

# Advanced Optimization

## Lecture 8: Benchmarking and Branch&Bound

December 1, 2021

CentraleSupélec / ESSEC Business School

[dimo.brockhoff@inria.fr](mailto:dimo.brockhoff@inria.fr)



Dimo Brockhoff  
Inria Saclay – Ile-de-France



INSTITUT  
POLYTECHNIQUE  
DE PARIS

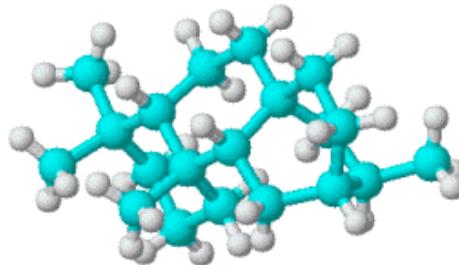


# Course Overview

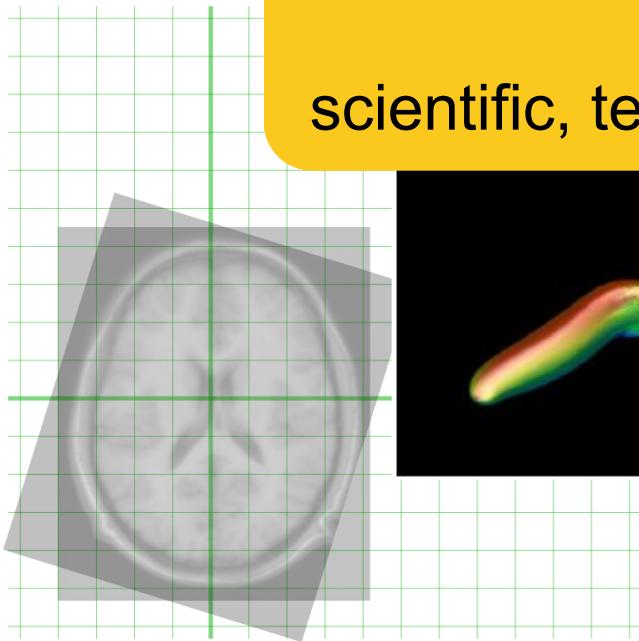
		Topic
Wed, 13.10.2021	PM	Introduction, examples of problems, problem types
Wed, 20.10.2021	PM	Continuous (unconstrained) optimization: convexity, gradients, Hessian, ... [technical test Evalmee]
Wed, 27.10.2021	PM	Continuous optimization II: [1 <sup>st</sup> mini-exam] Constrained optimization: Lagrangian, optimality conditions
Wed, 03.11.2021	PM	gradient descent, Newton direction, quasi-Newton (BFGS) Linear programming: duality, maxflow/mincut, simplex algo
Wed, 10.11.2021	PM	derivative-free algorithms: Nelder-Mead and CMA-ES
Wed, 17.11.2021	PM	CMA-ES, Part II, Stochastic Gradient Descent, Bayesian optimization
Wed, 24.11.2021	PM	Benchmarking solvers: runtime distributions, the COCO platform, recommendations [2 <sup>nd</sup> mini-exam]
Wed, 01.12.2021	PM	Benchmarking (cont.), discrete optimization: branch and bound
<b>Fri, 3.12.2021</b>	23:59	<b>Deadline open source project (PDF sent by email)</b>
Wed, 15.12.2021	PM	<b>Exam</b>

# Benchmarking Optimization Algorithms

or: critical performance assessment

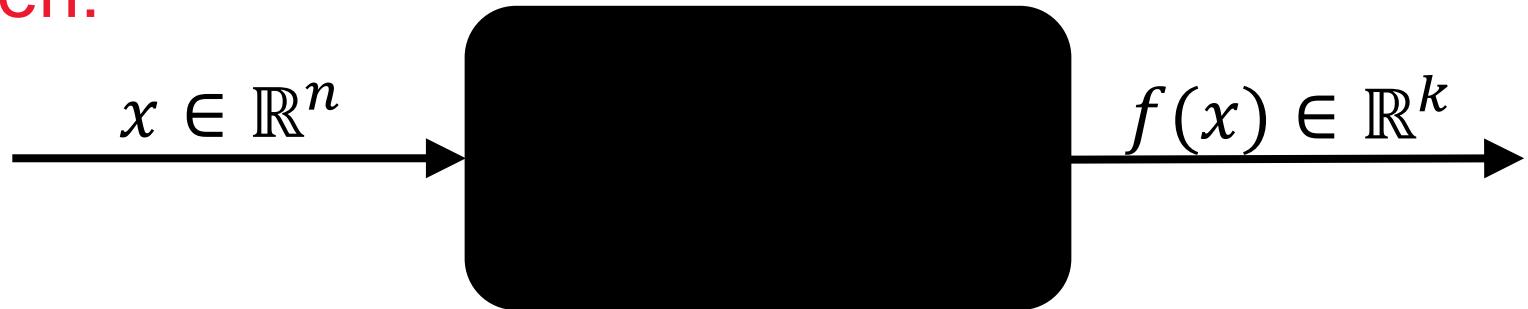


challenging optimization problems  
appear in many  
scientific, technological and industrial domains



# Practical (Numerical) Blackbox Optimization

Given:



*derivatives not available or not useful*

Not clear:

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

# 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

wouldn't  
**automatized benchmarking**  
be cool?

**for this, we developed COCO**

**Comparing Continuous Optimizers Platform**  
<https://github.com/numbbo/coco>

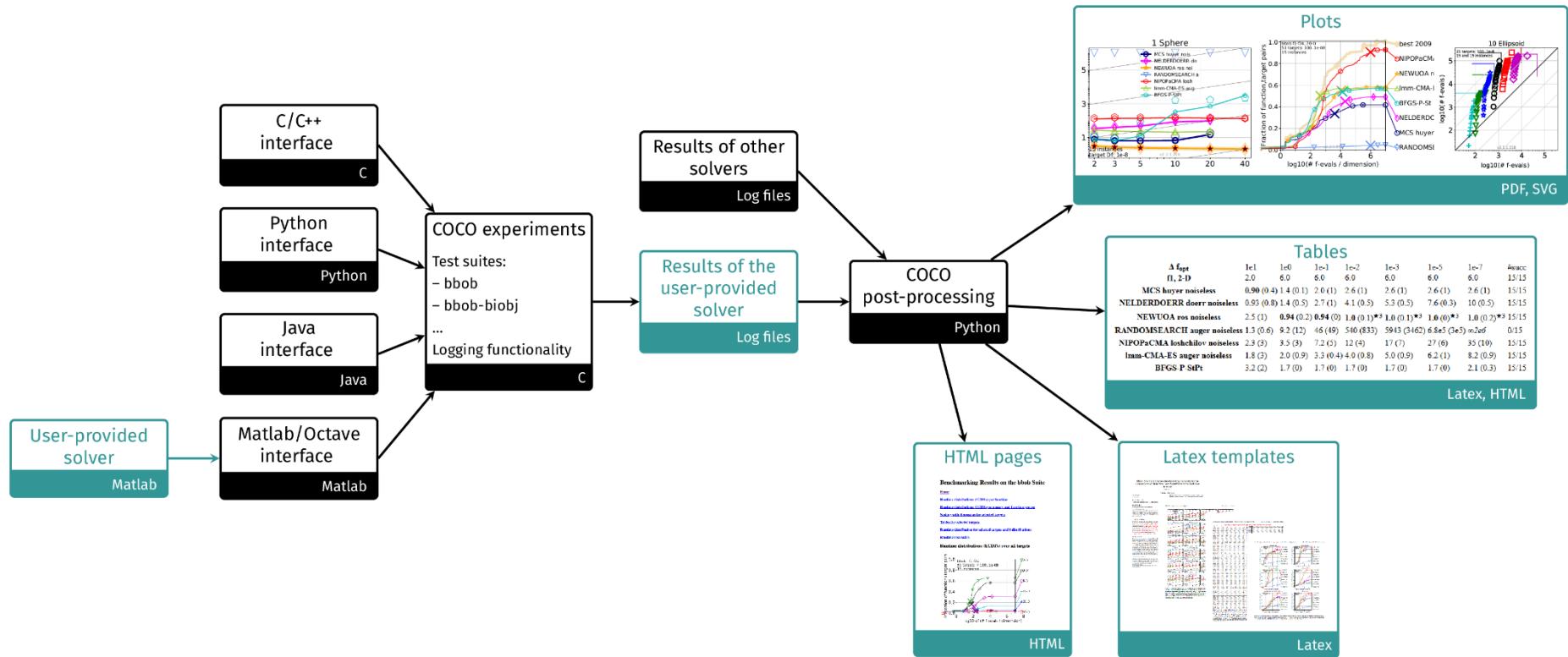


**benchmarking is non-trivial**

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

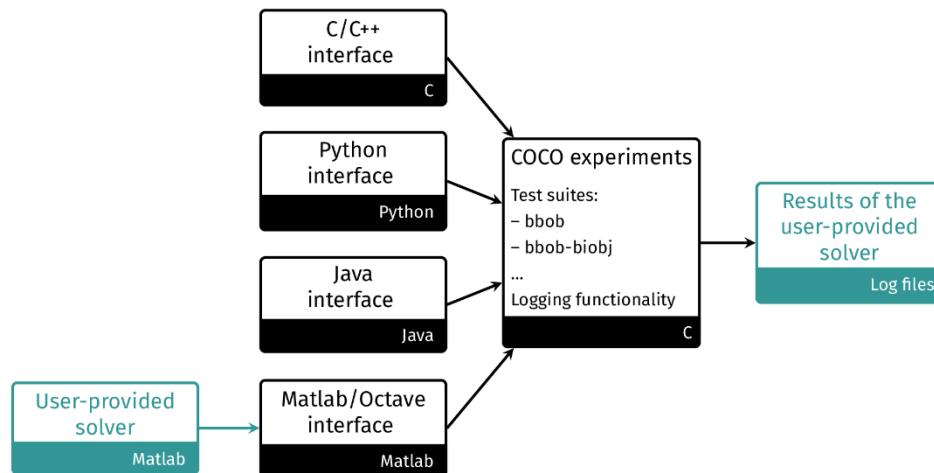
# The Comparing Continuous Optimizers Platform

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



# The Comparing Continuous Optimizers Platform

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



# Experiments are as simple as (after installation)...

```
import cocoex
import scipy.optimize

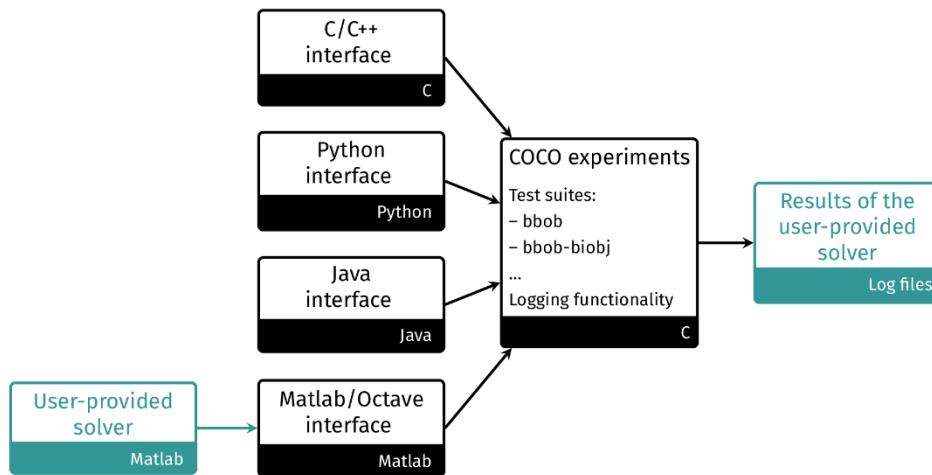
### input
suite_name = "bbob" # or "bbob-biobj" or ...
output_folder = "scipy-optimize-fmin"

### prepare
suite = cocoex.Suite(suite_name, "", "")
observer = cocoex.Observer(suite_name,
                           "result_folder: " + output_folder)

### go
for problem in suite: # this loop will take several minutes
    problem.observe_with(observer) # generates the data for
                                    # cocopp post-processing
    scipy.optimize.fmin(problem, problem.initial_solution)
```

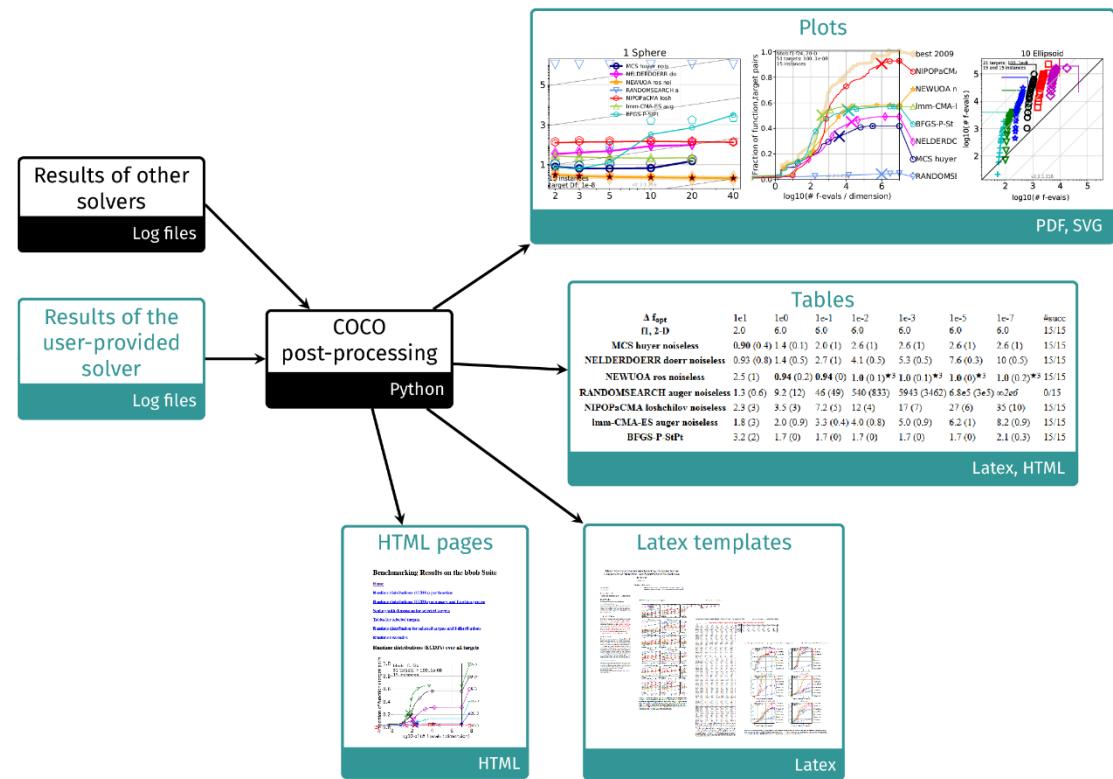
# The Comparing Continuous Optimizers Platform

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



# The Comparing Continuous Optimizers Platform

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



# **How to benchmark algorithms with COCO?**

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

The screenshot shows the GitHub repository page for `numbbo/coco`. The page includes the repository name, a search bar, and navigation links for Pull requests, Issues, Marketplace, and Gist. Key statistics are displayed: 16,007 commits, 11 branches, 31 releases, and 15 contributors. A list of recent commits is shown, and a modal window for cloning the repository via HTTPS is open.

**numbbo / coco**

Pull requests Issues Marketplace Gist

Unwatch 15 Unstar 38 Fork 24

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

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 ...

code-experiments A little more verbose error message when suite regression test fail

code-postprocessing Hashes are back on the plots.

code-preprocessing Fixed preprocessing to work correctly with the extended biobjective

howtos Update create-a-suite-howto.md

.clang-format raising an error in bbob2009\_logger.c when best\_value is NULL. Plus s...

.hgignore raising an error in bbob2009\_logger.c when best\_value is NULL. Plus s...

AUTHORS small correction in AUTHORS

LICENCE Update LICENCE

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 4 months ago

2 years ago 2 years ago

a year ago 11 months ago

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

The screenshot shows the GitHub repository page for `numbbo/coco`. The page includes the repository name, a summary bar with commit, branch, release, and contributor counts, a list of recent commits, and a 'Clone or download' dropdown menu.

**Repository Summary:**

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

**Recent Commits:**

File	Message	Time
<code>code-experiments</code>	A little more verbose error message when suite regression test fail	4 months ago
<code>code-postprocessing</code>	Hashes are back on the plots.	2 years ago
<code>code-preprocessing</code>	Fixed preprocessing to work correctly with the extended biobjective	2 years ago
<code>howtos</code>	Update <code>create-a-suite-howto.md</code>	4 months ago
<code>.clang-format</code>	raising an error in <code>bbob2009_logger.c</code> when <code>best_value</code> is <code>NULL</code> . Plus s...	2 years ago
<code>.hgignore</code>	raising an error in <code>bbob2009_logger.c</code> when <code>best_value</code> is <code>NULL</code> . Plus s...	2 years ago
<code>AUTHORS</code>	small correction in <code>AUTHORS</code>	a year ago
<code>LICENSE</code>	Update <code>LICENSE</code>	11 months ago
<code>README.md</code>	Added link to #1335 before closing.	a month ago

**Clone with HTTPS** <https://github.com/numbbo/coco.git>

[Open in Desktop](#) [Download ZIP](#)

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

numbbo/coco: Numerical ... [+](#)

[GitHub, Inc. \(US\)](#) | <https://github.com/numbbo/coco> | [Search](#)

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

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** ...

[code-experiments](#) A little more verbose error message when suite regression test fail

[code-postprocessing](#) Hashes are back on the plots.

[code-preprocessing](#) Fixed preprocessing to work correctly with the extended biobjective

[howtos](#) Update create-a-suite-howto.md

[.clang-format](#) raising an error in bbob2009\_logger.c when best\_value is NULL. Plus s...

[.hgignore](#) raising an error in bbob2009\_logger.c when best\_value is NULL. Plus s...

[AUTHORS](#) small correction in AUTHORS

[LICENSE](#) Update LICENSE

[README.md](#) Added link to #1335 before closing.

[do.py](#) refactoring here and there in do.py to get closer to PEP8 specifications

[doxygen.ini](#) moved all files into code-experiments/ folder besides the do.py scri...

[README.md](#)

**Clone with HTTPS** [Use SSH](#)

Use Git or checkout with SVN using the web URL.

<https://github.com/numbbo/coco.git> [Copy](#)

[Open in Desktop](#) [Download ZIP](#)

4 months ago 2 years ago 2 years ago a year ago 11 months ago a month ago 2 months ago 2 years ago

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

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 ...

code-experiments A little more verbose error message when suite regression test fail

code-postprocessing Hashes are back on the plots.

code-preprocessing Fixed preprocessing to work correctly with the extended biobjective

howtos Update create-a-suite-howto.md

.clang-format raising an error in bbob2009\_logger.c when best\_value is NULL. Plus s...

.hgignore raising an error in bbob2009\_logger.c when best\_value is NULL. Plus s...

AUTHORS small correction in AUTHORS

LICENSE Update LICENSE

README.md Added link to #1335 before closing.

do.py refactoring here and there in do.py to get closer to PEP8 specifications

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

README.md

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

4 months ago 2 years ago 2 years ago a year ago 11 months ago a month ago 2 months ago 2 years ago

## numbbo/coco: Comparing Continuous Optimizers

numbbo/coco: Numerical ... [GitHub, Inc. \(US\)](#) | <https://github.com/numbbo/coco>

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

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

## 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`

The screenshot shows a GitHub repository page for 'numbbo/coco'. The top navigation bar includes links for 'GitHub, Inc. (US)', 'https://github.com/numbbo/coco', 'Search', and various Inria-related links like 'RandOpt', 'CMAP', 'Inria GitLab', and 'RER B from lab'. Below the navigation is a list of recent commits:

File	Commit Message	Date
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

The README.md file content is as follows:

## 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,

- read our [benchmarking guidelines introduction](#)
- read the [COCO experimental setup description](#)

numbbo/coco: Numerical ... + GitHub, Inc. (US) https://github.com/numbbo/coco Search Most Visited Getting Started COCO-Algorithms numbbo/numbbo · Gi... RandOpt CMAP Inria GitLab RER B from lab

## 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,

- 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.

The screenshot shows a web browser window with the URL <https://github.com/numbbo/coco>. The page title is "numbbo/coco: Numerical Optimization". The "Getting Started" section is highlighted with a red box. A red callout bubble on the right contains the text "requirements & download".

