Introduction to Blackbox Optimization Benchmarking

(original slides by Nikolaus Hansen)

Black-Box Optimization (Search)

Minimize an objective function (also: cost, loss, error, or fitness function)

$$f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}, \quad x \mapsto f(x)$$

in a black-box scenario (direct search, no gradients)

$$x \longrightarrow \blacksquare \longrightarrow f(x)$$

where the black box can be

- non-linear, non-convex, discontinuous, dynamic, stochastic
- from milli-seconds to hours to evaluate

Objective:

- convergence to a global essential infimum of f as fast as possible
- (informally, time-finite) find $x \in \mathcal{X}$ with small $f(x)$ value using as few back-box calls (function evaluations) as possible

Why Do We Want to Evaluate Optimizers?

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

How to Evaluate Algorithms

We can measure performance on

- real world problems
  - expensive
  - comparison is typically limited to certain domains
  - experts have limited interest to publish
- "artificial" benchmark functions
  - cheap
  - controlled
  - data acquisition is comparatively easy
  - problem of representativity
- caveat: parameter of algorithms

How to Evaluate Search Algorithms

we need

- Meaningful quantitative measure on benchmark functions or real world problems
- Account for meta-parameter tuning

  tuning, if needed, can be quite expensive

- Account for invariance properties

  prediction of performance is based on “similarity”, ideally equivalence classes of functions

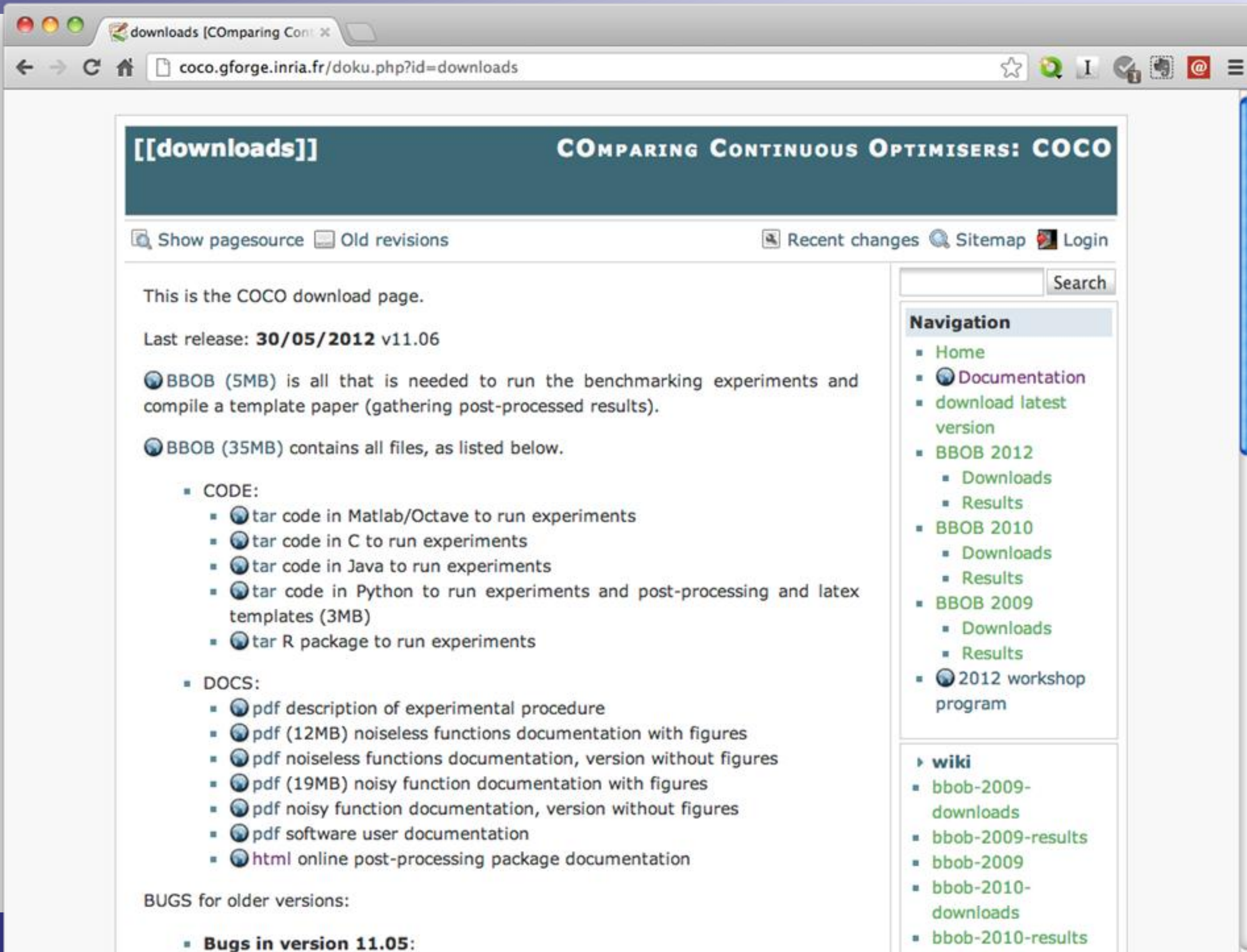
- Account for algorithm internal costs

  often negligible, depending on the objective function cost

BBOB in Practice (using COCO)

COCO (COmparing Continuous Optimizers): a tool for black-box optimization benchmarking

BBOB in practice



downloads [COmparing Con

coco.gforge.inria.fr/doku.php?id=downloads

[[downloads]] COmparing Continuous Optimisers: COCO

Show pagesource Old revisions Recent changes Sitemap Login

Search

This is the COCO download page.

Last release: **30/05/2012** v11.06

BBOB (5MB) is all that is needed to run the benchmarking experiments and compile a template paper (gathering post-processed results).

BBOB (35MB) contains all files, as listed below.

- CODE:
  - tar code in Matlab/Octave to run experiments
  - tar code in C to run experiments
  - tar code in Java to run experiments
  - tar code in Python to run experiments and post-processing and latex templates (3MB)
  - tar R package to run experiments
- DOCS:
  - pdf description of experimental procedure
  - pdf (12MB) noiseless functions documentation with figures
  - pdf noiseless functions documentation, version without figures
  - pdf (19MB) noisy function documentation with figures
  - pdf noisy function documentation, version without figures
  - pdf software user documentation
  - html online post-processing package documentation

BUGS for older versions:

- **Bugs in version 11.05**:

Navigation

- Home
- Documentation
- download latest version
- BBOB 2012
  - Downloads
  - Results
- BBOB 2010
  - Downloads
  - Results
- BBOB 2009
  - Downloads
  - Results
- 2012 workshop program

wiki

- bbob-2009-downloads
- bbob-2009-results
- bbob-2009
- bbob-2010-downloads
- bbob-2010-results

BBOB in practice

Name	Date Modified	Size	Kind
bbob.v11.06	Today, 1:15	--	Folder
c	May 30, 2012 12:07	--	Folder
docs	October 27, 2012 0:58	--	Folder
java	May 30, 2012 12:07	--	Folder
latextemplates	May 30, 2012 12:06	--	Folder
matlab	May 30, 2012 12:06	--	Folder
python	May 30, 2012 12:06	--	Folder
r	May 30, 2012 12:07	--	Folder

Macintosh HD ▸ Users ▸ hansen ▸ Downloads ▸ bbob.v11.06

BBOB in practice

Name	Date Modified	Size	Kind
bbob.v11.06	Today, 1:15	--	Folder
c	May 30, 2012 12:07	--	Folder
docs	October 27, 2012 0:58	--	Folder
java	May 30, 2012 12:07	--	Folder
latextemplates	May 30, 2012 12:06	--	Folder
matlab	May 30, 2012 12:06	--	Folder
benchmarkinfos.txt	February 9, 2009 16:29	4 KB	Gedit …ument
benchmarks.m	February 10, 2011 16:24	86 KB	Objec…rce File
benchmarksnoisy.m	February 10, 2011 16:24	102 KB	Objec…rce File
exampleexperiment.m	February 1, 2012 19:52	4 KB	Objec…rce File
exampletiming.m	March 7, 2012 14:40	4 KB	Objec…rce File
fgeneric.m	December 7, 2011 17:56	33 KB	Objec…rce File
LICENSE.txt	February 1, 2012 20:23	4 KB	Gedit …ument
MY_OPTIMIZER.m	February 14, 2011 19:30	4 KB	Objec…rce File
README.txt	May 19, 2011 10:47	4 KB	Gedit …ument
python	May 30, 2012 12:06	--	Folder
r	May 30, 2012 12:07	--	Folder

Macintosh HD ▸ Users ▸ hansen ▸ Downloads ▸ bbob.v11.06

Matlab script (`exampleexperiment.m`):

```
dimensions = [2, 3, 5, 10, 20, 40];  % small dimensions first, for CPU reasons
functions = benchmarks('FunctionIndices');  % or benchmarksnoisy(...)
instances = [1:5, 31:40];  % 15 function instances

for dim = dimensions
  for ifun = functions
    for iinstance = instances
      fgeneric('initialize', ifun, iinstance, datapath, opt);
      MY_OPTIMIZER('fgeneric', dim, fgeneric('ftarget'), eval(maxfunevals) - f
      disp(sprintf(['  f%d in %d-D, instance %d: FEs=%d with %d restarts, fbes
      fgeneric('finalize');
    end
    disp(['      date and time: ' num2str(clock, ' %.0f')]);
  end
  disp(sprintf('---- dimension %d-D done ----', dim));
end
```

Interface: MY_OPTIMIZER(function_name, dimension, optional_args)

BBOB in practice

Running the experiment at an OS shell:

```
$ nohup nice octave < exampleexperiment.m > output.txt &
$ less output.txt
```

```
GNU Octave, version 3.6.3
Copyright (C) 2012 John W. Eaton and others.
This is free software; see the source code for copying conditions.
[...]
Read http://www.octave.org/bugs.html to learn how to submit bug reports.

For information about changes from previous versions, type `news'.

  f1 in 2-D, instance 1: FEs=242, fbest-ftarget=-8.1485e-10, elapsed time [h]: 0.00
  f1 in 2-D, instance 2: FEs=278, fbest-ftarget=-6.0931e-09, elapsed time [h]: 0.00
  f1 in 2-D, instance 3: FEs=242, fbest-ftarget=-9.2281e-09, elapsed time [h]: 0.00
  f1 in 2-D, instance 4: FEs=302, fbest-ftarget=-4.5997e-09, elapsed time [h]: 0.00
  f1 in 2-D, instance 5: FEs=230, fbest-ftarget=-9.8350e-09, elapsed time [h]: 0.00
  f1 in 2-D, instance 6: FEs=284, fbest-ftarget=-7.0829e-09, elapsed time [h]: 0.00
  f1 in 2-D, instance 7: FEs=278, fbest-ftarget=-6.5999e-09, elapsed time [h]: 0.00
  f1 in 2-D, instance 8: FEs=272, fbest-ftarget=-8.7044e-09, elapsed time [h]: 0.00
  f1 in 2-D, instance 9: FEs=248, fbest-ftarget=-2.6316e-09, elapsed time [h]: 0.00
  f1 in 2-D, instance 10: FEs=302, fbest-ftarget=-4.6779e-09, elapsed time [h]: 0.00
  f1 in 2-D, instance 11: FEs=272, fbest-ftarget=-5.1499e-09, elapsed time [h]: 0.00
[...]
      date and time: 2013 3 29 19 59 26
  f2 in 2-D, instance 1: FEs=824, fbest-ftarget=-7.0206e-09, elapsed time [h]: 0.00
  f2 in 2-D, instance 2: FEs=572, fbest-ftarget=-9.2822e-09, elapsed time [h]: 0.00
[...]
```

BBOB in practice

Name	Date Modified	Size	Kind
bbob.v13.05	March 8, 2013 13:04	--	Folder
c	March 5, 2013 23:04	--	Folder
docs	March 6, 2013 13:56	--	Folder
java	March 5, 2013 23:04	--	Folder
latextemplates	March 5, 2013 23:04	--	Folder
matlab	March 5, 2013 23:02	--	Folder
python	Today, 20:08	--	Folder
bbob_pproc	March 5, 2013 23:03	--	Folder
bbobbenchmarks.py	November 12, 2012 16:56	74 KB	Python script
benchmarkinfos.txt	February 9, 2009 16:29	4 KB	Gedit ...ument
exampleexperiment.py	February 22, 2013 14:26	4 KB	Python script
exampletiming.py	November 12, 2012 16:56	4 KB	Python script
fgeneric.py	March 3, 2013 19:33	25 KB	Python script
LICENSE.txt	February 1, 2012 20:23	4 KB	Gedit ...ument
README.txt	November 12, 2012 16:56	4 KB	Gedit ...ument
r	March 5, 2013 23:05	--	Folder

Macintosh HD ▸ Users ▸ hansen ▸ Downloads ▸ bbob.v13.05 ▸ python ▸ bbob_pproc

BBOB in practice

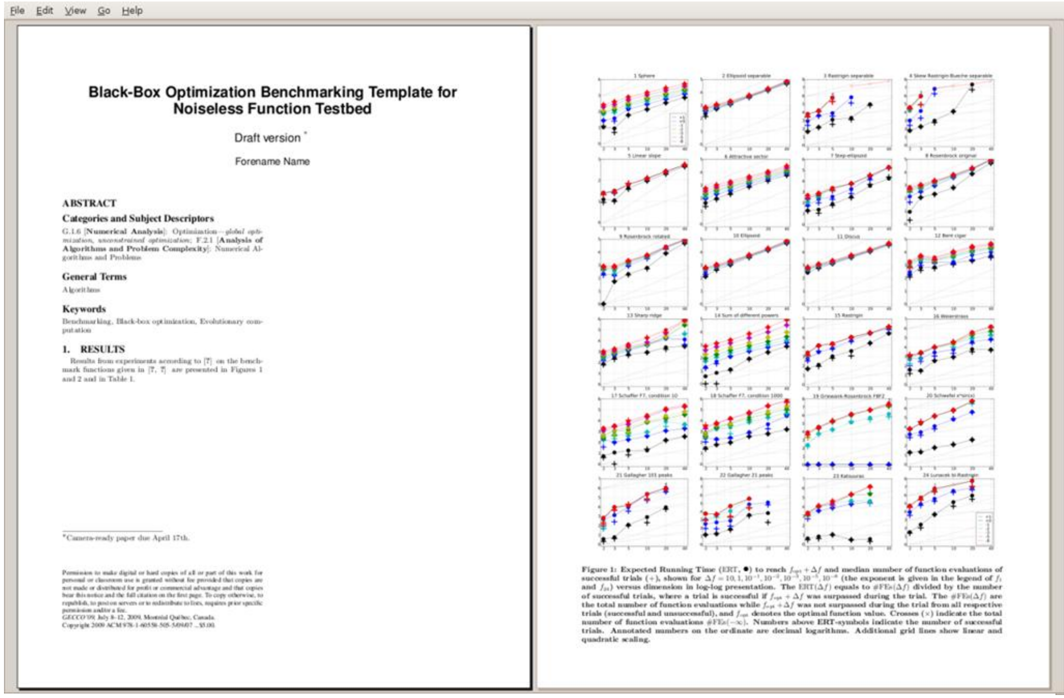
Post-processing at the OS shell:

