Advanced Optimization

Lecture/Exercise 5: Critically Looking at Data

January 17, 2017

Master AIC

Université Paris-Saclay, Orsay, France

Anne Auger

INRIA Saclay – Ile-de-France

Dimo Brockhoff

INRIA Saclay – Ile-de-France

Course Overview

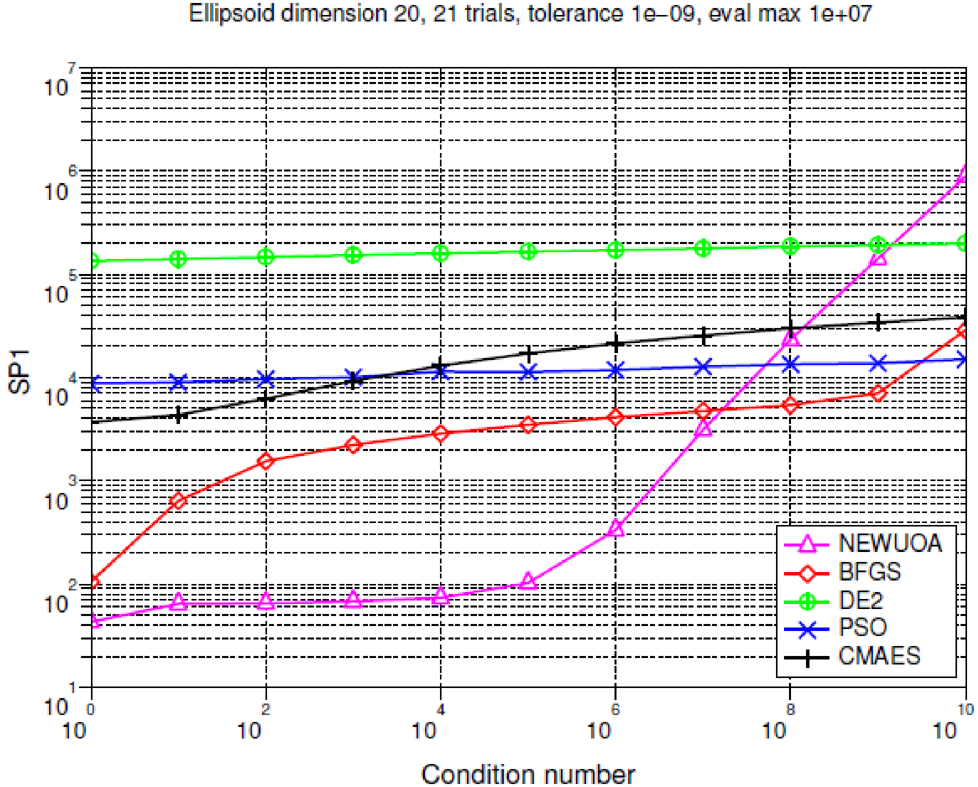
	Date		Topic
1	Tue, 22.11.2016	Dimo	Randomized Algorithms for Discrete Problems
2	Tue, 29.11.2016	Dimo	Exercise: The Travelling Salesperson Problem
3	Tue, 6.12.2016	Anne	Continuous Optimization I
	vacation		
4	Tue, 3.1.2017	Anne	Continuous Optimization II
5	Tue, 10.1.2017	Anne	Continuous Optimization III
6	Tue, 17.1.2017	Dimo	Evolutionary Multiobjective Optimization I
7	Tue, 31.1.2017	Dimo	Evolutionary Multiobjective Optimization II
	???		oral presentations (individual time slots)

all from 14:00 till 17:15 in PUIO - E213

Experimental Considerations around CMA-ES and invariances

Comparison to BFGS, NEWUOA, PSO and DE

f convex quadratic, separable with varying condition number α



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

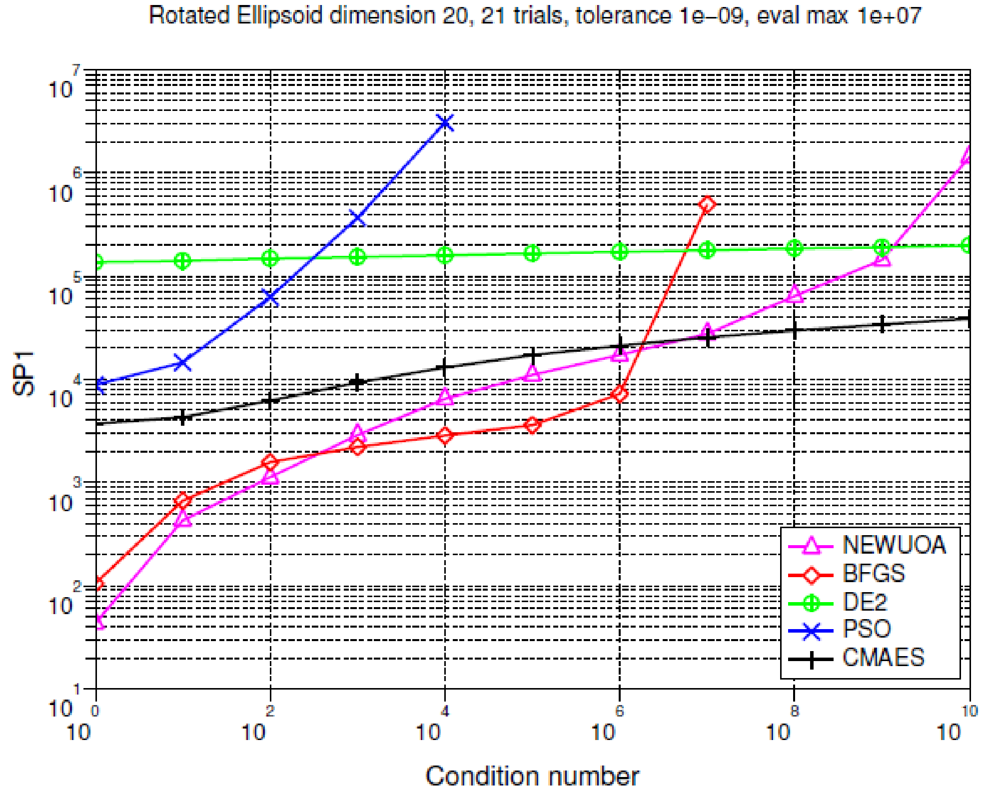
$f(\boldsymbol{x}) = g(\boldsymbol{x}^{\mathrm{T}}\mathbf{H}\boldsymbol{x})$ with
$\boldsymbol{H}$ diagonal
g identity (for BFGS and NEWUOA)
g any order-preserving = strictly increasing function (for all other)

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

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

Comparison to BFGS, NEWUOA, PSO and DE

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



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

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

$\boldsymbol{H}$ full

g identity (for BFGS and NEWUOA)

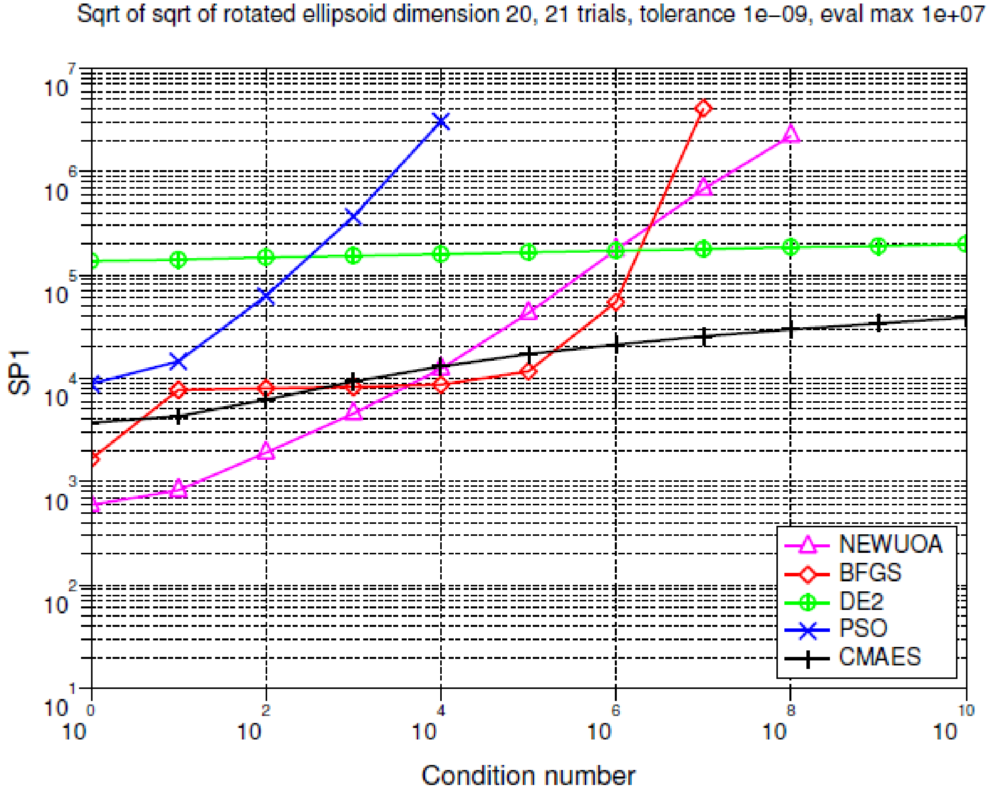
g any order-preserving = strictly increasing function (for all other)