**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 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
```

The screenshot shows a web browser window with the URL <https://github.com/numbbo/coco>. The page title is "Getting Started". The content includes:

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 commands:

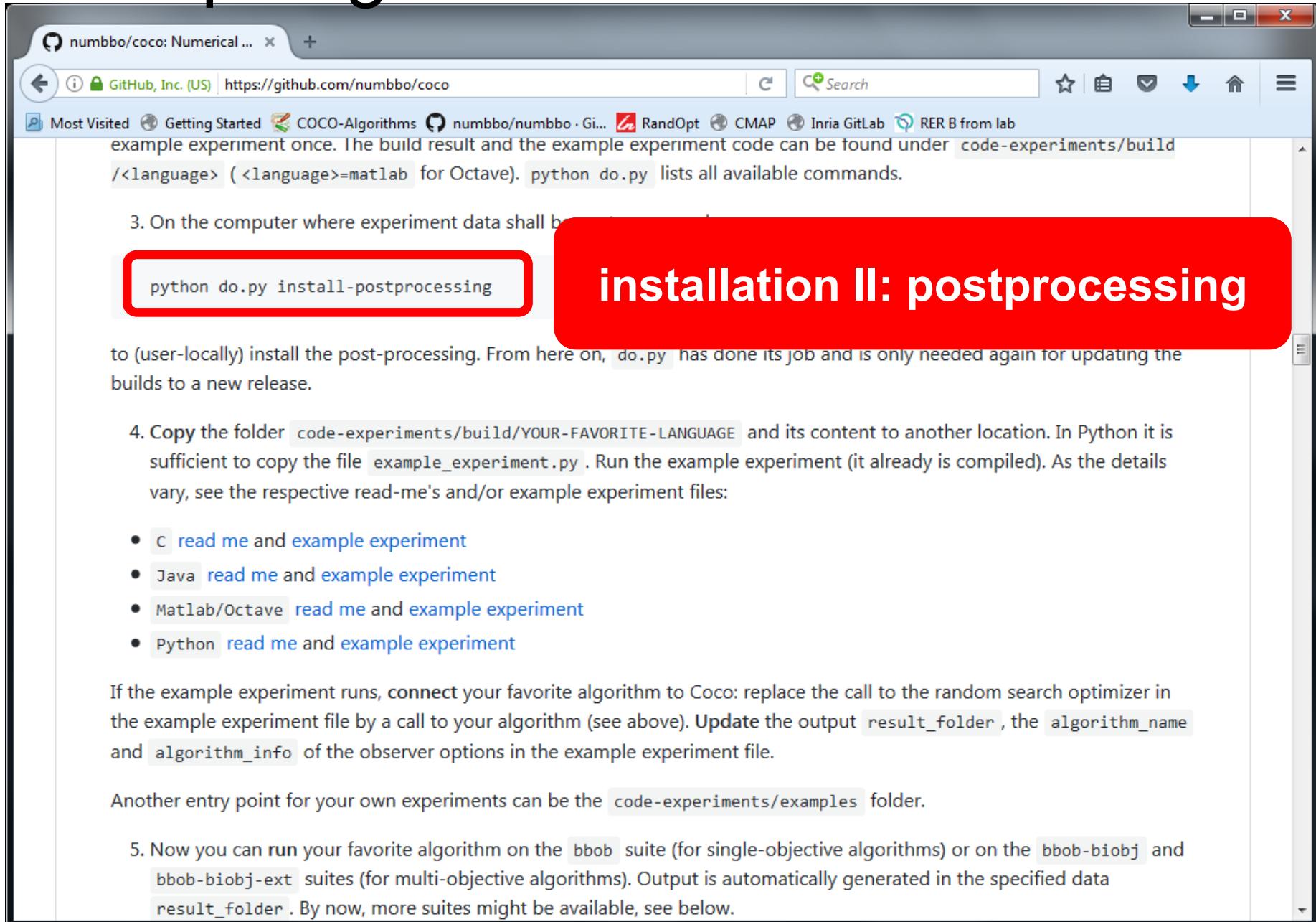
```
python do.py run-c  
python do.py run-java  
python do.py run-matlab  
python do.py run-octave  
python do.py run-python
```

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
```

**installation I: experiments**



The screenshot shows a web browser window with the URL <https://github.com/numbbo/coco>. The page content discusses the Coco optimization framework. A red box highlights the command `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.

numbbo/coco: Numerical ... + GitHub, Inc. (US) https://github.com/numbbo/coco Search Most Visited Getting Started COCO-Algorithms numbbo/numbbo · Gi... RandOpt CMAP Inria GitLab RER B from lab 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](#)

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.

**coupling algo + COCO**

# Simplified Example Experiment in Python

```
import cocoex
import scipy.optimize

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

### prepare
suite = cocoex.Suite(suite_name, "", "")
observer = cocoex.Observer(suite_name,
                           "result_folder: " + output_folder)

### go
for problem in suite: # this loop will take several minutes
    problem.observe_with(observer) # generates the data for
                                    # cocopp post-processing
    fmin(problem, problem.initial_solution)
```