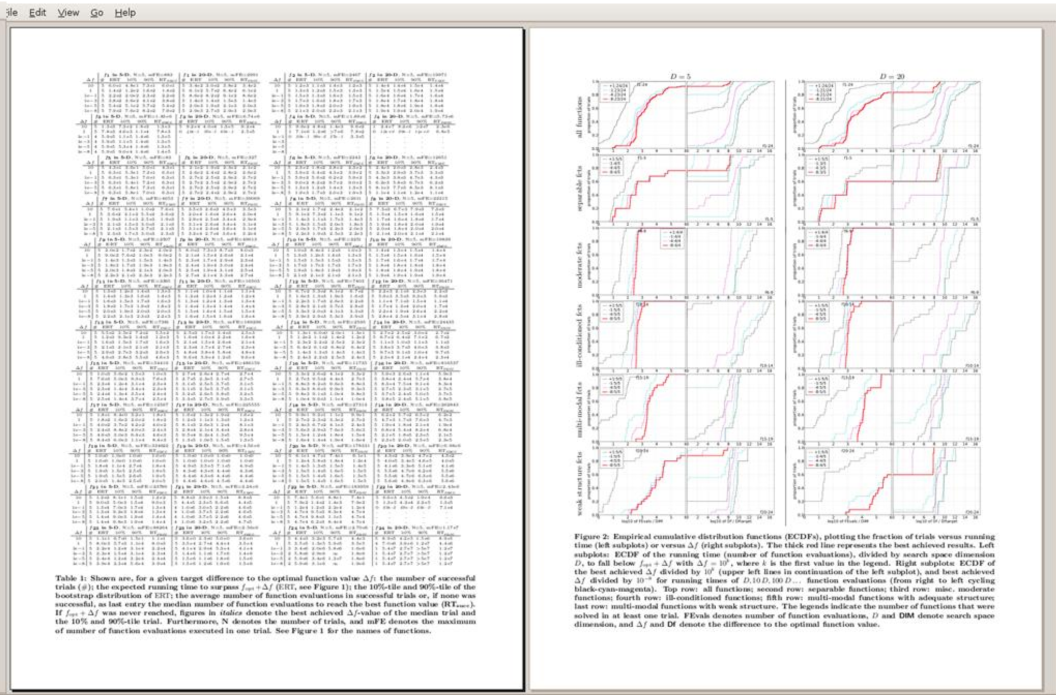
```
$ python codepath/bbob_pproc/rungeneric.py datapath

[...]

$ pdflatex templateACMarticle.tex

[...]
```





e Edit View Go Help

Black-Box Optimization Benchmarking Template for Noiseless Function Testbed

Draft version *

Forename Name

ABSTRACT

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*global optimization, unconstrained optimization*; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems

General Terms

Algorithms

Keywords

Benchmarking, Black-box optimization, Evolutionary computation

1. RESULTS

Results from experiments according to [?] on the benchmark functions given in [?, ?] are presented in Figures 1 and 2 and in Table 1.

*Camera-ready paper due April 17th.

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

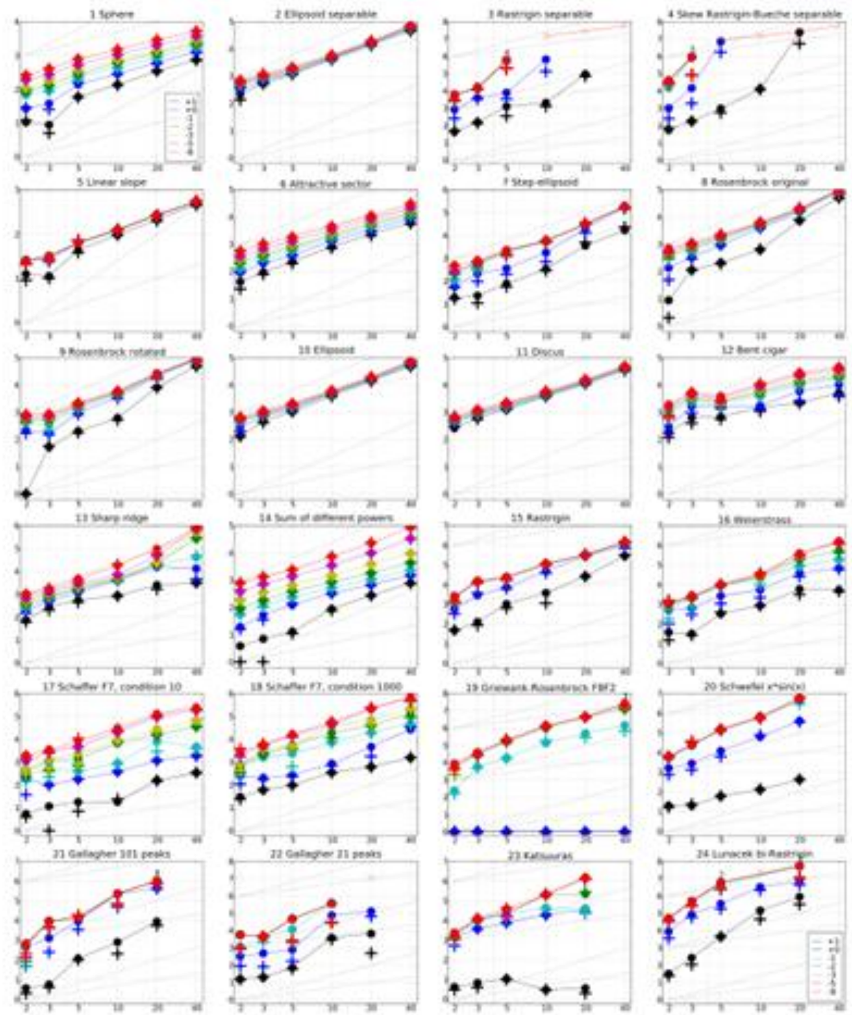


Figure 1: Expected Running Time (ERT, ●) to reach $f_{opt} + \Delta f$ and median number of function evaluations of successful trials (+), shown for $\Delta f = 10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}$ (the exponent is given in the legend of f_1 and f_{24}) versus dimension in log-log presentation. The ERT(Δf) equals to #FEs(Δf) divided by the number of successful trials, where a trial is successful if $f_{opt} + \Delta f$ was surpassed during the trial. The #FEs(Δf) are the total number of function evaluations while $f_{opt} + \Delta f$ was not surpassed during the trial from all respective trials (successful and unsuccessful), and f_{opt} denotes the optimal function value. Crosses (×) indicate the total number of function evaluations #FEs($-\infty$). Numbers above ERT-symbols indicate the number of successful trials. Annotated numbers on the ordinate are decimal logarithms. Additional grid lines show linear and quadratic scaling.

Page 4 of 6

File Edit View Go Help

Table 1: Shown are, for a given target difference to the optimal function value Δf: the number of successful trials (#); the expected running time to surpass $f_{opt} + \Delta f$ (ERT, see Figure 1); the 10%-tile and 90%-tile of the bootstrap distribution of ERT; the average number of function evaluations in successful trials or, if none was successful, as last entry the median number of function evaluations to reach the best function value ($\mathbf{RT}_{succ}$). If $f_{opt} + \Delta f$ was never reached, figures in *italics* denote the best achieved Δf-value of the median trial and the 10% and 90%-tile trial. Furthermore, N denotes the number of trials, and mFE denotes the maximum of number of function evaluations executed in one trial. See Figure 1 for the names of functions.

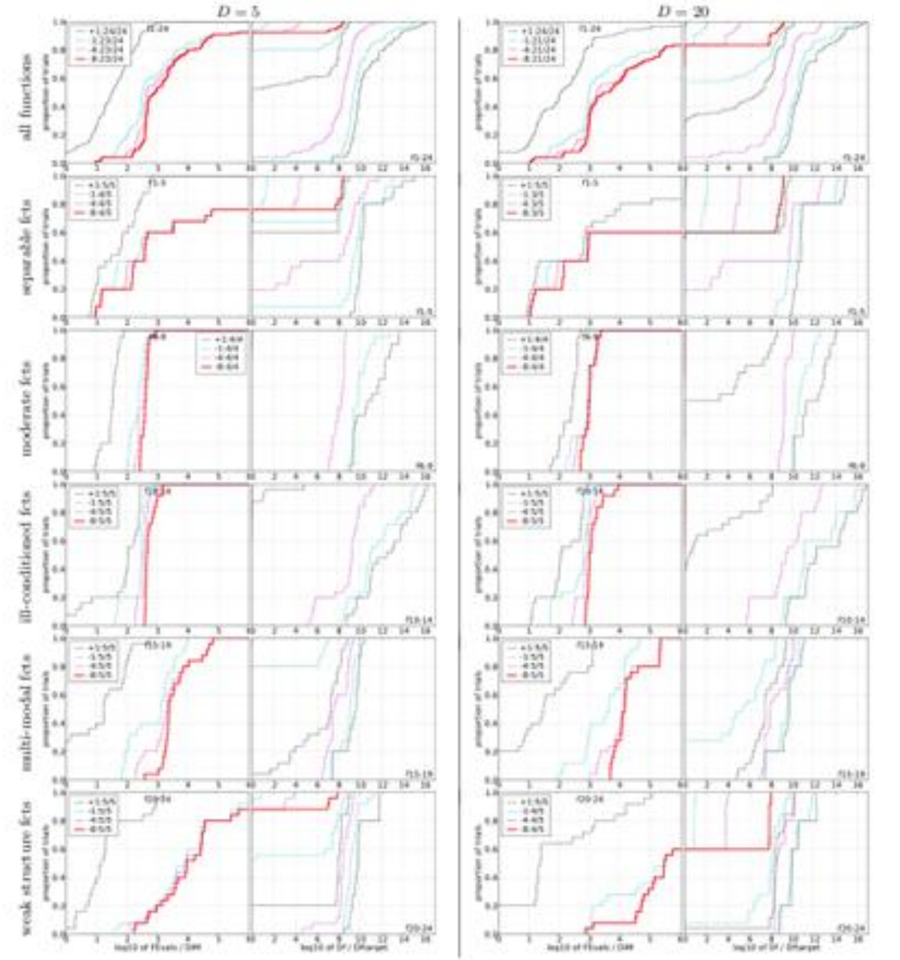


Figure 2: Empirical cumulative distribution functions (ECDFs), plotting the fraction of trials versus running time (left subplots) or versus Δf (right subplots). The thick red line represents the best achieved results. Left subplots: ECDF of the running time (number of function evaluations), divided by search space dimension D, to fall below $f_{opt} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. Right subplots: ECDF of the best achieved Δf divided by 10^k (upper left lines in continuation of the left subplot), and best achieved Δf divided by 10^{-8} for running times of $D, 10\,D, 100\,D \ldots$ function evaluations (from right to left cycling black-cyan-magenta). Top row: all functions; second row: separable functions; third row: misc. moderate functions; fourth row: ill-conditioned functions; fifth row: multi-modal functions with adequate structure; last row: multi-modal functions with weak structure. The legends indicate the number of functions that were solved in at least one trial. FEvals denotes number of function evaluations, D and DIM denote search space dimension, and Δf and Df denote the difference to the optimal function value.

Test Functions

Test Functions

Test functions

- define the "scientific question"

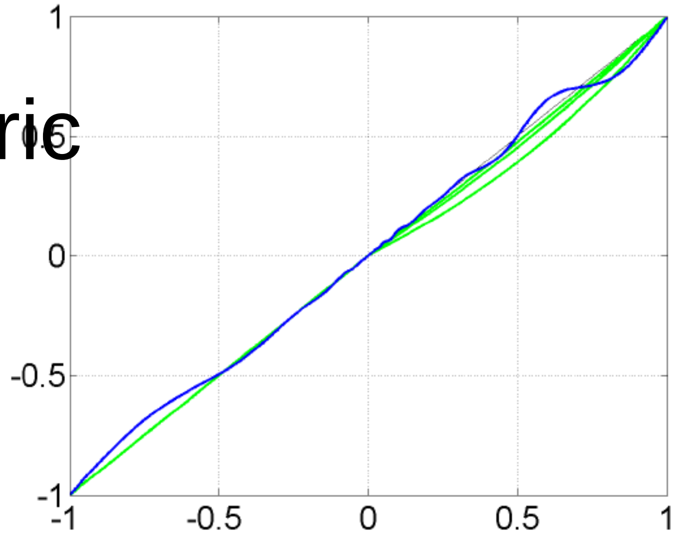
  the relevance can hardly be overestimated

- should represent "reality"
- are often too simple?

  remind separability

- a number of testbeds are around

BBOB: the noiseless functions



functions are not perfectly symmetric and are locally deformed

24 functions within five sub-groups

- Separable functions
- Essential unimodal functions
- Ill-conditioned unimodal functions
- Multimodal structured functions
- Multimodal functions with weak or without structure

GECCO-BBOB

Test Functions

http://coco.gforge.inria.fr

Submitted Data Sets

- 2009: 31 noiseless and 21 noisy “data sets”
- 2010: 24 noiseless and 16 noisy “data sets”
- 2012: 30 noiseless and 4 noisy “data sets”
- 2013: 31 noiseless “data sets“
- 2015: 26 noiseless “data sets”
- Algorithms: RCGAs (e.g. plain, PCX), EDAs (e.g. IDEA), BFGS, NEWUAO, Simplex & (many) other “classical” methods, ESs (e.g. CMA), PSO, DE, Memetic Alg, Ant-Stigmergy Alg, Bee Colony, EGS, SPSA, Meta-Strategies...

How do we measure performance?