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

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

Comparison to BFGS, NEWUOA, PSO and DE

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



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

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

$\boldsymbol{H}$ full

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

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

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

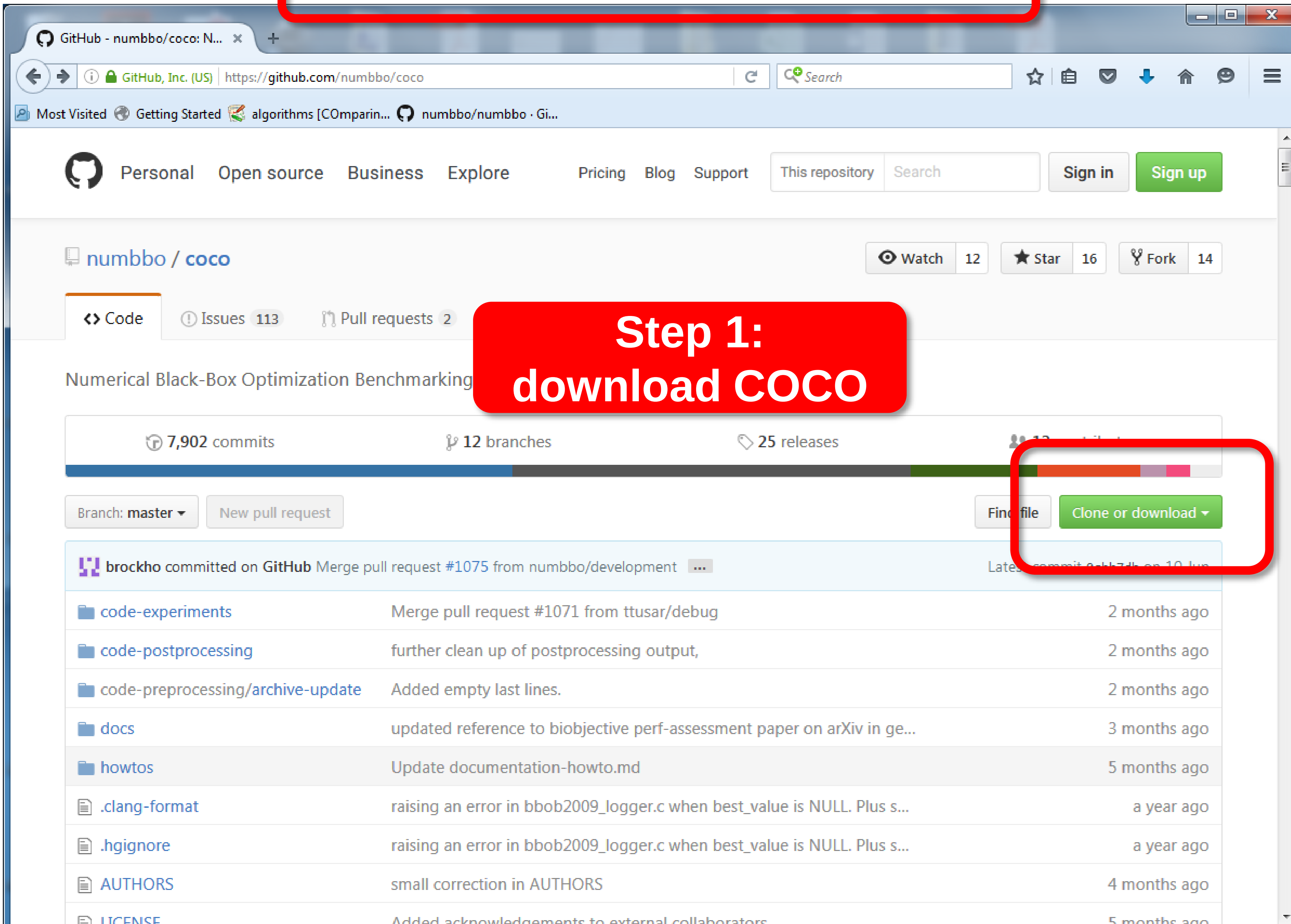
[16] Auger et.al. (2009): Experimental comparisons of derivative free optimization algorithms, SEA

Paper 1: "Improving Covariance Matrix Adaptation Evolution Strategy with Difference..."

Paper 2: "Dynamic Search in Fireworks Algorithm"

Exercise: Looking at COCO Data

https://github.com/numbbo/coco



Step 1: download COCO

https://github.com/numbbo/coco

GitHub - numbbo/coco: N...

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

Search

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

corresponds to the master branch as linked above.

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

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

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

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

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

Step 2:
installation of post-processing

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

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

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

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

Step 3: downloading data

[[algorithms]] COMPARING CONTINUOUS OPTIMISERS: COCO

Show pagesource Old revisions Recent changes Sitemap Login

Search

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

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

Navigation

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

for the moment: IPOP-CMA-ES

https://github.com/numbbo/coco

GitHub - numbbo/coco: N...

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

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

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]
```

postprocess

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

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