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

numbbo/coco at develop... GitHub, Inc. (US) https://github.com/numbbo/coco/tree/development Search Most Visited Getting Started COCO-Algorithms numbbo/numbbo · Gi... RandOpt CMAP Inria GitLab RER B from lab 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] YOURDATA
```

Any subfolder in the folder arguments will be searched for 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, p  
file, useful  
the output

A summary  
template  
template

7. Once

independ

running the experiment

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

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

numbbo/coco at develop... GitHub, Inc. (US) https://github.com/numbbo/coco/tree/development Search Most Visited Getting Started COCO-Algorithms numbbo/numbbo · Gi... RandOpt CMAP Inria GitLab RER B from lab

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" `YOURDATAFOLDER`, specifying several data result folders generated by different alg...

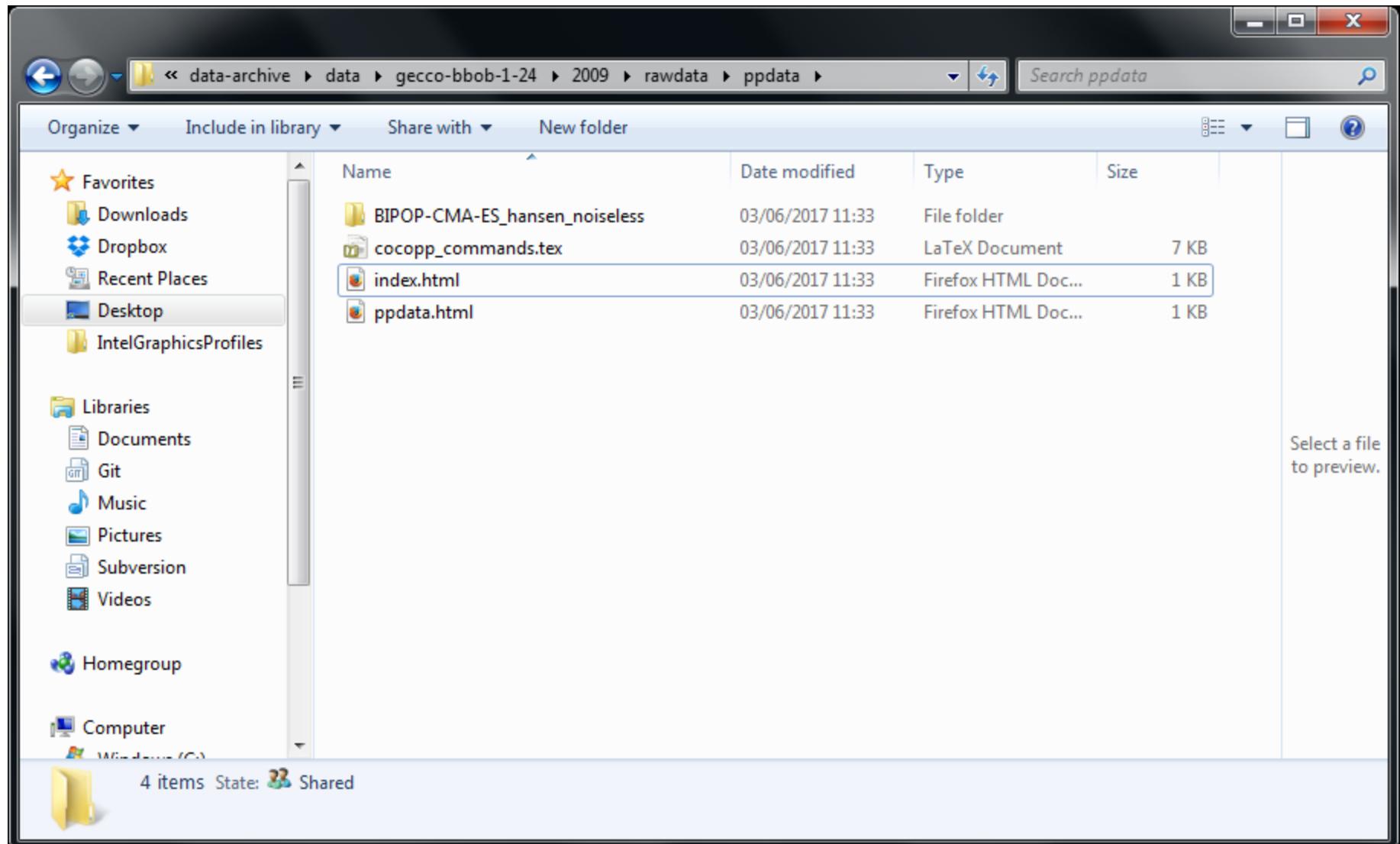
A folder, `ppdata` by default, will be generated, which contains a file, useful as main entry point to explore the result with a browser. The output folder name with the `-o OUTPUT_FOLDERNAME` option.

**postprocessing**

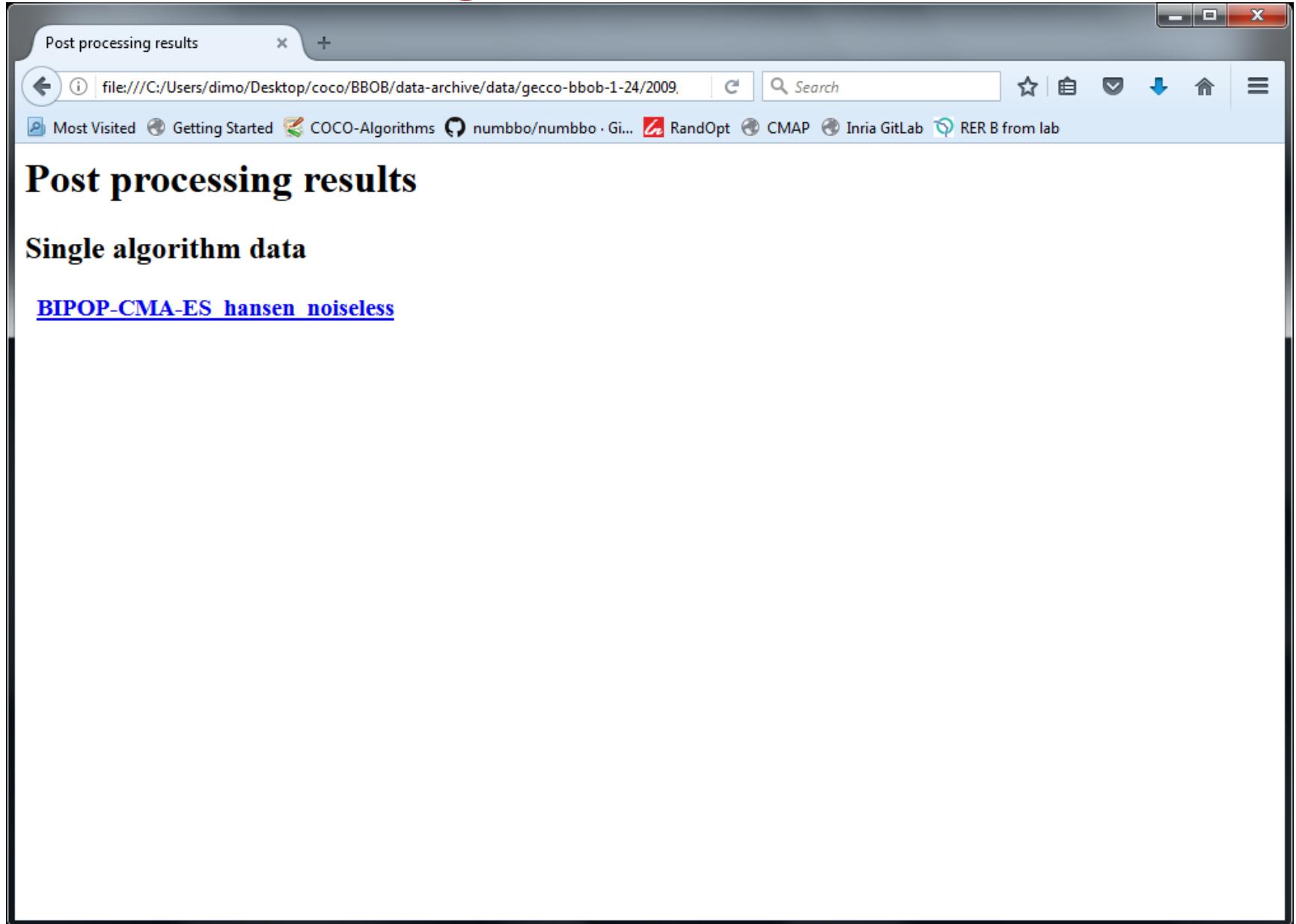
**data from 200+ algorithms can be accessed directly through its name  
(see <https://numbbo.github.io/data-archive/>)**

`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

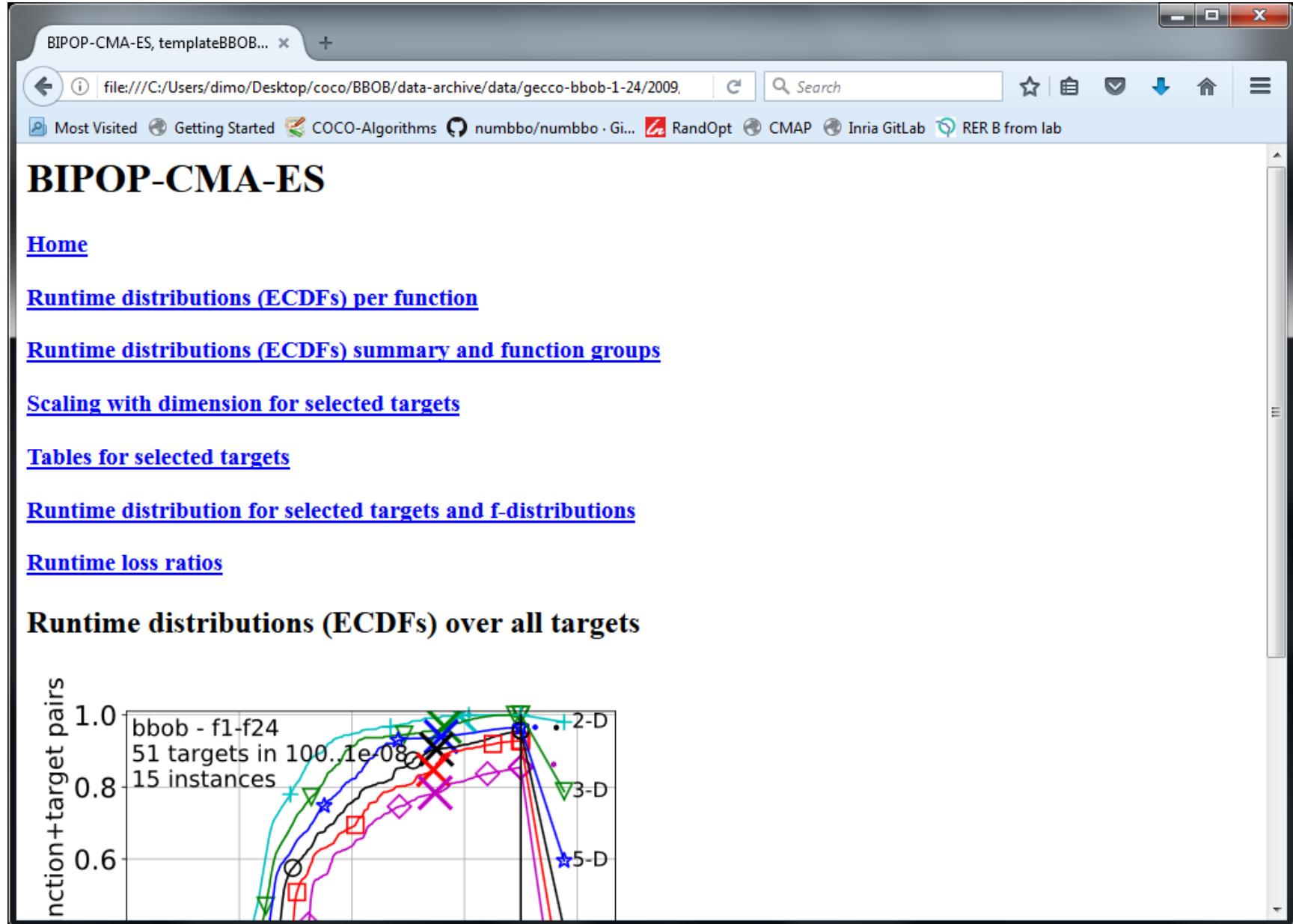
# Result Folder



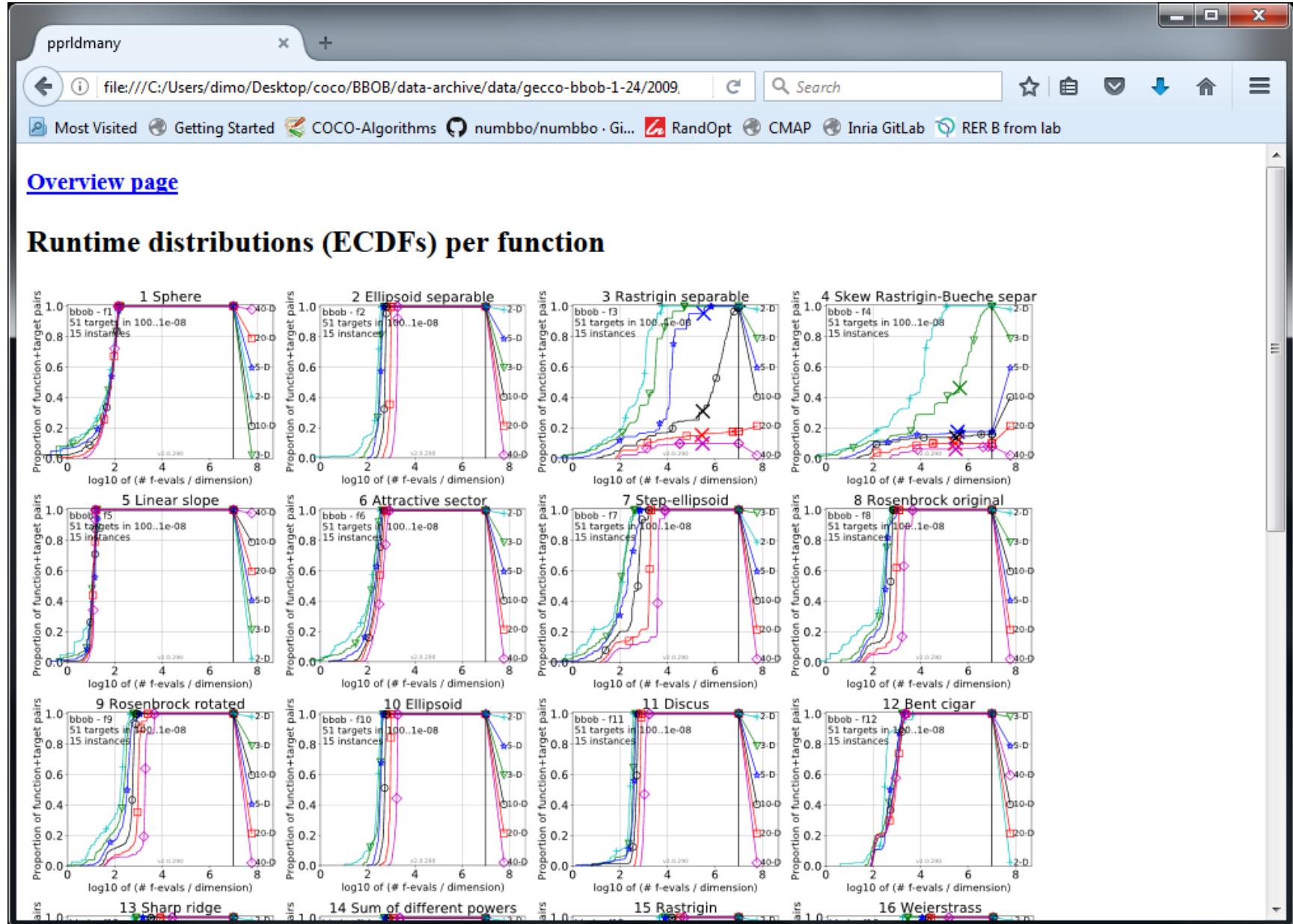
# Automatically Generated Results



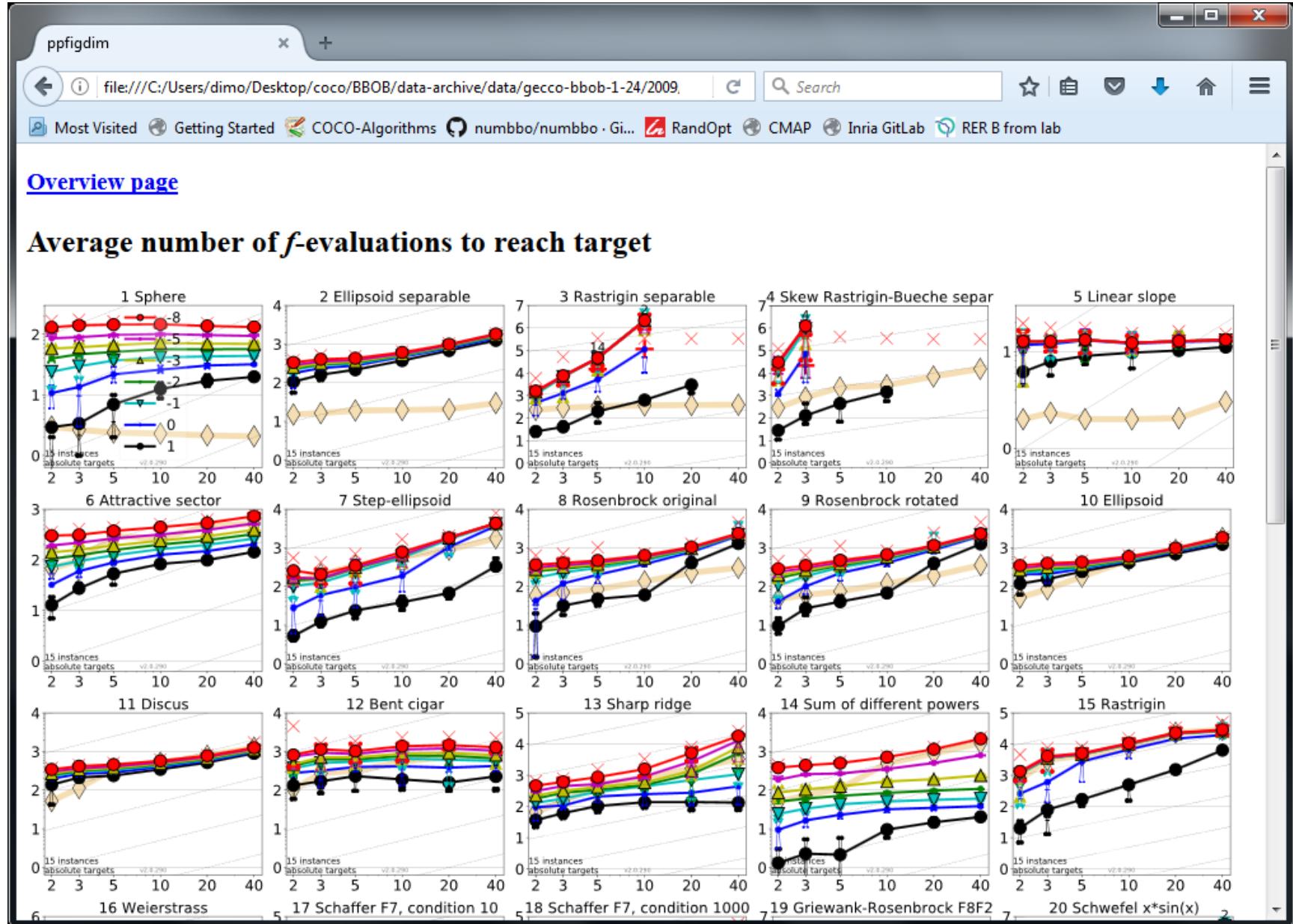
# Automatically Generated Results



# Automatically Generated Results



# Automatically Generated Results



**so far:**

data for 300+ algorithm variants  
(some of which on noisy/multiobjective/mixed-integer test functions)

# 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

- bbo**b** 24 noiseless fcts 220+ algo data sets
- bbo**b**-noisy 30 noisy fcts 40+ algo data sets
- bbo**b**-biobj 55 bi-objective fcts 30+ algo data sets
- bbo**b**-largescale 24 noiseless fcts 11 algo data sets