Evaluation of Search Algorithms

Behind the scene

a performance measure should be

- quantitative on the ratio scale (highest possible)
  - “algorithm A is two *times* better than algorithm 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)

Measuring Performance

...empirically...

convergence graphs is all we have to start with

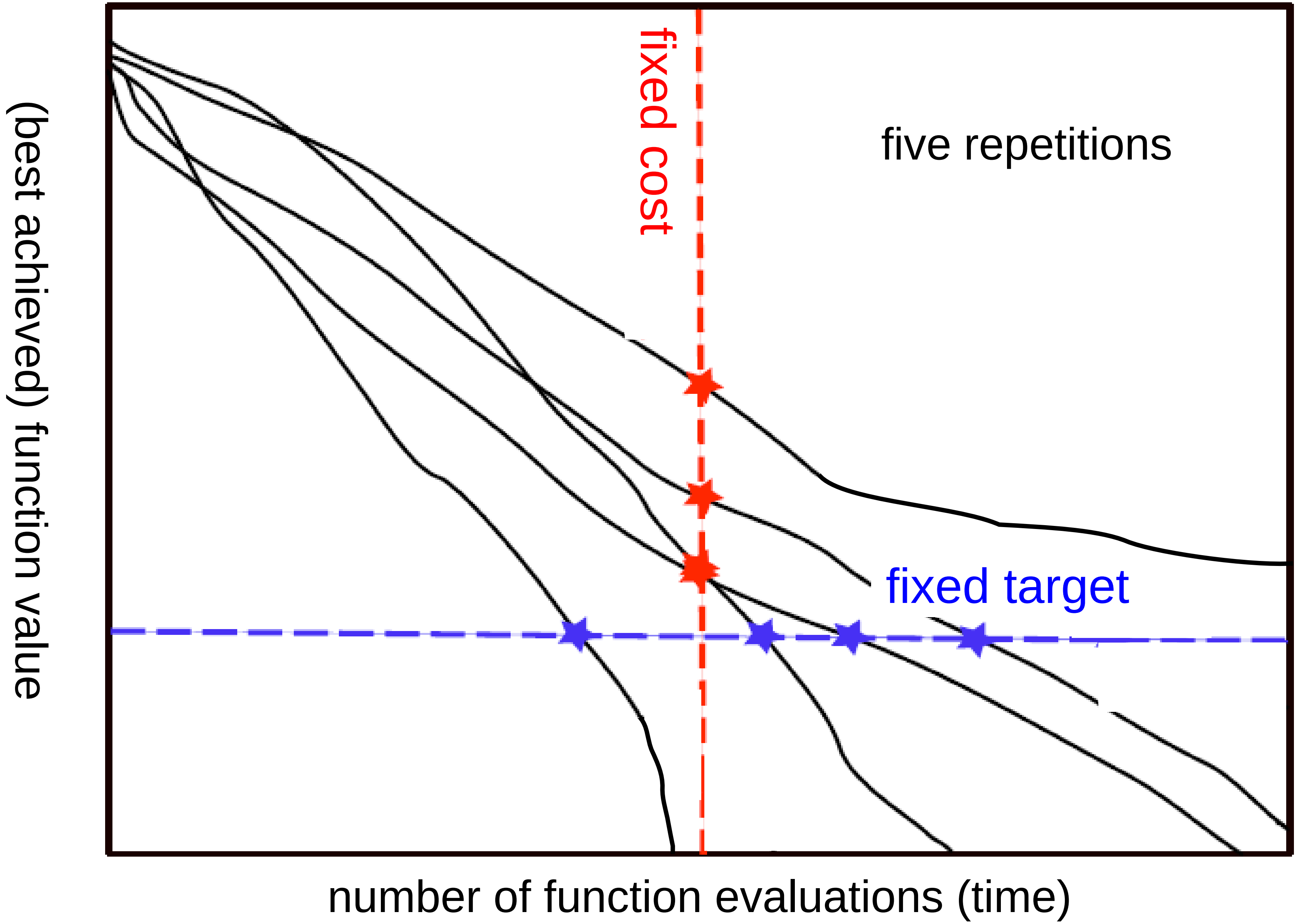
(recall) Black-Box Optimization

Two objectives:

- Find solution with small(est possible) function value
- With the least possible search costs (number of function evaluations)
- For measuring performance: fix one and measure the other

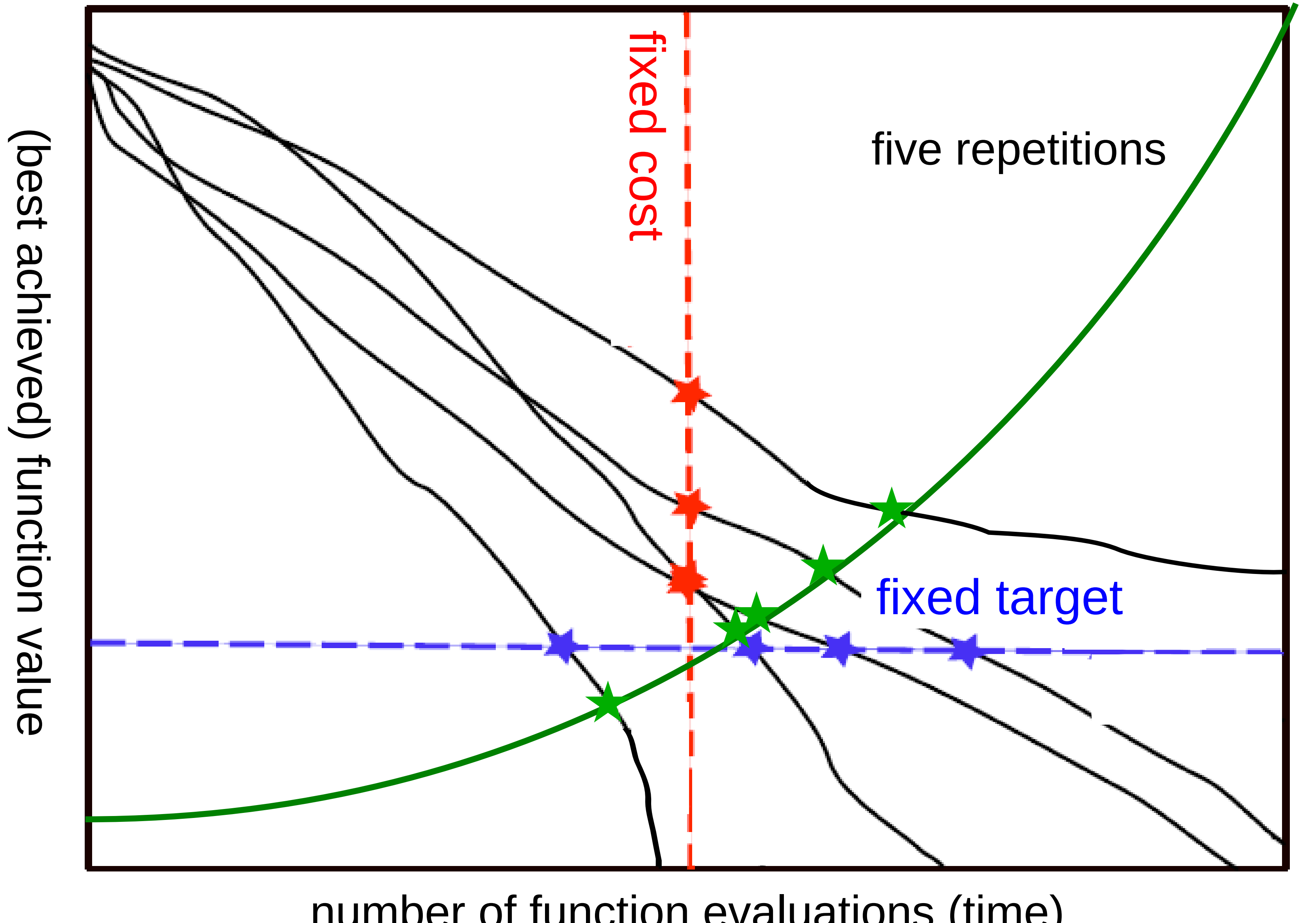
Measuring Performance from Convergence Graphs

fixed-cost versus fixed-target



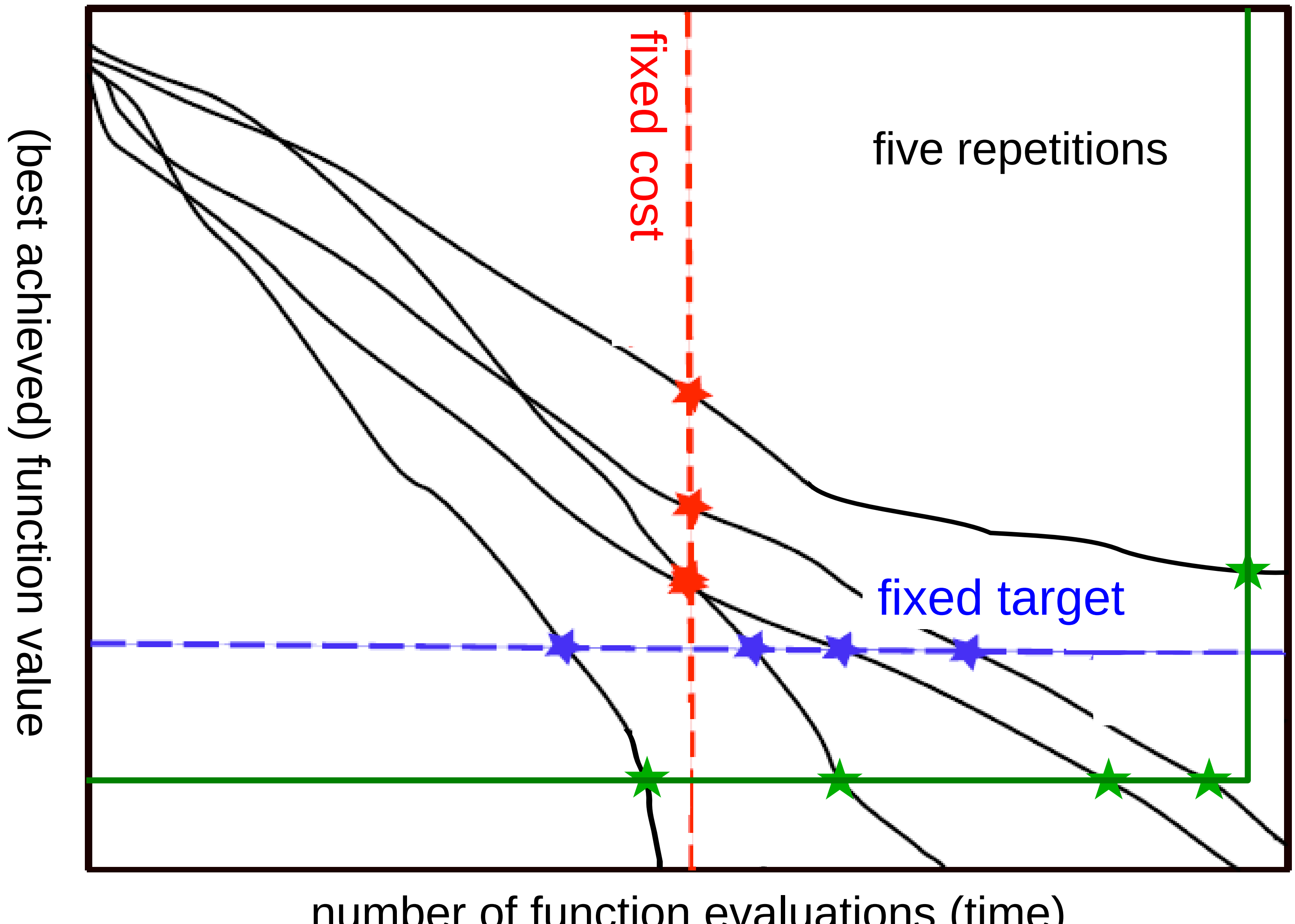
Measuring Performance from Convergence Graphs

fixed-cost versus fixed-target



Measuring Performance from Convergence Graphs

fixed-cost versus fixed-target



The performance measure we use

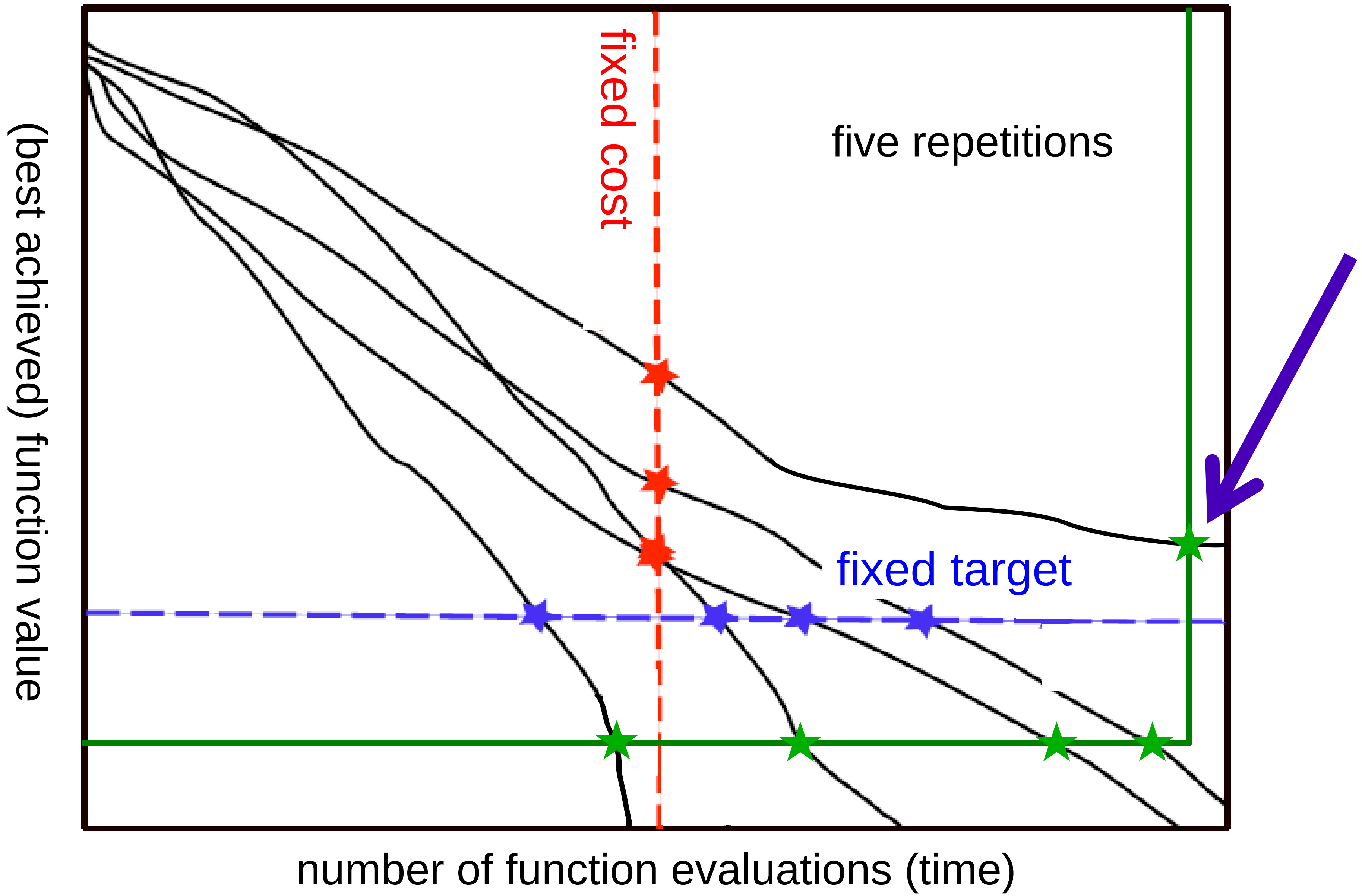
Run length or runtime or first hitting time to a given target function value measured in number of fitness function evaluations

equivalent to first hitting time of a sublevel set in search space

How can we deal with "missing values"?