```
python -m bbob_pproc IPOP-CMA-ES
```

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

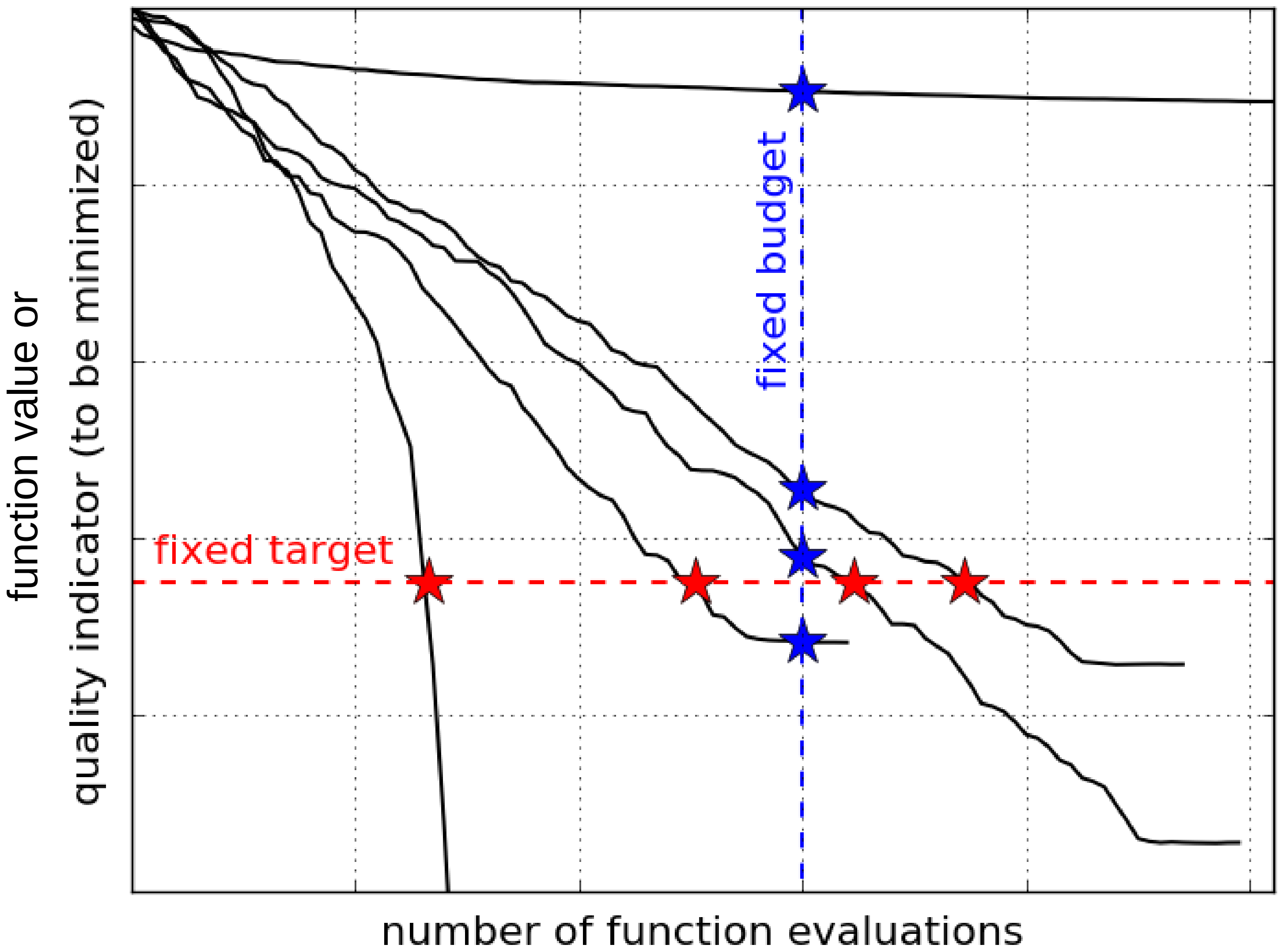
8. Once your algorithm runs well, **increase the budget** in your experiment script, if necessary implement randomized independent restarts, and follow the above steps successively until you are happy.

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

Description by Folder

Measuring Performance Empirically

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

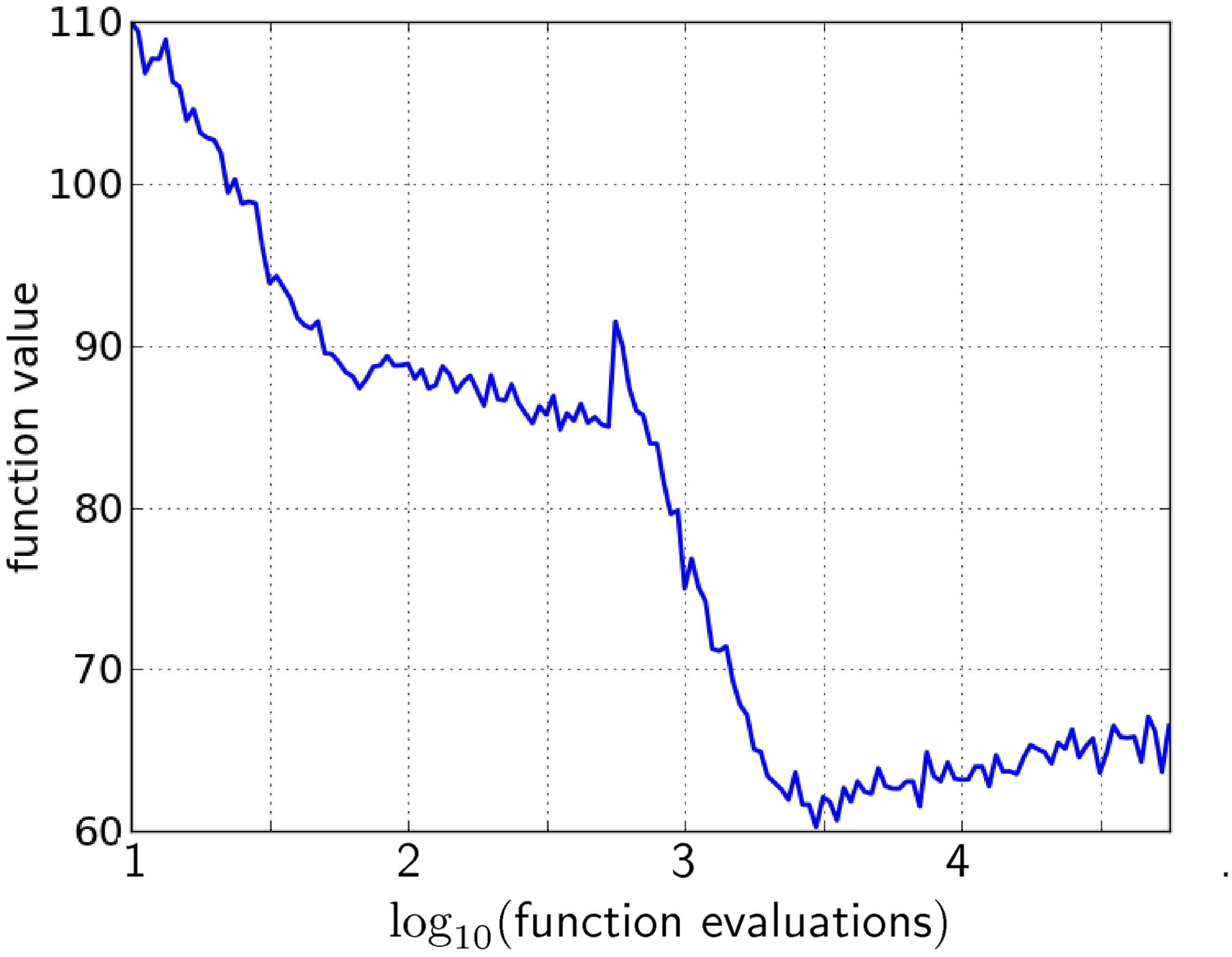


ECDF:

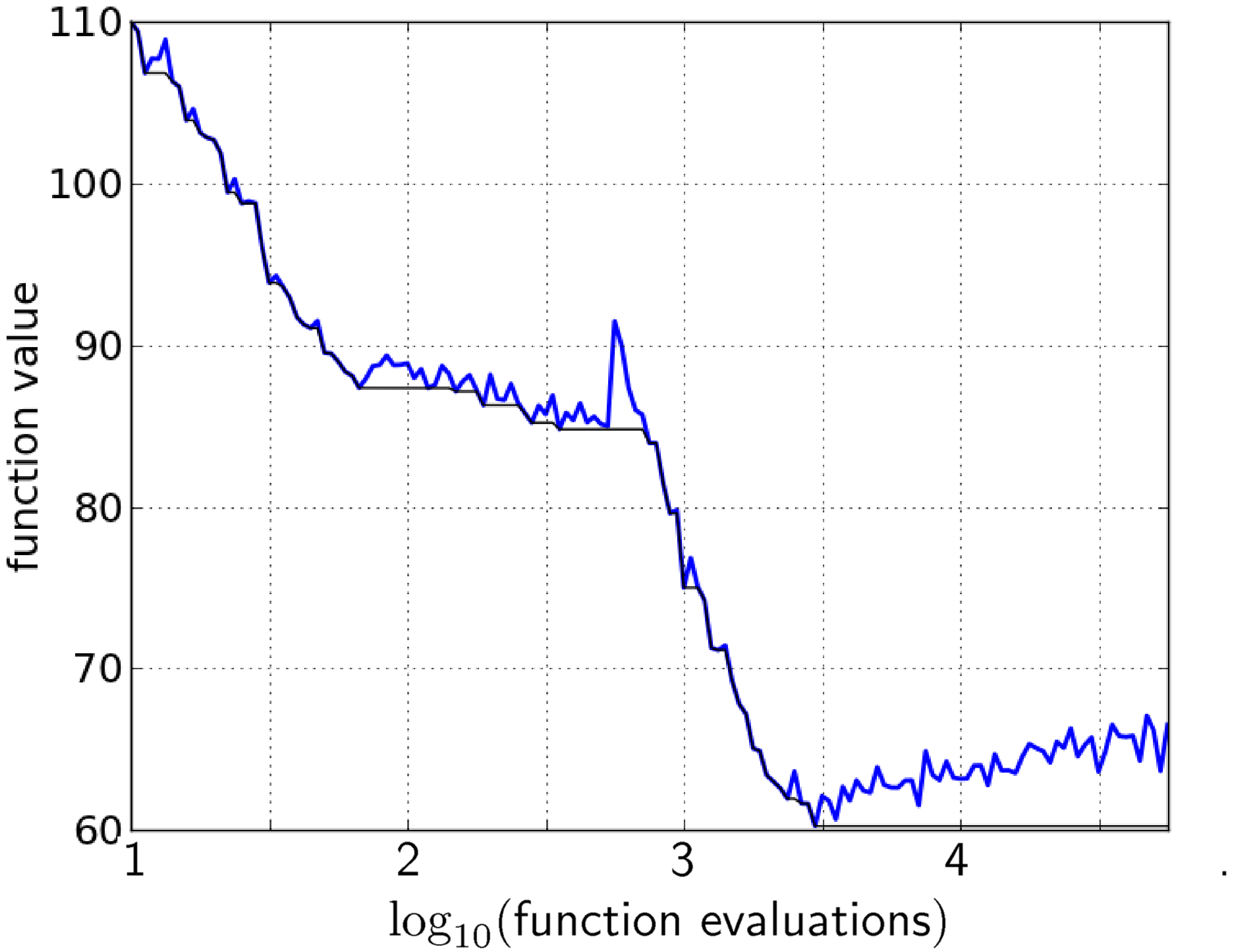
Empirical Cumulative Distribution Function of the Runtime