- bbo**b**-mixint 24 mixed integer fcts
- bbo**b**-biobj-mixint 92 mixed integer fcts

# 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?

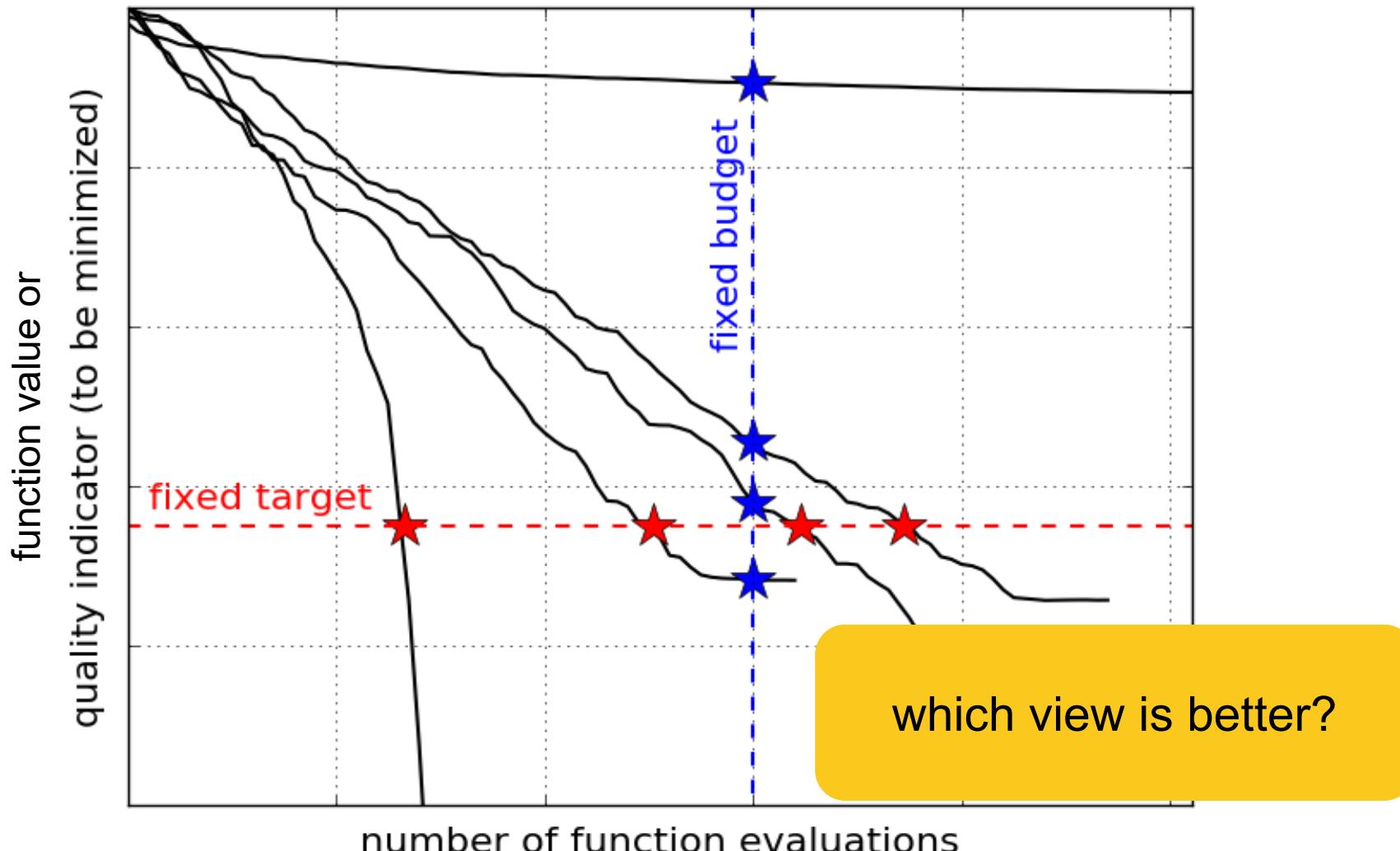
**Two objectives:**

- 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...



# Runtime is the prime candidate

## Why?

- runtimes are easier to interpret than f-values
- runtimes are interpretable in a meaningful way
- runtimes have a natural zero (hence: ratio scale)

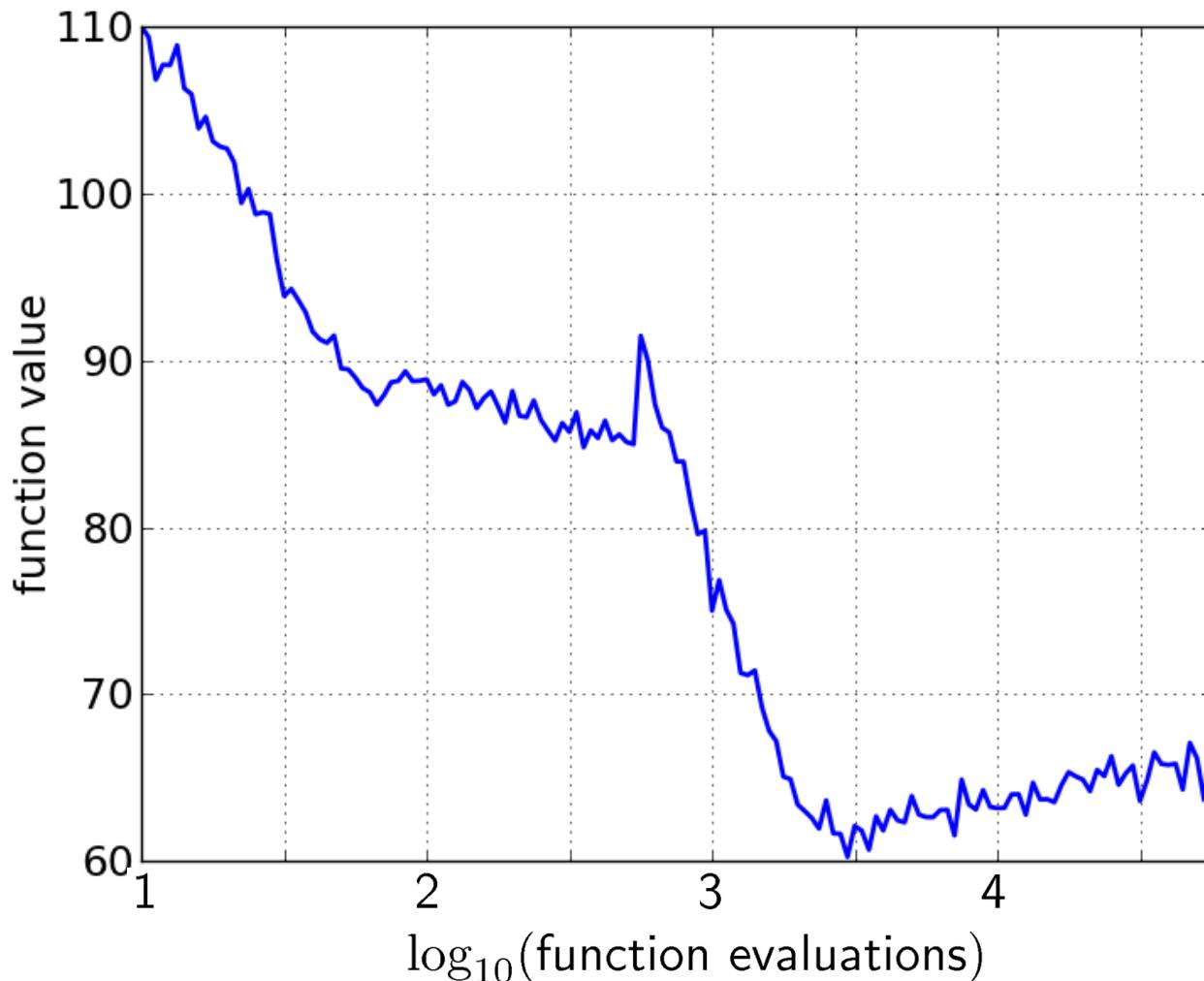
## Another reason:

- in both cases, we have missing values
- in the fixed budget view, we miss values of "too good" algorithms
- in the fixed target view, we miss values of "bad/slow" algorithms

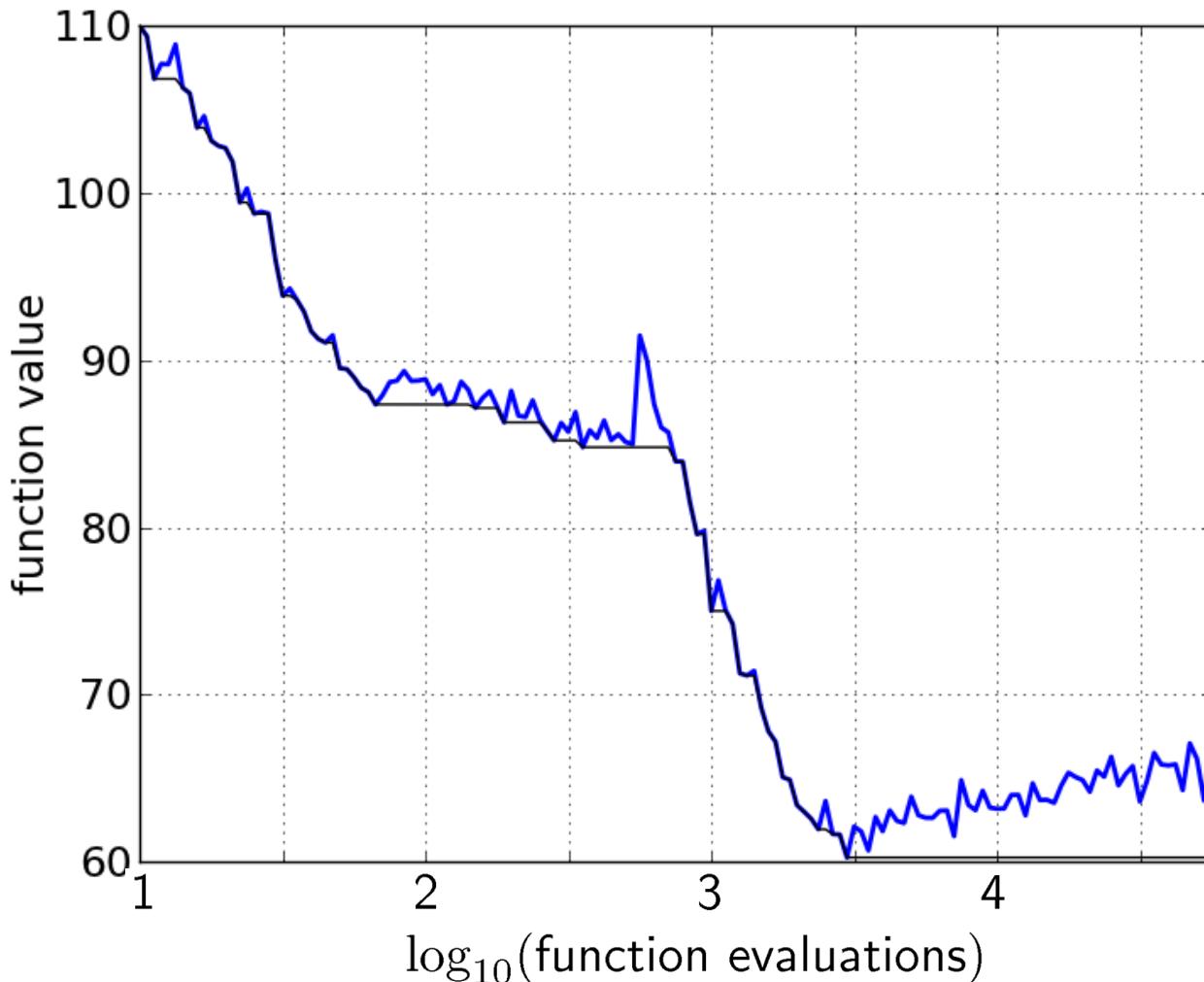
**ECDF:**

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

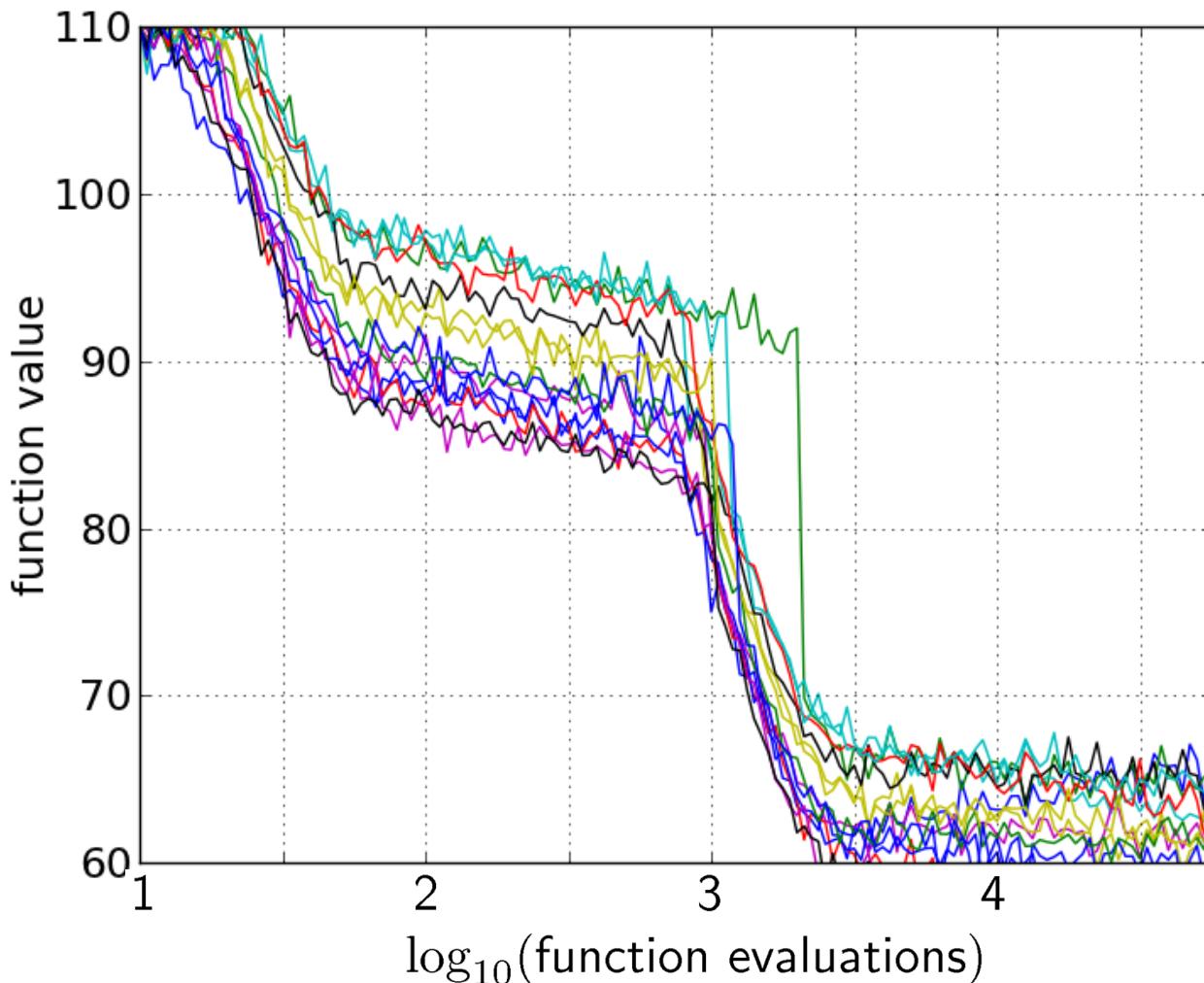
# A Convergence Graph



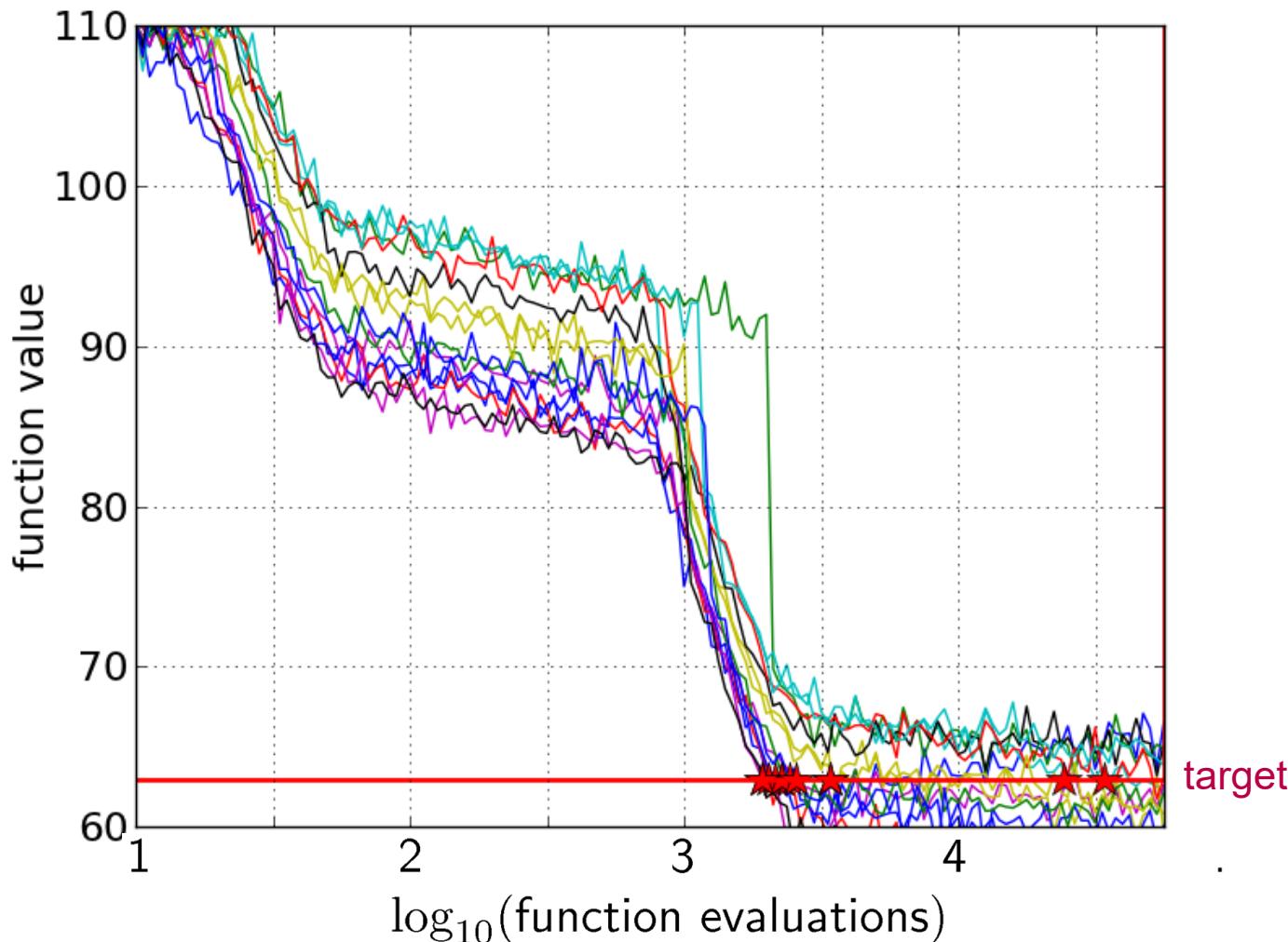
# First Hitting Time is Monotonous



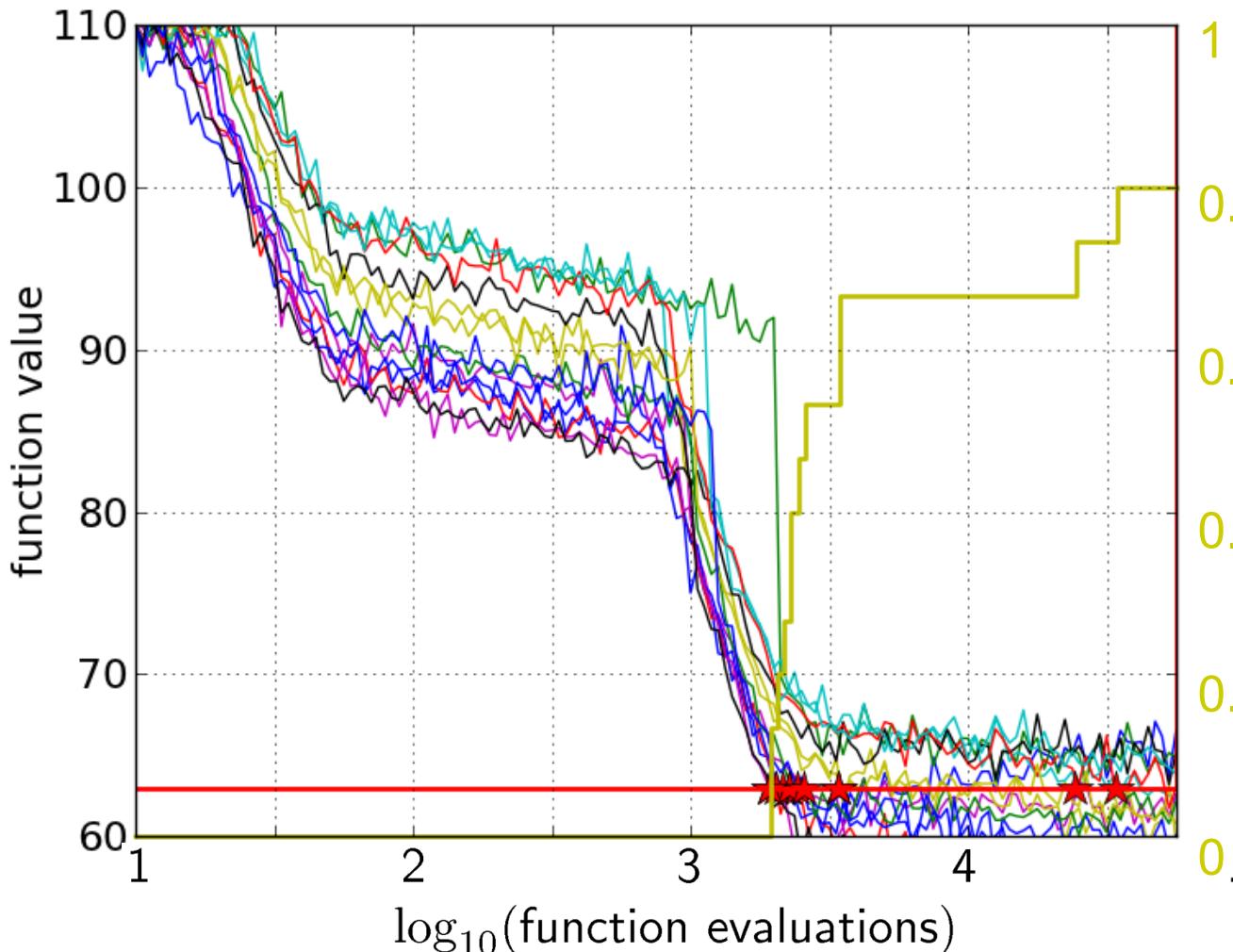
# 15 Runs



# 15 Runs $\leq$ 15 Runtime Data Points

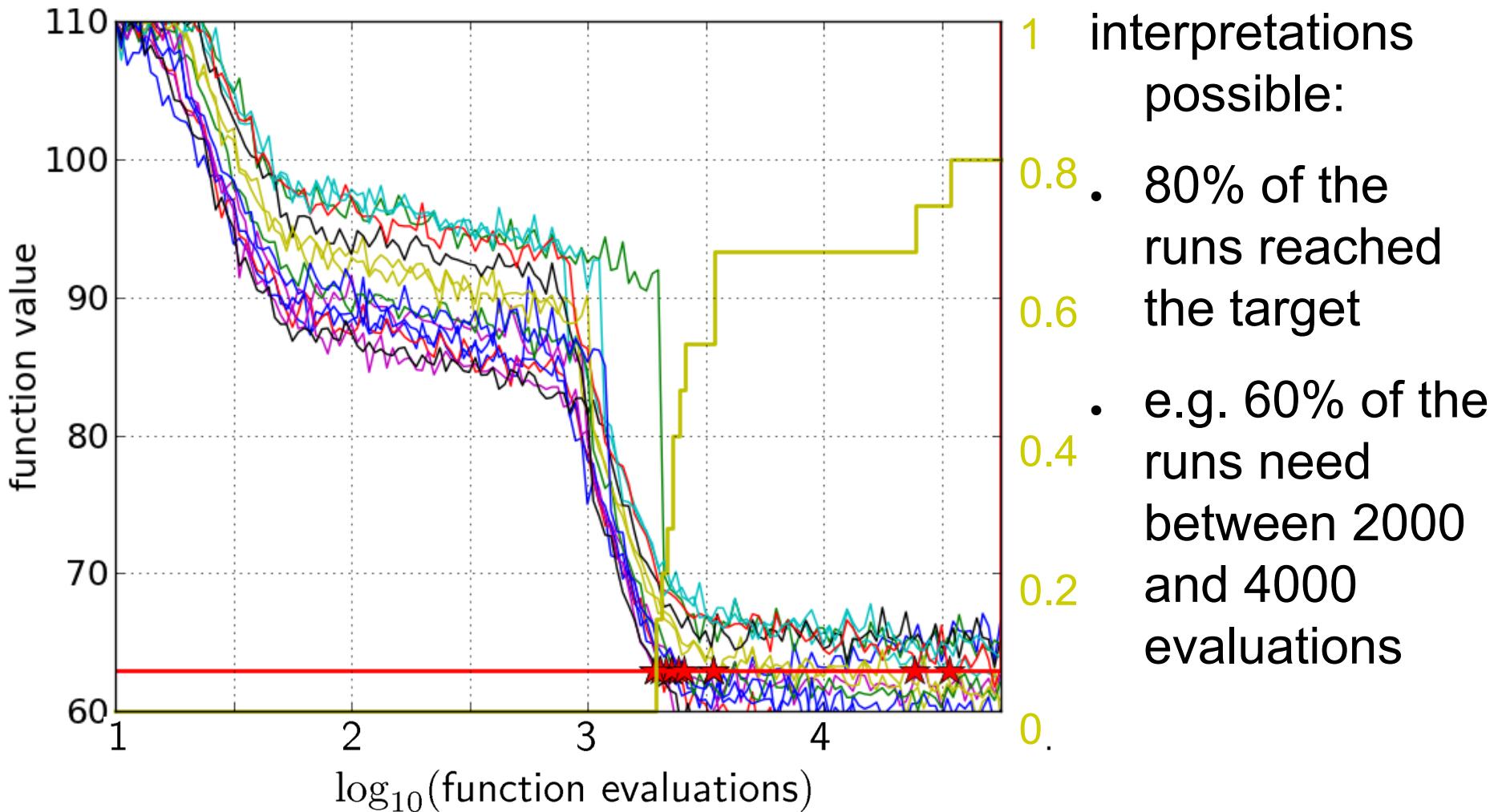


# Empirical Cumulative Distribution

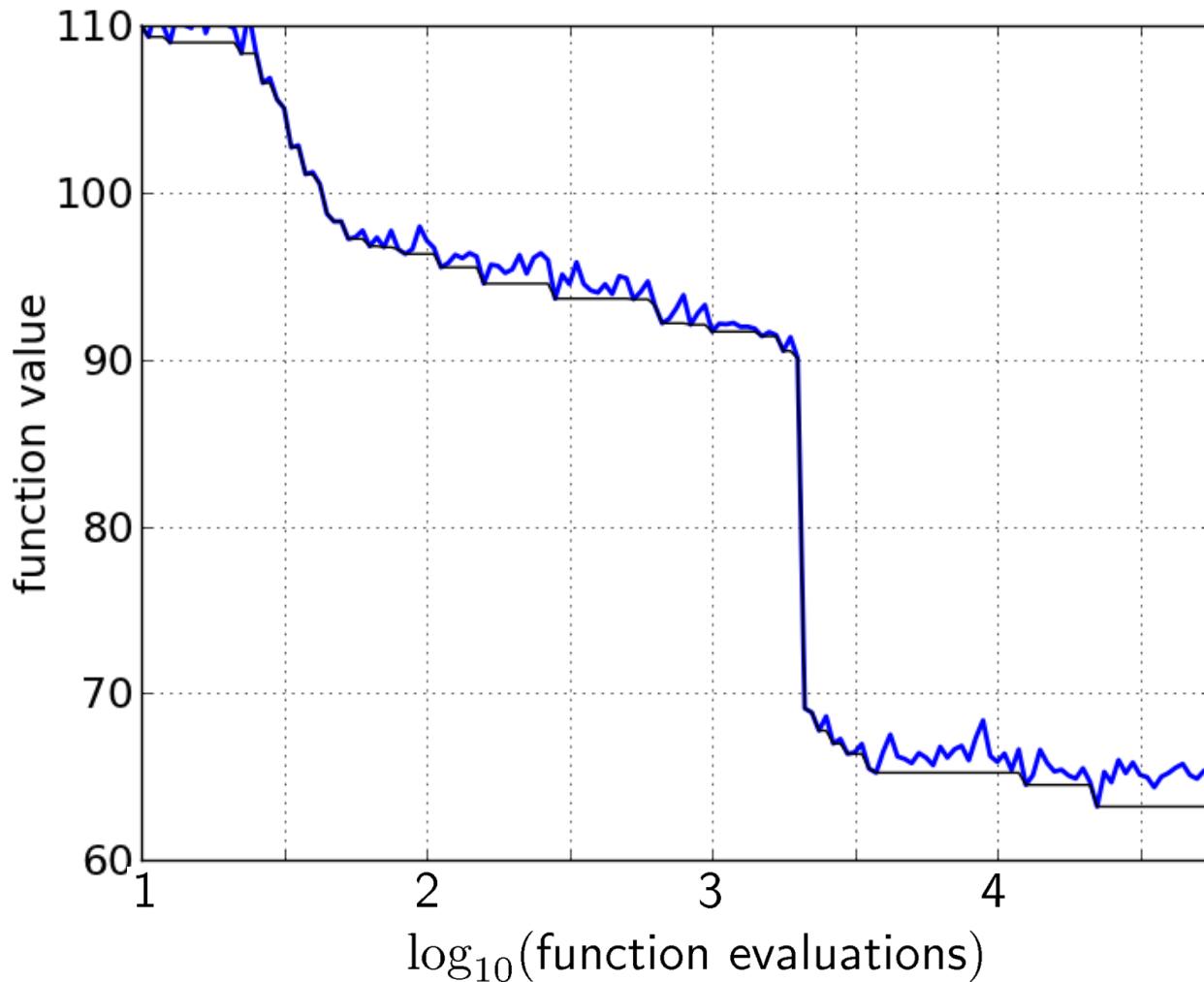


- 1 the ECDF of run lengths to reach the target
  - has for each data point a vertical step of constant size
  - displays for each x-value (budget) the count of observations to the left (first hitting times)

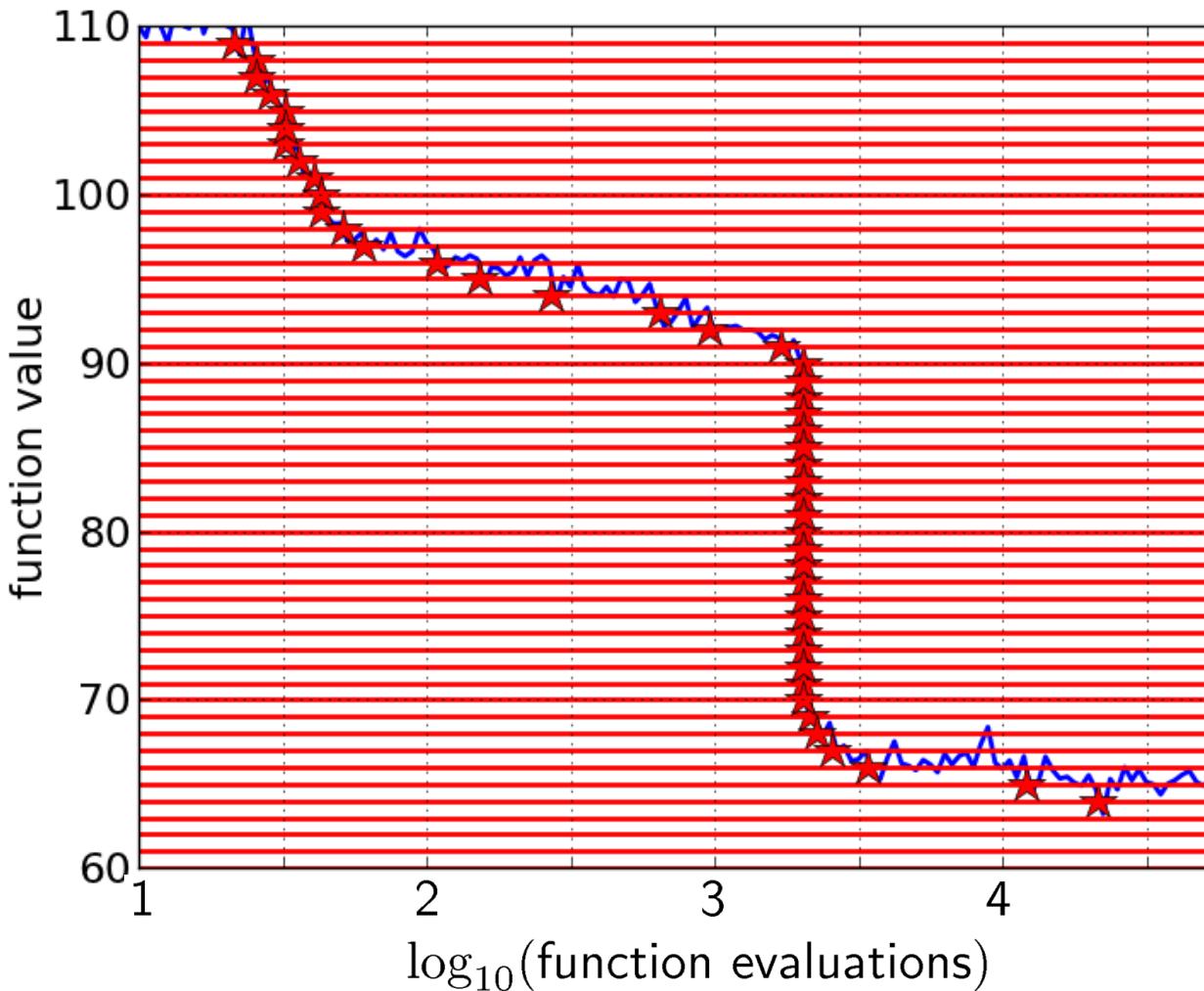
# Empirical Cumulative Distribution



# Reconstructing A Single Run

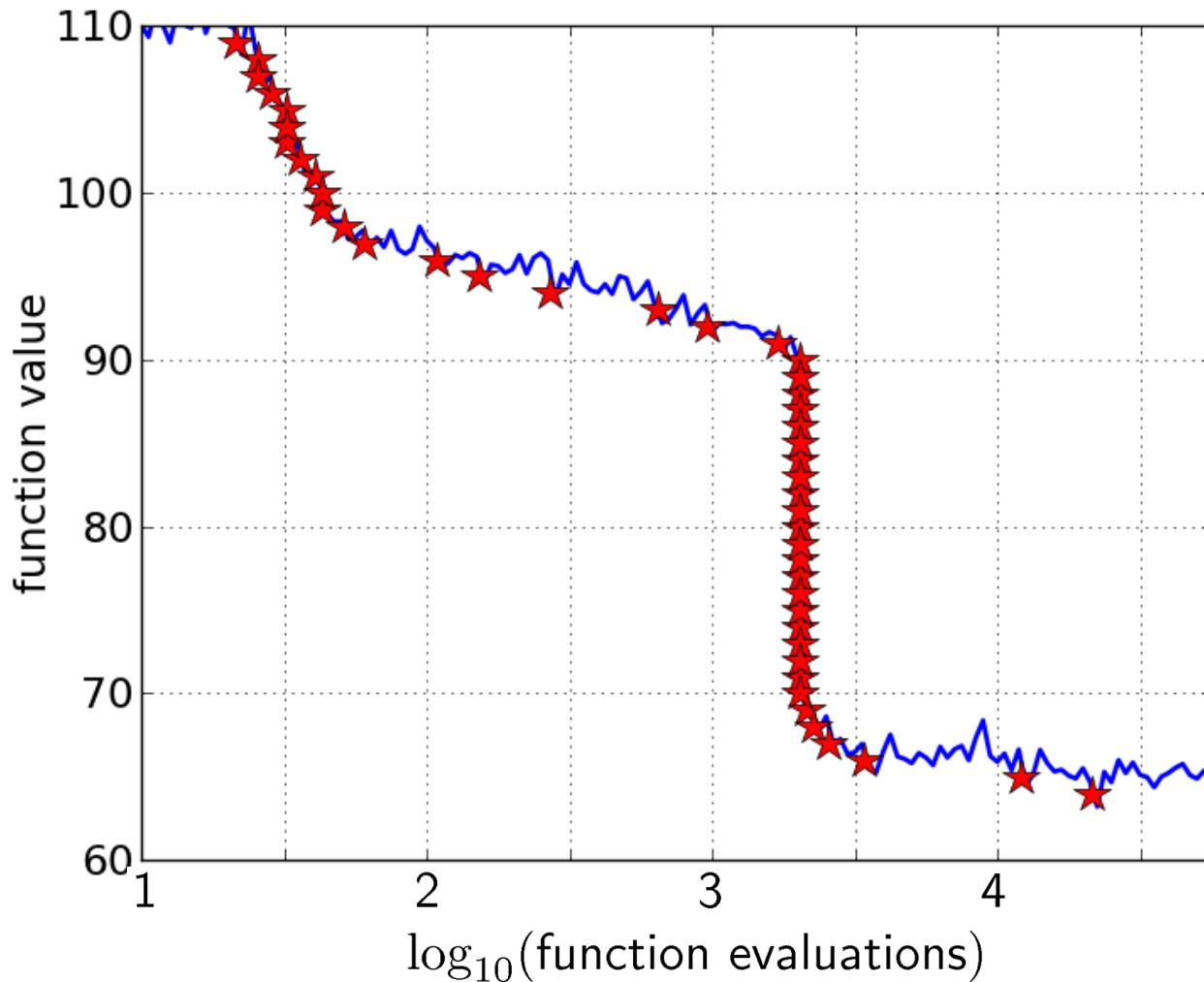


# Reconstructing A Single Run

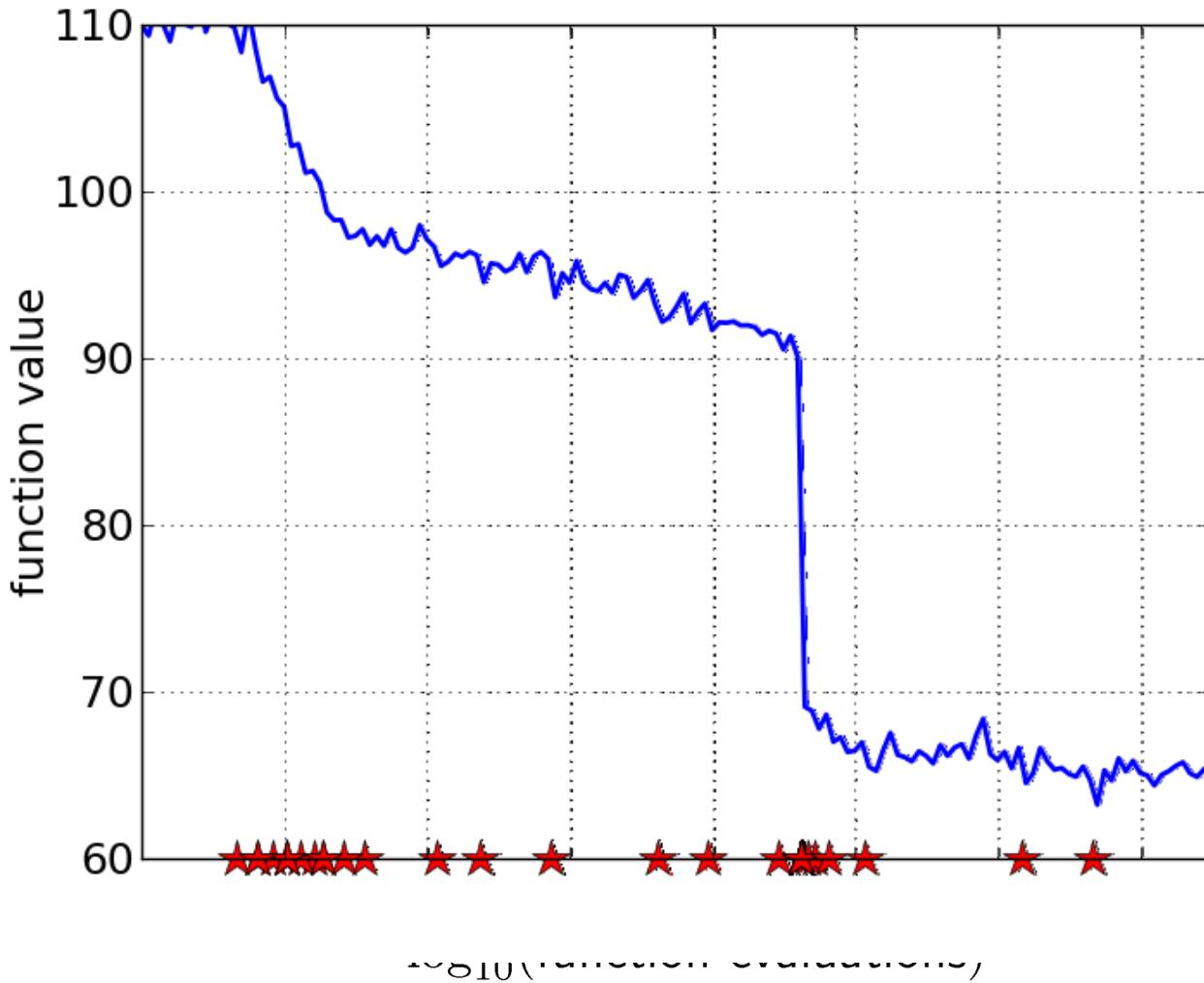


50 equally  
spaced targets

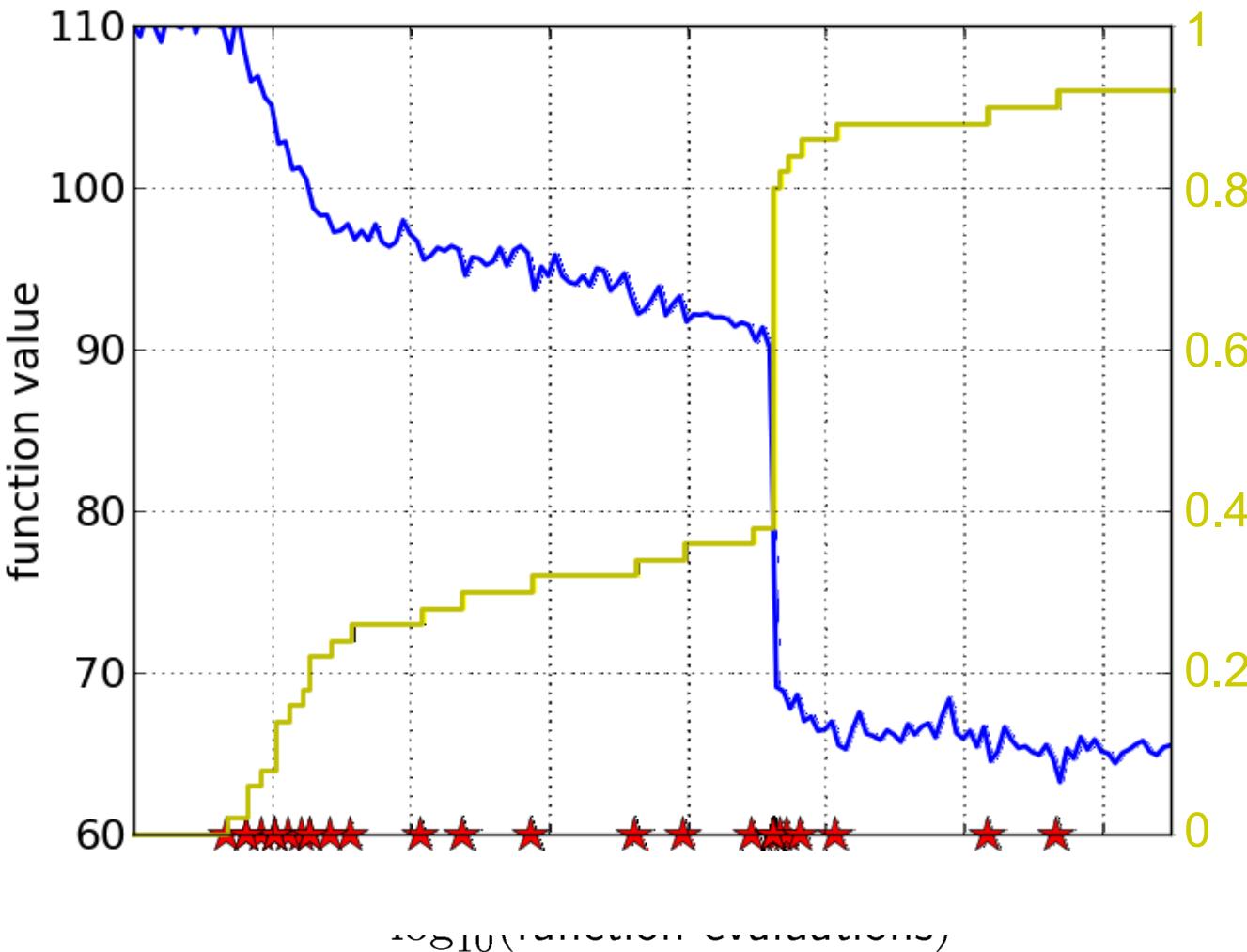
# Reconstructing A Single Run



# Reconstructing A Single Run

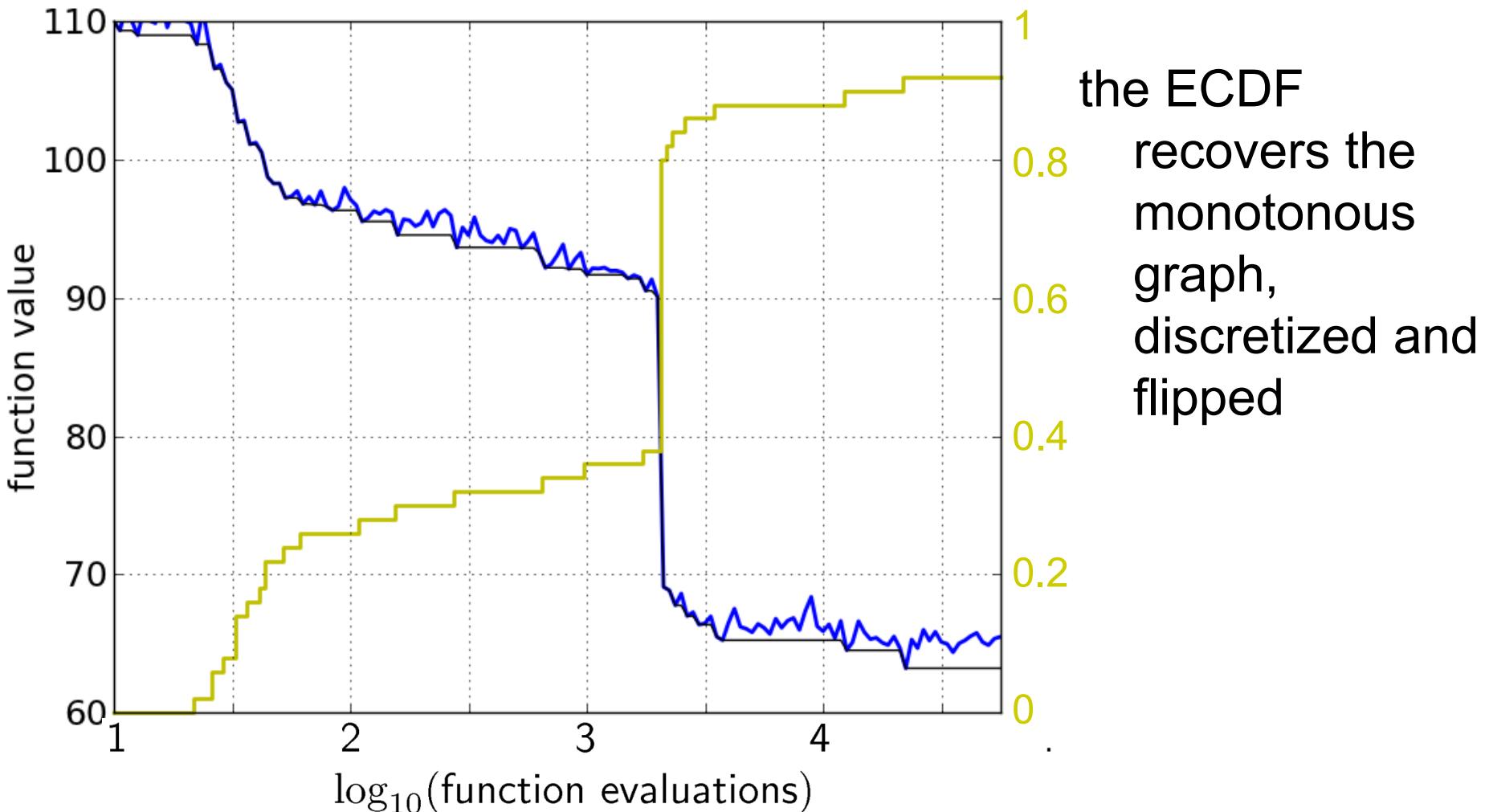


# Reconstructing A Single Run

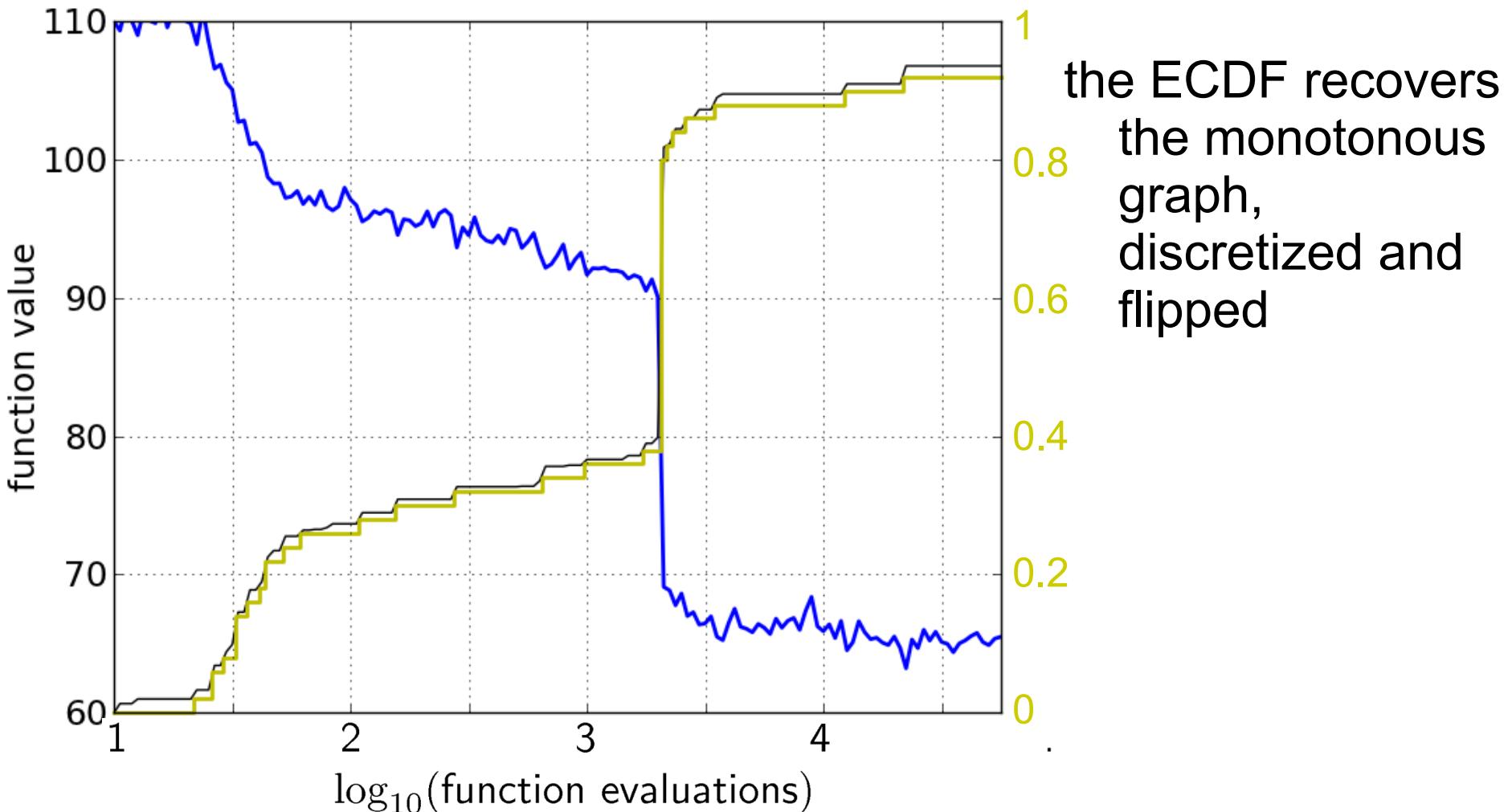


i.e. empirical  
CDF makes a  
step for each  
star, is  
monotonous  
and displays  
for each  
budget the  
fraction of  
targets  
achieved within  
the budget

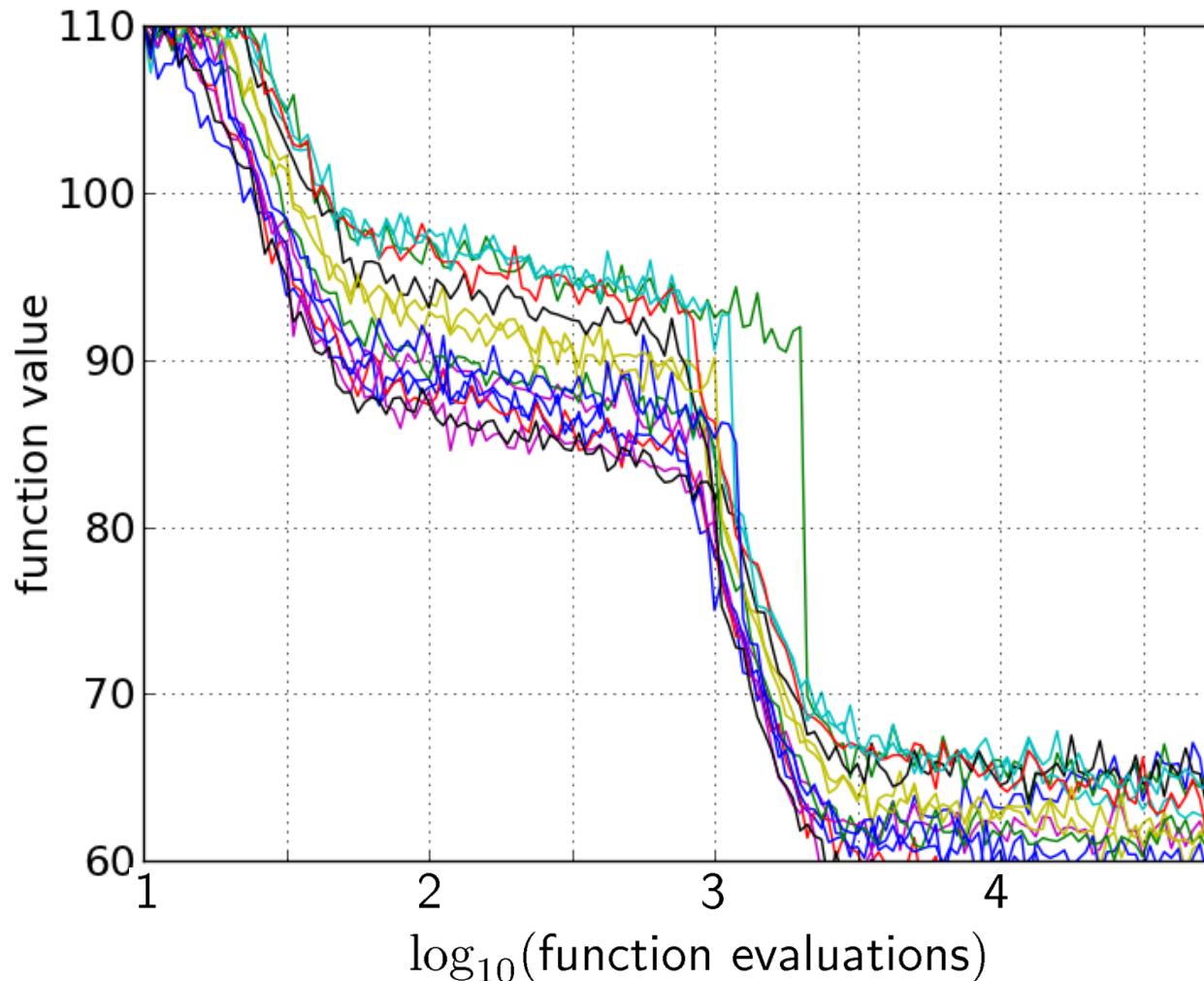
# Reconstructing A Single Run



# Reconstructing A Single Run

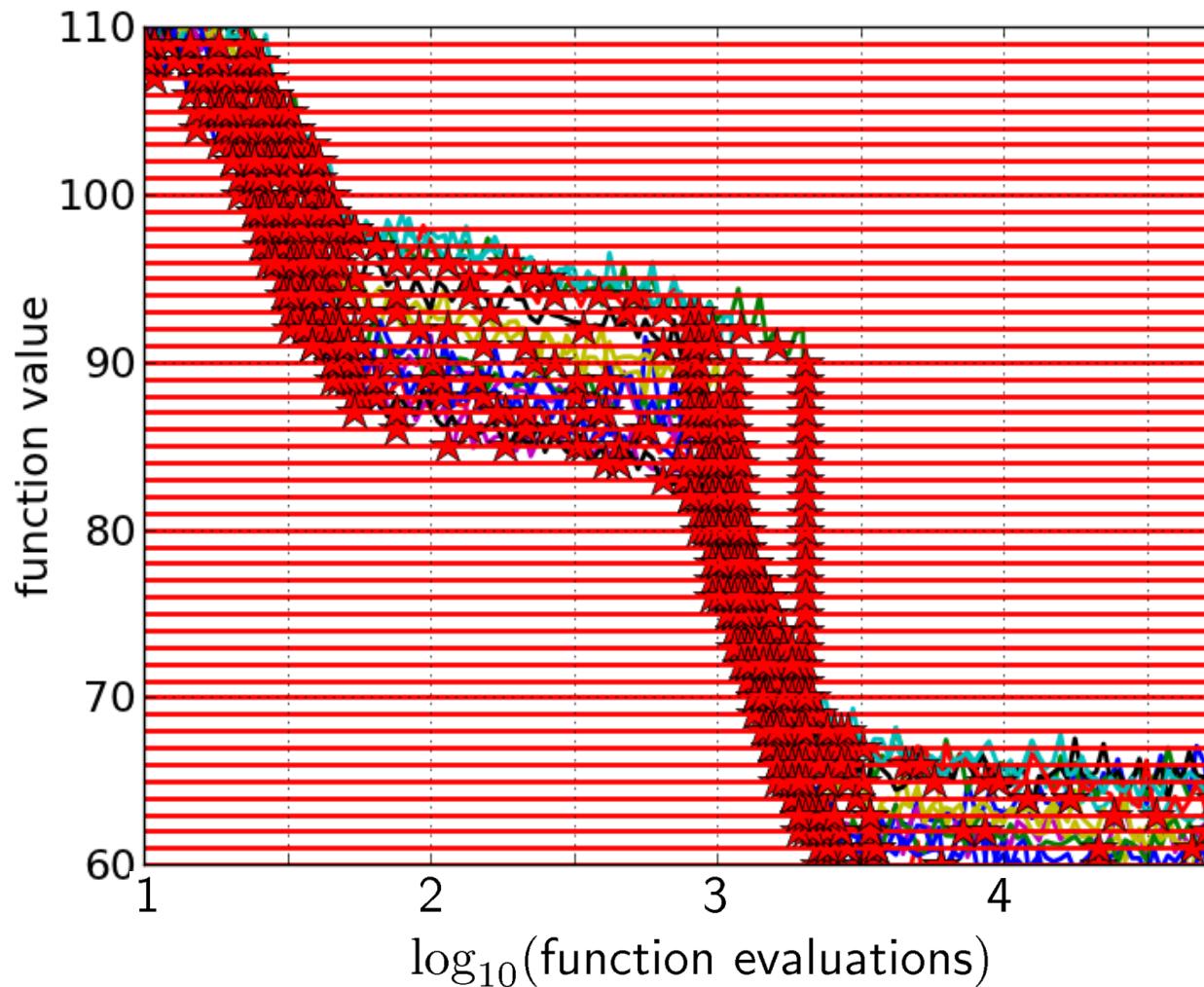


# Aggregation



15 runs