Measuring Performance from Convergence Graphs

fixed-cost versus fixed-target



Fixed-target: Measuring Runtime

1. Fix a target f-value (most difficult part)

2. Compute the success rate $\widehat{p}$ as

$$\widehat{p} = \frac{\text{\# of successful runs (that reached the target)}}{\text{\# of all runs}} \in [0,1]$$

$$\widehat{R} = \frac{1-\widehat{p}}{\widehat{p}} = \frac{\text{\# of unsuccessful runs}}{\text{\# of successful runs}} \in [0,\infty]$$

$\widehat{R}$ is the odds ratio to be unsuccessful

$\widehat{R}$ is the number of unsuccessful runs observed *for each single successful* run (i.e. normalized by # of successful runs)

Fixed-target: Measuring Runtime

$$\widehat{R} = \frac{1-\widehat{p}}{\widehat{p}} = \frac{\text{\# of unsuccessful runs}}{\text{\# of successful runs}} \in [0, \infty]$$

$\widehat{R}$ is the odds ratio to be unsuccessful

$\widehat{R}$ is the number of unsuccessful runs observed *for each single successful* run (i.e. normalized by # of successful runs)

3. Compute "expected runtime" to hit the target

$$\mathrm{ERT} := \overbrace{\overline{\mathrm{RT}}_{\mathrm{succ}}}^{\text{average runtime for a single successful run}} + \underbrace{\widehat{R} \times \overline{\mathrm{RT}}_{\mathrm{unsucc}}}_{\text{average runtime spent in unsuccessful runs to achieve one successful run}}$$

$\mathrm{SP1} := \overline{\mathrm{RT}}_{\mathrm{succ}} + \widehat{R} \times \overline{\mathrm{RT}}_{\mathrm{succ}} = \overline{\mathrm{RT}}_{\mathrm{succ}}(1+\widehat{R})$ disregarding runlength of unsuccessful runs

if $\widehat{R} < \infty$, else we can assume $\mathrm{ERT} \geq \sum \mathrm{RT}_{\mathrm{unsucc}}$

Fixed-target: Measuring Runtime

$\widehat{R}$ is the number of unsuccessful runs observed *for each single successful* run (i.e. normalized by # of successful runs)

3. Compute "expected runtime" to hit the target

$$\mathsf{ERT} := \overbrace{\overline{\mathsf{RT}}_{\mathrm{succ}}}^{\text{average runtime for a single successful run}} + \underbrace{\widehat{R} \times \overline{\mathsf{RT}}_{\mathrm{unsucc}}}_{\text{average runtime spent in unsuccessful runs to achieve one successful run}}$$

$\mathrm{SP1} := \overline{\mathsf{RT}}_{\mathrm{succ}} + \widehat{R} \times \overline{\mathsf{RT}}_{\mathrm{succ}} = \overline{\mathsf{RT}}_{\mathrm{succ}}(1 + \widehat{R})$ disregarding runlength of unsuccessful runs

We can simulate a single runtime by "restarting" until the first success

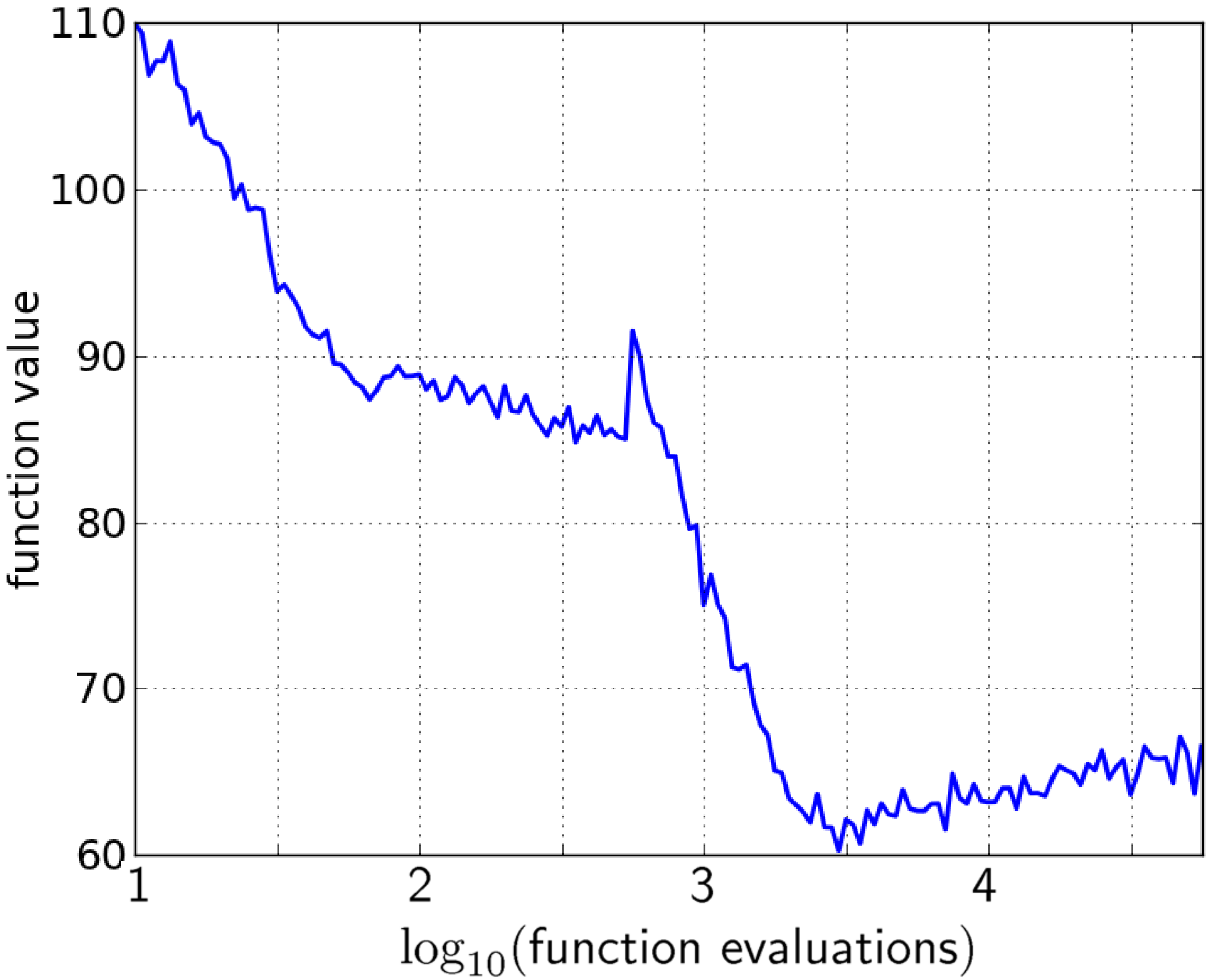
$$\mathsf{RT} = \mathsf{RT}_{\mathrm{succ}} + \sum \mathsf{RT}_{\mathrm{unsucc}}$$

$\Longrightarrow$ distribution of runtimes incorporating unsuccessful runs
$\Longrightarrow$ display the distribution or a statistic of it

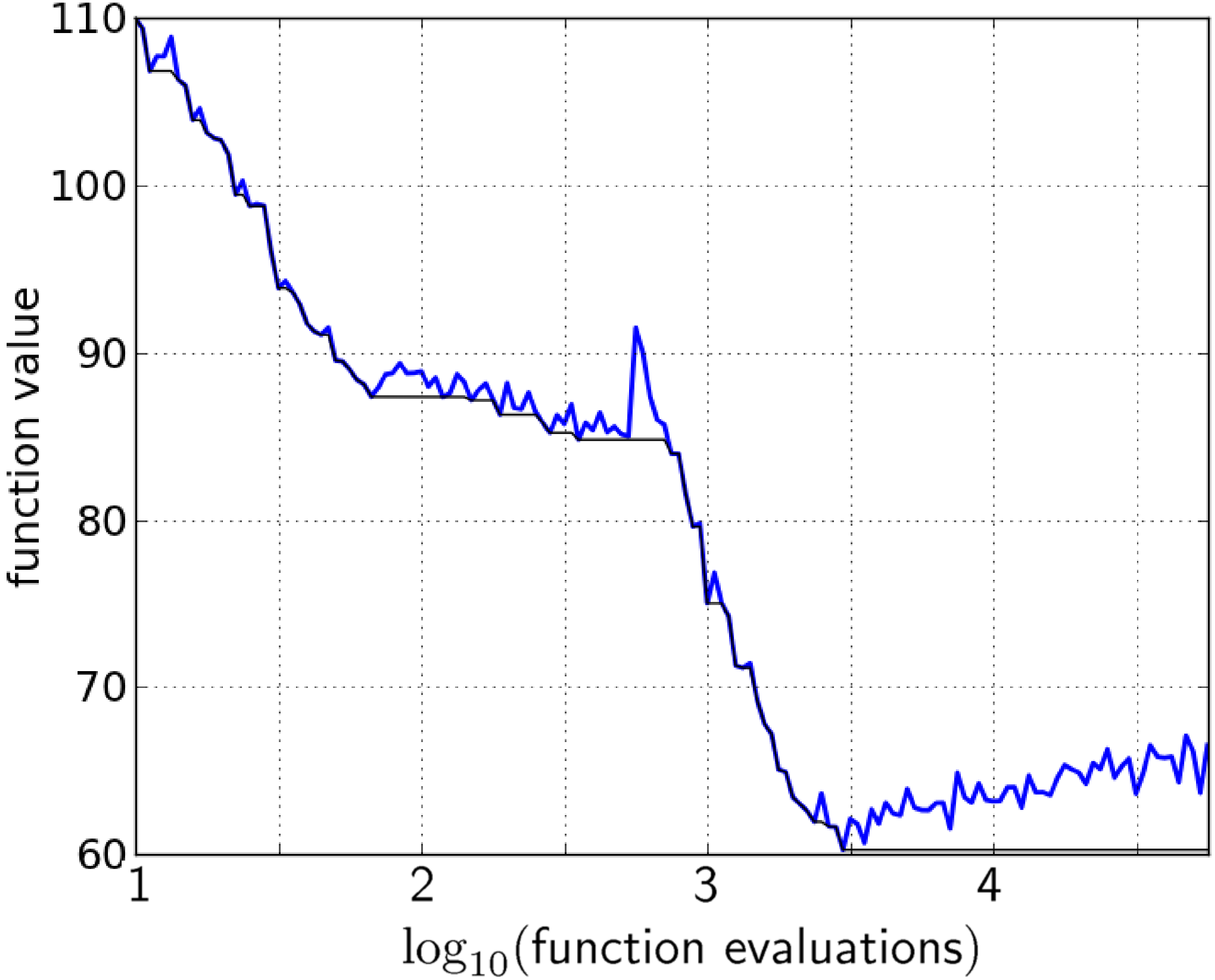
ECDF:

Empirical Cumulative Distribution Function of the Runtime

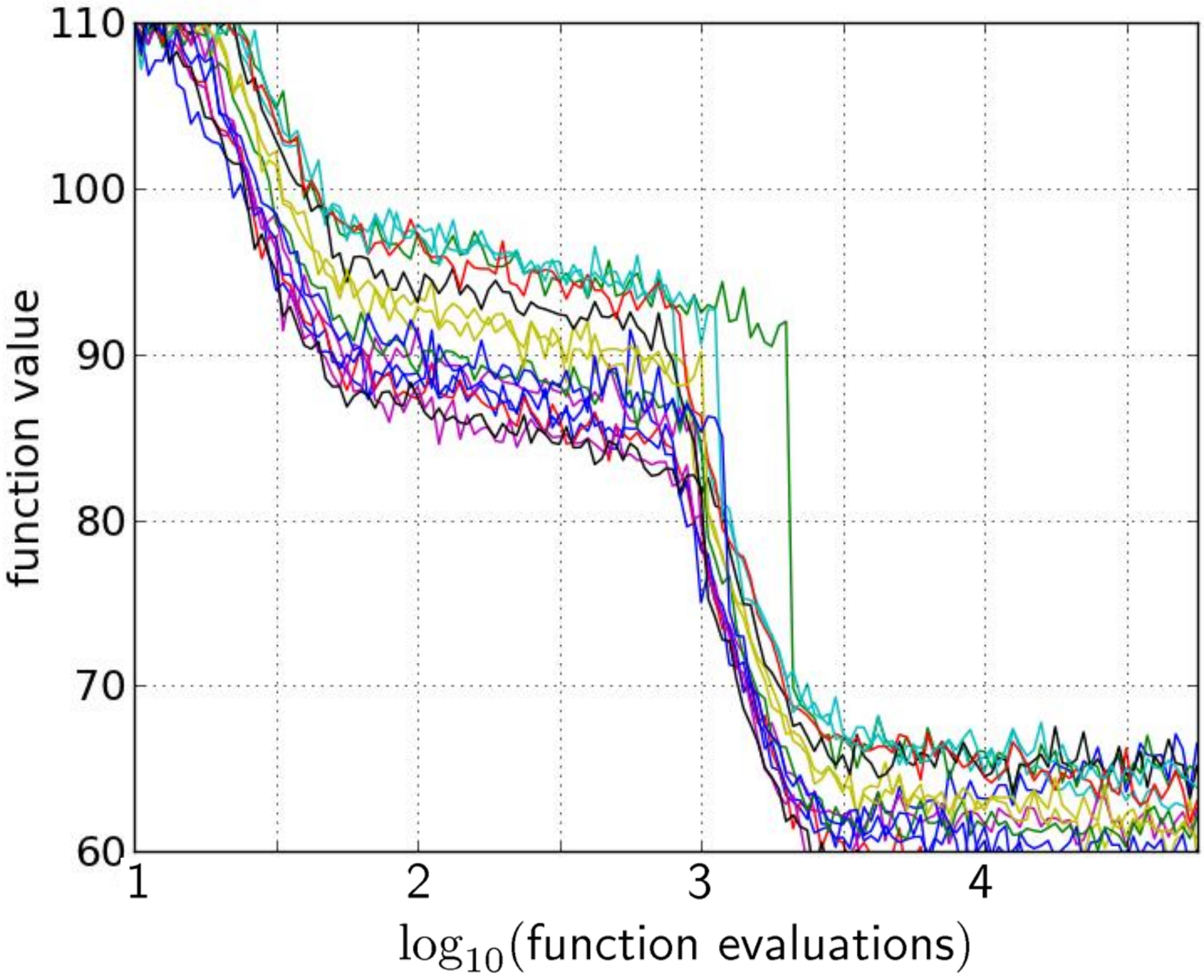
A Convergence Graph



First hitting time is monotonous

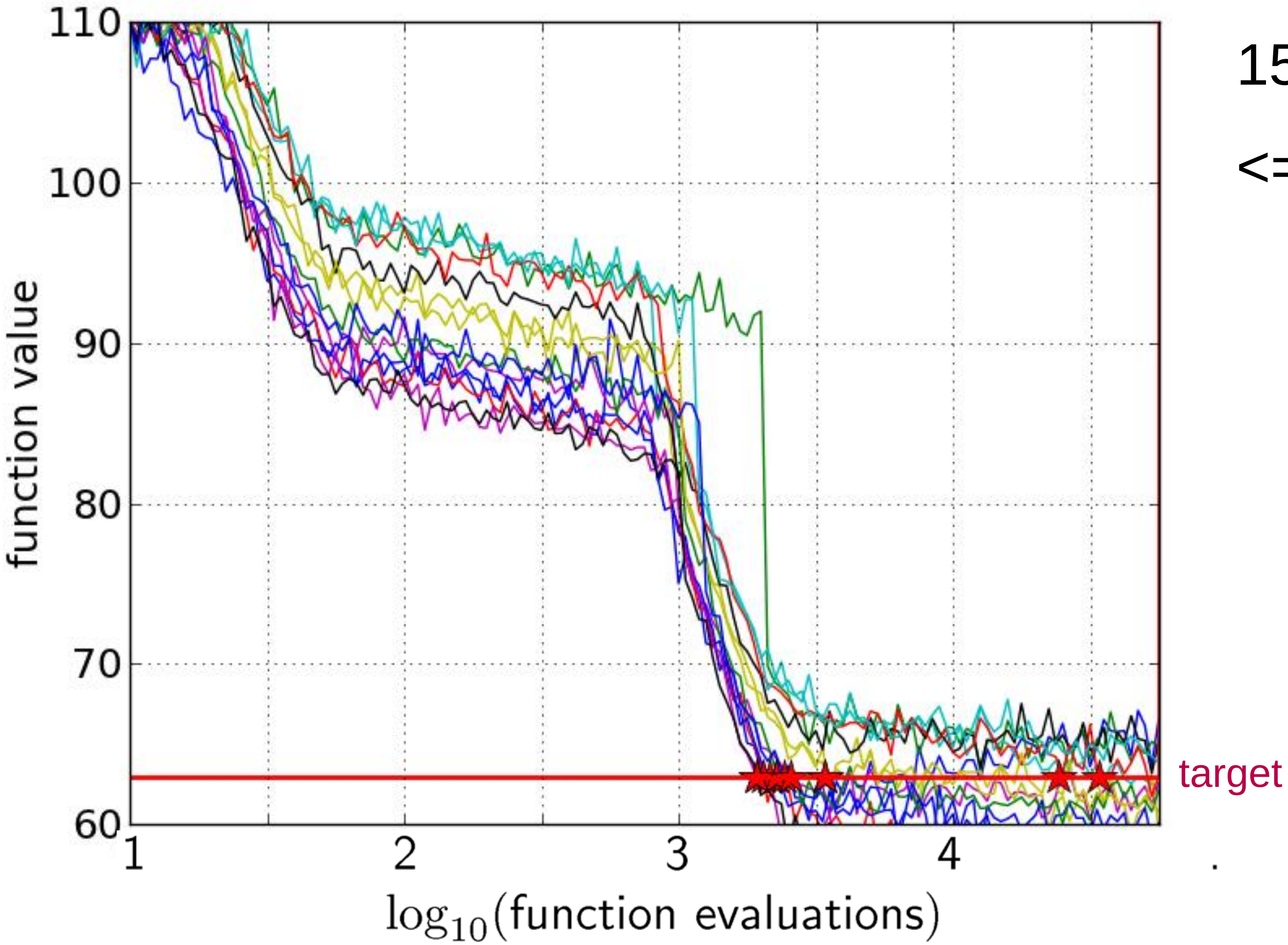


- first hitting time: a monotonous graph



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

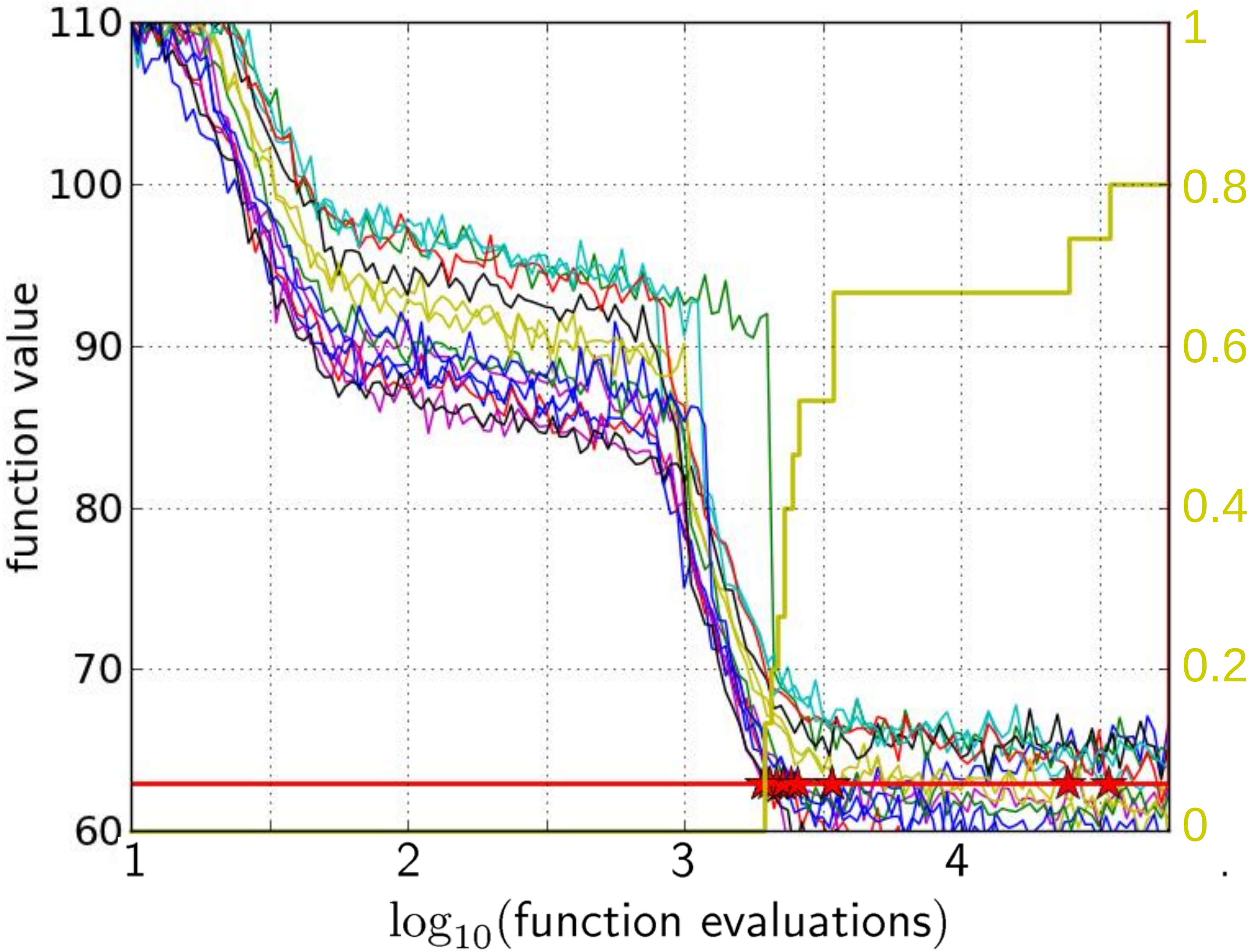
15 runs



15 runs

<=15 first hitting times (runtimes)

Empirical CDF

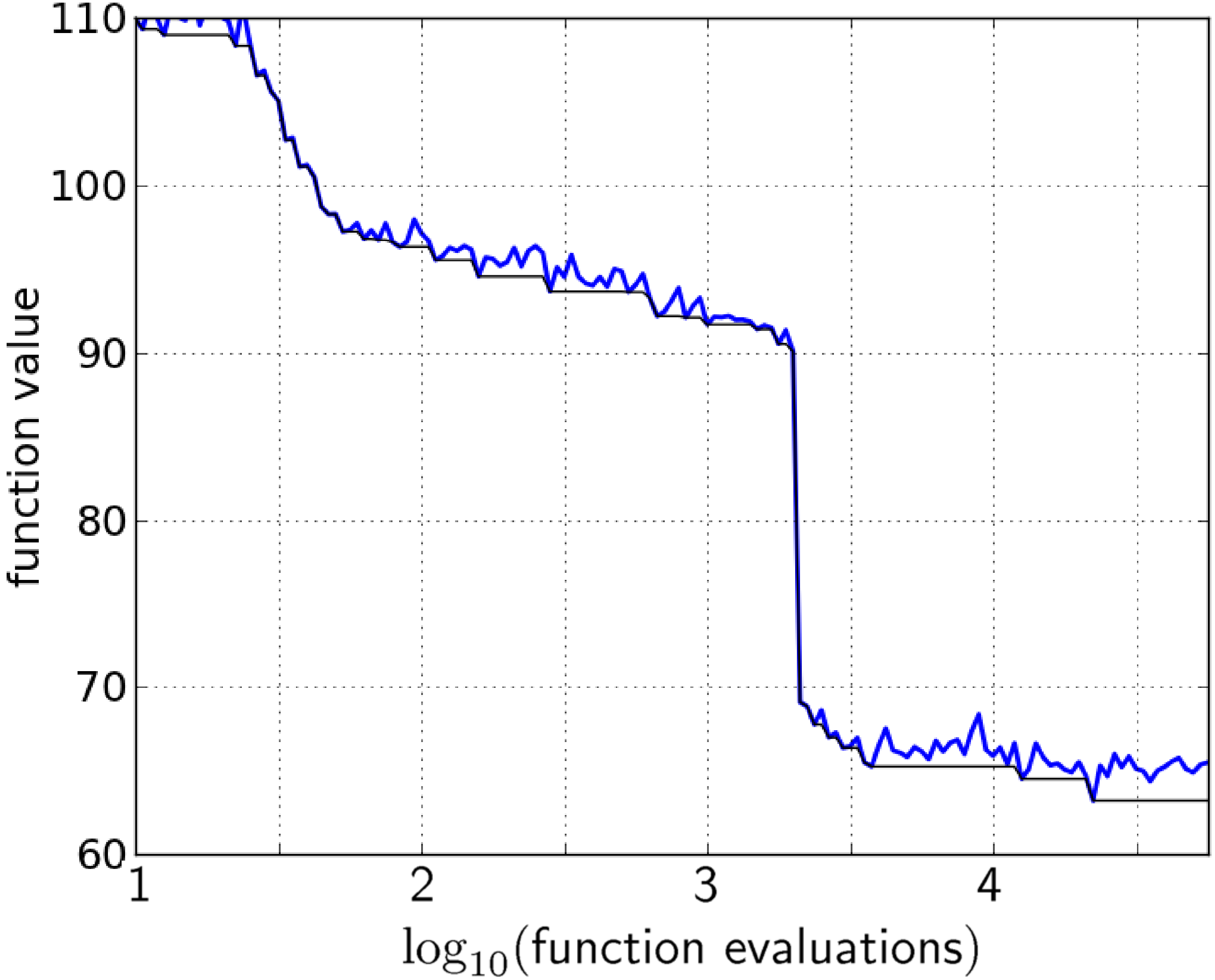


the ECDF of run lengths (runtimes) to reach the target

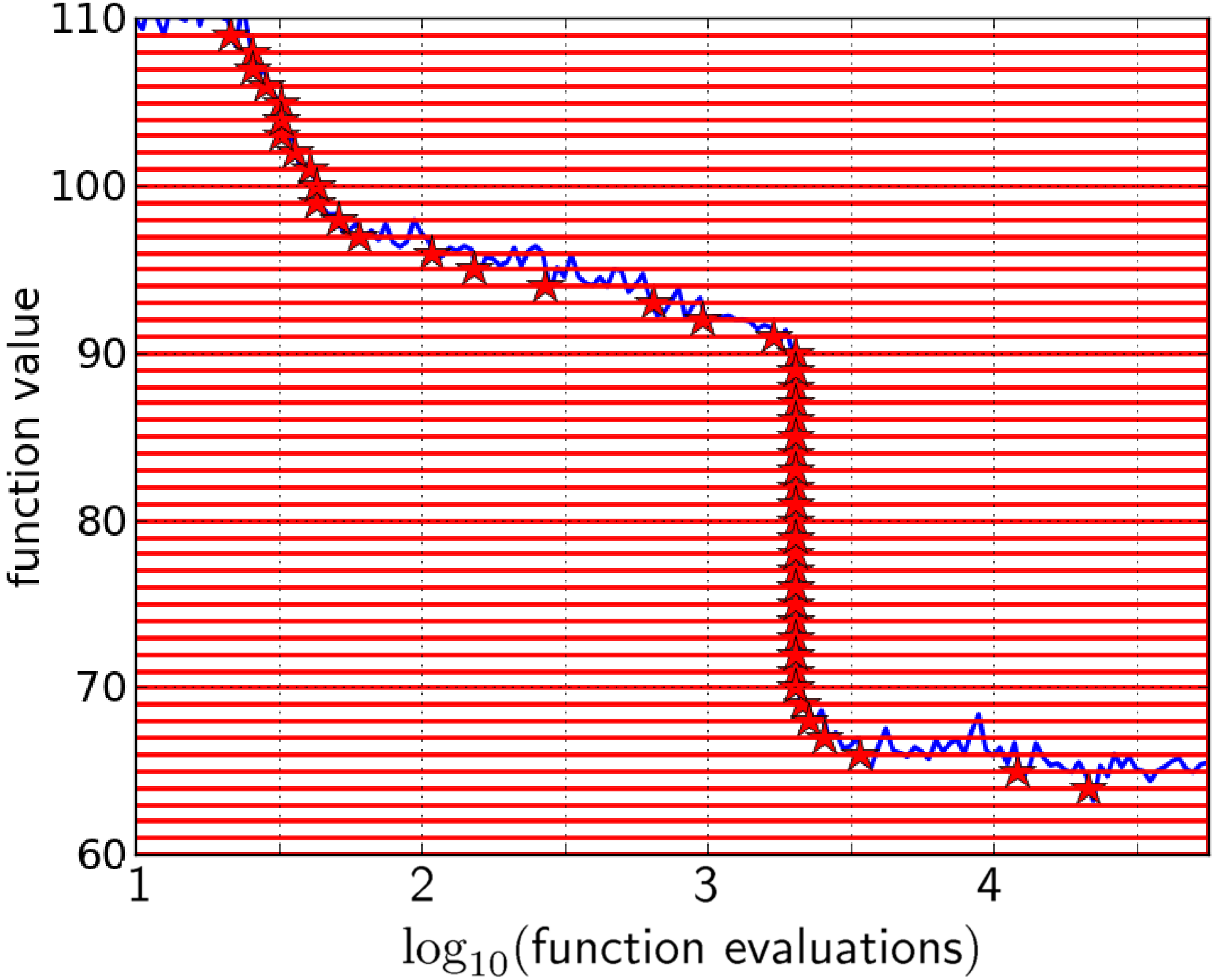
- has for each data point a vertical step of constant size
- displays for each x-value (budget) the count of observations to the left (first hitting times)

