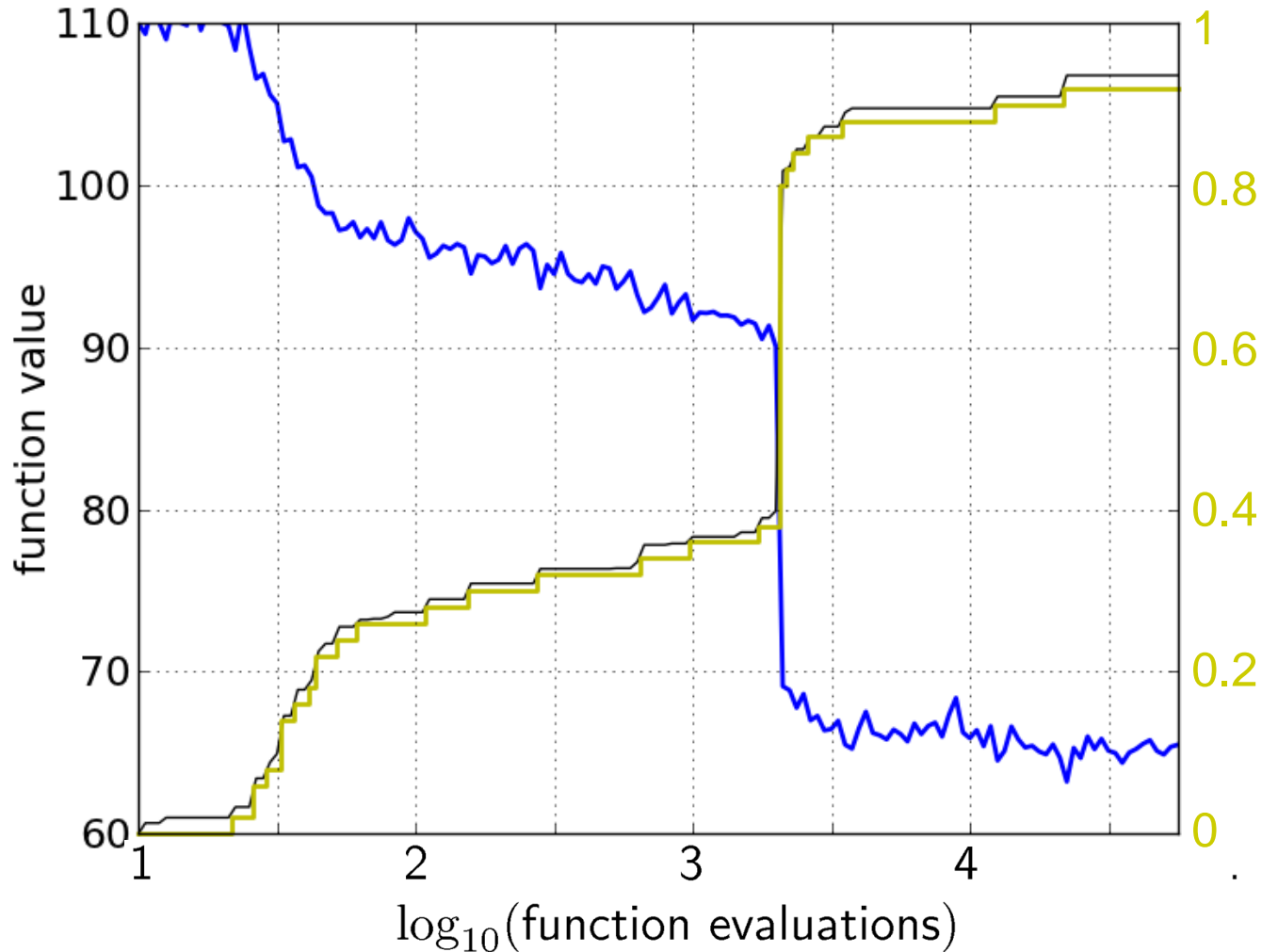
the empirical CDF makes a step for each star, is monotonous and displays for each budget the fraction of targets achieved within the budget

Reconstructing A Single Run



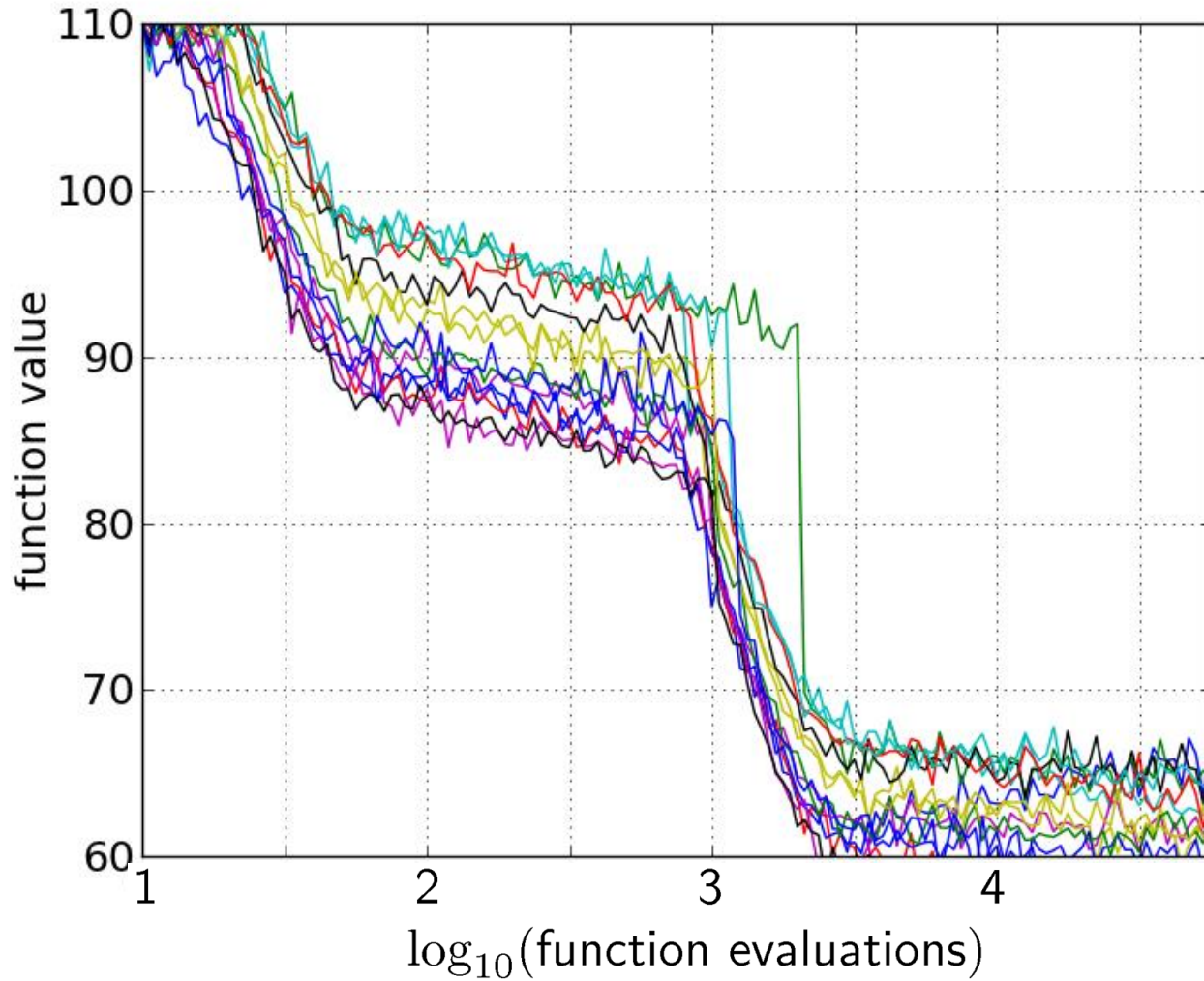
the ECDF recovers
the monotonous
graph,
discretised and
flipped

Reconstructing A Single Run



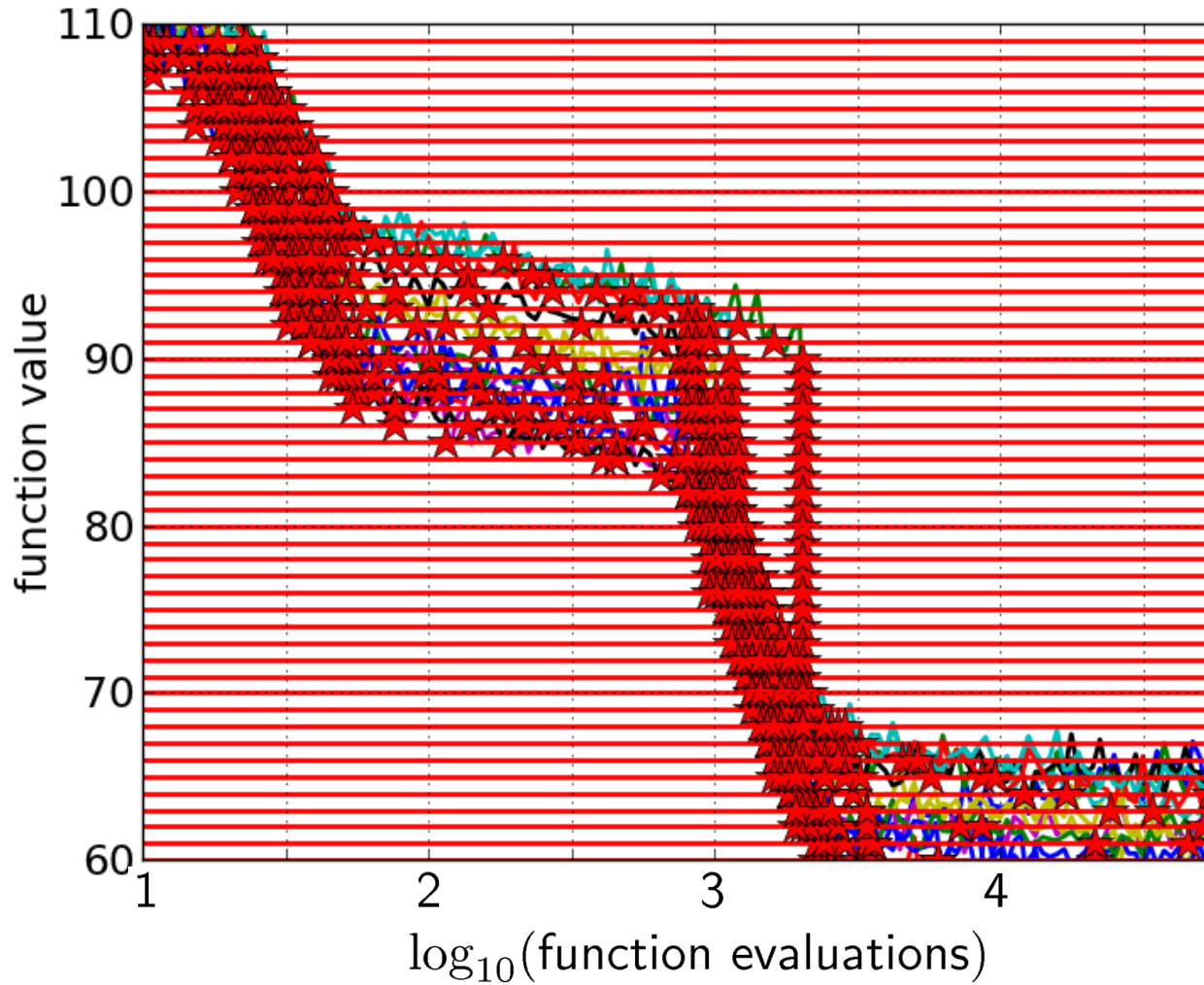
the ECDF recovers
the monotonous
graph,
discretised and
flipped