[aka data profile]

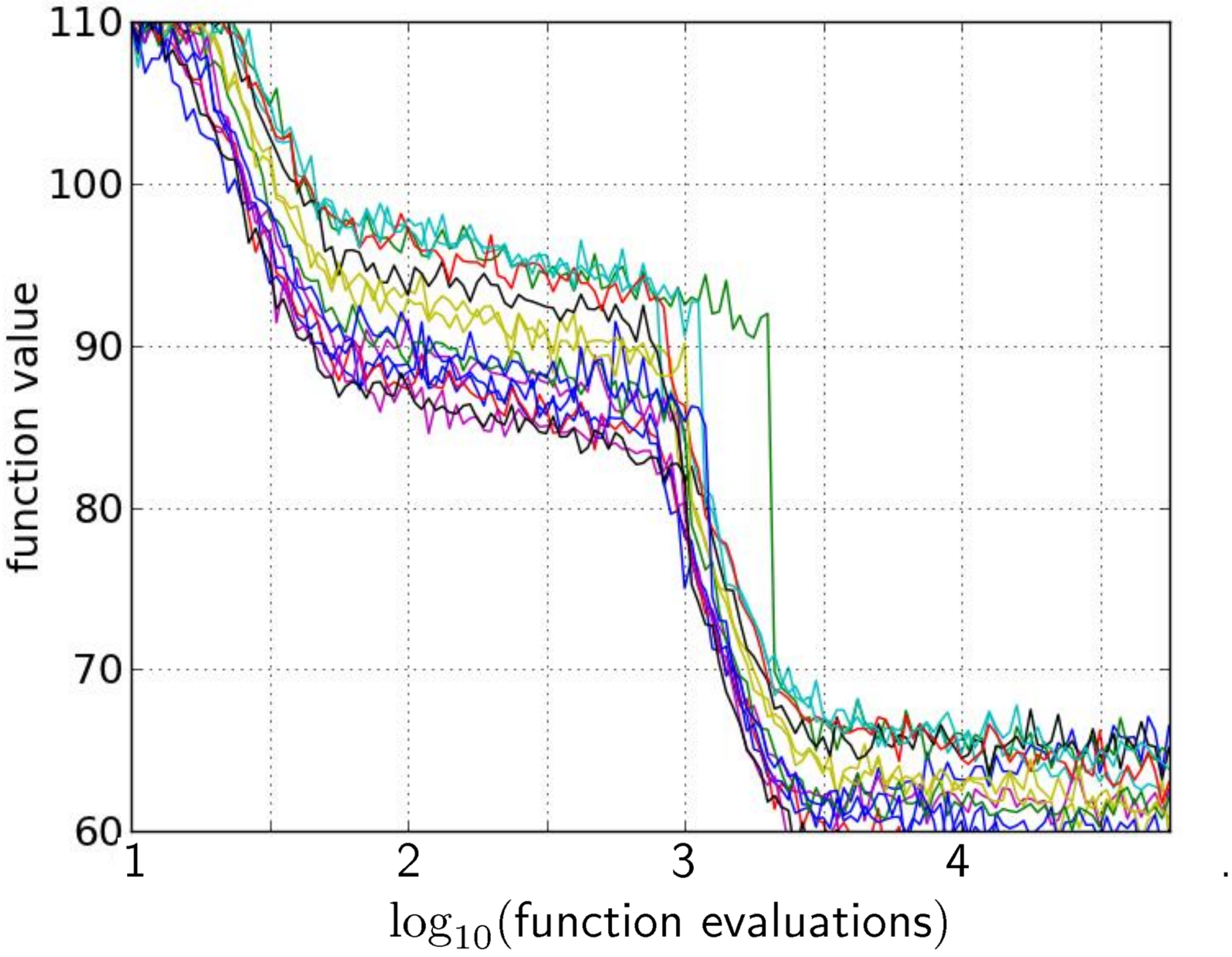
A Convergence Graph



First Hitting Time is Monotonous

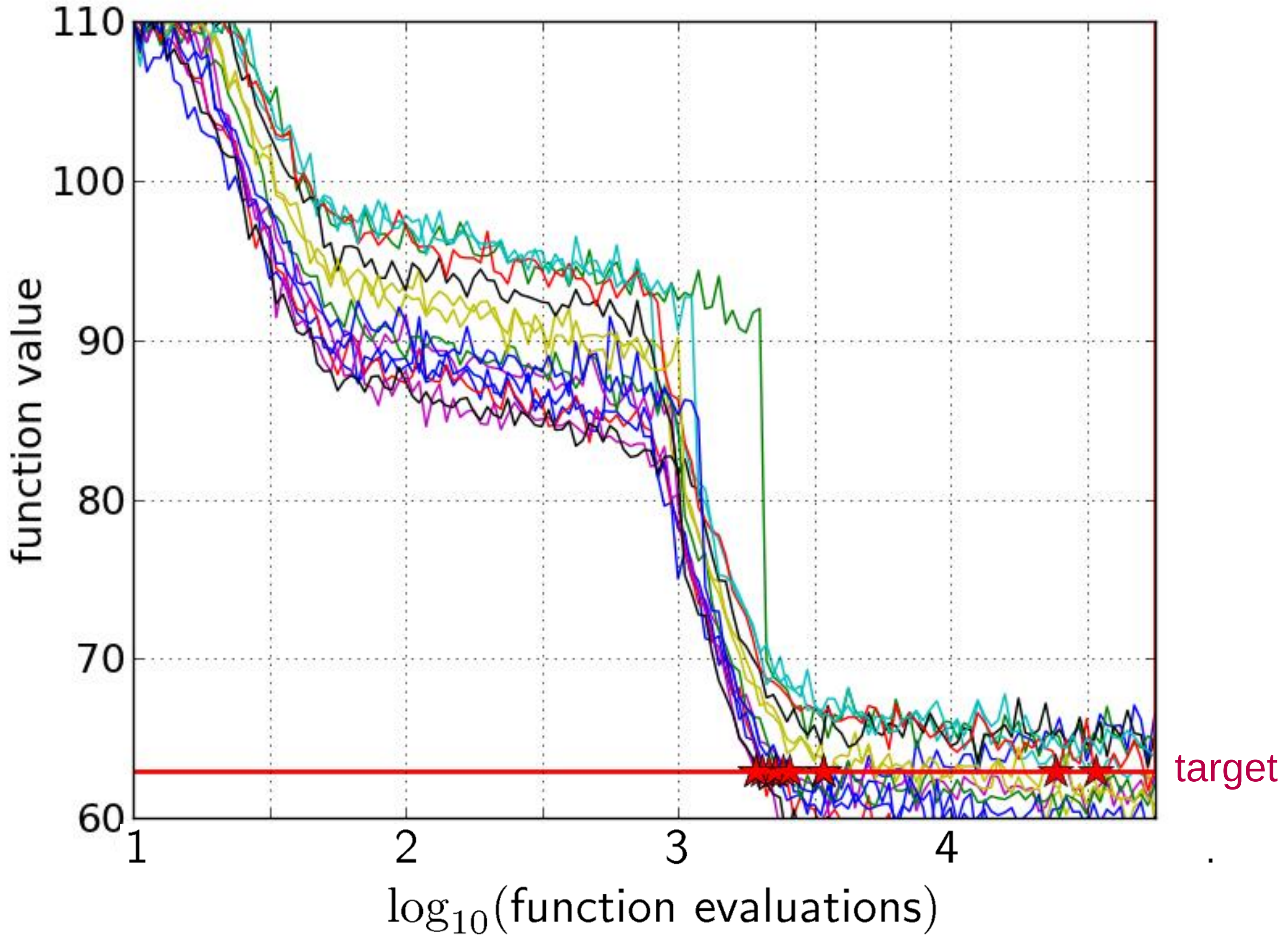


15 Runs

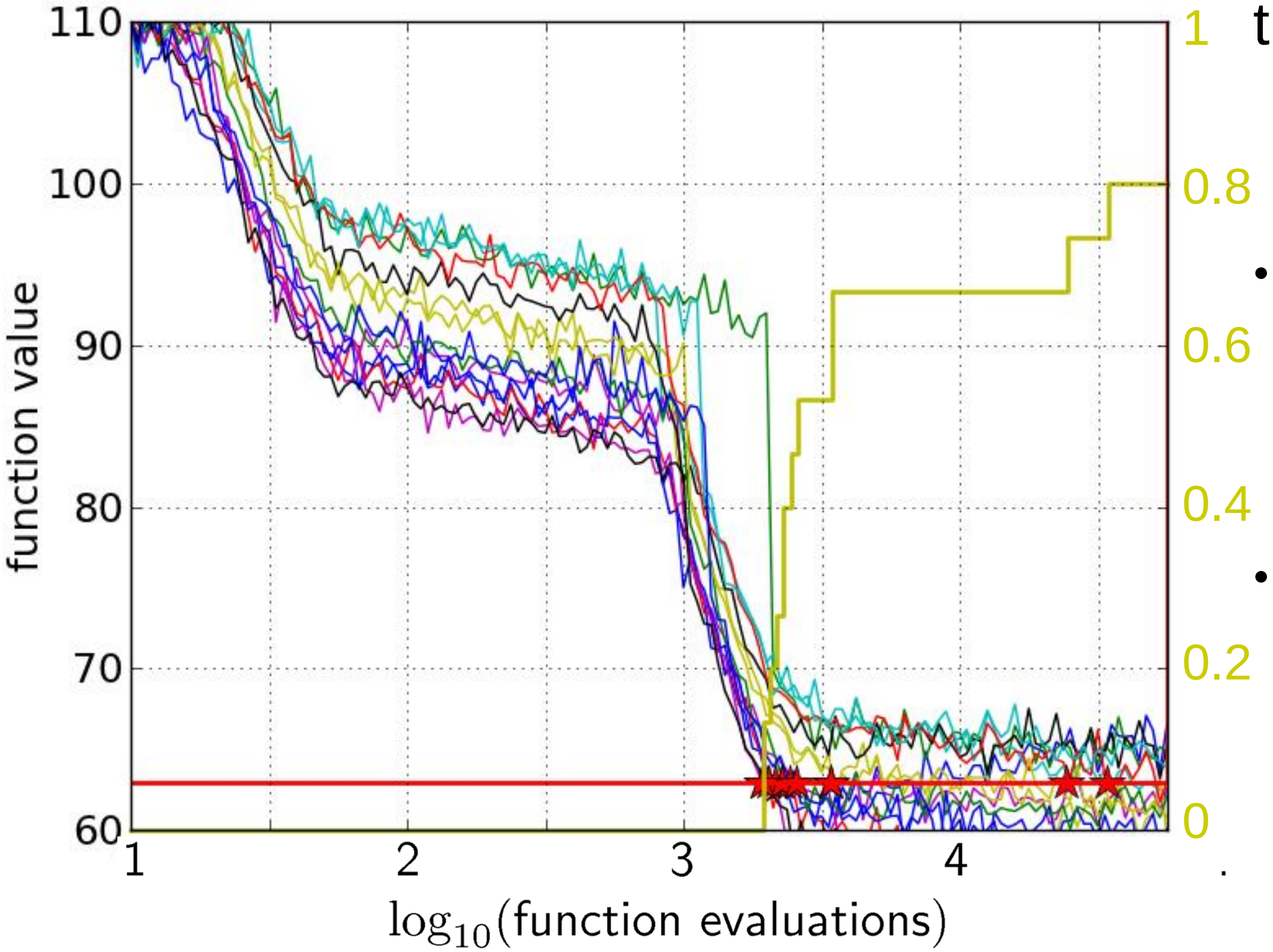


110
100
90
80
70
60
function value
1
2
3
4
$\log_{10}$(function evaluations)

15 Runs ≤ 15 Runtime Data Points

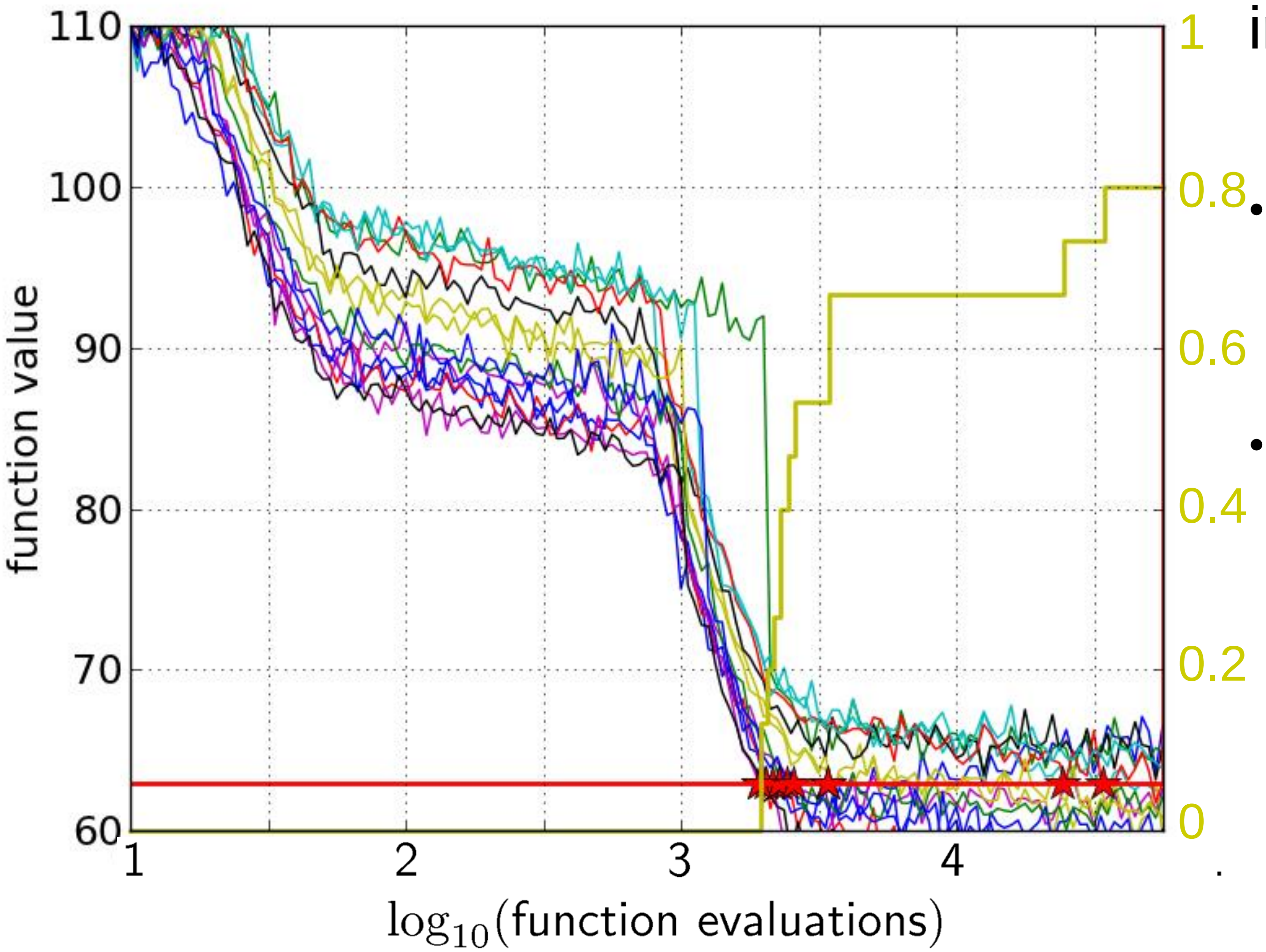


Empirical Cumulative Distribution



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

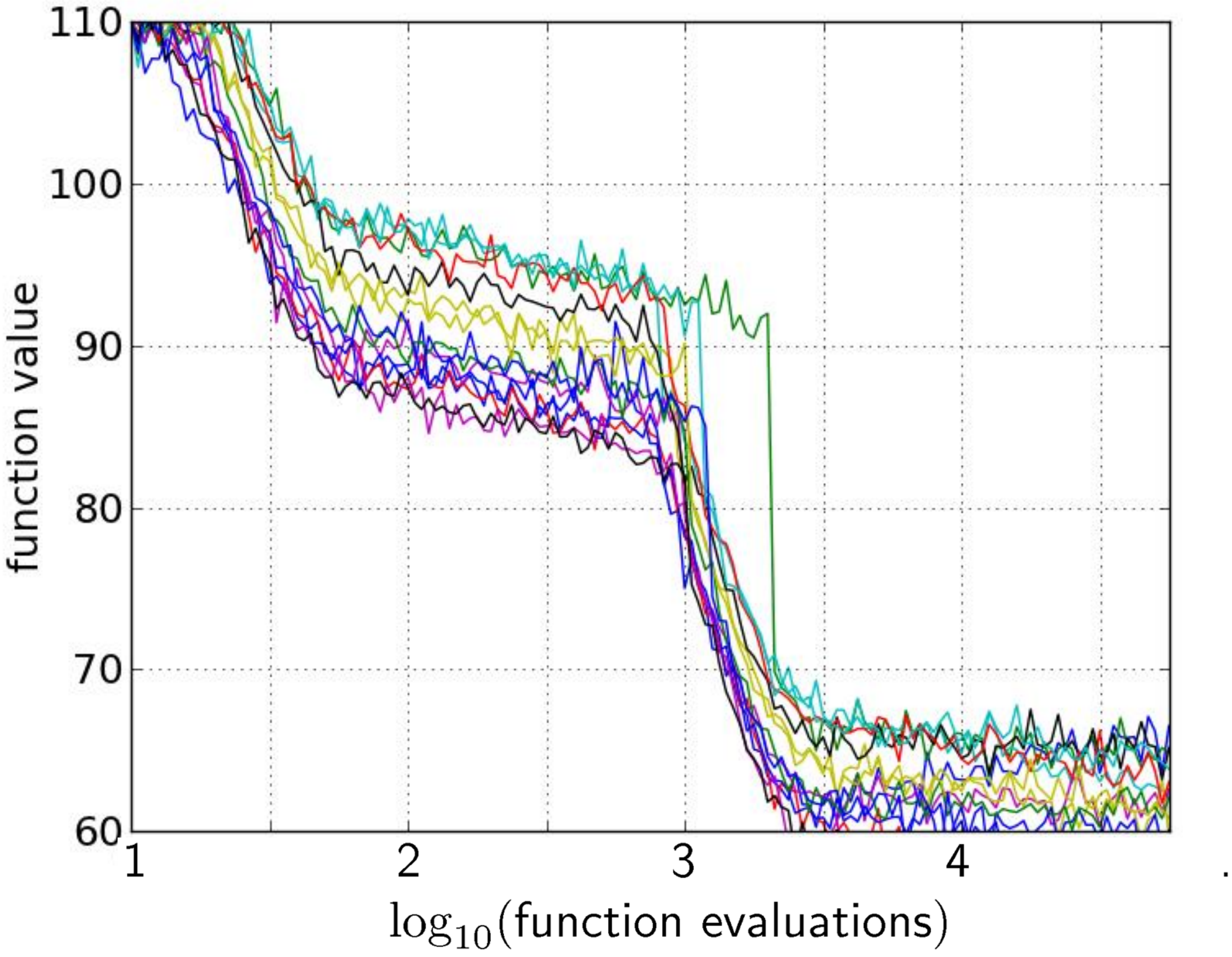
Empirical Cumulative Distribution



interpretations possible:

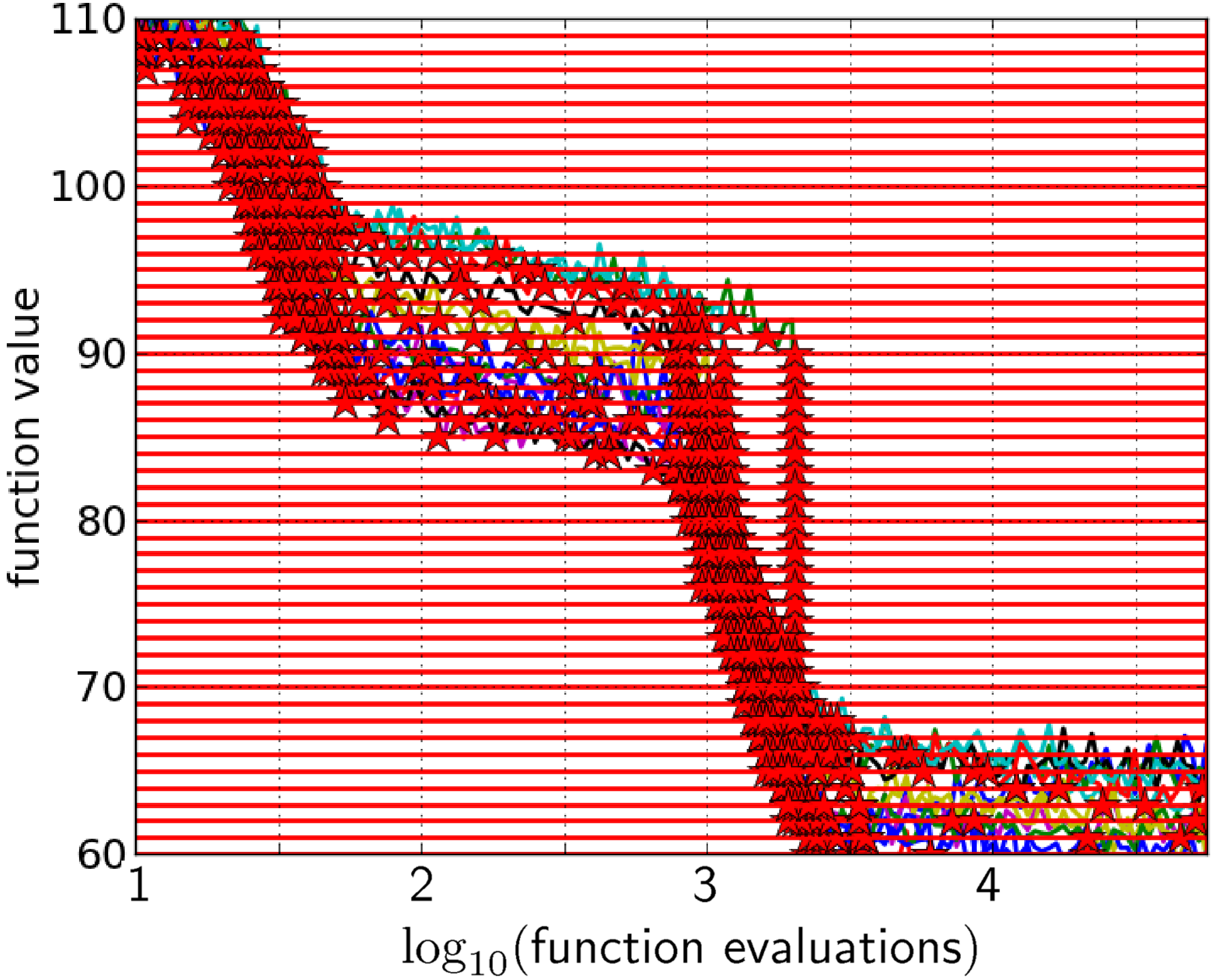