60% of the runs need between 2000 and 4000 evaluations

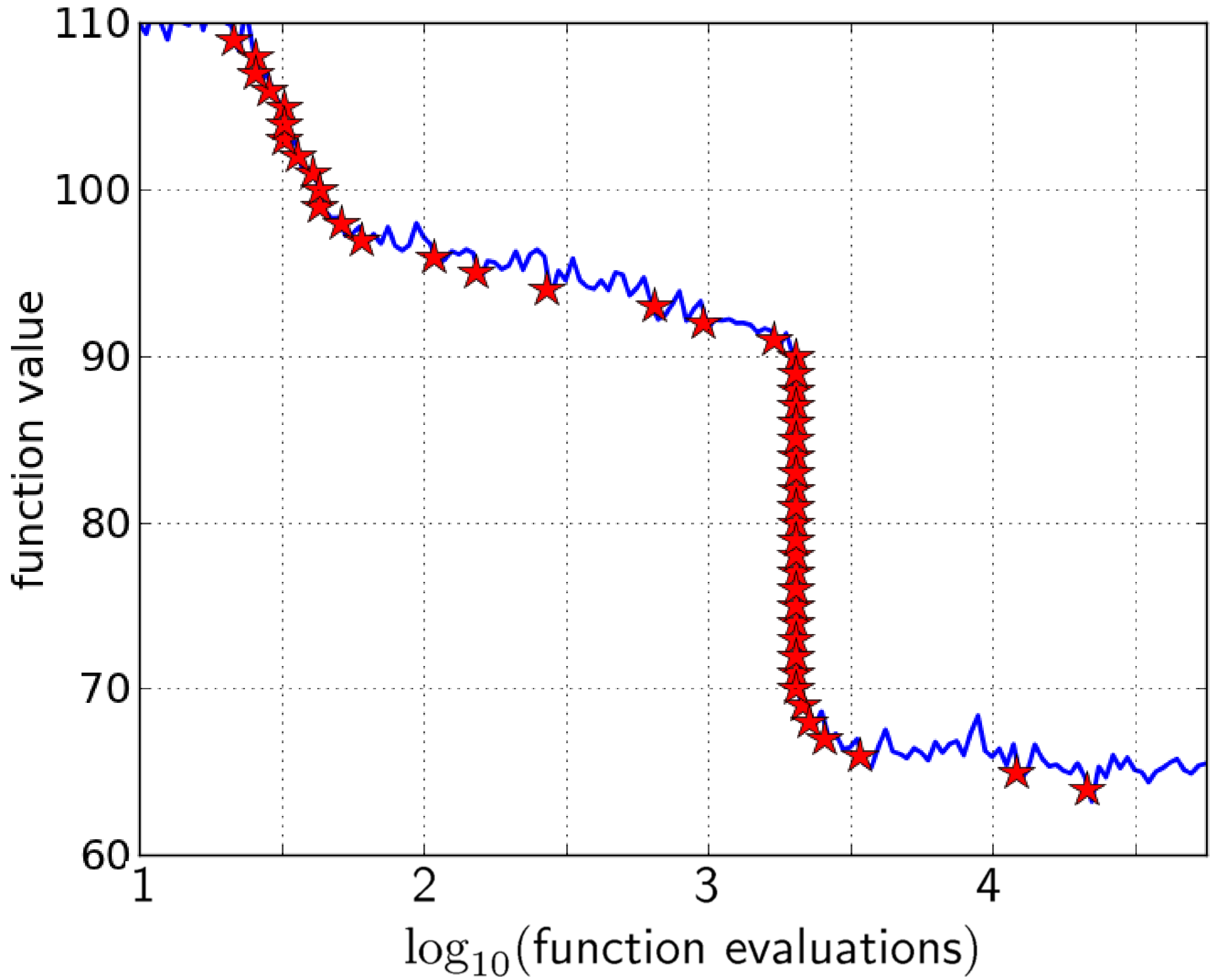
80% of the runs reached the target



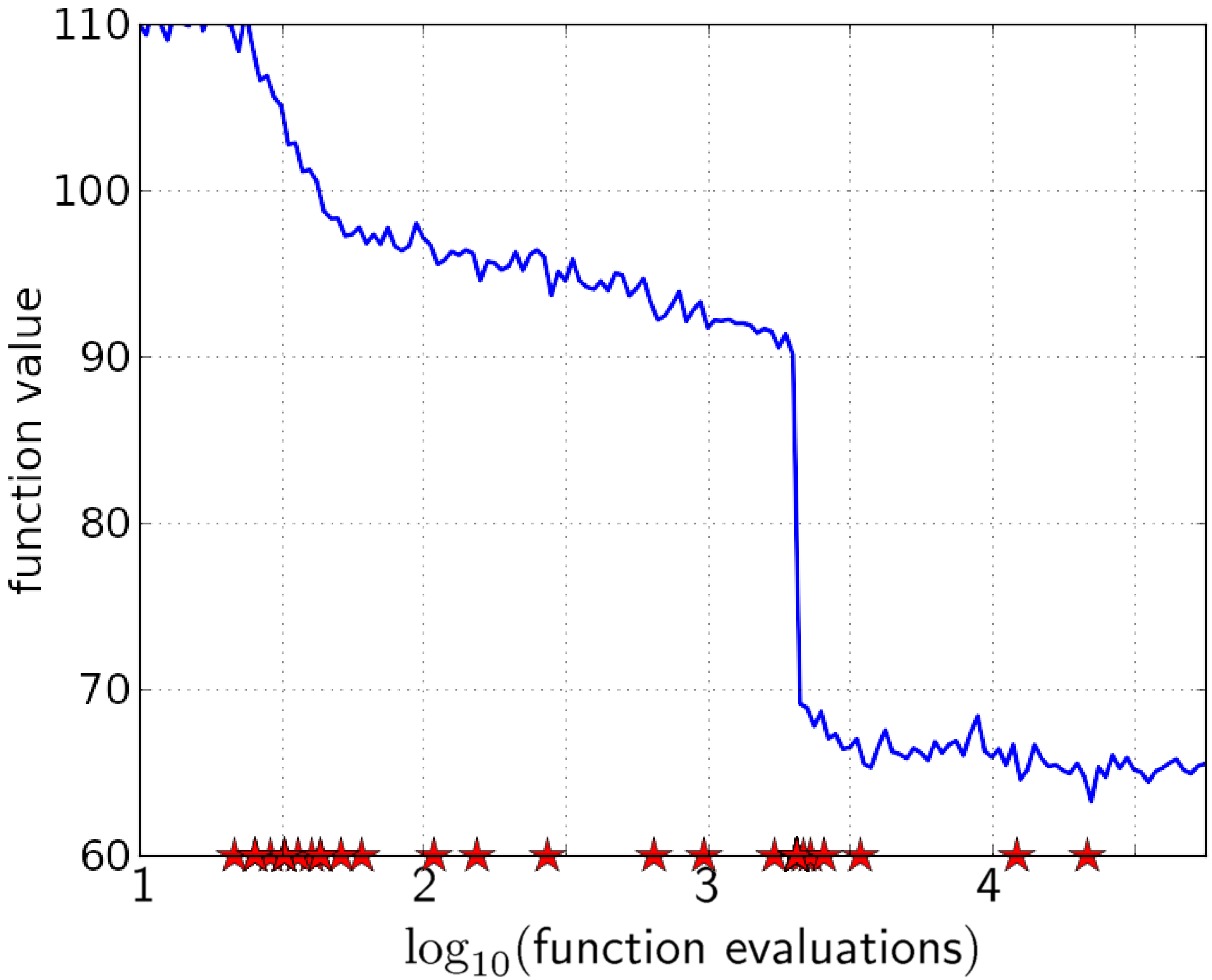
- reconstructing a single run using different target values



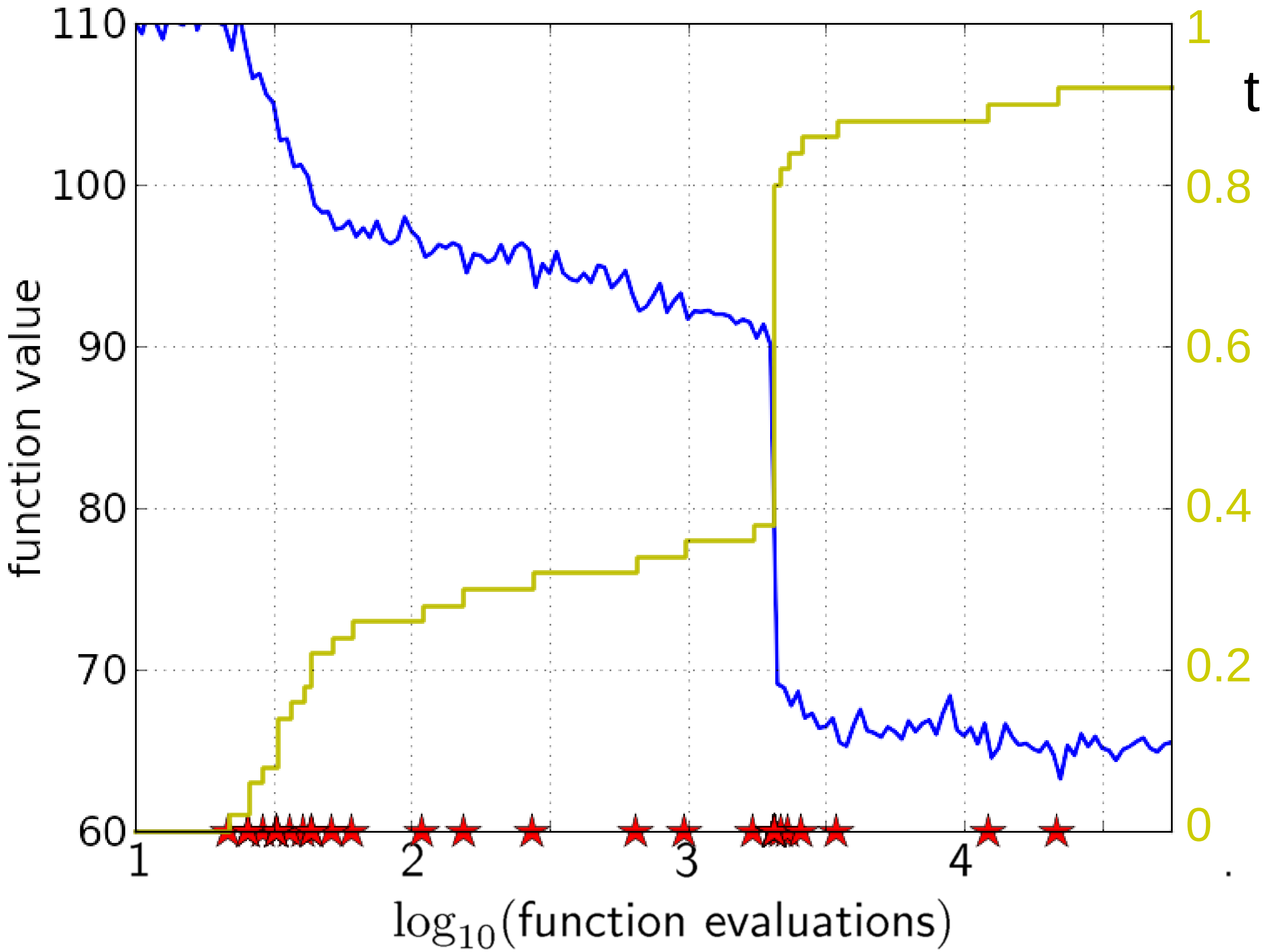
50 equally spaced targets


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

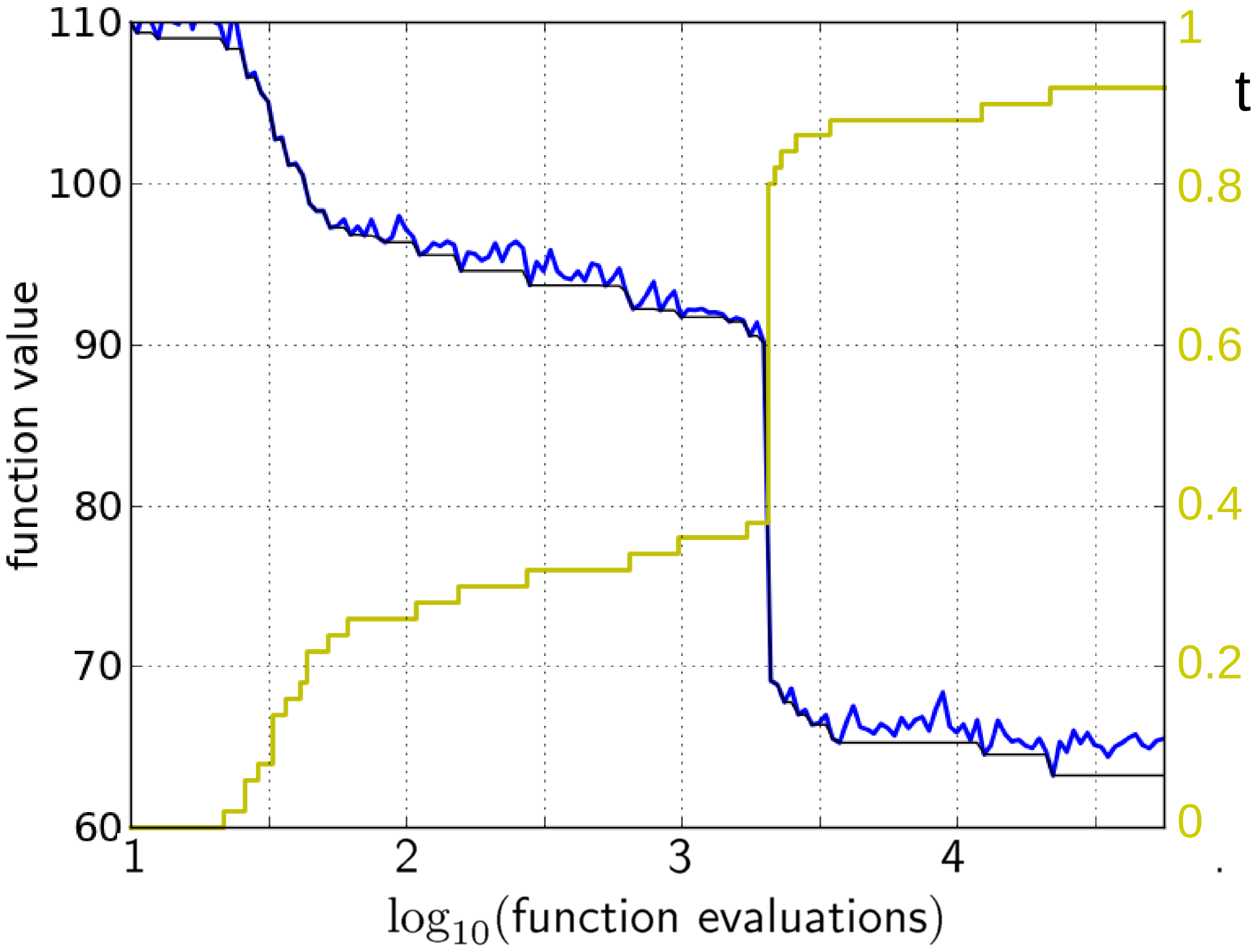
.