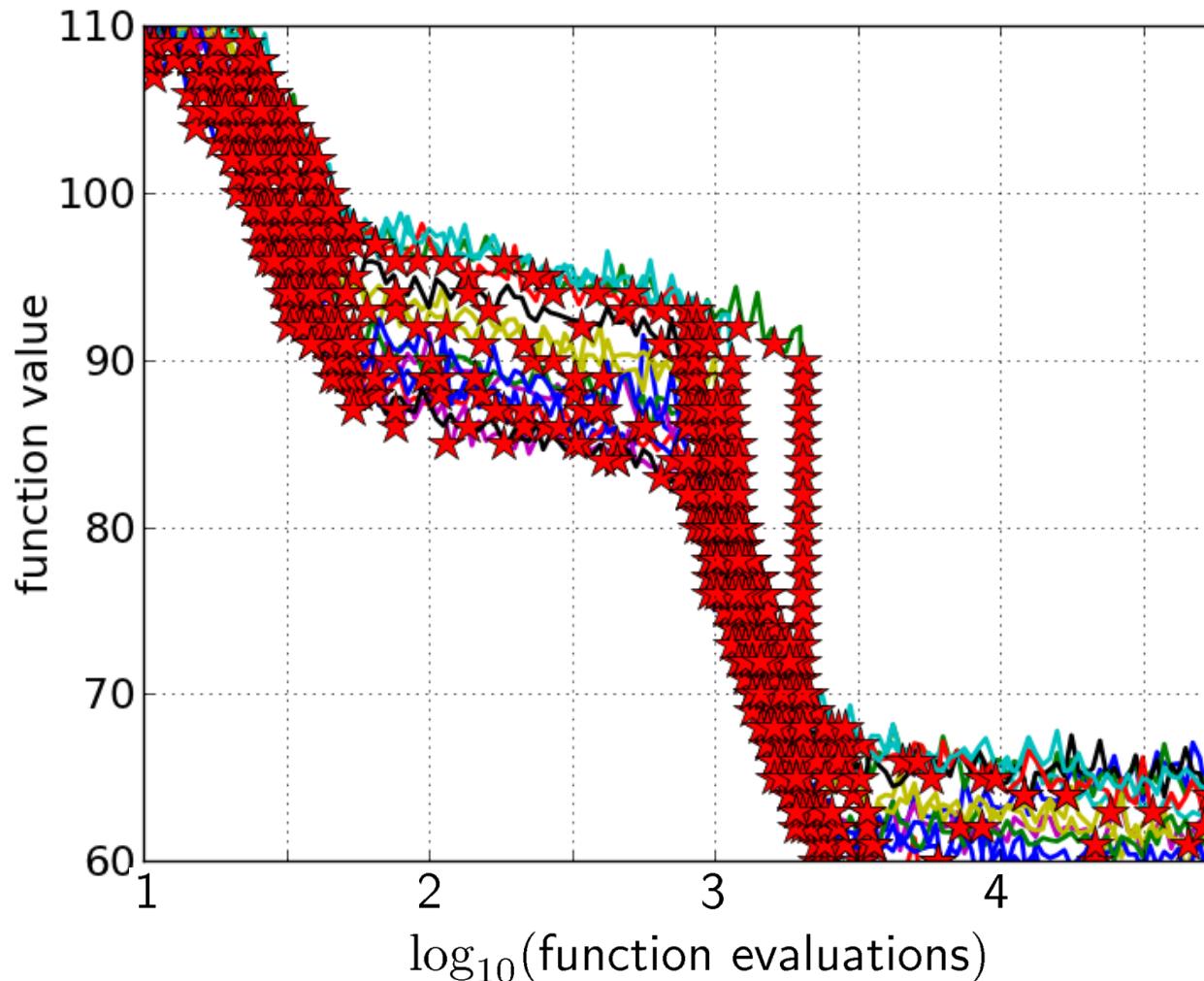
# Aggregation



15 runs

50 targets

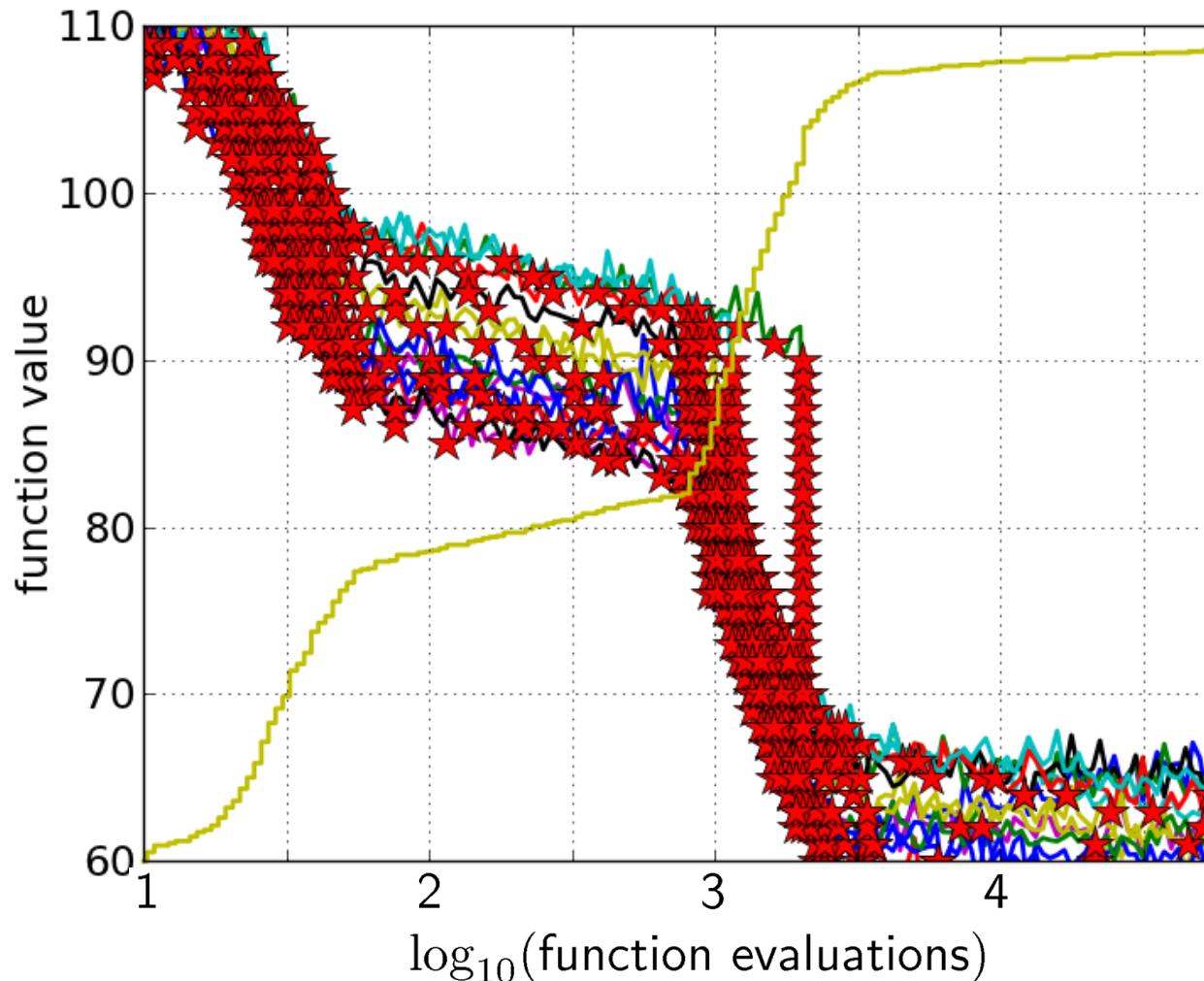
# Aggregation



15 runs

50 targets

# Aggregation

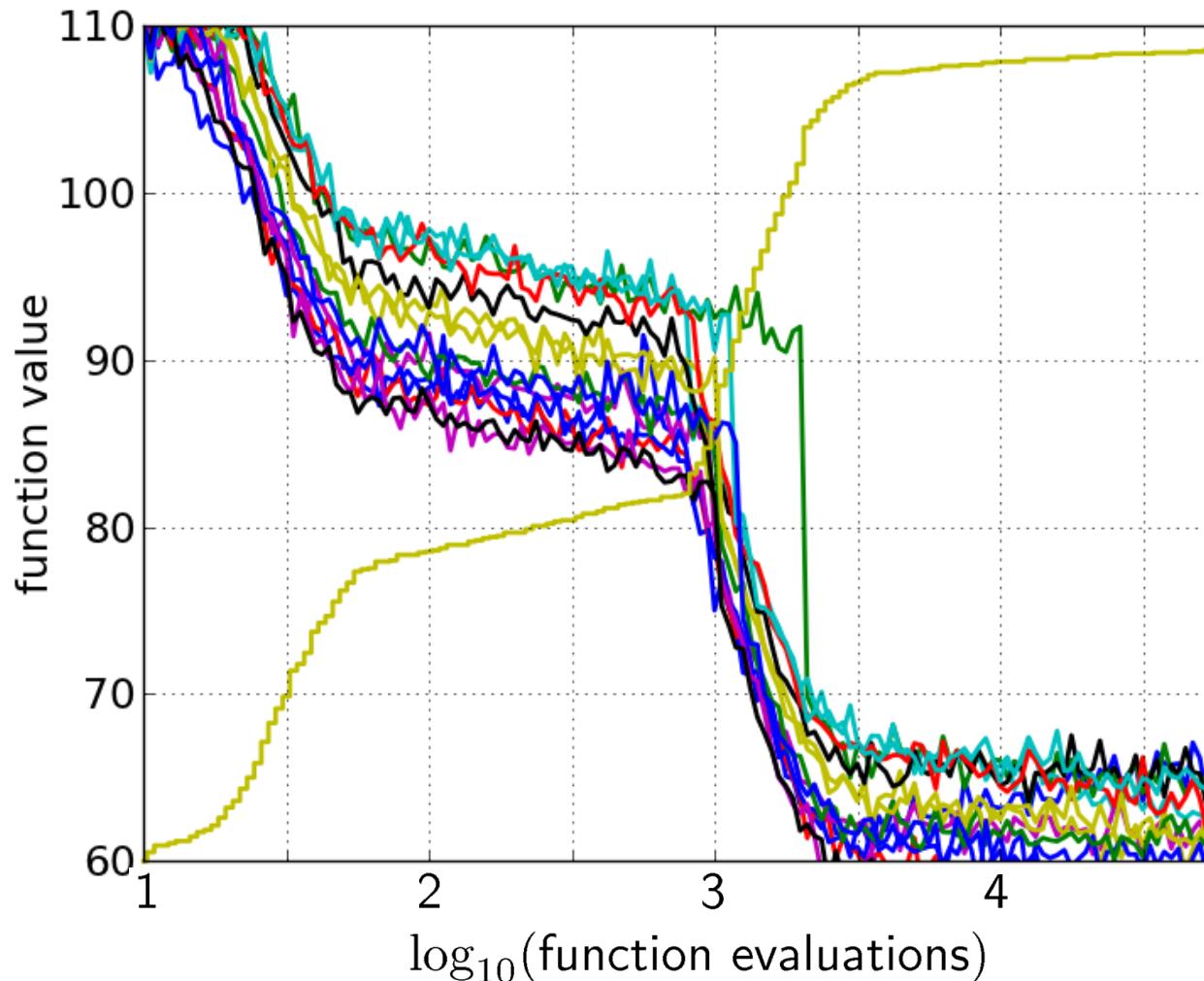


15 runs

50 targets

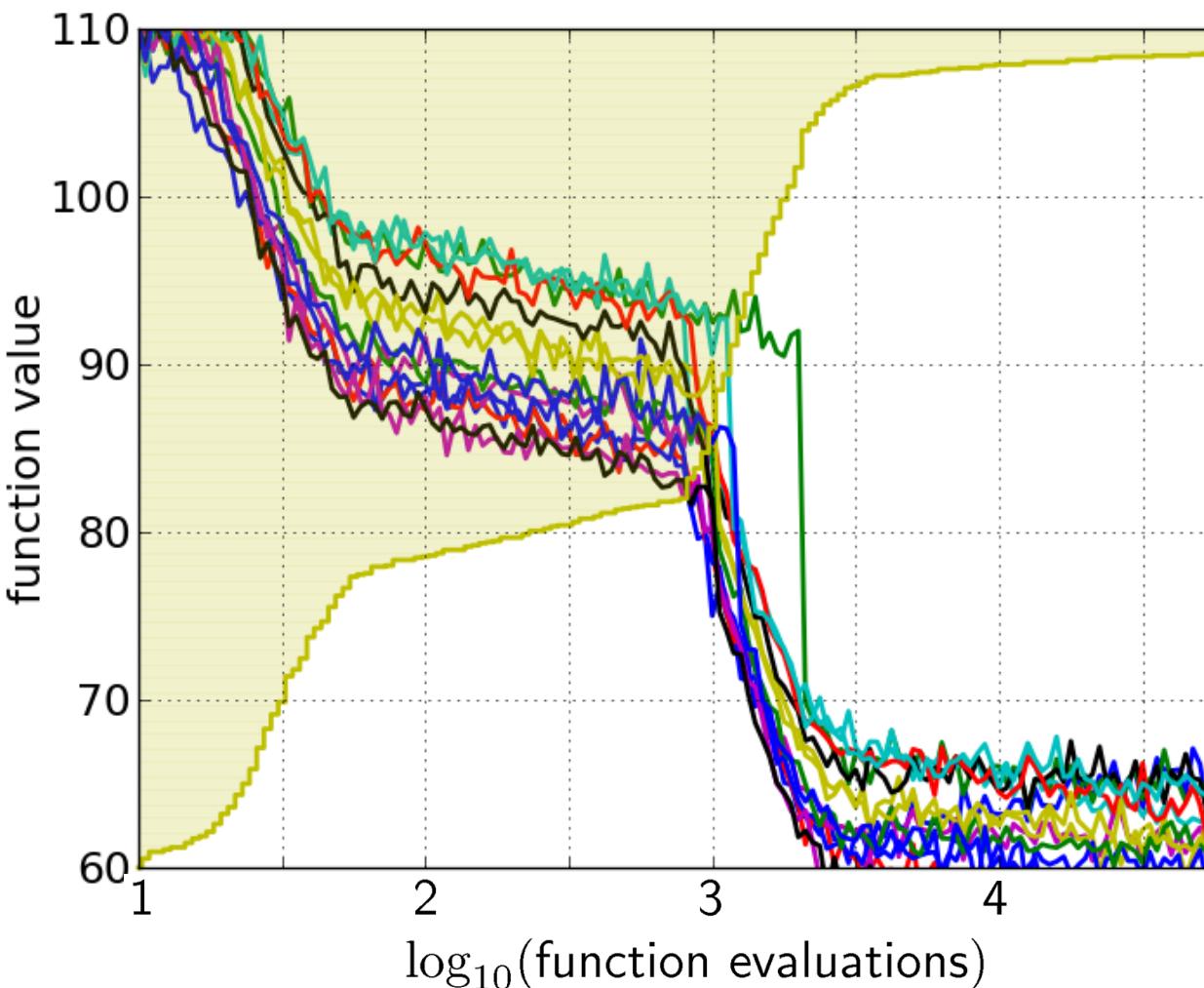
ECDF with 750  
steps

# 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

=

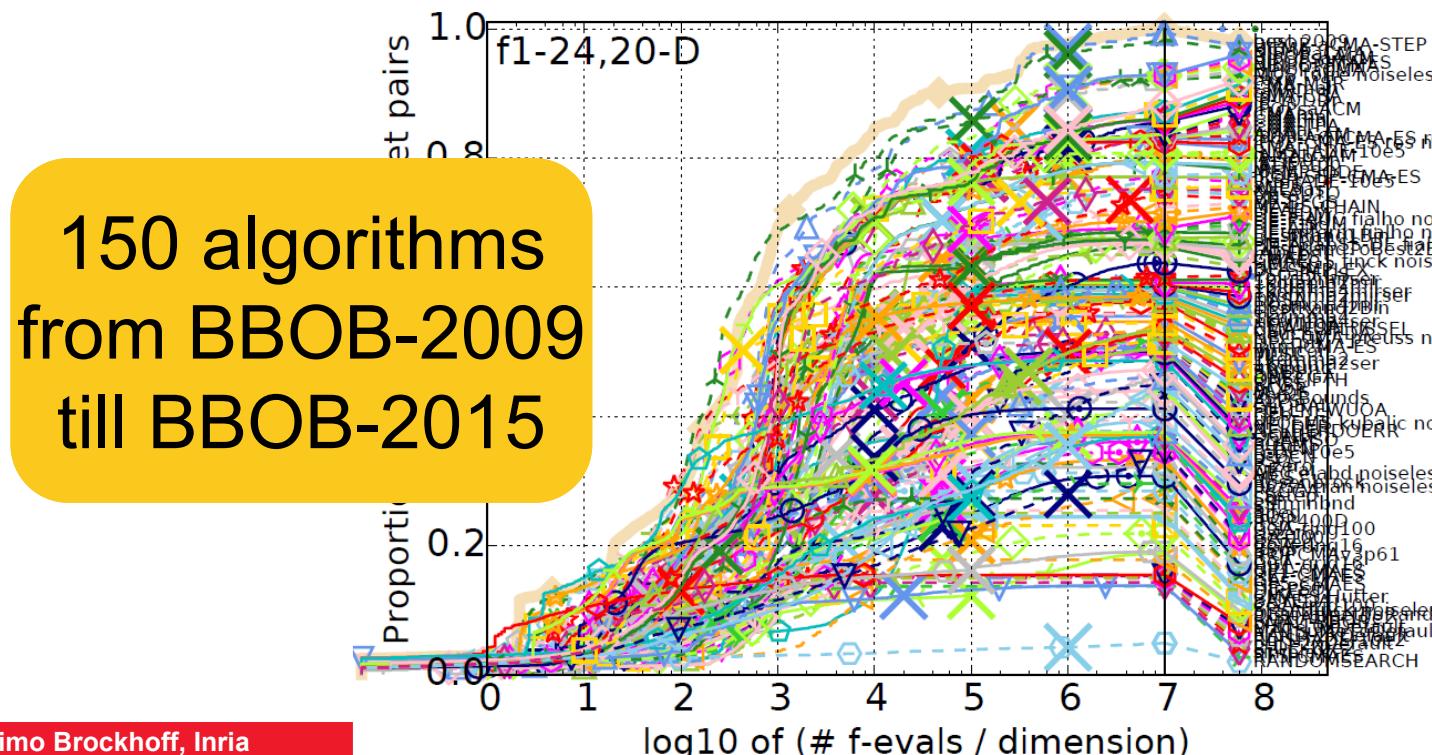
average log  
runtime

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

# Worth to Note: ECDFs in COCO

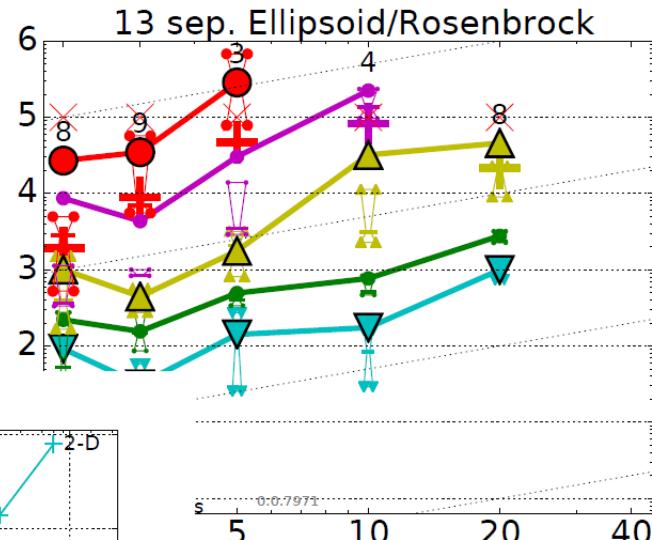
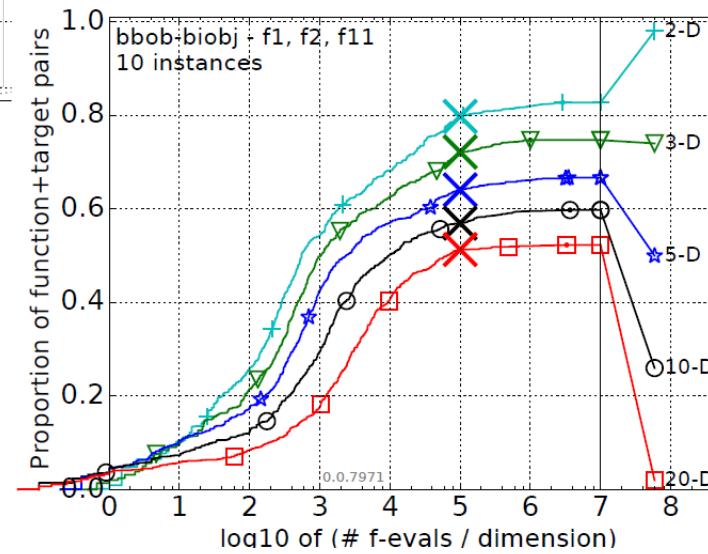
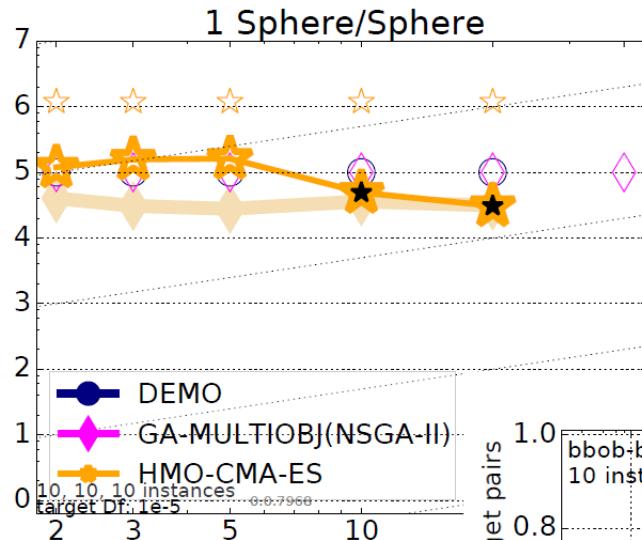
In COCO, ECDF graphs

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



# More Automated Plots with COCO

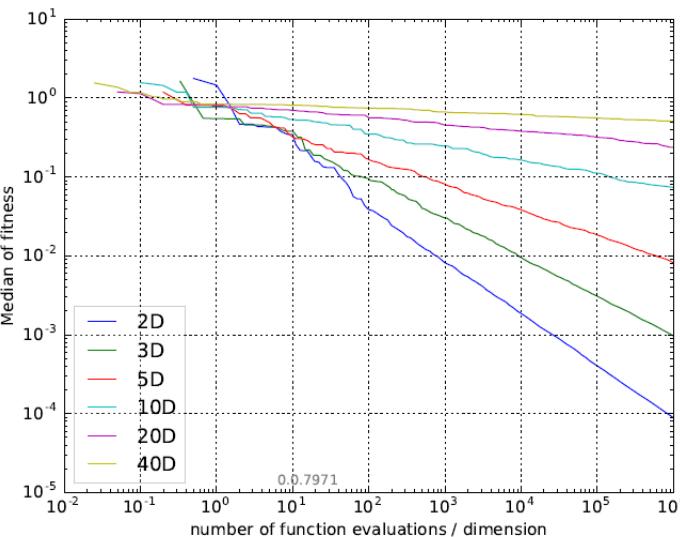
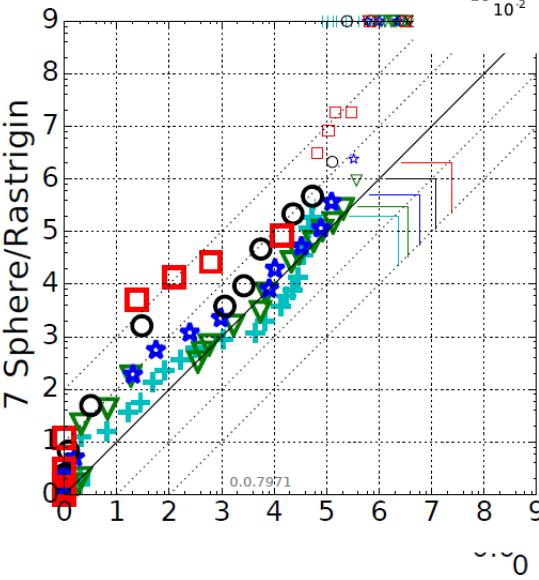
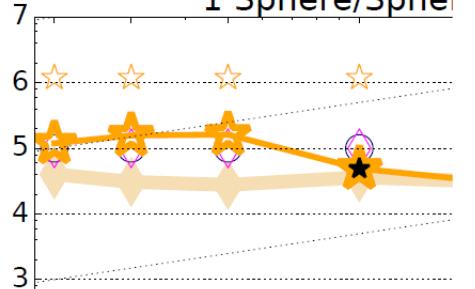
...but no time to explain them here ☹



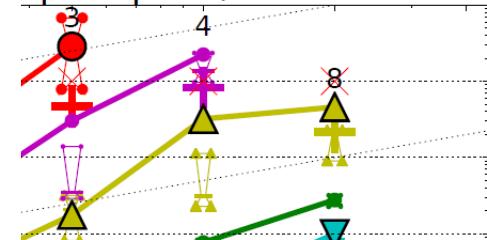
# More Automated Plots with COCO

...but no time to ex

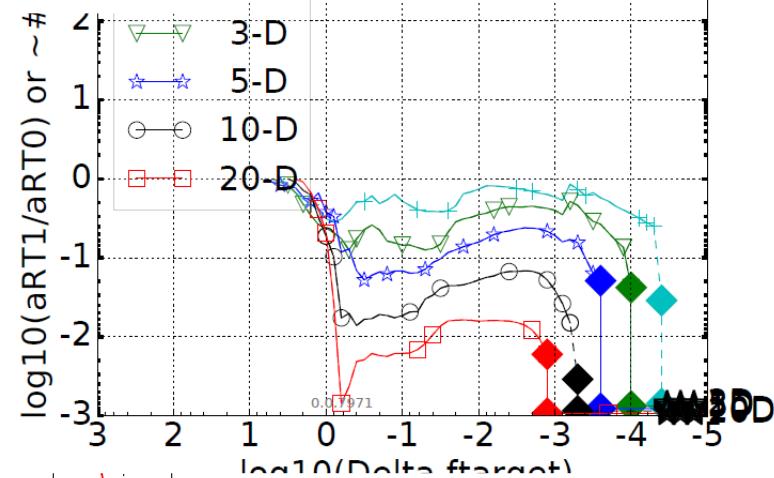
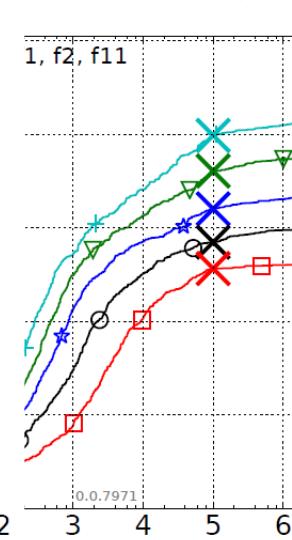
1 Sphere/Sphere



ep. Ellipsoid/Rosenbrock



1 Sphere/Sphere



# Take Home Messages Benchmarking

I hope it became clear...

...that benchmarking is a non-trivial task

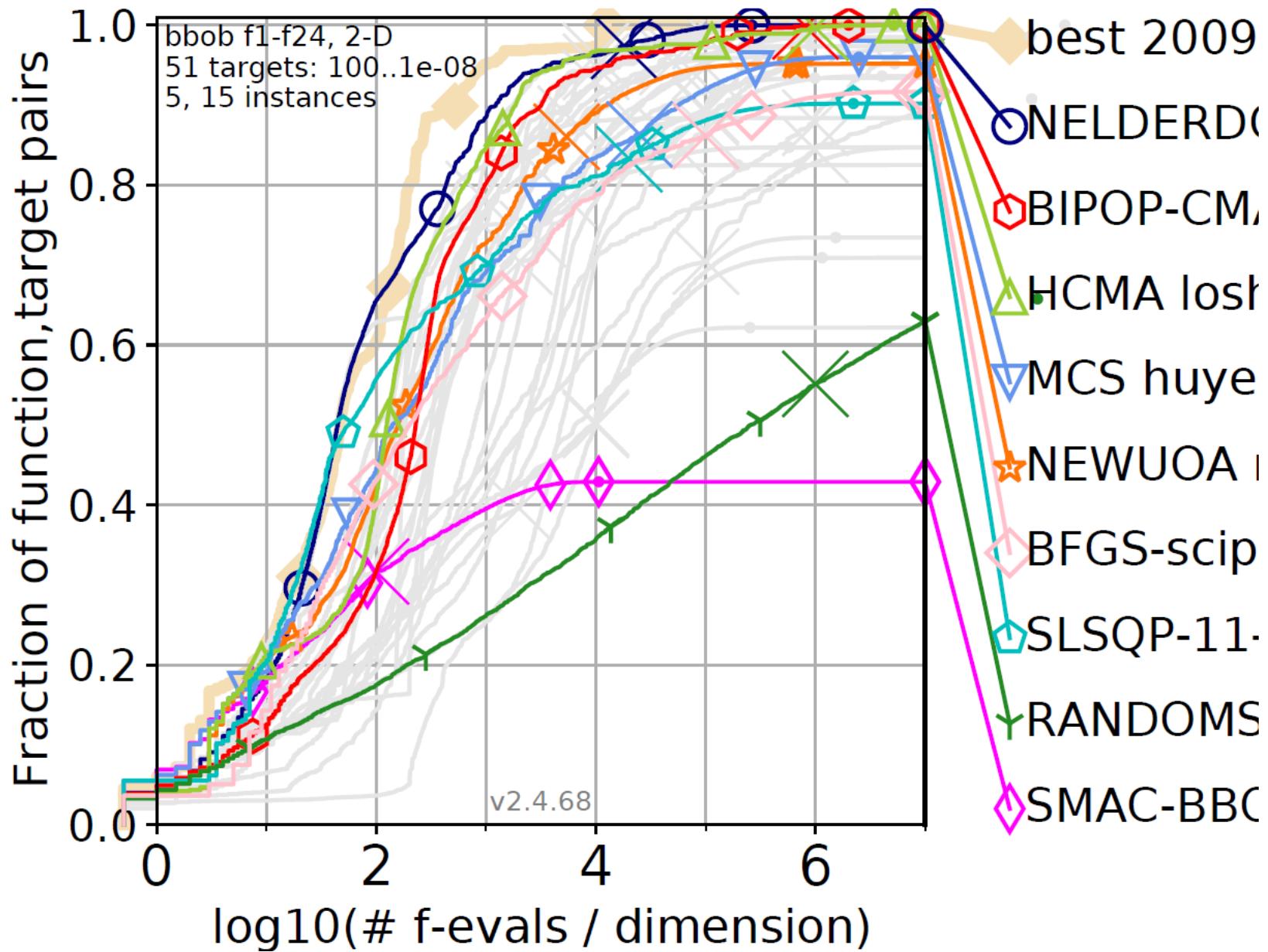
...details matter when comparing algorithms

...and that the COCO platform allows for an automated benchmarking  
and provides data from hundreds of benchmarking experiments

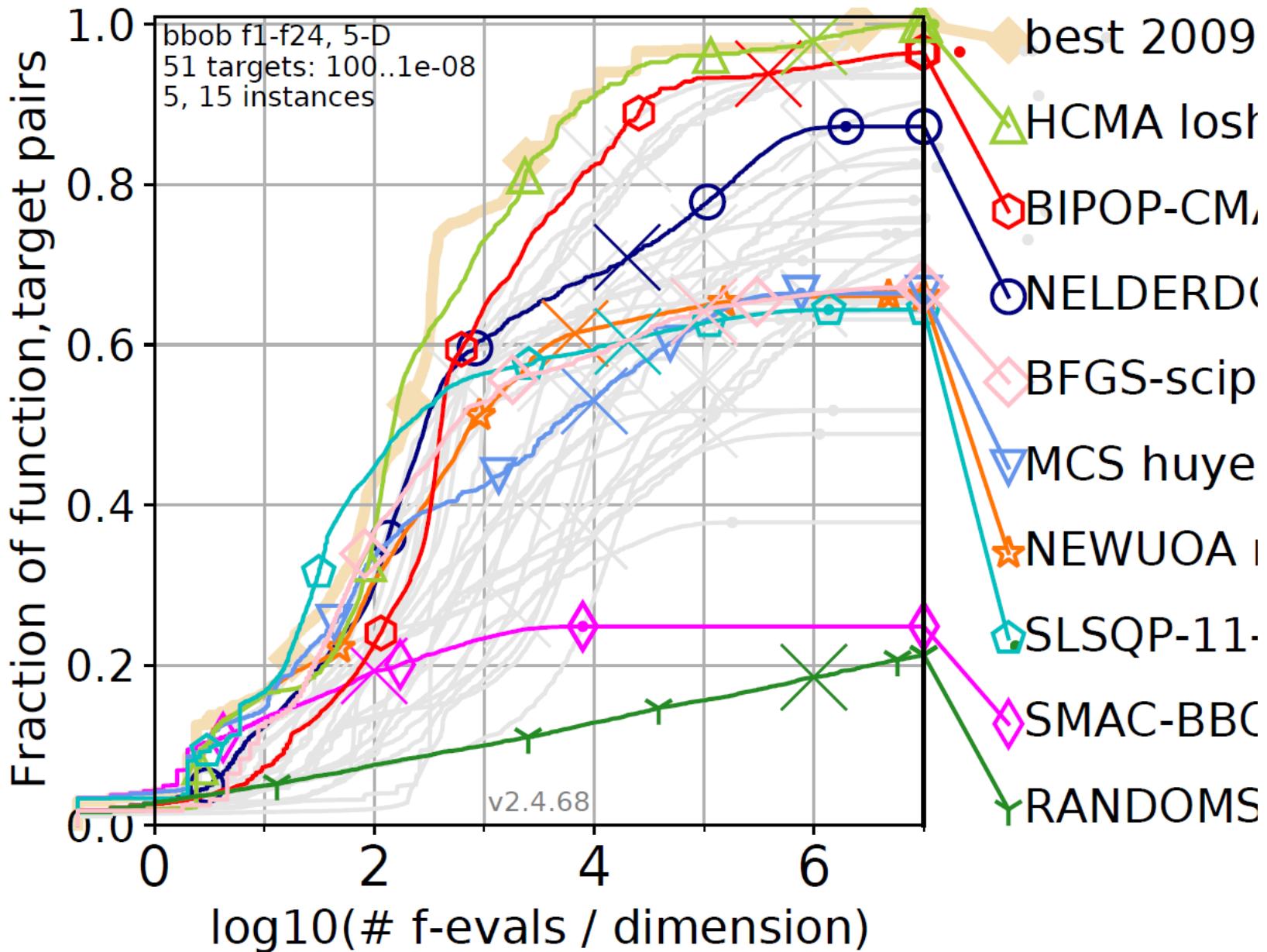
# Final Observations

bringing it all together

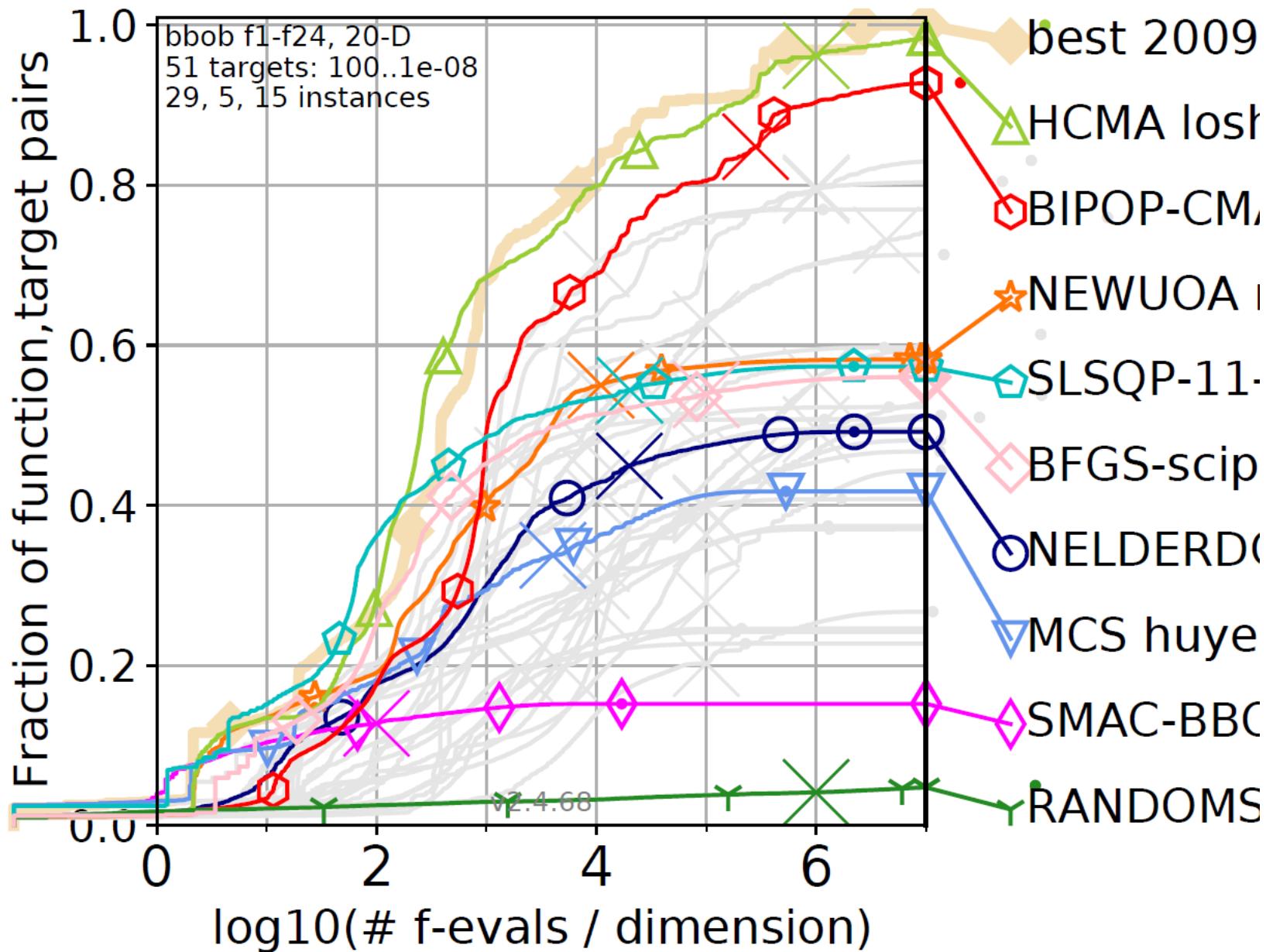
# Some Algorithms in 2-D



# Some Algorithms in 5-D



# Some Algorithms in 20-D



# Recommendations

## Results of 240+ algorithms (noiseless test suite)

- low dimension (2-D, 3-D), small budget: **Nelder Mead** (implementation by B. Doerr et al.)
- very small budget (<50D funevals): **BBOB-SMAC** (an EGO variant) or **SLSQP**
- larger dimension, small budget (<500D funevals): **NEWUOA**, **SLSQP**, **BFGS**
- larger dimension, large budget (>500D funevals): **CMA-ES** variants
- portfolio algorithms can combine the best of all worlds, e.g. **HCMA**