Aggregation



15 runs

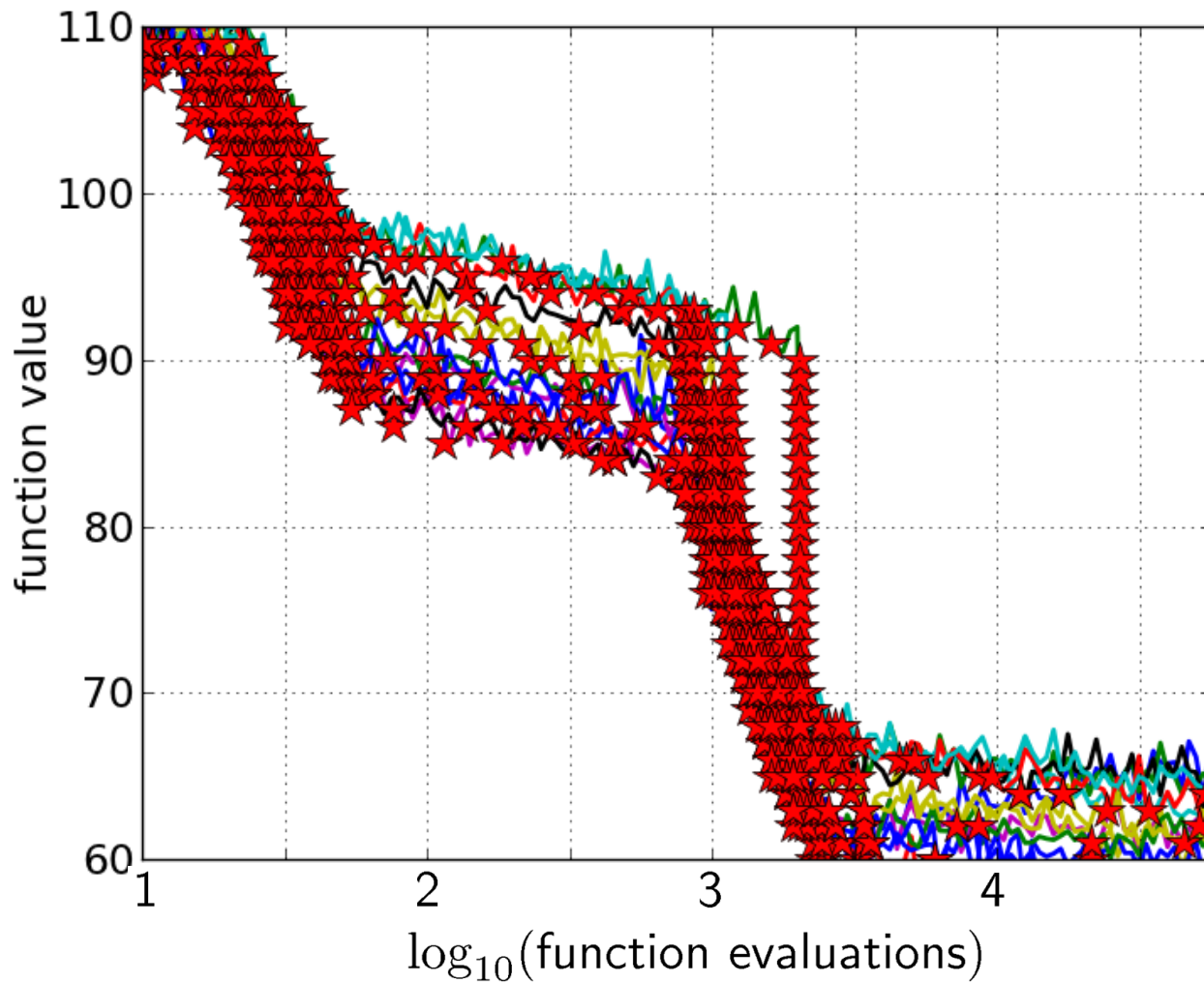
Aggregation



15 runs

50 targets

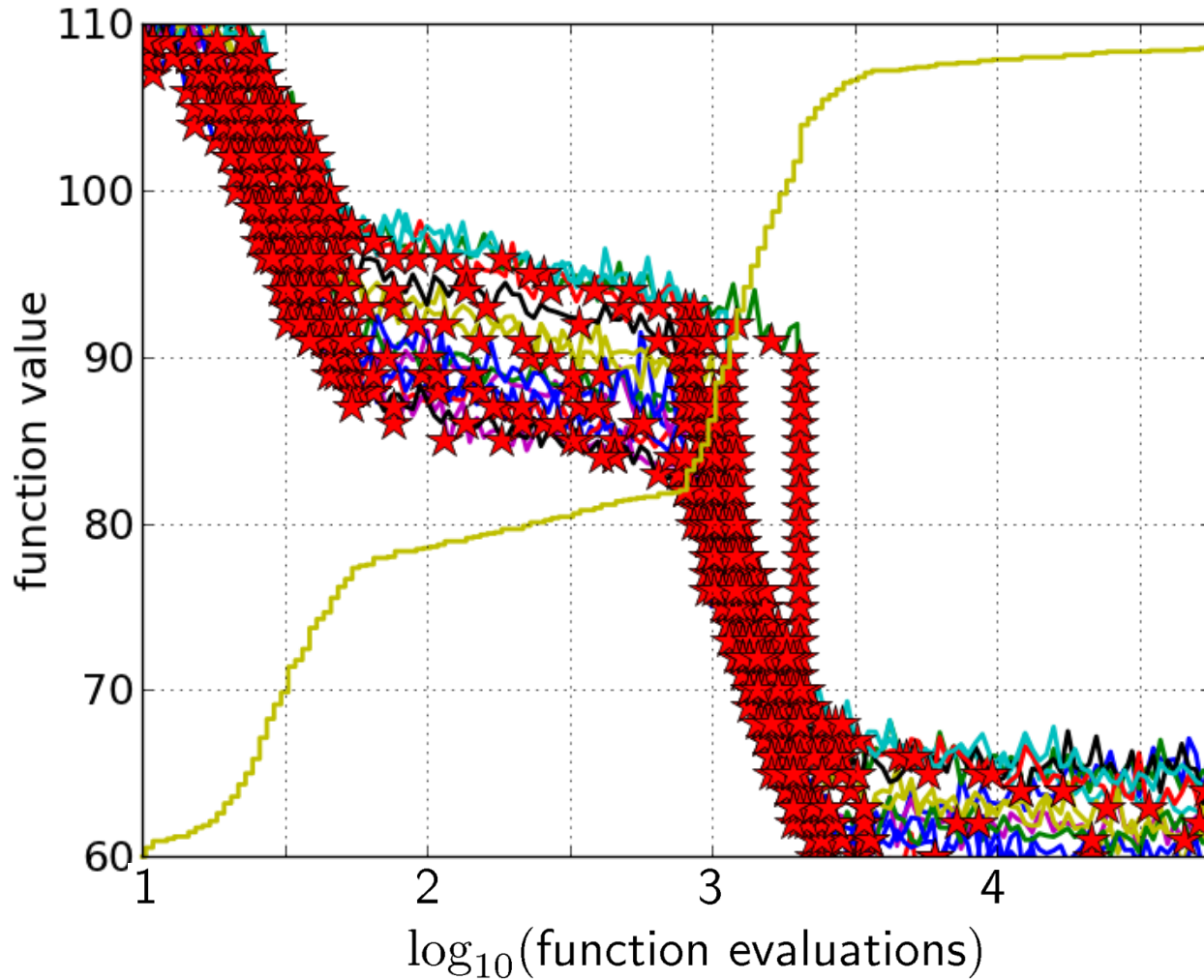
Aggregation



15 runs

50 targets

Aggregation

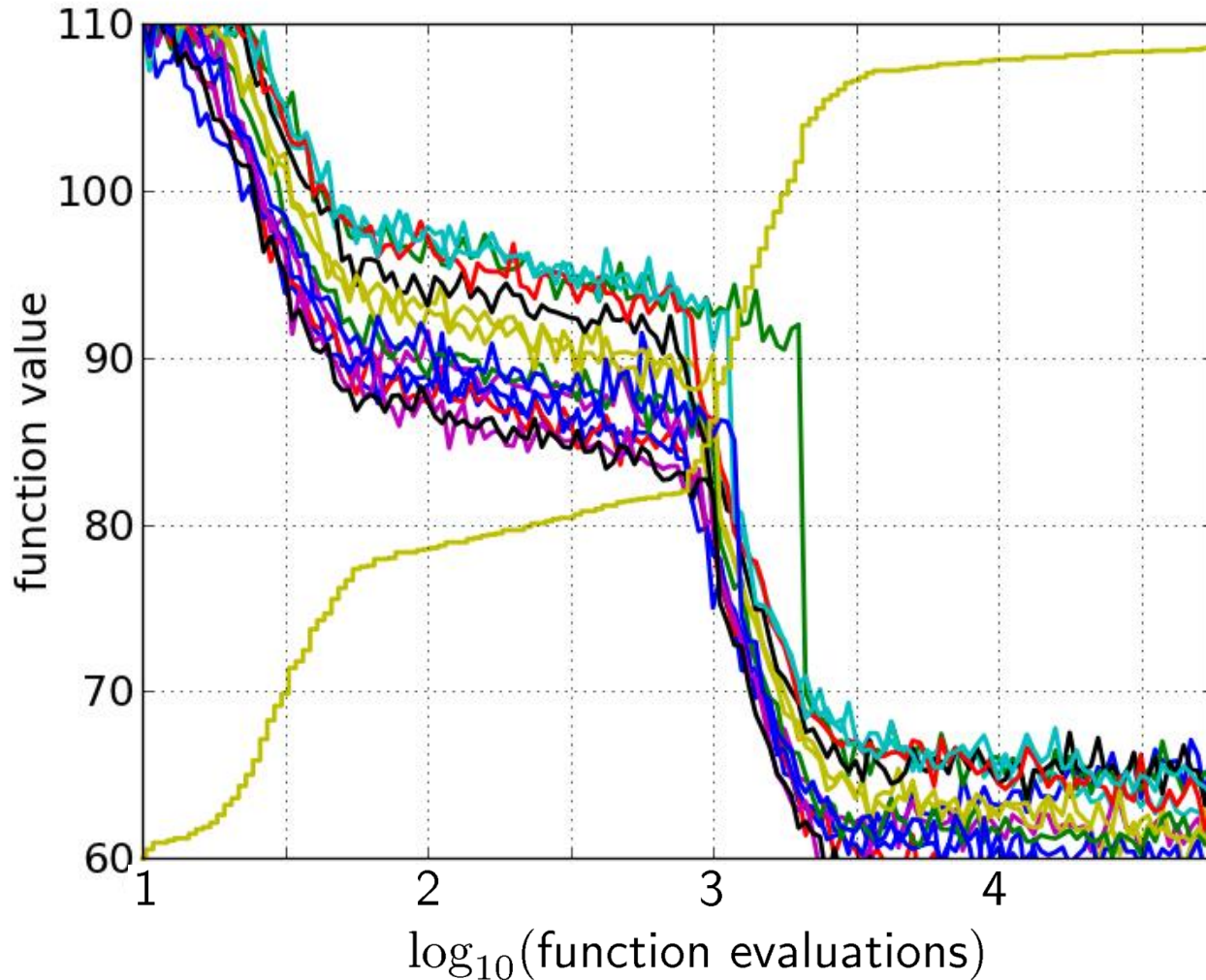


15 runs

50 targets

ECDF with 750
steps

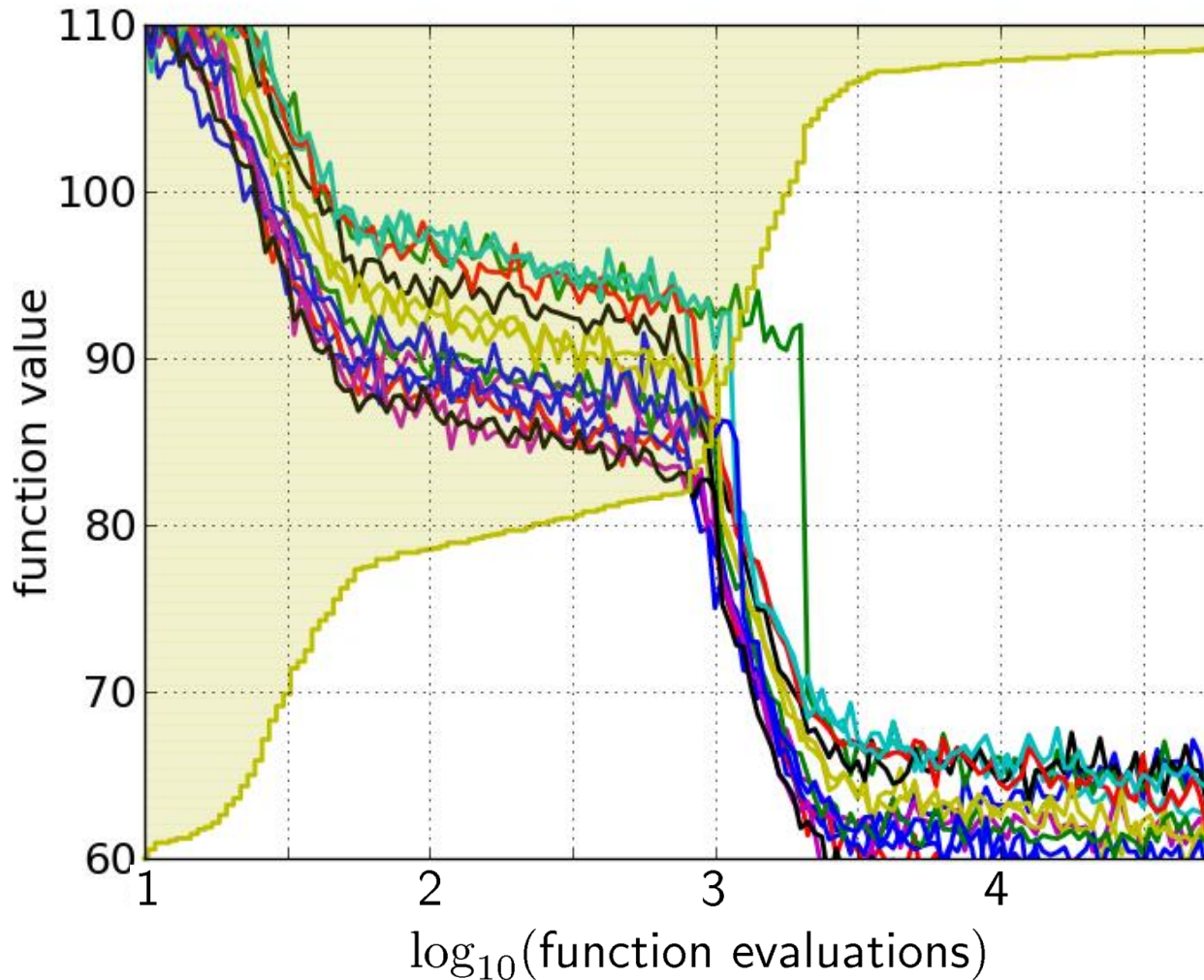
Aggregation