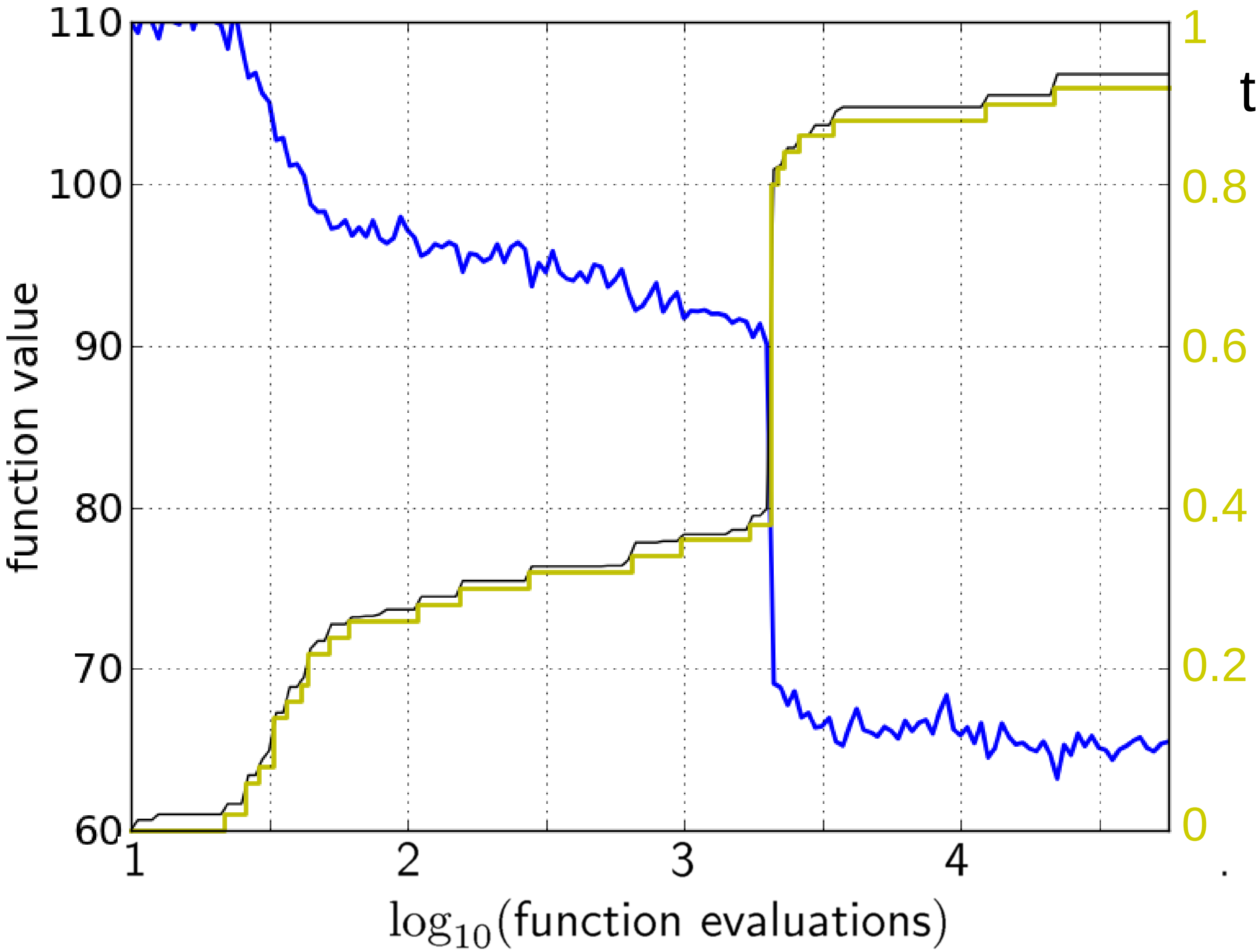
110
100
90
80
70
60
function value
1
2
3
4
$\log_{10}$(function evaluations)



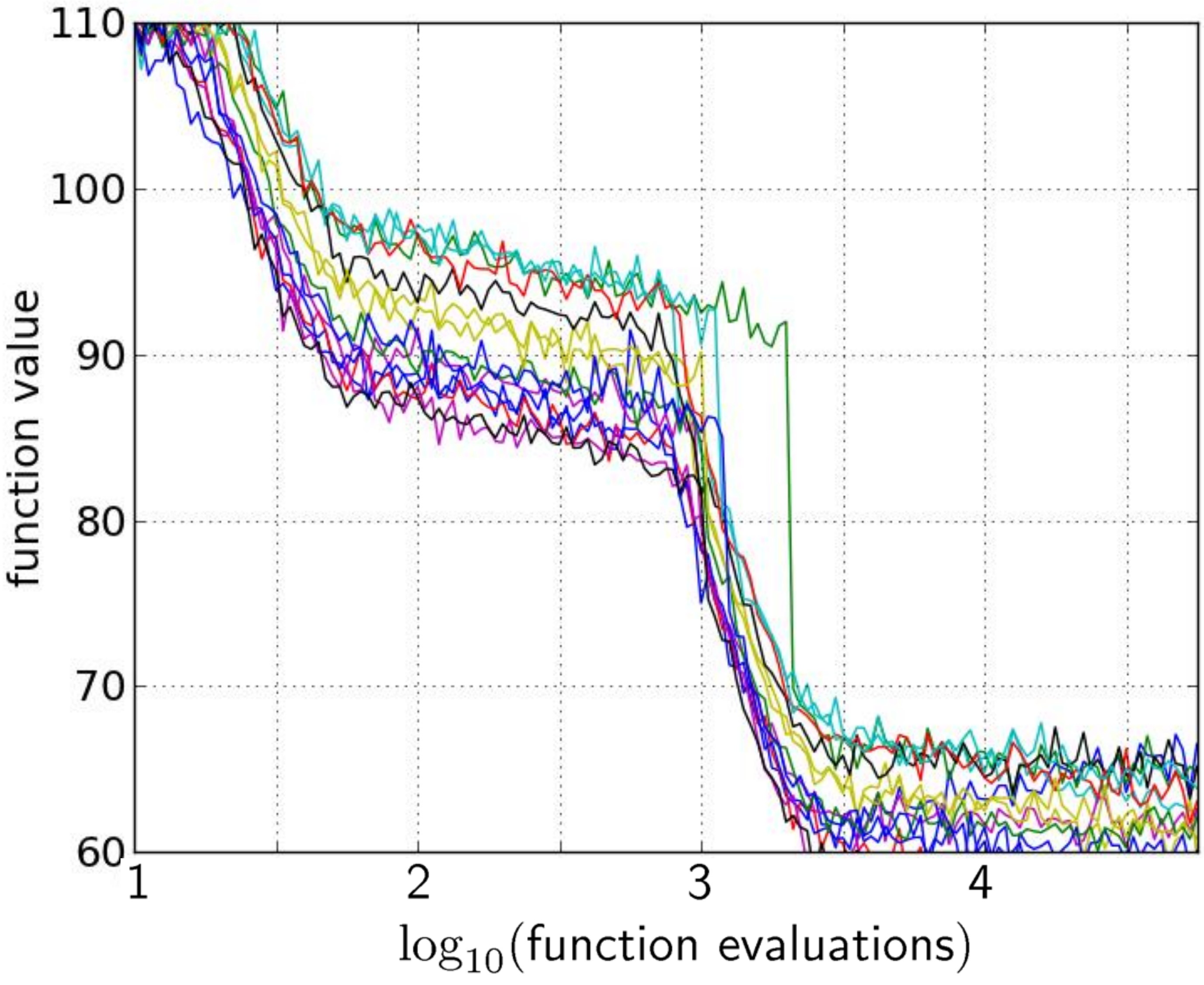
the empirical CDF
makes a step for each star, is monotonous and displays for each budget the fraction of targets achieved within the budget



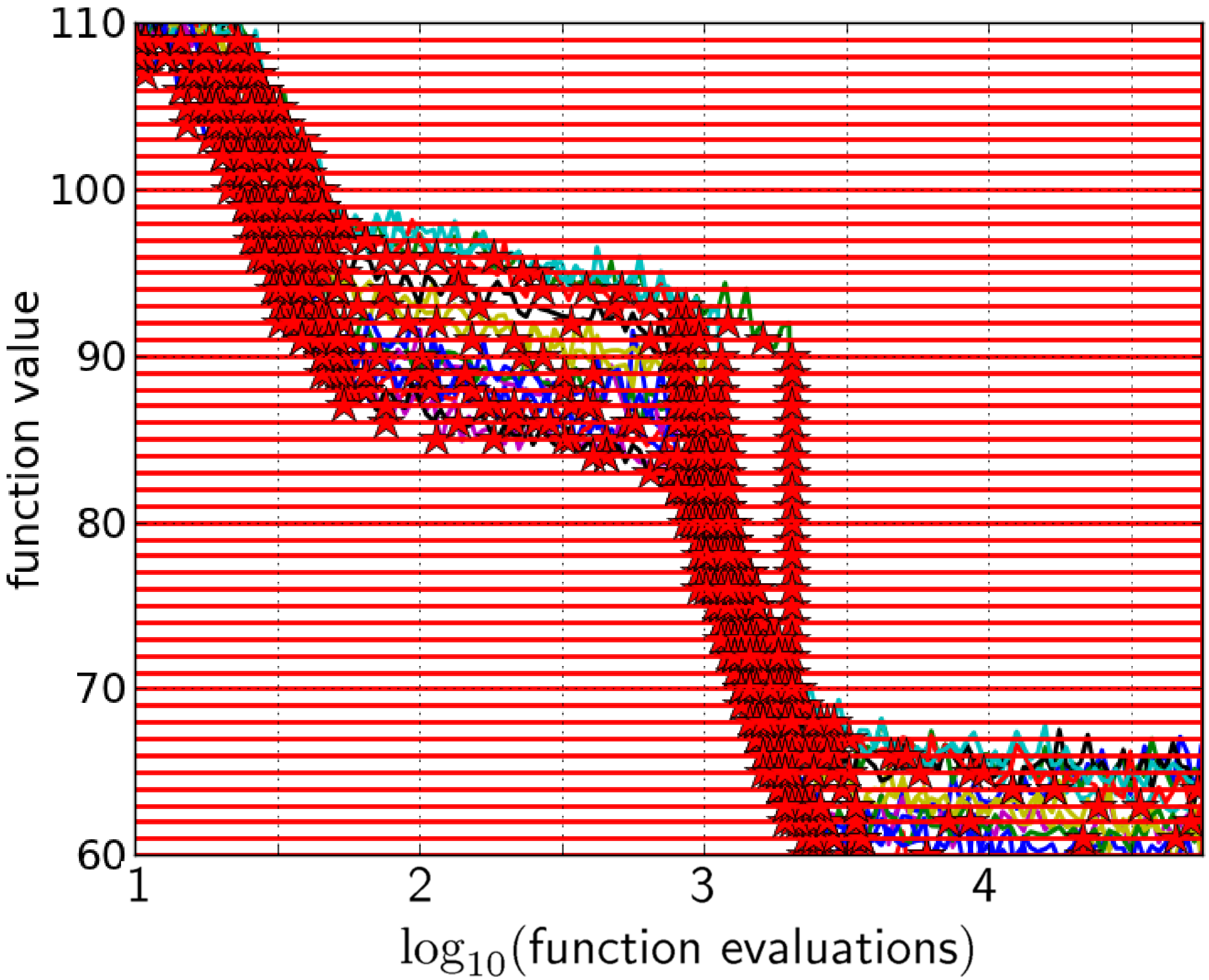
the ECDF recovers the monotonous graph, discretised and flipped



the ECDF recovers the monotonous graph, discretised and flipped

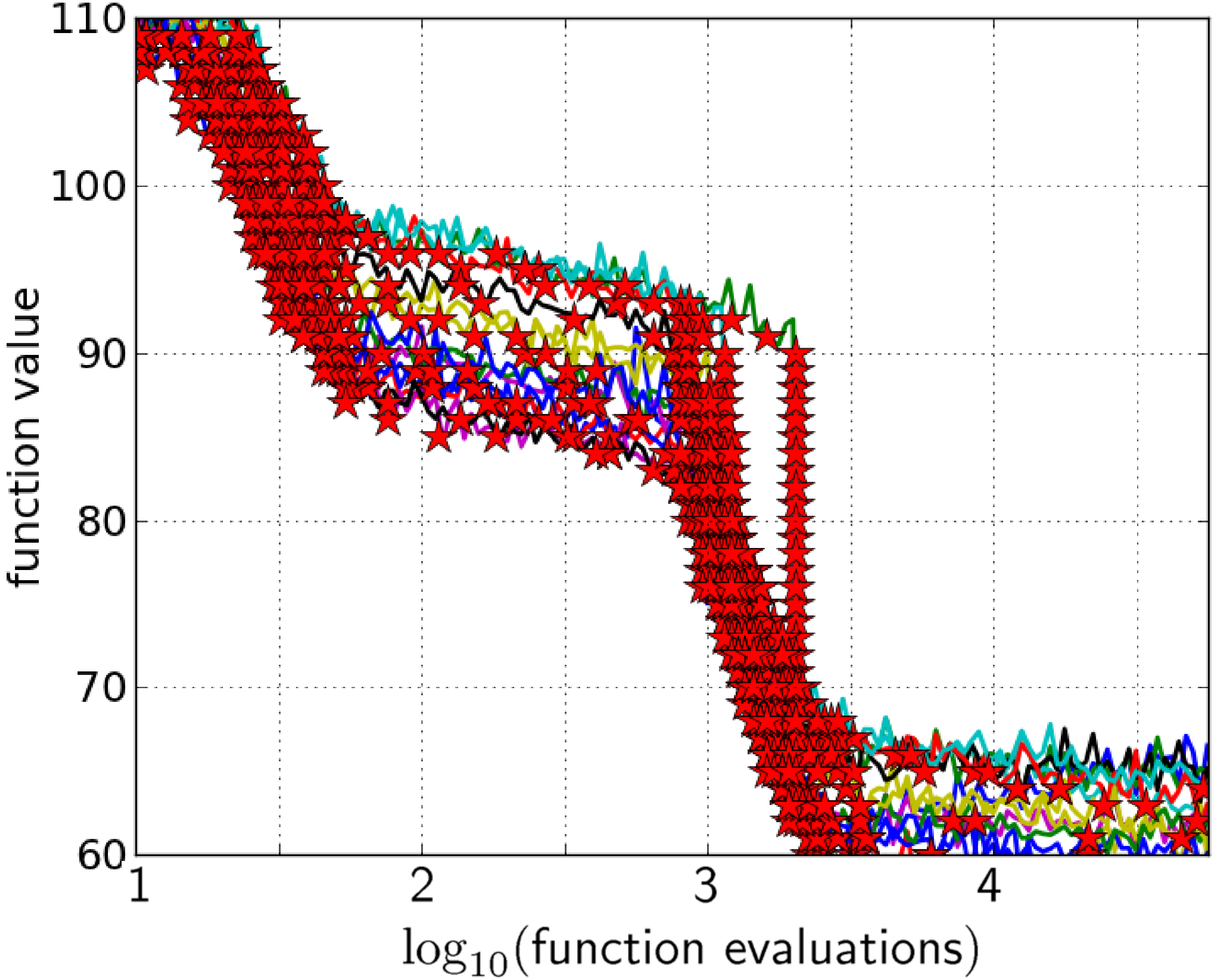

110
100
90
80
70
60
function value
1
2
3
4
log10(function evaluations)