- restarts of quick (local) algorithms for multimodality

# Further Observations

## Invariance

- many algorithms are not invariant under affine transformations of the search space (for example PSO), others like CMA-ES variants are
- as a consequence, other types of algorithms are good for separable and non-separable problems

## Robustness of Algorithms

- CMA-ES was tested in several variants, in several programming languages and with several, slightly changing settings
- nevertheless, it showed throughout its variants consistently good behavior and can be considered as a robust algorithm if the budget is not too low

final topic...

# Branch and Bound

# Branch and Bound: Scenario and Idea

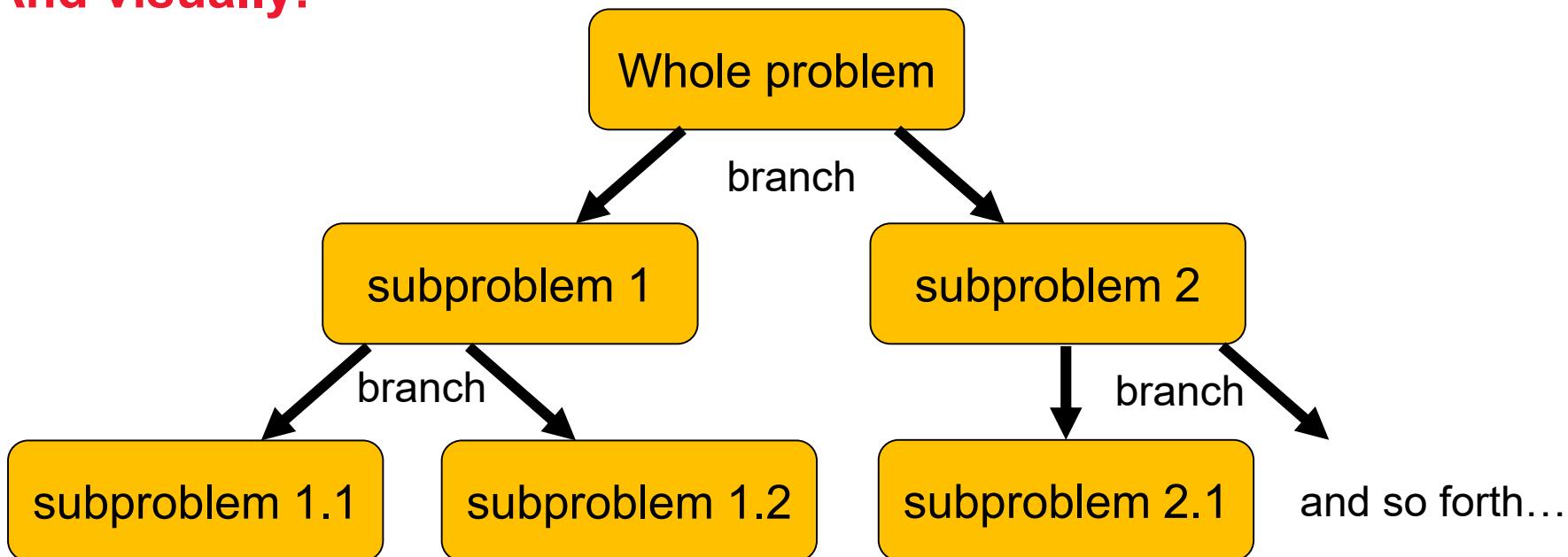
- A general optimization heuristic for discrete optimization
- Split the finite search space iteratively and solve each subproblem separately (“branch”)
- Why is this better than exhaustive search?
  - If we can get upper and lower bounds on the optimal function value of a subproblem (“bound”)...  
*should be quick, i.e. without computing the actual optimal value*
  - ...we can avoid solving subproblems if, for example the upper bound for problem P1 < lower bound for problem P2

# Visualization of Branch and Bound

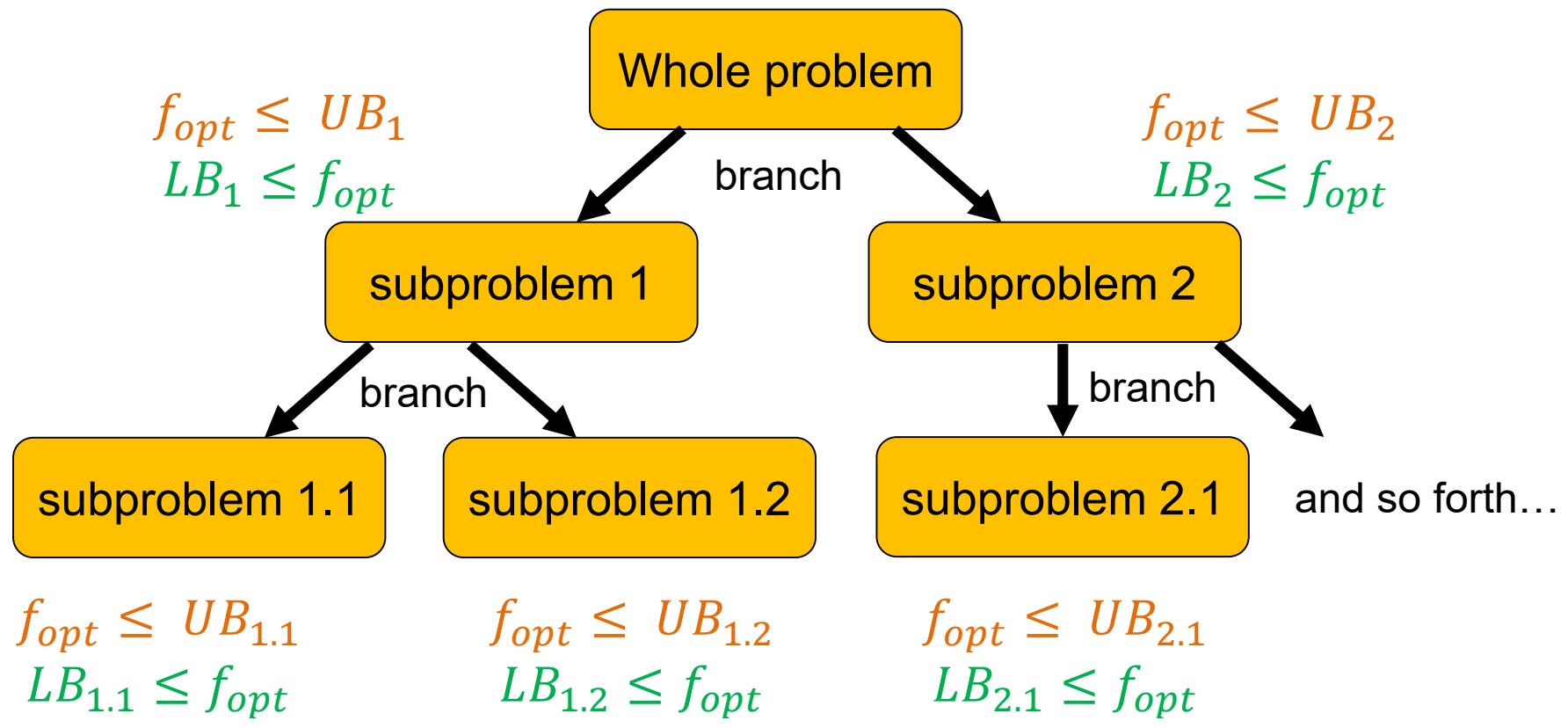
In other words:

- Basically enumerates the entire search space
- But uses clever strategies to avoid enumerations in bad areas

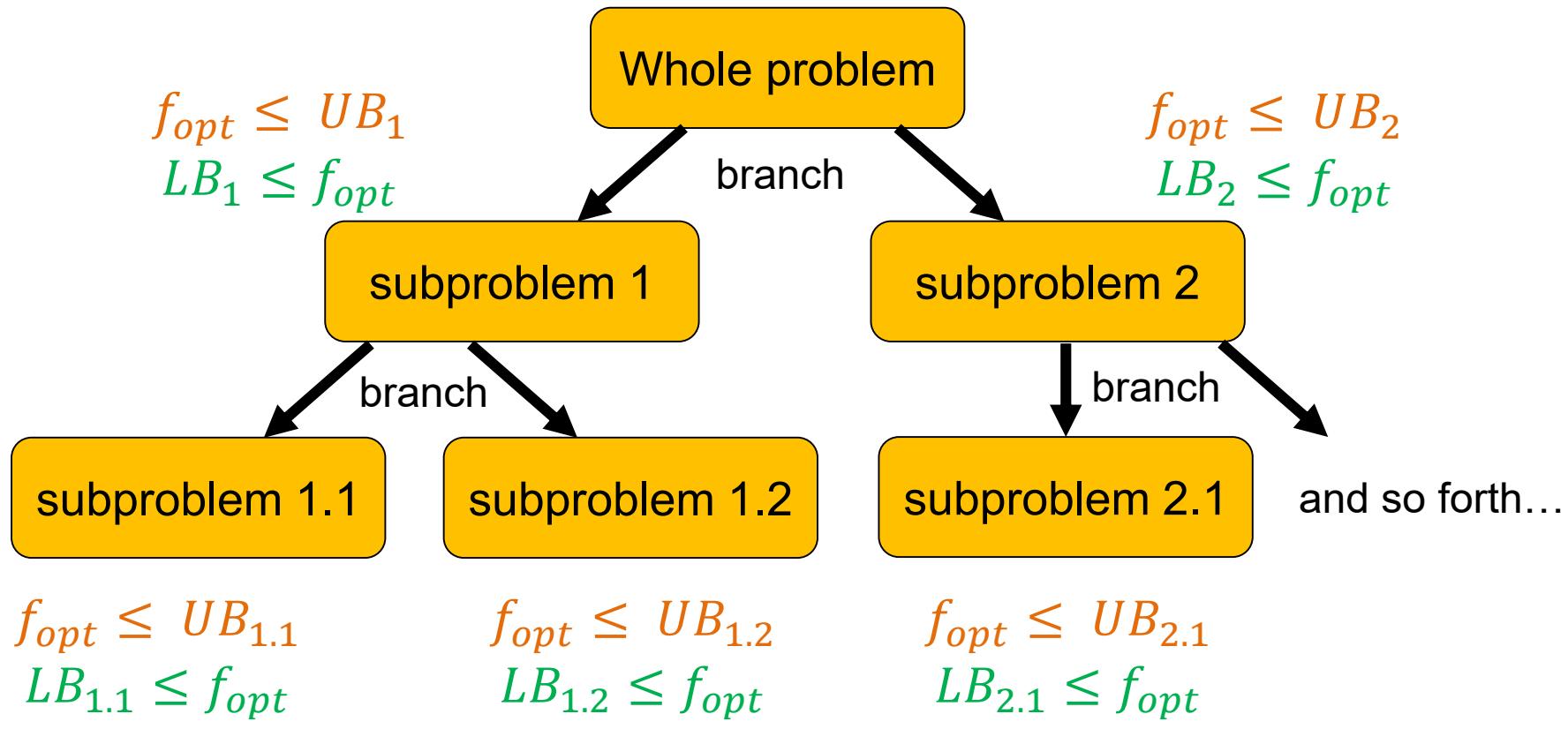
And visually:



# Idea Behind Branch and Bound

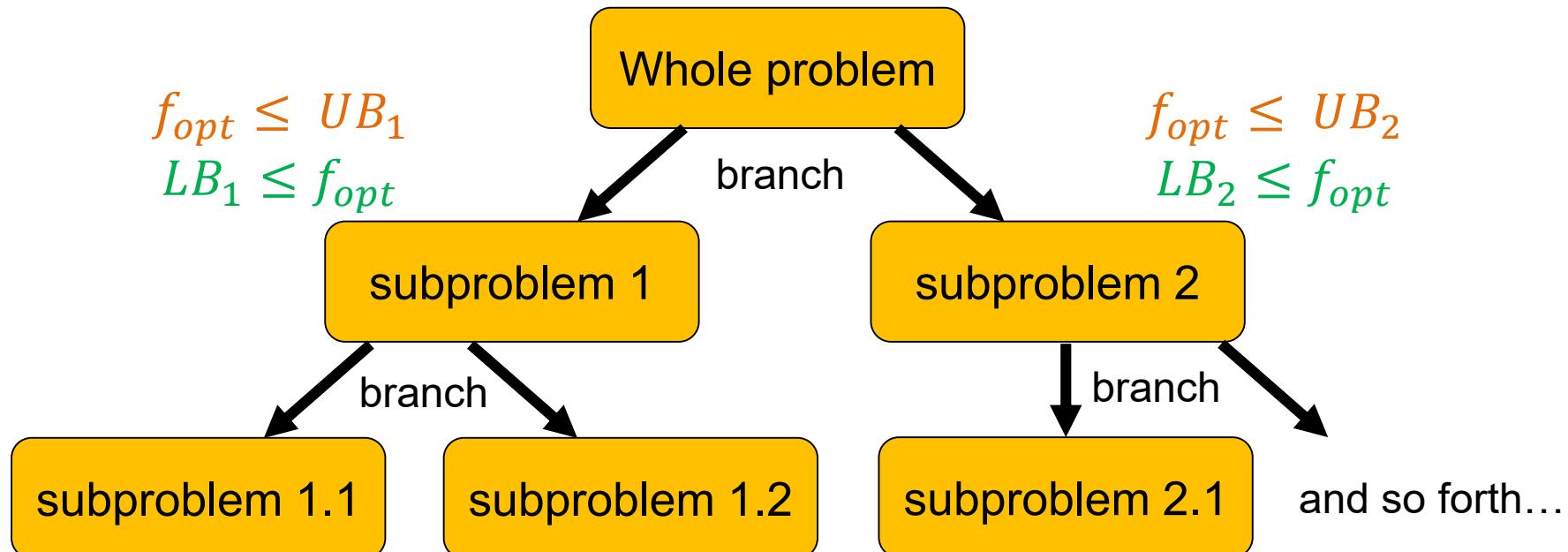


# Idea Behind Branch and Bound



when can we actually avoid evaluating all solutions?

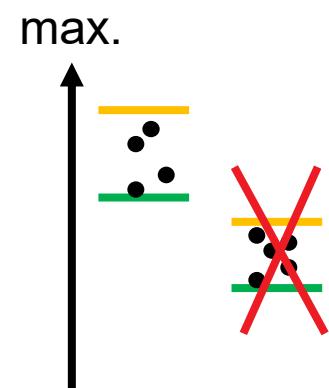
# Idea Behind Branch and Bound



We can stop exploring/branching if

- $UB = LB$
- $UB$  for new subproblem lower than  $LB$  for another

[when maximizing]



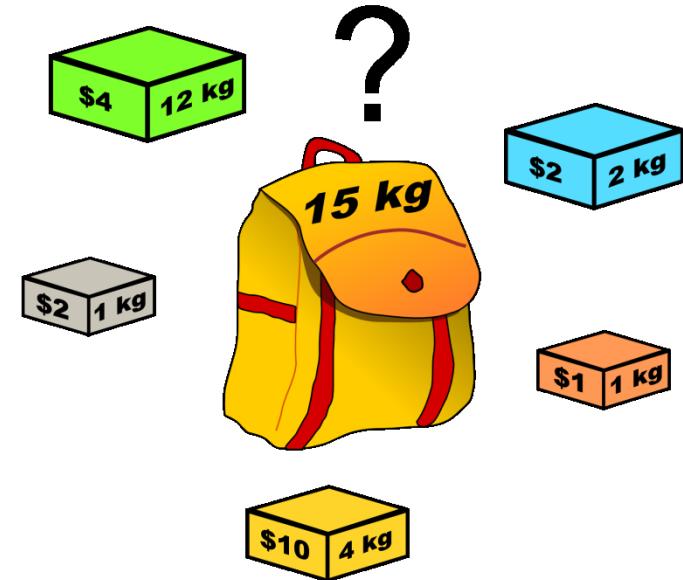
# How do we get Upper and Lower Bounds?

We assume again maximization here...

- A feasible solution gives us a lower bound  
*the optimum will be at least as good as a solution, we know*
- Hence, fast (non-exact) algorithms such as greedy can give us lower bounds