50 targets from
15 runs

...integrated in a
single graph

Interpretation



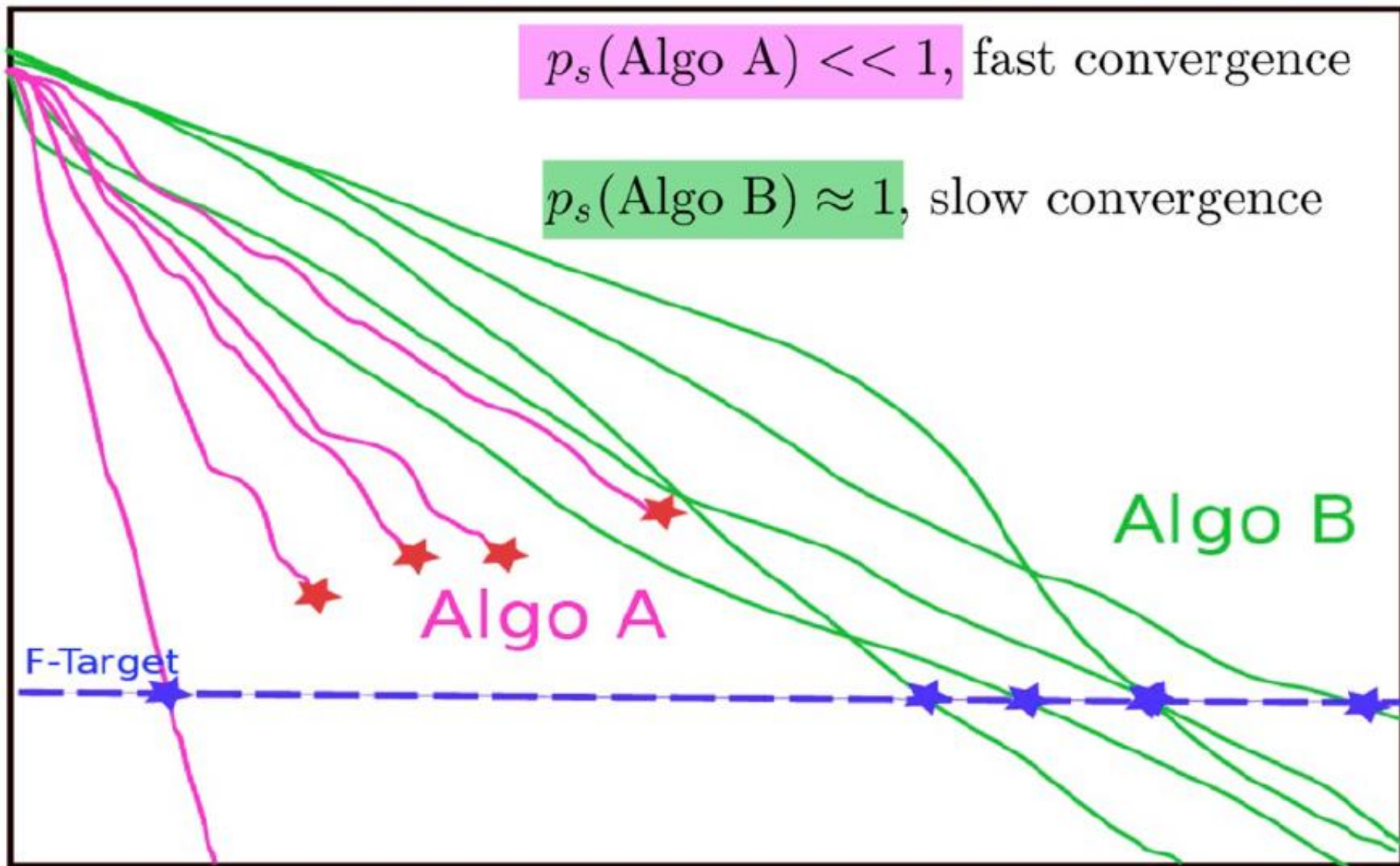
50 targets from
15 runs
integrated in a
single graph

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:



- Algo Restart B:



Fixed-target: Measuring Runtime

- Expected running time of the restarted algorithm:

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

- Estimator average running time (aRT):

$$\hat{p}_s = \frac{\text{\#successes}}{\text{\#runs}}$$

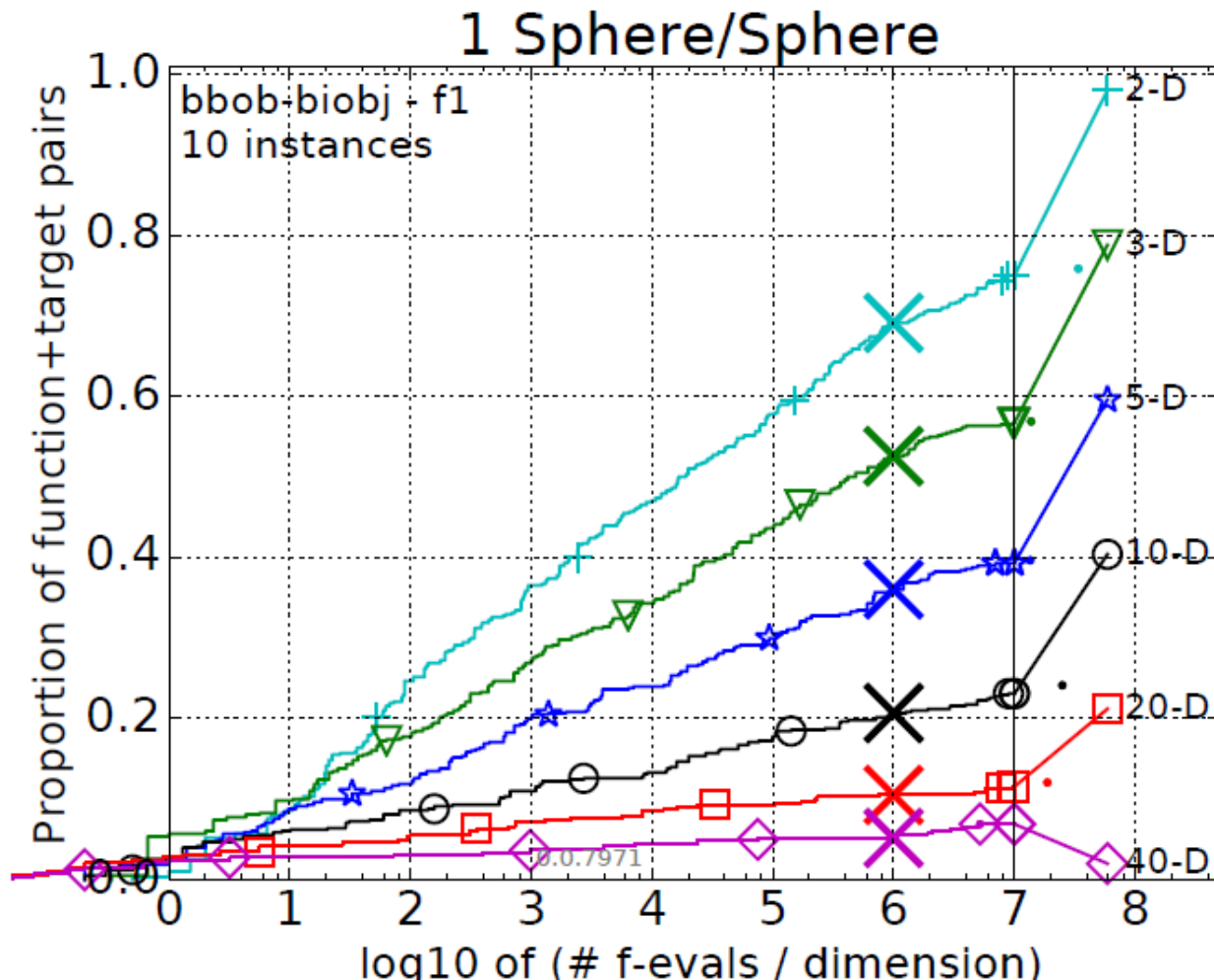
\widehat{RT}_{unsucc} = Average evals of unsuccessful runs

\widehat{RT}_{succ} = Average evals of successful runs

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

ECDFs with Simulated Restarts

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

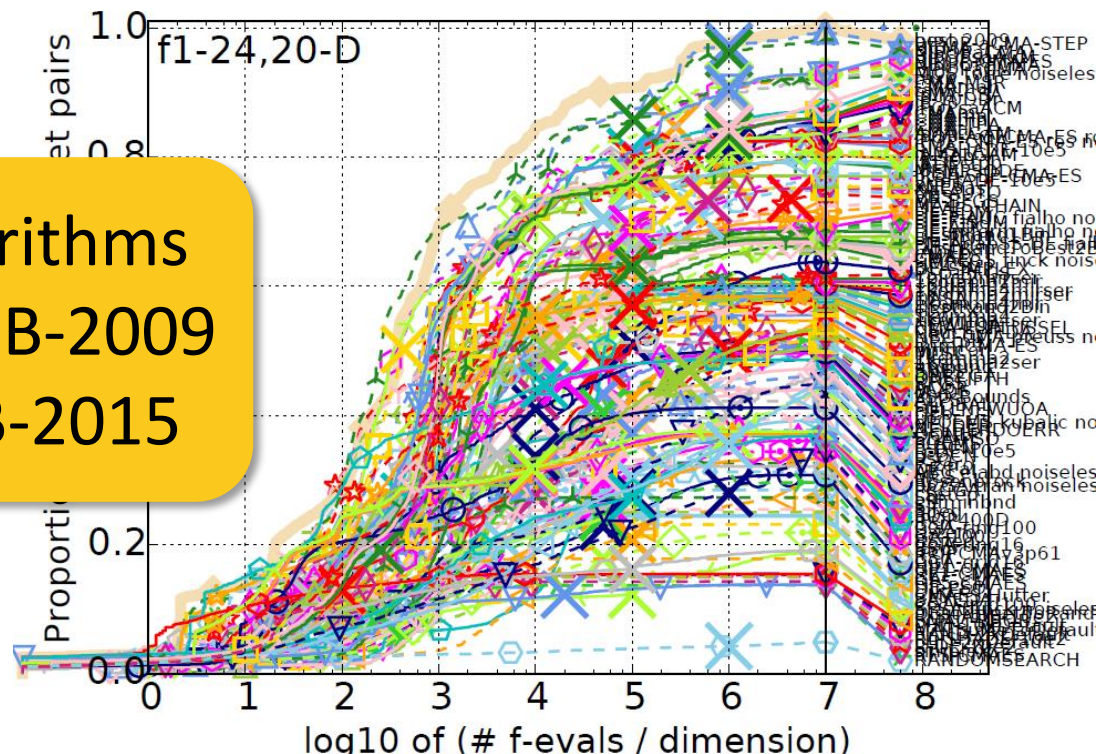


Worth to Note: ECDFs in COCO

In COCO, ECDF graphs

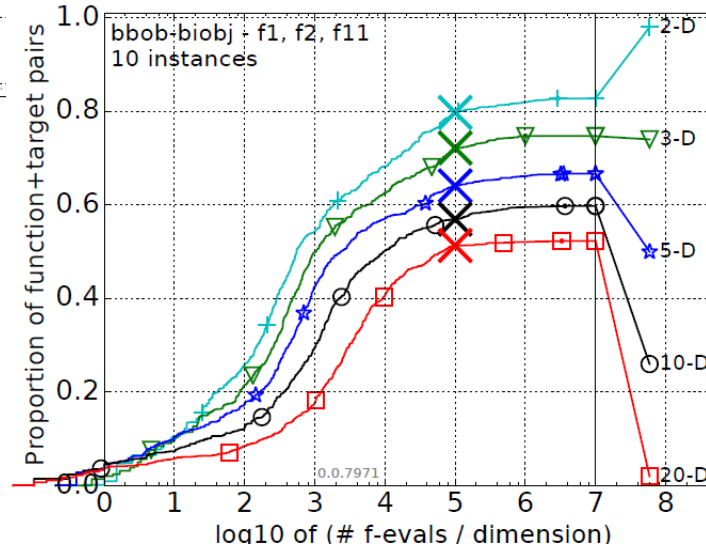
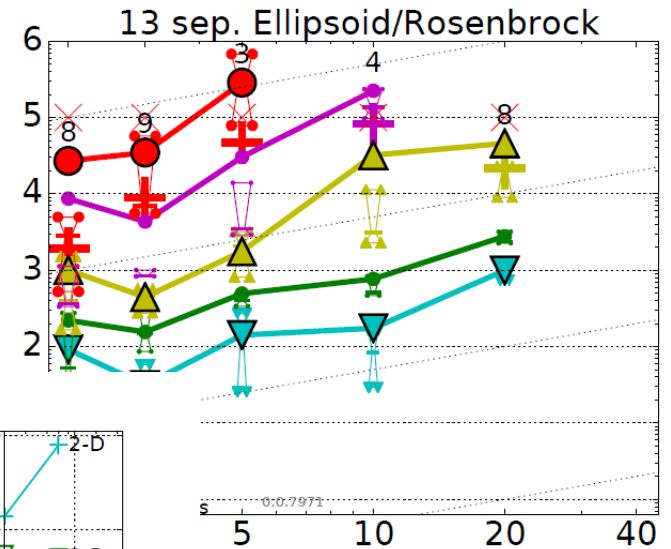
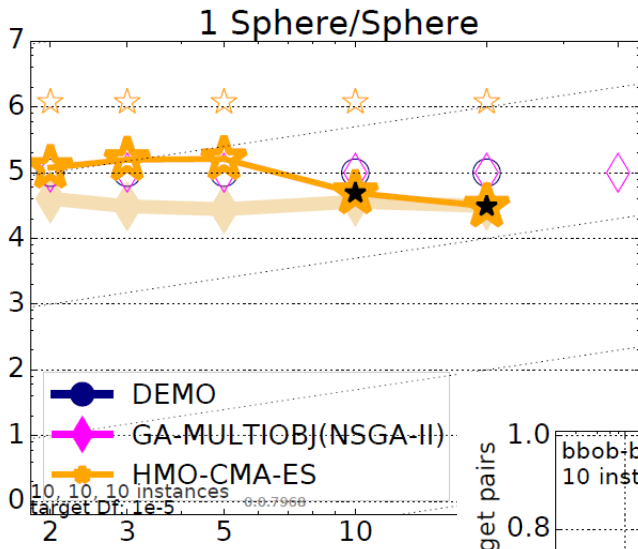
- never aggregate over dimension
 - but often over targets and functions
- can show data of more than 1 algorithm at a time