- 80% of the runs reached the target
- e.g. 60% of the runs need between 2000 and 4000 evaluations

Aggregation



15 runs

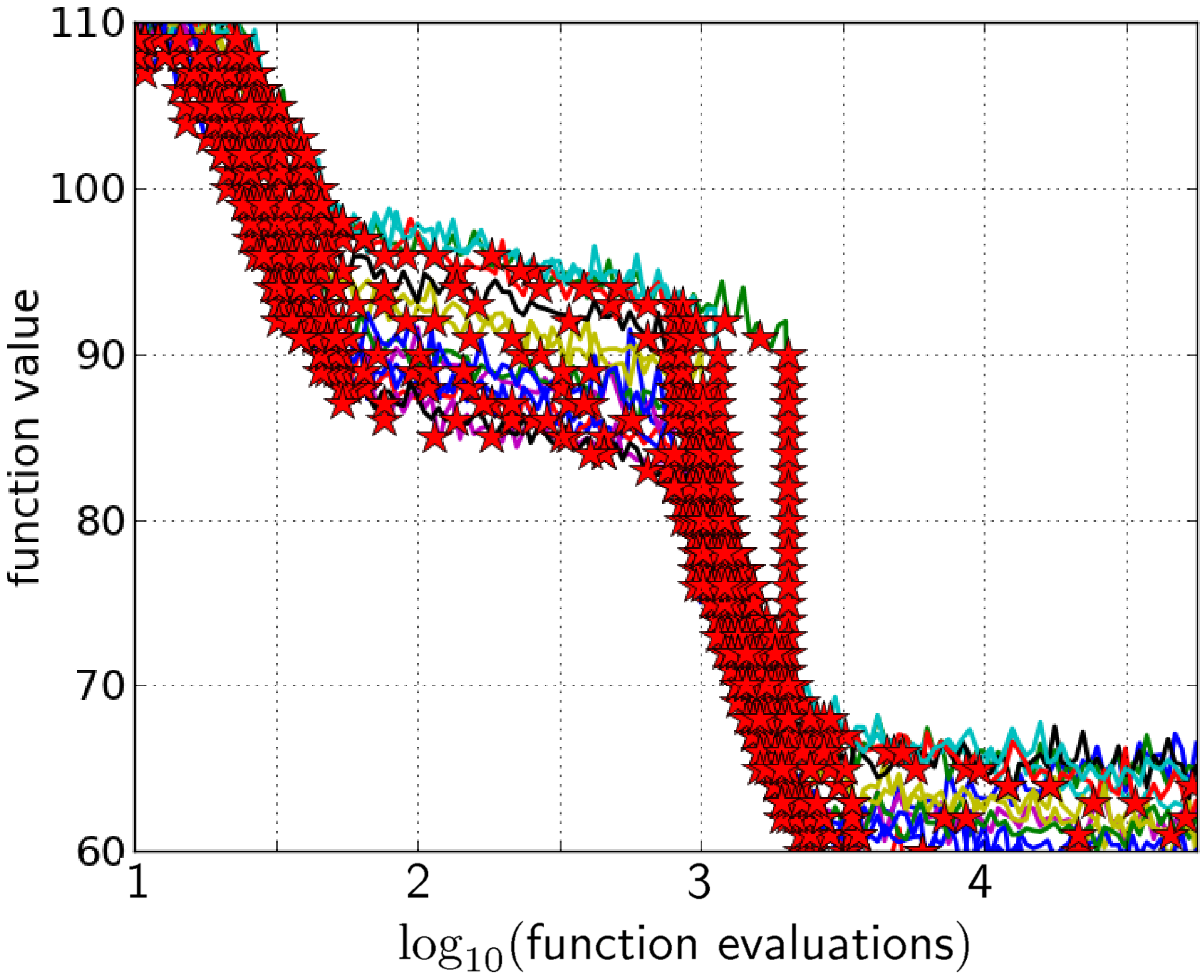
Aggregation



15 runs

50 targets

Aggregation

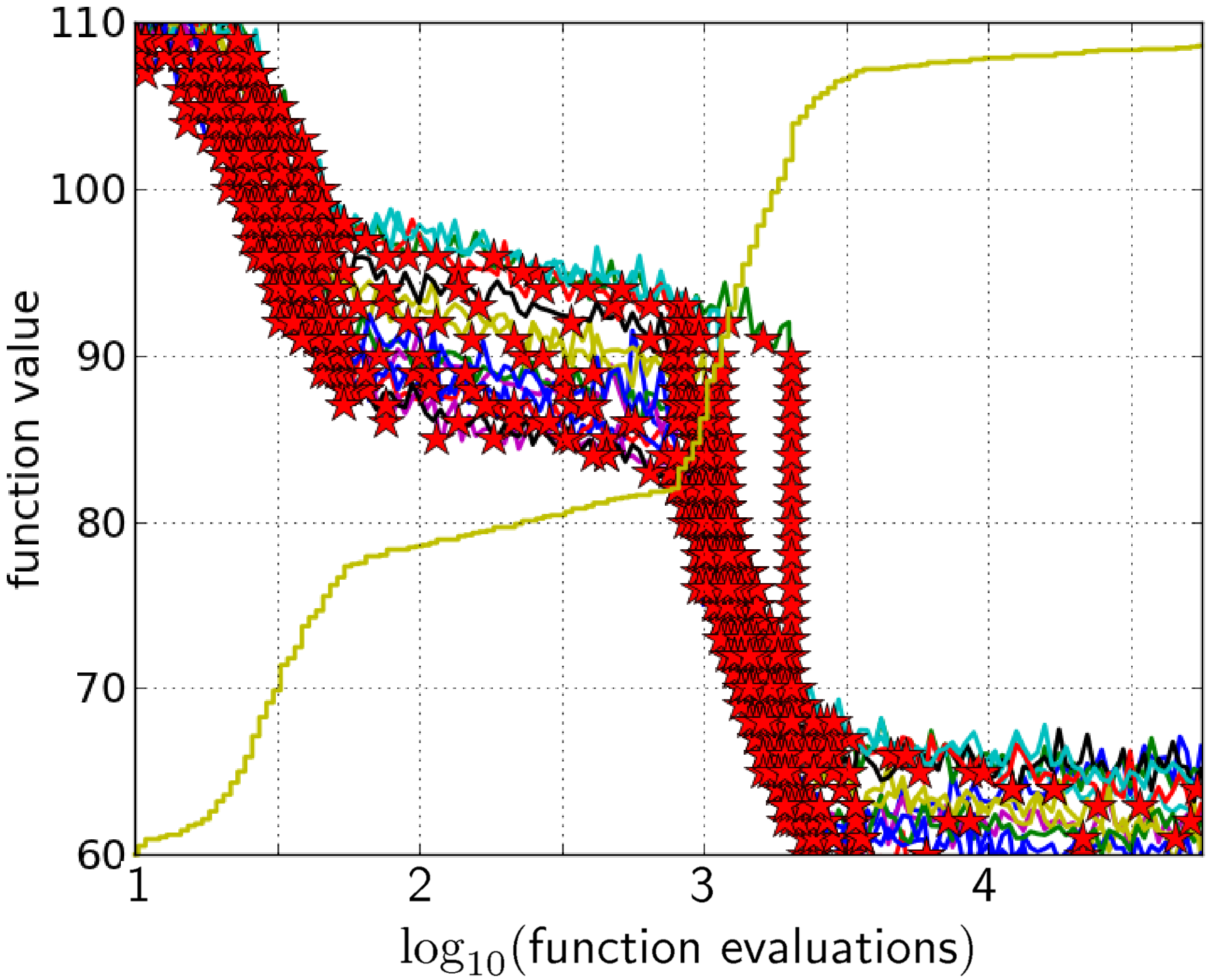


15 runs

50 targets

.

Aggregation

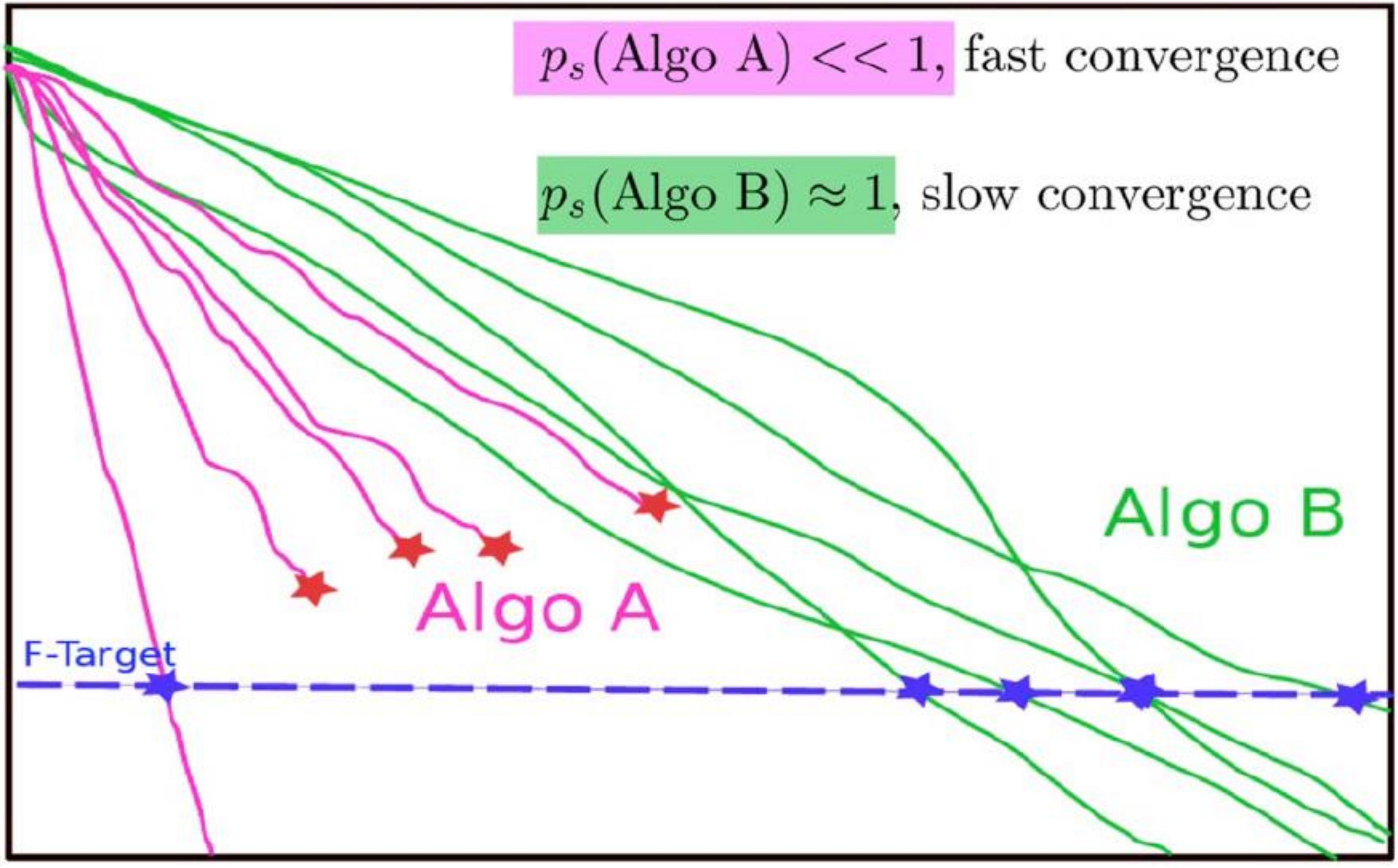


15 runs

50 targets

ECDF with 750 steps

Fixed-target: Measuring Runtime

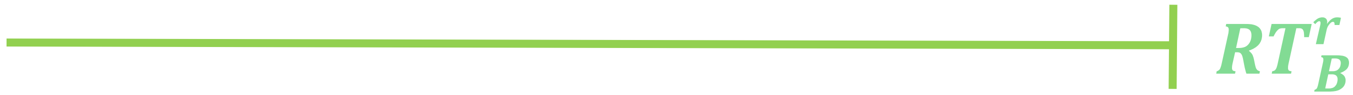


Fixed-target: Measuring Runtime

- Algo Restart A:

p_s(Algo Restart A) = 1

- Algo Restart B:



p_s(Algo Restart B) = 1

Fixed-target: Measuring Runtime

- Expected running time of the restarted algorithm:

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

- Estimator average running time (aRT):

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

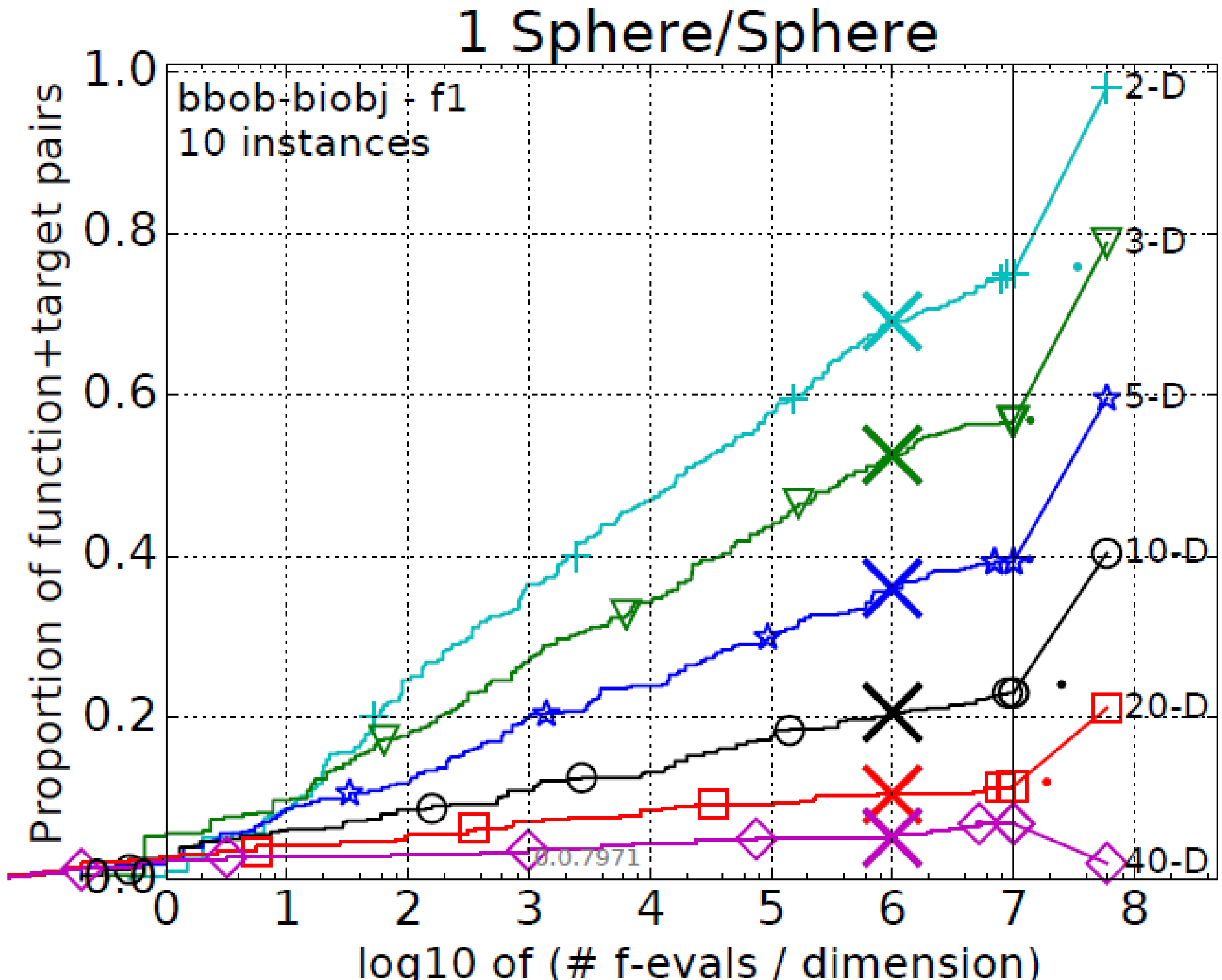
$$\widehat{RT_{unsucc}} = \text{Average evals of unsuccessful runs}$$

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

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

ECDFs with Simulated Restarts

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



Exercise (Part 2)

Objectives:

- investigate the performance of algorithms, available at http://coco.gforge.inria.fr
  - CMA-ES ("IPOP-CMA-ES" version)