- For upper bounds, we can relax the problem  
*for example, by removing constraints*

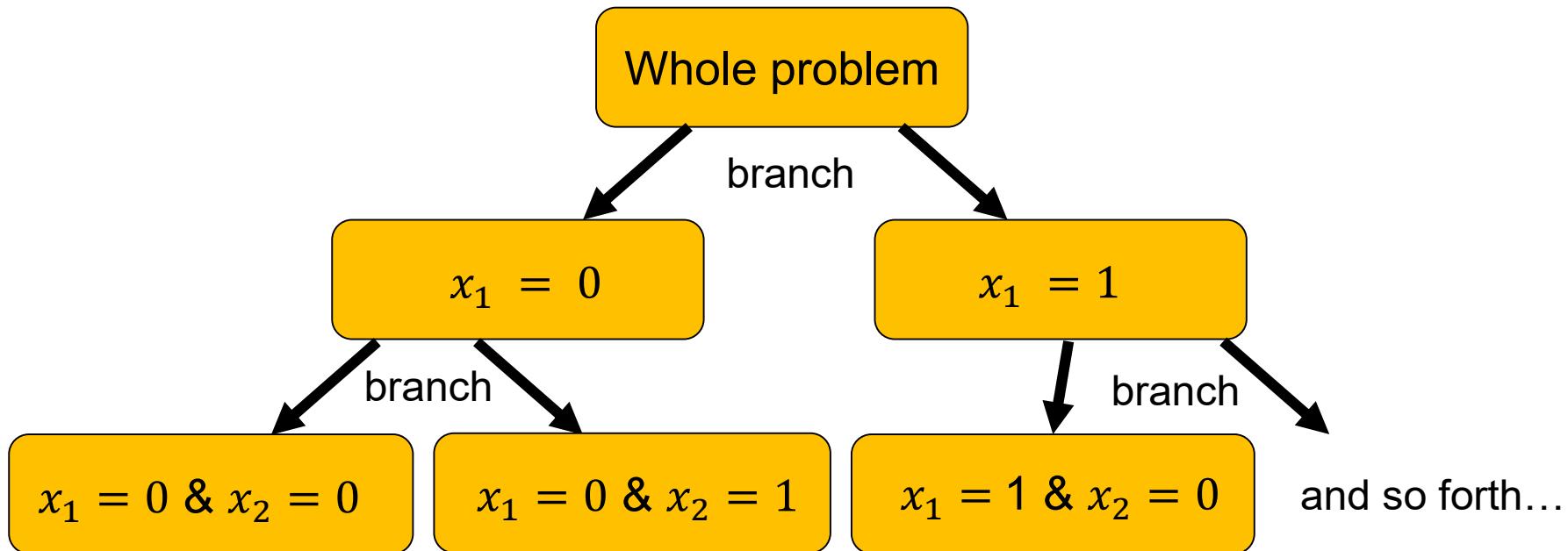
# An Example: Branch&Bound for the KP

$$\begin{aligned} & \geq \sum_{j=1}^n p_j x_j \text{ with } x_j \in \{0, 1\} \\ \text{s.t. } & \sum_{j=1}^n w_j x_j \leq W \end{aligned}$$



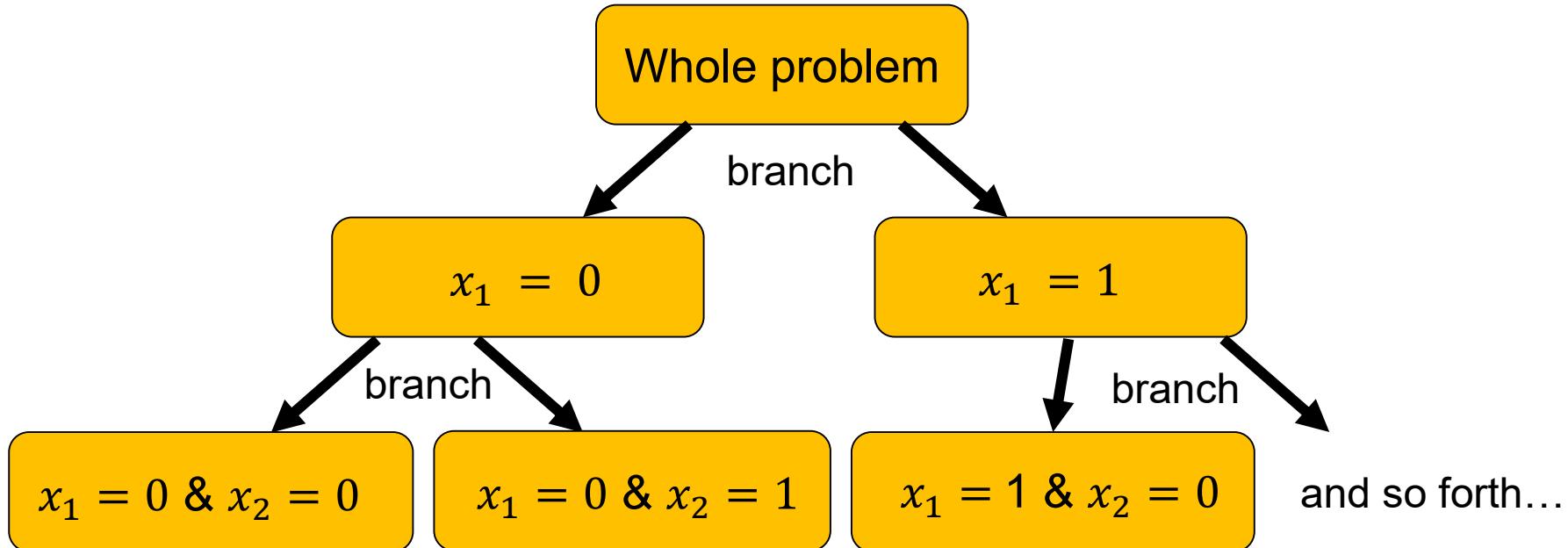
Dake

# KP: How to Branch?



! order of variables plays an important role  
optimally, the subproblems don't overlap

# KP: How to Bound?



Maximization, so LB by greedy approach for example:

Choose items in decreasing profit/weight ratio until knapsack full

UB by relaxation of constraints (on the variables here):

Use greedy algorithm and pack add. item partially if there is space  
*...this variable can be used to branch next*

# A More General Example: Application to ILPs

$$\begin{array}{ll}\text{imize} & c^T x \\ \text{subject to} & Ax \leq b \\ & x \geq 0 \\ \text{and} & x \in \mathbb{Z}^n\end{array}$$

The ILP formalization covers many problems such as

- Traveling Salesperson Person (TSP)
- Vertex Cover and other covering problems
- Set packing and other packing problems
- Boolean satisfiability (SAT)

# Different Approaches to Solve an ILP

- Do not restrict the solutions to integers and **round the solution of this so-called relaxed problem** (=remove the integer constraints) by a continuous solver (e.g. Dantzig or interior point methods)  
→ no guarantee to be exact
- Exploiting the instance property of **A being total unimodular**:
  - unimodular matrix: square integer matrix with determinant +1 or -1  
(equivalent: integer matrix that is invertible over the integers)
  - total unimodular = every square submatrix is unimodular
- feasible solutions are guaranteed to be integer in this case
- algorithms for continuous relaxation can be used (e.g. the simplex algorithm)
- With heuristic methods (typically without any quality guarantee)
  - for example genetic algorithms
- Using exact algorithms such as **branch and bound**

# Branch and Bound for ILPs

Here, we just give an idea instead of a concrete algorithm...

- How to compute upper bounds (assuming maximization)?
- How to split a problem into subproblems (“branching”)?