150 algorithms
from BBOB-2009
till BBOB-2015



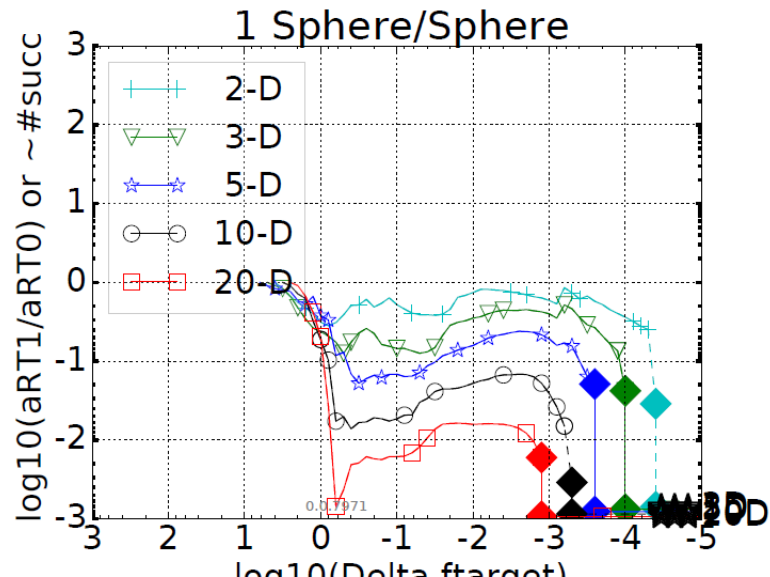
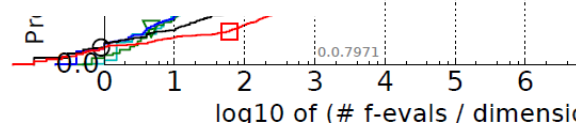
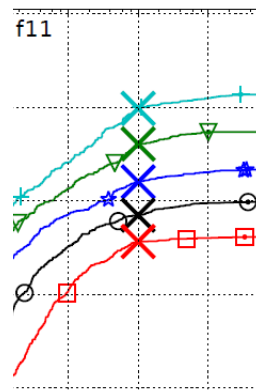
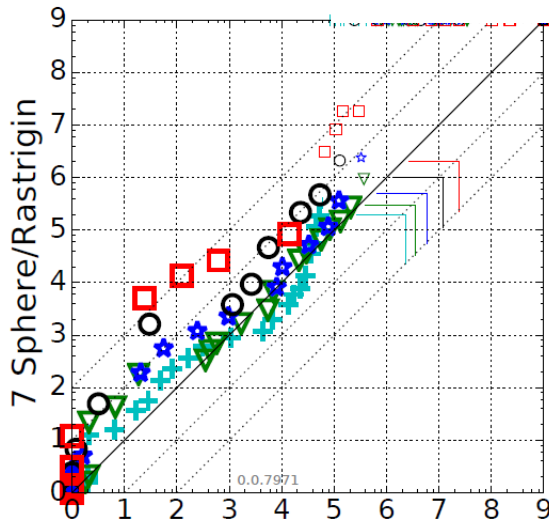
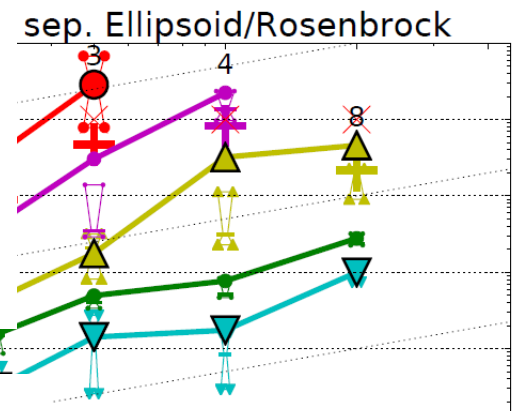
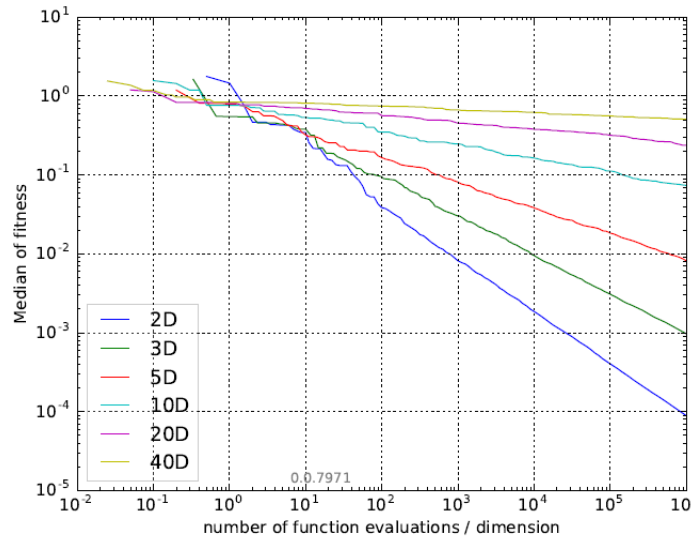
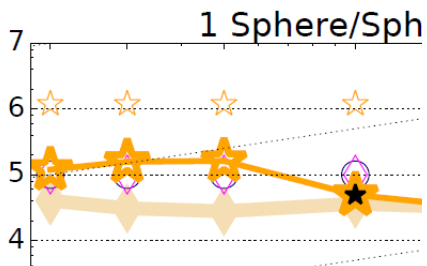
More Automated Plots...

...but no time to explain them here ☹️



More Automated Plots...















...but no time t













The single-objective BBOB functions

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

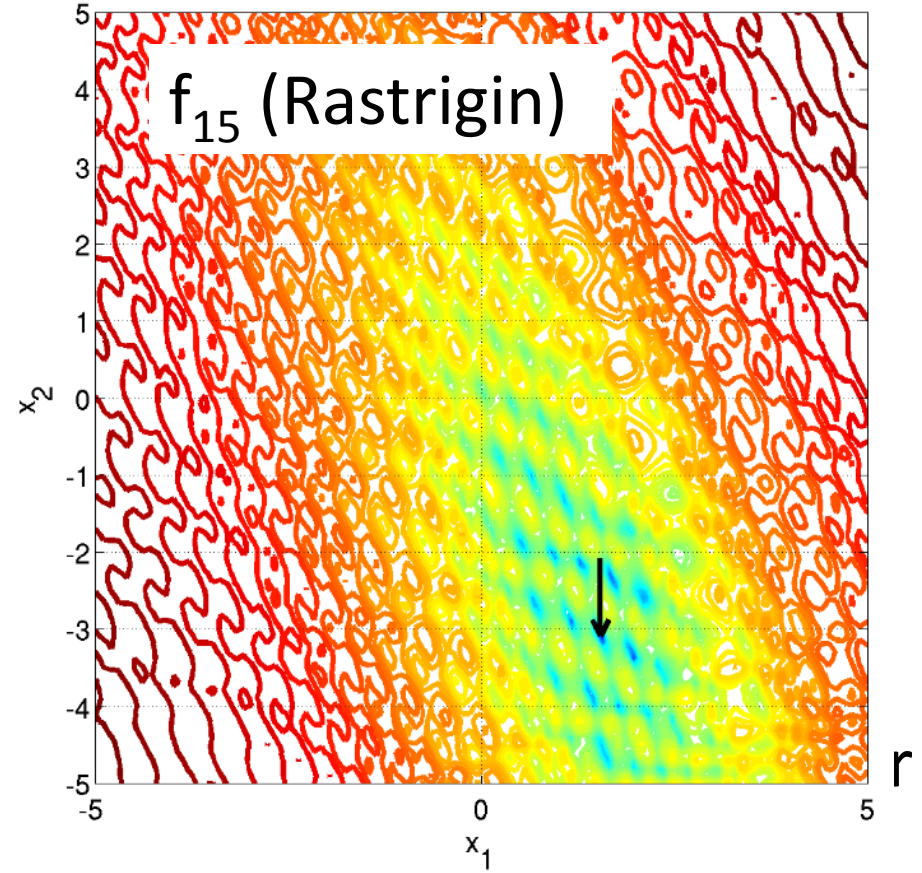
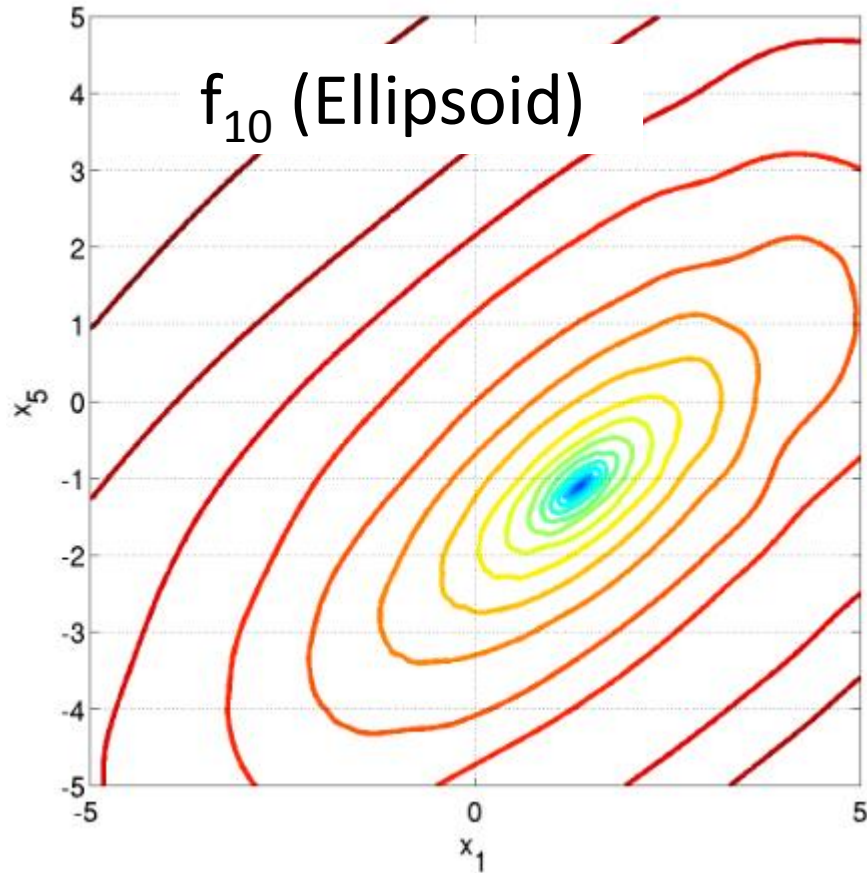
- All COCO problems come in form of instances
 - e.g. as translated/rotated versions of the same function
- Prescribed instances typically change from year to year
 - avoid overfitting
 - 5 instances are always kept the same