  - CMA-ES ("BIPOP-CMA-ES" version)
  - Nelder-Mead simplex (use "NelderDoerr" version here)
  - BFGS quasi-Newton
  - Genetic Algorithm: discretization of cont. variables ("GA")
  - ONEFIFTH: (1+1)-ES with 1/5 rule
- postprocess (now) and investigate the data (after a few more slides)

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

The single-objective BBOB functions

The bbob Testbed

- 24 functions in 5 groups:

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

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

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

Notion of Instances

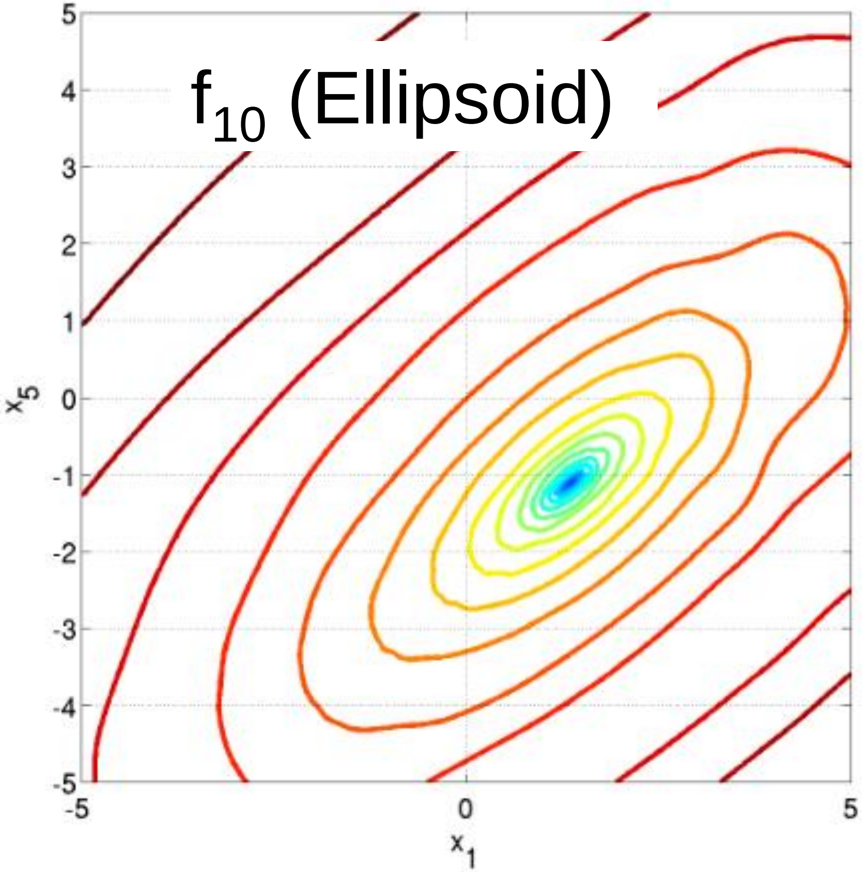