- Optional: how to compute lower bounds?
- How to decide which next tree node to split?

# Branch and Bound for ILPs

## How to compute upper bounds (assuming maximization)?

- drop the integer constraints and solve the so-called LP-relaxation
- can be solved by standard LP algorithms such as simplex method by Dantzig

## What's then?

- The LP has no feasible solution. Fine. Prune.
- We found an integer solution. Fine as well. Might give us a new lower bound to the overall problem.
- The LP problem has an optimal solution which is worse than the highest lower bound over all already explored subproblems. Fine. Prune.
- Otherwise: Branch on this subproblem: e.g. if optimal solution has  $x_i = 2.7865$ , use  $x_i \leq 2$  and  $x_i \geq 3$  as new constraints

# Branch and Bound for ILPs

## How to split a problem into subproblems (“branching”)?

- mainly needed if the solution of the LP-relaxation is not integer
- branch on a variable which is rational

## Not discussed here in depth due to time:

- Optional: how to compute lower bounds?
- How to decide which next tree node to split?
  - seems to be good choice: subproblem with largest upper bound of LP-relaxation
- Advanced algorithms such as branch and cut (cut part: add specific constraints aka “cutting planes” that ensure integer solutions)

# Conclusions

I hope it became clear...

...what the algorithmic ideas behind **branch and bound** are  
...and for which problem types it is supposed to be **suitable**

# Internships/Master's Theses in Blackbox Optimization



RandOpt team  
Inria and Ecole Polytechnique

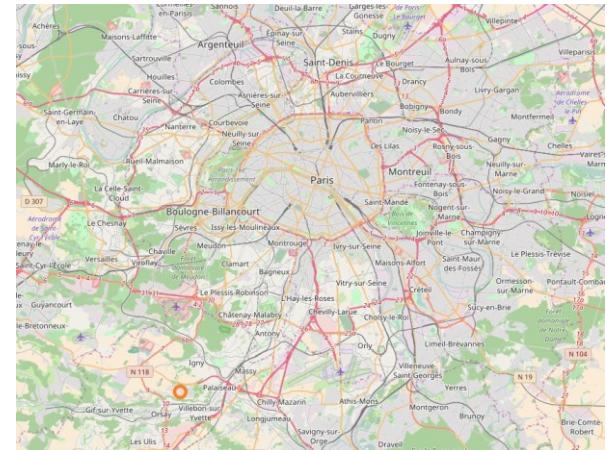


## Permanent members:

Anne Auger, Dimo Brockhoff, Nikolaus Hansen

<https://team.inria.fr/randopt/team-members/>

3 (soon 4) PhD students



## Interests:

constrained  
large-scale      multiobjective  
CMA-ES      blackbox optimization      expensive  
theory      applications  
algorithm design      benchmarking

# Thank you all!

## Don't forget:

- send your group project's report this Friday  
as PDF to [dimo.brockhoff@inria.fr](mailto:dimo.brockhoff@inria.fr)
- be prepared for the final exam on December 15 (in 2 weeks)