Plus:

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

Notion of Instances

- All COCO problems come in form of instances



bbob-noisy Testbed

- 30 functions with various kinds of noise types and strengths
 - 3 noise types: Gaussian, uniform, and seldom Cauchy
 - Functions with moderate noise
 - Functions with severe noise
 - Highly multi-modal functions with severe noise
 - **bbob** functions included: Sphere, Rosenbrock, Step ellipsoid, Ellipsoid, Different Powers, Schaffers' F7, Composite Griewank-Rosenbrock
- 6 dimensions: 2, 3, 5, 10, 20, (40 optional)

**the recent extension to
multi-objective optimization**

bbob-biobj Testbed (new in 2016)

- 55 functions by combining 2 bbob functions

1 Separable Functions	
f1	<input type="checkbox"/> Sphere Function ✓
f2	<input type="checkbox"/> Ellipsoidal Function ✓
f3	<input type="checkbox"/> Rastrigin Function
f4	<input type="checkbox"/> Büche-Rastrigin Function
f5	<input type="checkbox"/> Linear Slope
2 Functions with low or moderate conditioning	
f6	<input type="checkbox"/> Attractive Sector Function ✓
f7	<input type="checkbox"/> Step Ellipsoidal Function
f8	<input type="checkbox"/> Rosenbrock Function, original ✓
f9	<input type="checkbox"/> Rosenbrock Function, rotated
3 Functions with high conditioning and unimodal	
f10	<input type="checkbox"/> Ellipsoidal Function
f11	<input type="checkbox"/> Discus Function
f12	<input type="checkbox"/> Bent Cigar Function
f13	<input type="checkbox"/> Sharp Ridge Function ✓
f14	<input type="checkbox"/> Different Powers Function ✓

4 Multi-modal functions with adequate global structure	
f15	<input type="checkbox"/> Rastrigin Function ✓
f16	<input type="checkbox"/> Weierstrass Function
f17	<input type="checkbox"/> Schaffers F7 Function ✓
f18	<input type="checkbox"/> Schaffers F7 Functions, moderately ill-conditioned
f19	<input type="checkbox"/> Composite Griewank-Rosenbrock Function F8F2
5 Multi-modal functions with weak global structure	
f20	<input type="checkbox"/> Schwefel Function ✓
f21	<input type="checkbox"/> Gallagher's Gaussian 101-me Peaks Function ✓
f22	<input type="checkbox"/> Gallagher's Gaussian 21-hi Peaks Function
f23	<input type="checkbox"/> Katsuura Function
f24	<input type="checkbox"/> Lunacek bi-Rastrigin Function

bbob-biobj Testbed (new in 2016)

- 55 functions by combining 2 bbob functions

1 Separable Functions		4 Multi-modal functions with adequate global structure										
f1	<input checked="" type="checkbox"/> Sphere Function ✓	f15	<input checked="" type="checkbox"/> Rastrigin Function ✓									
f2	<input checked="" type="checkbox"/> Ellipsoidal Function ✓	f16	<input type="checkbox"/> Weierstrass Function									
f3	<input type="checkbox"/> Rastrigin Function	f17	<input checked="" type="checkbox"/> Schaffers F7 Function ✓									
f4	<input type="checkbox"/> Büche-Rastrigin Function											
f5	<input type="checkbox"/> Linear Slope											
2 Functions with low or moderate conditioning			f_1	f_2	f_6	f_8	f_{13}	f_{14}	f_{15}	f_{17}	f_{20}	f_{21}
f6	<input checked="" type="checkbox"/> Attractive Sector Function ✓	f_1	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
f7	<input type="checkbox"/> Step Ellipsoidal Function	f_2		f11	f12	f13	f14	f15	f16	f17	f18	f19
f8	<input checked="" type="checkbox"/> Rosenbrock Function, original ✓	f_6			f20	f21	f22	f23	f24	f25	f26	f27
f9	<input type="checkbox"/> Rosenbrock Function, rotated	f_8				f28	f29	f30	f31	f32	f33	f34
3 Functions with high conditioning and unimodal		f_{13}					f35	f36	f37	f38	f39	f40
f10	<input type="checkbox"/> Ellipsoidal Function	f_{14}						f41	f42	f43	f44	f45
f11	<input type="checkbox"/> Discus Function	f_{15}							f46	f47	f48	f49
f12	<input type="checkbox"/> Bent Cigar Function	f_{17}								f50	f51	f52
f13	<input checked="" type="checkbox"/> Sharp Ridge Function ✓	f_{20}									f53	f54
f14	<input checked="" type="checkbox"/> Different Powers Function ✓	f_{21}										f55

bbob-biobj Testbed (new in 2016)

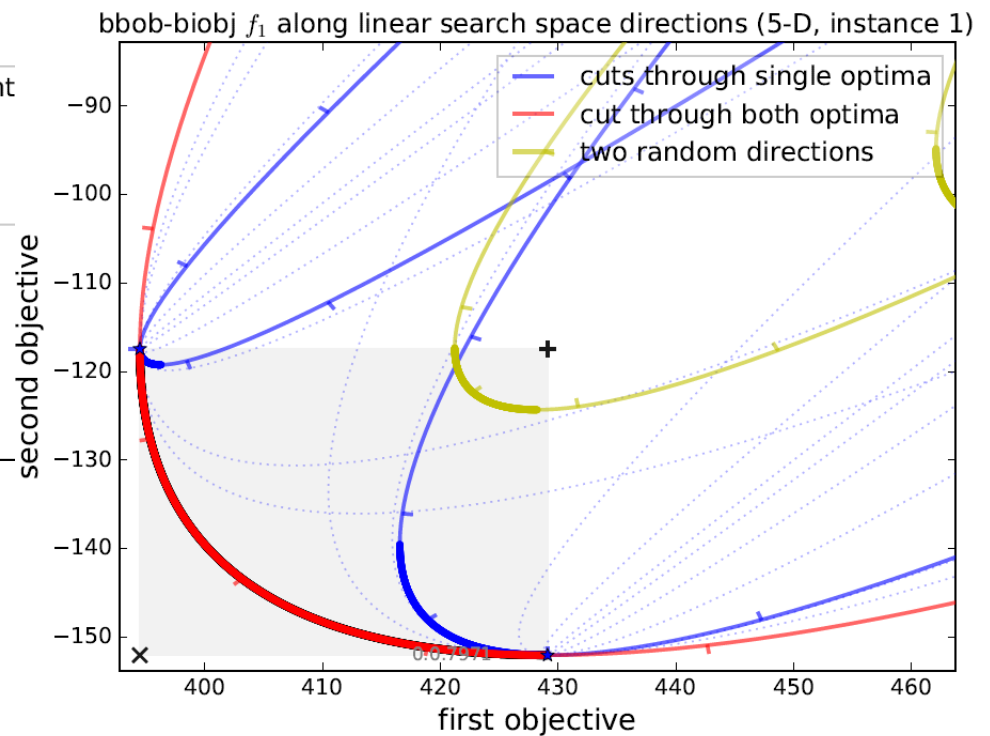
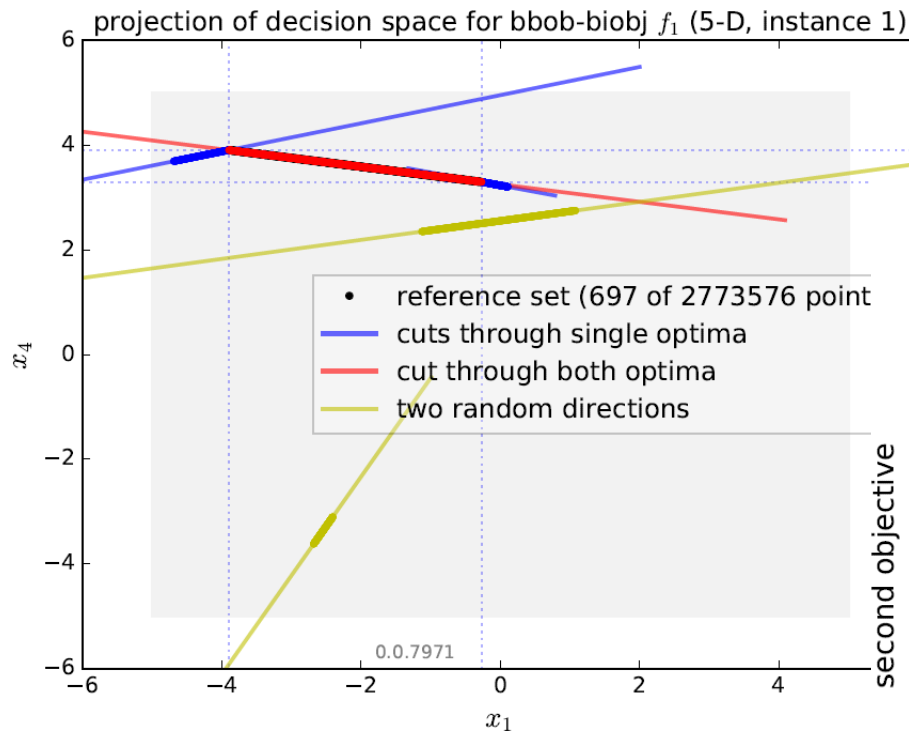
- **55 functions** by combining 2 **bbob** functions
- **15 function groups** with 3-4 functions each
 - separable – separable, separable – moderate, separable - ill-conditioned, ...
- **6 dimensions**: 2, 3, 5, 10, 20, (40 optional)
- instances derived from **bbob** instances:
 - more or less 2^{i+1} for 1st objective and 2^{i+2} for 2nd objective
 - exceptions: instances 1 and 2 and when optima are too close
- **no normalization** (algo has to cope with different orders of magnitude)
- for performance assessment: **ideal/nadir points known**

bbob-biobj Testbed (cont'd)

- Pareto set and Pareto front **unknown**
 - but we have a good idea of where they are by running quite some algorithms and keeping track of all non-dominated points found so far
- Various types of shapes