- All COCO problems come in form of instances
    - e.g. as translated/rotated versions of the same function
- Prescribed instances typically change from year to year
    - avoid overfitting
    - 5 instances are always kept the same

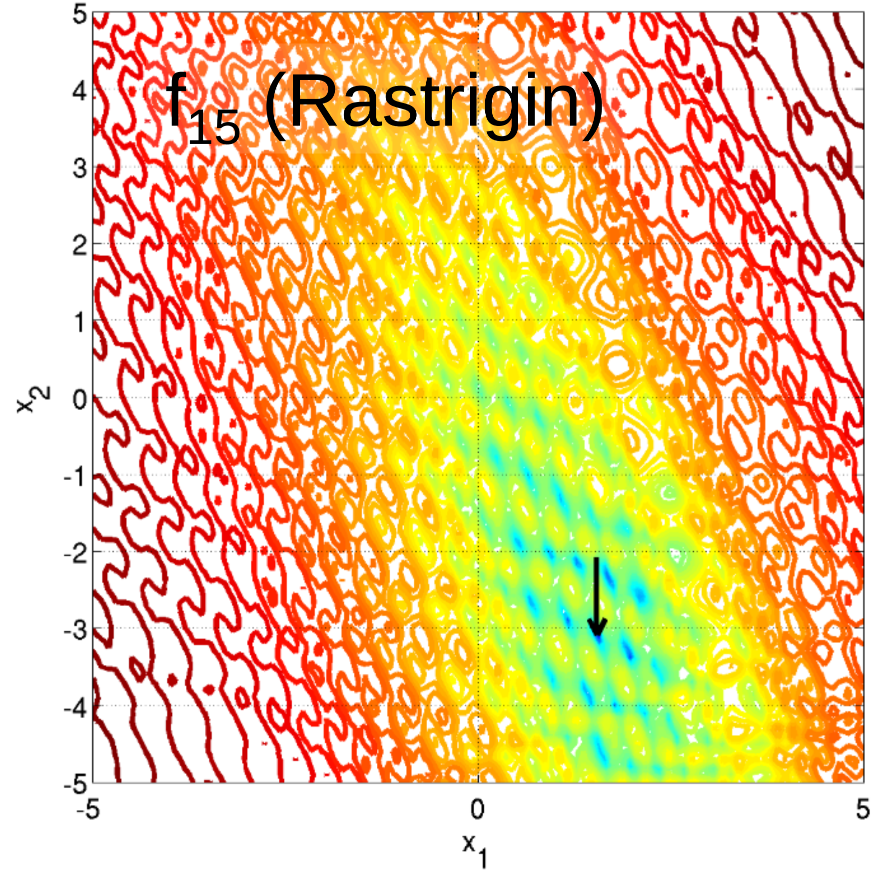
Plus:

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

Notion of Instances

• All COCO problems come in form of instances





linear transformations

Exercise (Part 3)

Objective:

investigate the data:

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

reminder: open thesis projects

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