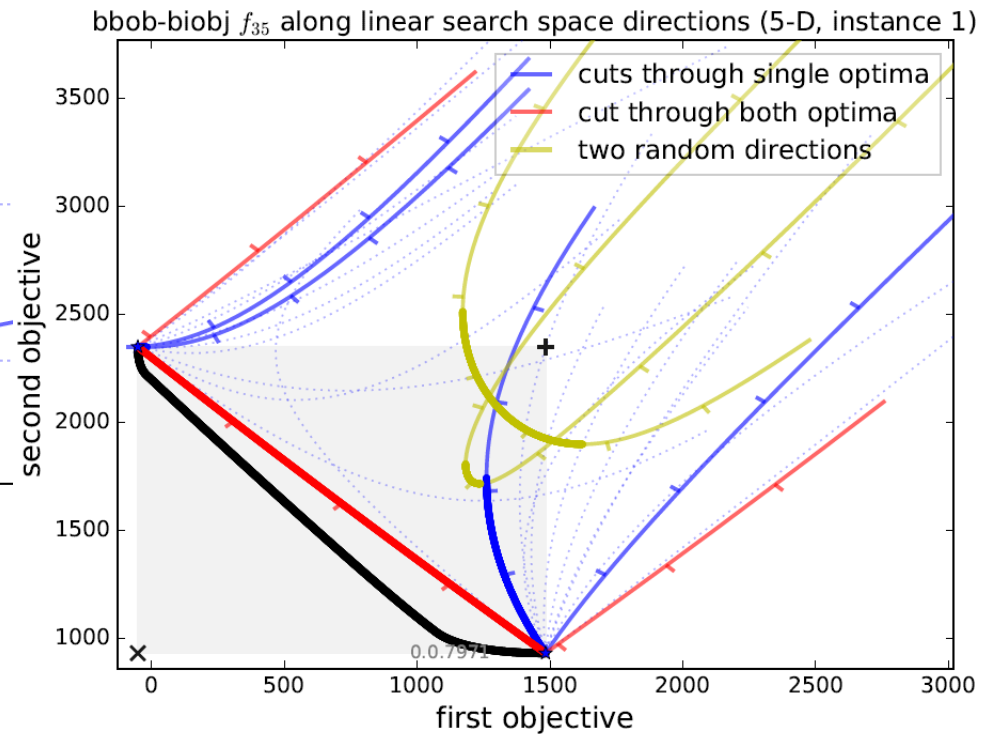
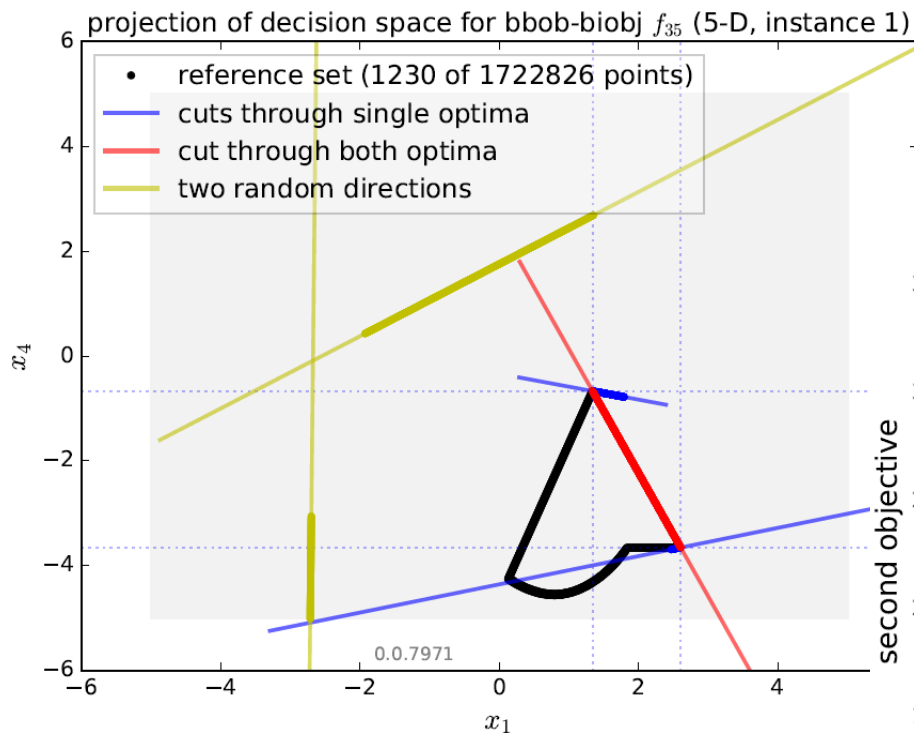
bbob-biobj Testbed (cont'd)

Example: sphere with sphere



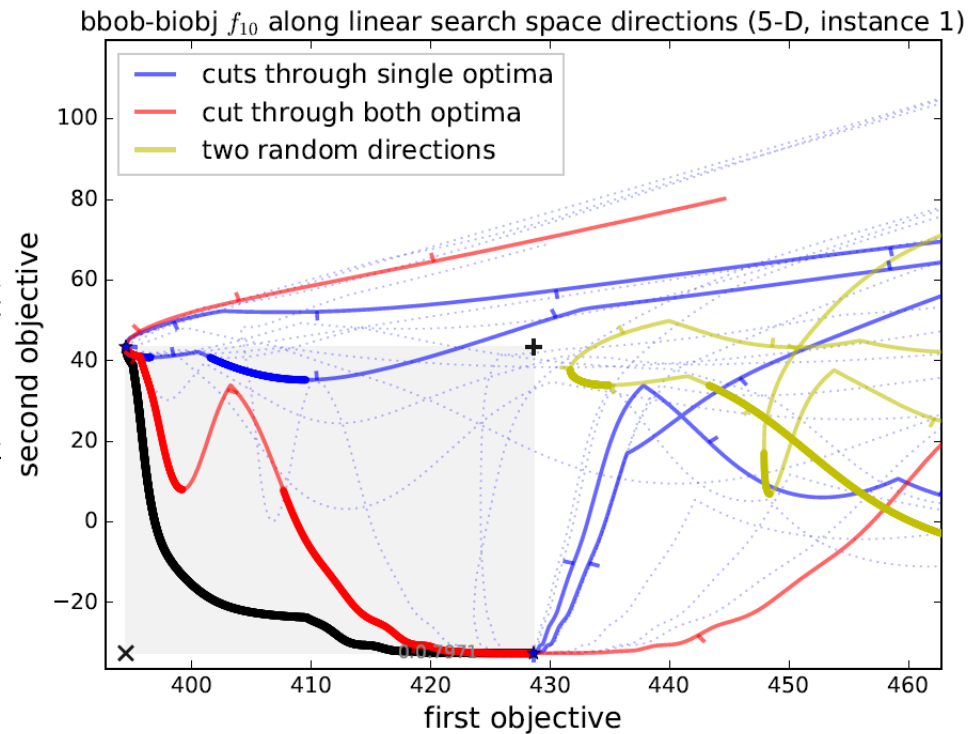
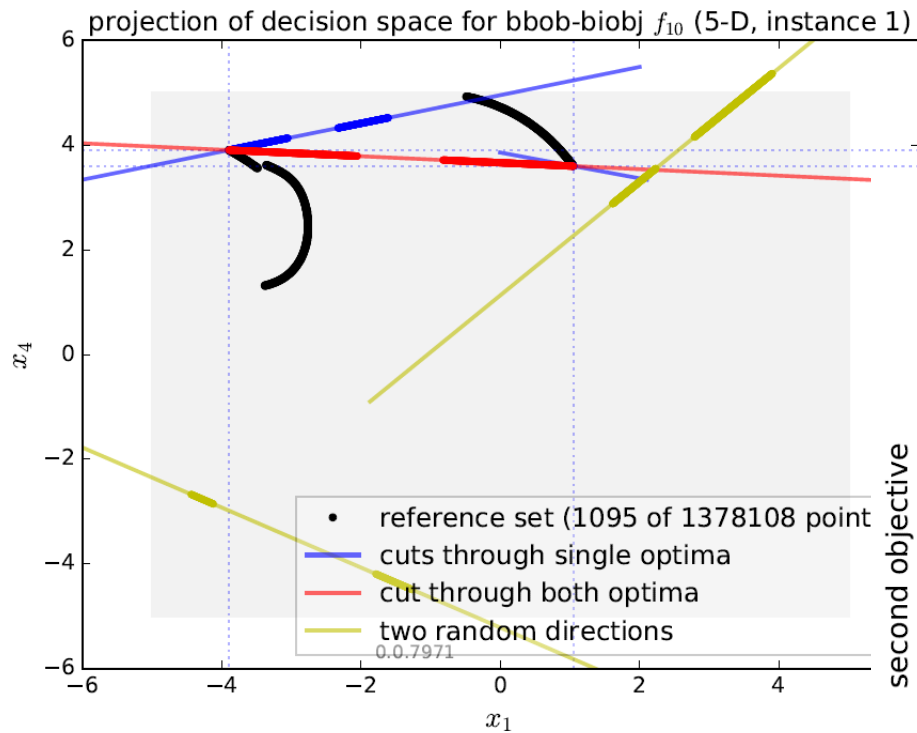
bbob-biobj Testbed (cont'd)

Example: sharp ridge with sharp ridge



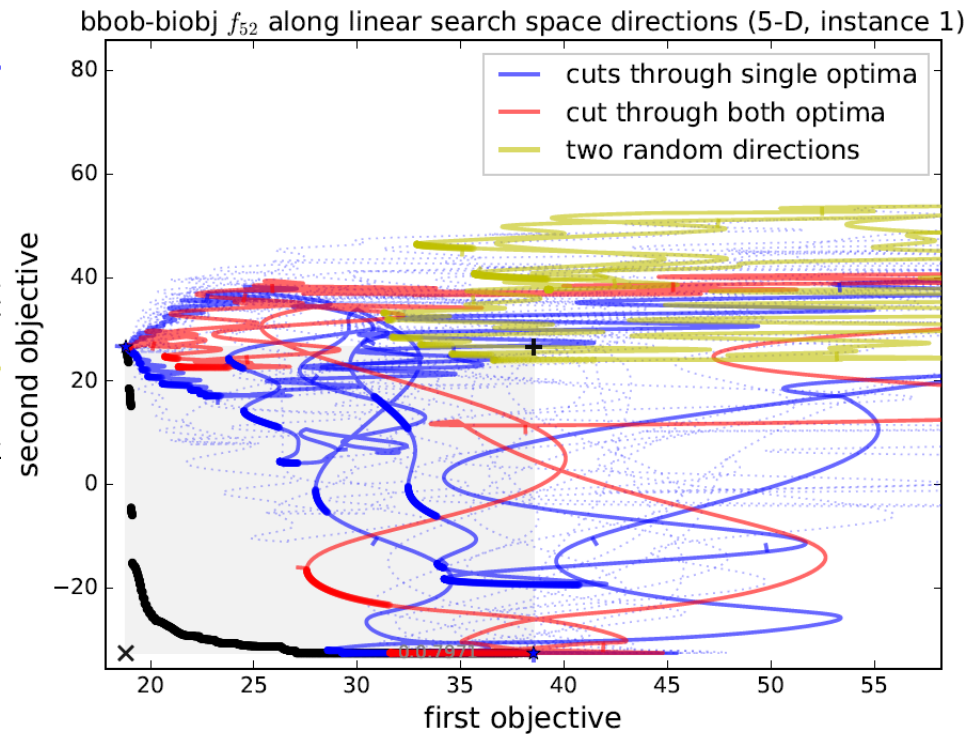
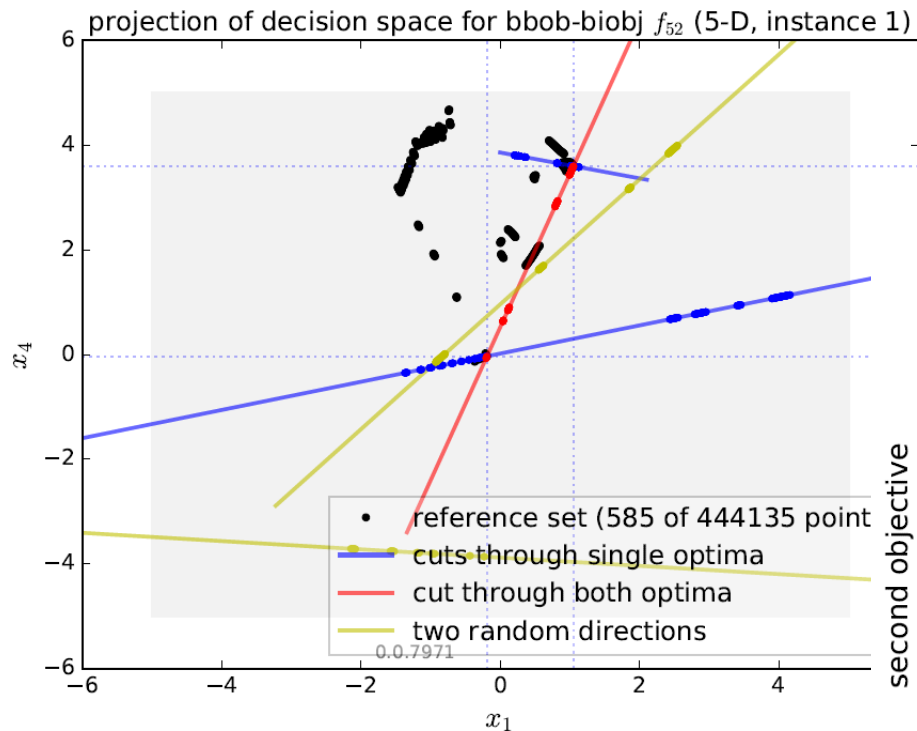
bbob-biobj Testbed (cont'd)

Example: sphere with Gallagher 101 peaks



bbob-biobj Testbed (cont'd)

Example: Schaffer F7, cond. 10 with Gallagher 101 peaks



Bi-objective Performance Assessment

algorithm quality =

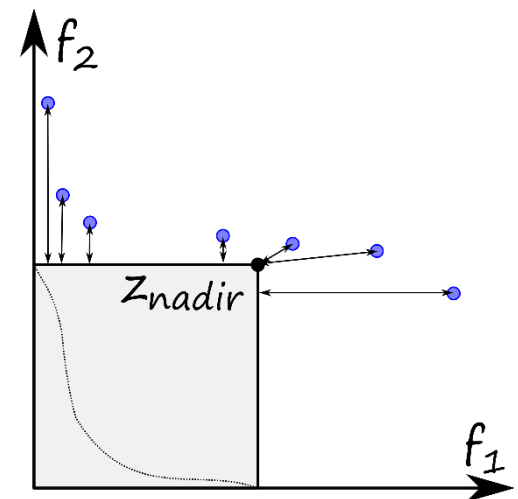
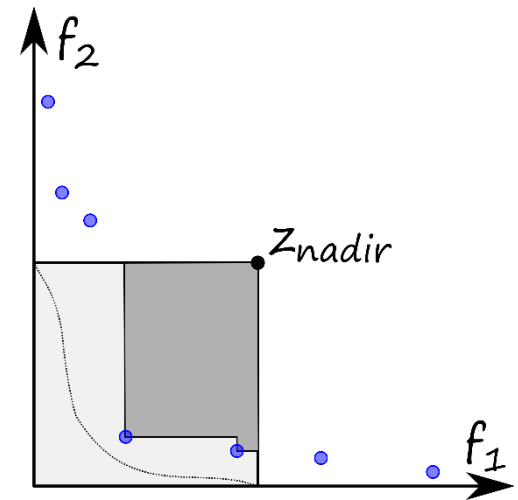
normalized* hypervolume (HV)
of all non-dominated solutions

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= $[1,1]$



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)
 - for the workshop: `before_workshop` values
 - from now on: updated `current_best` values incl. all non-dominated points found by the 15 workshop algos:
will be available soon and hopefully fixed for some time
- actual **absolute hypervolume targets** used are

$HV(\text{refset}) - \text{targetprecision}$

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

and now?

BBOB-2016

Enjoy the talks in this and the next two slots:

Session I

08:30 - 09:30	The BBOBies: Introduction to Blackbox Optimization Benchmarking
09:30 - 09:55	Tea Tušar*, Bogdan Filipič: Performance of the DEMO algorithm on the bi-objective BBOB test suite
09:55 - 10:20	Ilya Loshchilov, Tobias Glasmachers*: Anytime Bi-Objective Optimization with a Hybrid Multi-Objective CMA-ES (HMO-CMA-ES)

Session II

10:40 - 10:55	The BBOBies: Session Introduction
10:55 - 11:20	Cheryl Wong*, Abdullah Al-Dujaili, and Suresh Sundaram: Hypervolume-based DIRECT for Multi-Objective Optimisation
11:20 - 11:45	Abdullah Al-Dujaili* and Suresh Sundaram: A MATLAB Toolbox for Surrogate-Assisted Multi-Objective Optimization: A Preliminary Study
11:45 - 12:10	Oswin Krause*, Tobias Glasmachers, Nikolaus Hansen, and Christian Igel: Unbounded Population MO-CMA-ES for the Bi-Objective BBOB Test Suite
12:10 - 12:30	The BBOBies: Session Wrap-up

Session III

14:00 - 14:15	The BBOBies: Session Introduction
14:15 - 14:40	Kouhei Nishida* and Youhei Akimoto: Evaluating the Population Size Adaptation Mechanism for CMA-ES
14:40 - 15:05	The BBOBies: Wrap-up of all BBOB-2016 Results
15:05 - 15:30	Thomas Weise*: optimizationBenchmarking.org : An Introduction
15:30 - 15:50	Open Discussion

http://coco.gforge.inria.fr/

The screenshot shows a web browser window displaying the COCO website. The browser's address bar shows the URL `coco.gforge.inria.fr`. The page title is `COMPARING CONTINUOUS OPTIMISERS: COCO`. The main content area features a line graph on the right and a text description on the left. The graph plots the 'Proportion of functions' (y-axis, 0.0 to 1.0) against 'Running length / dimension' (x-axis, logarithmic scale from 10^0 to 10^8). Numerous optimization algorithms are represented by different colored lines, showing their convergence rates. A legend on the right side of the graph lists the algorithms, including BIPOP-CMA-ES, AMALGAM IDEA, IAmALGAM IDEA, MA-LS-Chain, VNS (Garcia), IPOPOP-SQP-CMA-ES, ALPS-GA, POEMS, Cauchy/EDA, EDA-PSO, (1+1)-CMA-ES, DASA, NELDER (Han), PSO Bounds, NELDER (Doe), PSO, (1+1)-ES, Full NEWUOA, GLOBAL, BFGS, Rosenbrock, MCS, simple GA, Siminband, L-Step, DIRECT, DE-PSO, BayEAcG, and Monte Carlo. A navigation menu on the right side of the page lists various categories such as Home, Documentation, download latest old code, new code homepage, download new code directly, BBOB 2016, BBOB 2015 @ GECCO, BBOB 2015 @ CEC, BBOB 2013, BBOB 2012, and BBOB 2010, each with sub-links for Algorithms, Results, Schedule, and Downloads.

[[start]] COMPARING CONTINUOUS OPTIMISERS: COCO

Show pagesource Old revisions Recent changes Sitemap Login

COCO (Comparing Continuous Optimisers) is a platform for systematic and sound comparisons of real-parameter global optimisers. COCO provides benchmark function testbeds, experimentation templates which are easy to parallelize, and tools for processing and visualizing data generated by one or several optimizers. The COCO platform has been used for the Black-Box-Optimization-Benchmarking (BBOB) workshops that took place during the GECCO conference in 2009, 2010, 2012, 2013 and 2015. It was also used at the IEEE Congress on Evolutionary Computation (CEC'2015) in Sendai, Japan. The COCO source code is available at the [downloads](#) page.

- Black-Box Optimization Benchmarking (BBOB) 2016
- Black-Box Optimization Benchmarking (BBOB) 2015
- CEC'2015 special session on Black-Box Optimization Benchmarking (CEC-BBOB 2013)
- Black-Box Optimization Benchmarking (BBOB) 2013
- Black-Box Optimization Benchmarking (BBOB) 2012
- Black-Box Optimization Benchmarking (BBOB) 2010
- Black-Box Optimization Benchmarking (BBOB) 2009
- Downloads and documentations

To subscribe to (or unsubscribe from) the bbob discussion mailing list follow this link <http://lists.lri.fr/cgi-bin/mailman/listinfo/bbob-discuss>.

To receive announcements related to the BBOB workshops simply send an email to BBOB team

Navigation

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

by the way...

we are hiring!

at the moment:

**1 engineer position for 1 year in Paris
+ potential PhD, postdoc, and internship positions**

if you are interested, please talk to:

Anne Auger or Dimo Brockhoff