15 runs



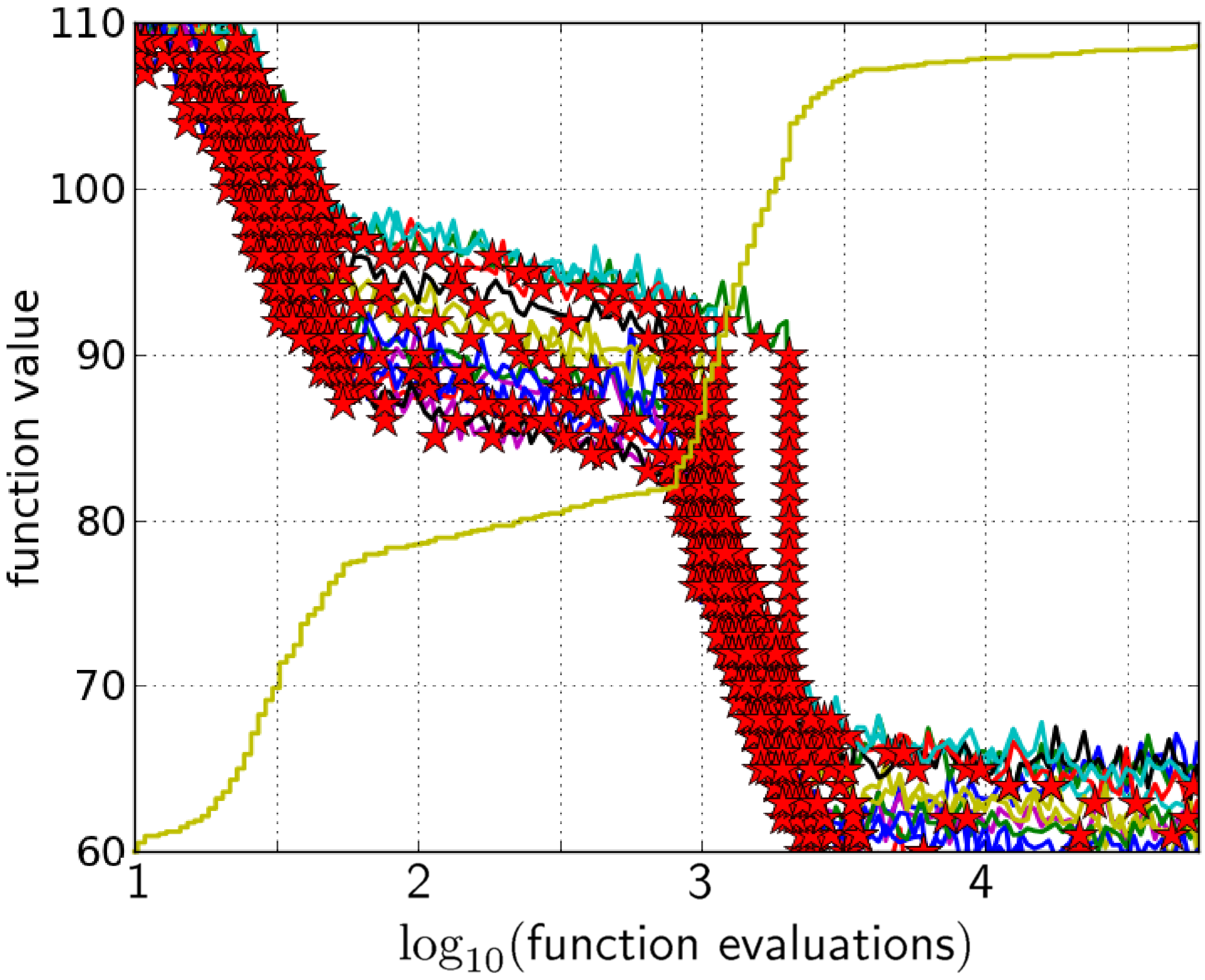
15 runs

50 targets



15 runs

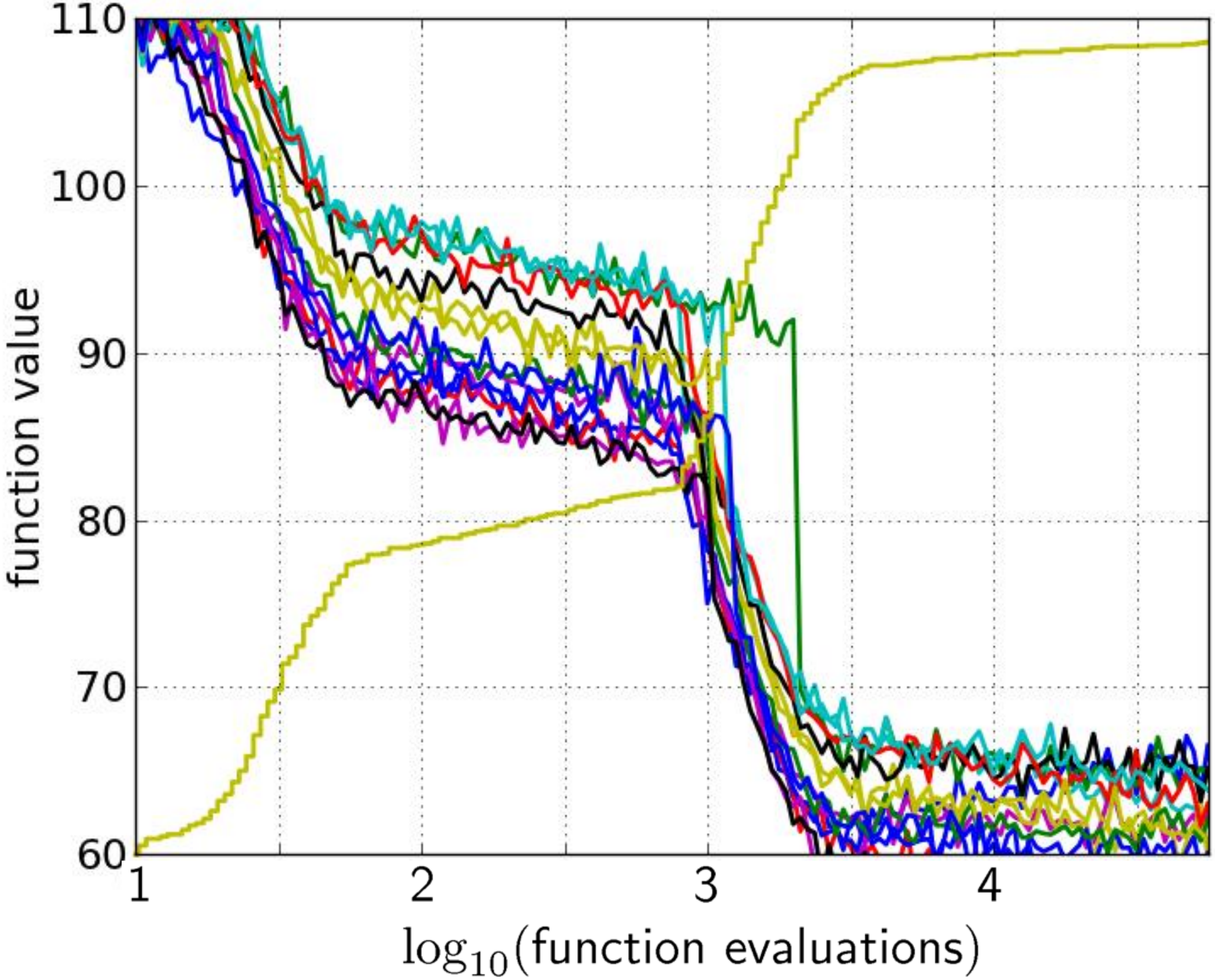
50 targets



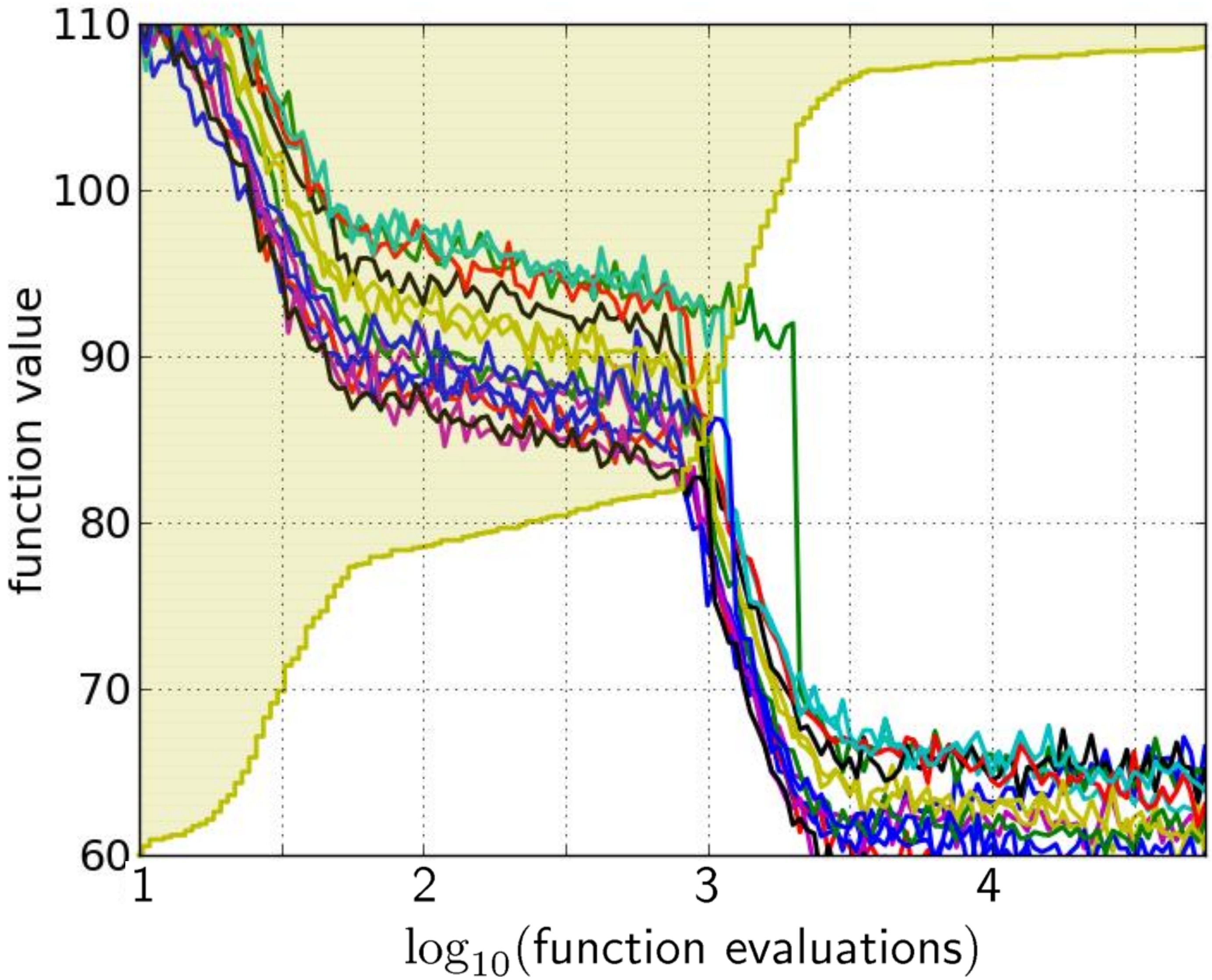
15 runs

50 targets

ECDF with 750 steps

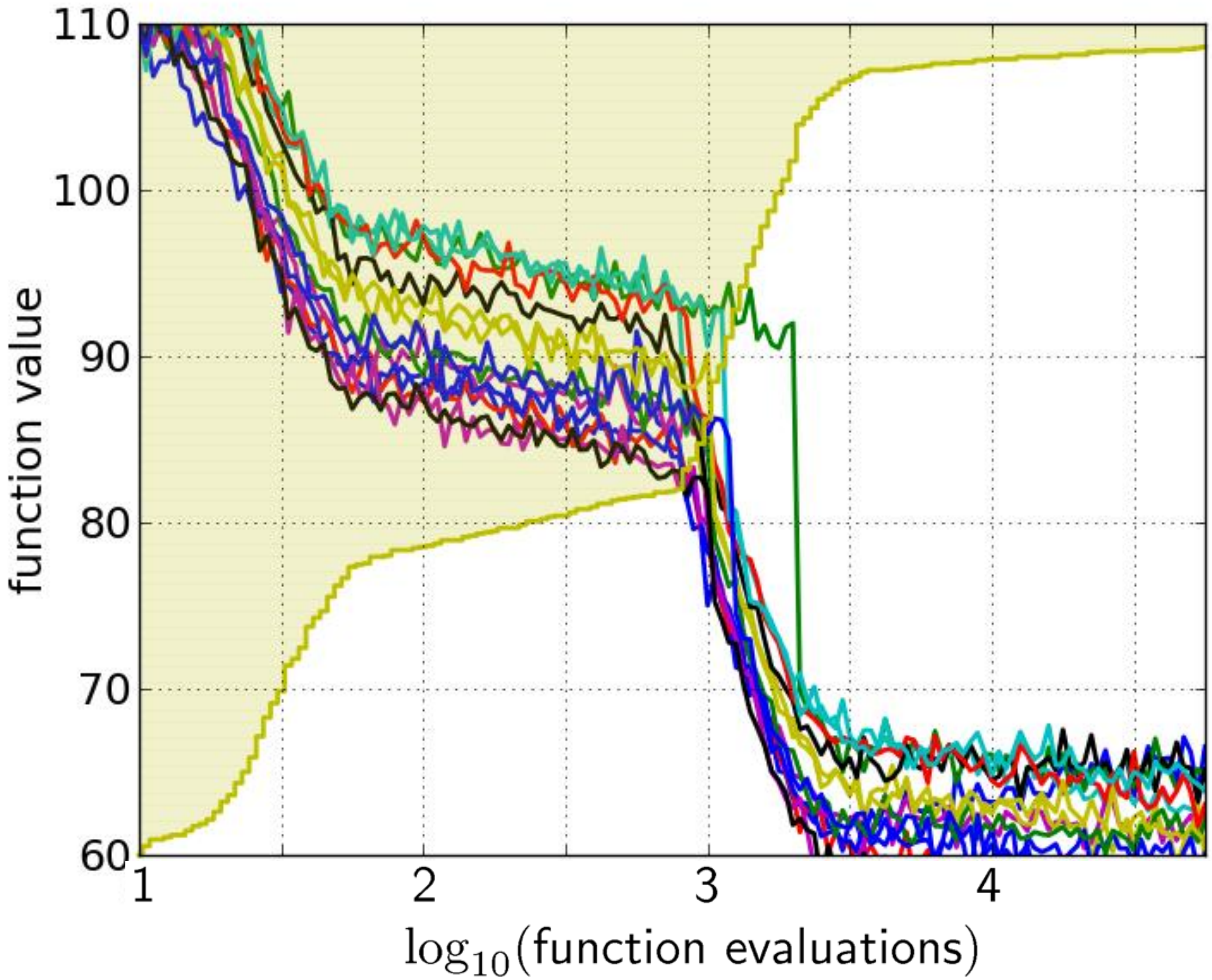


50 targets from 15 runs integrated in a single graph



50 targets from 15 runs integrated in a single graph

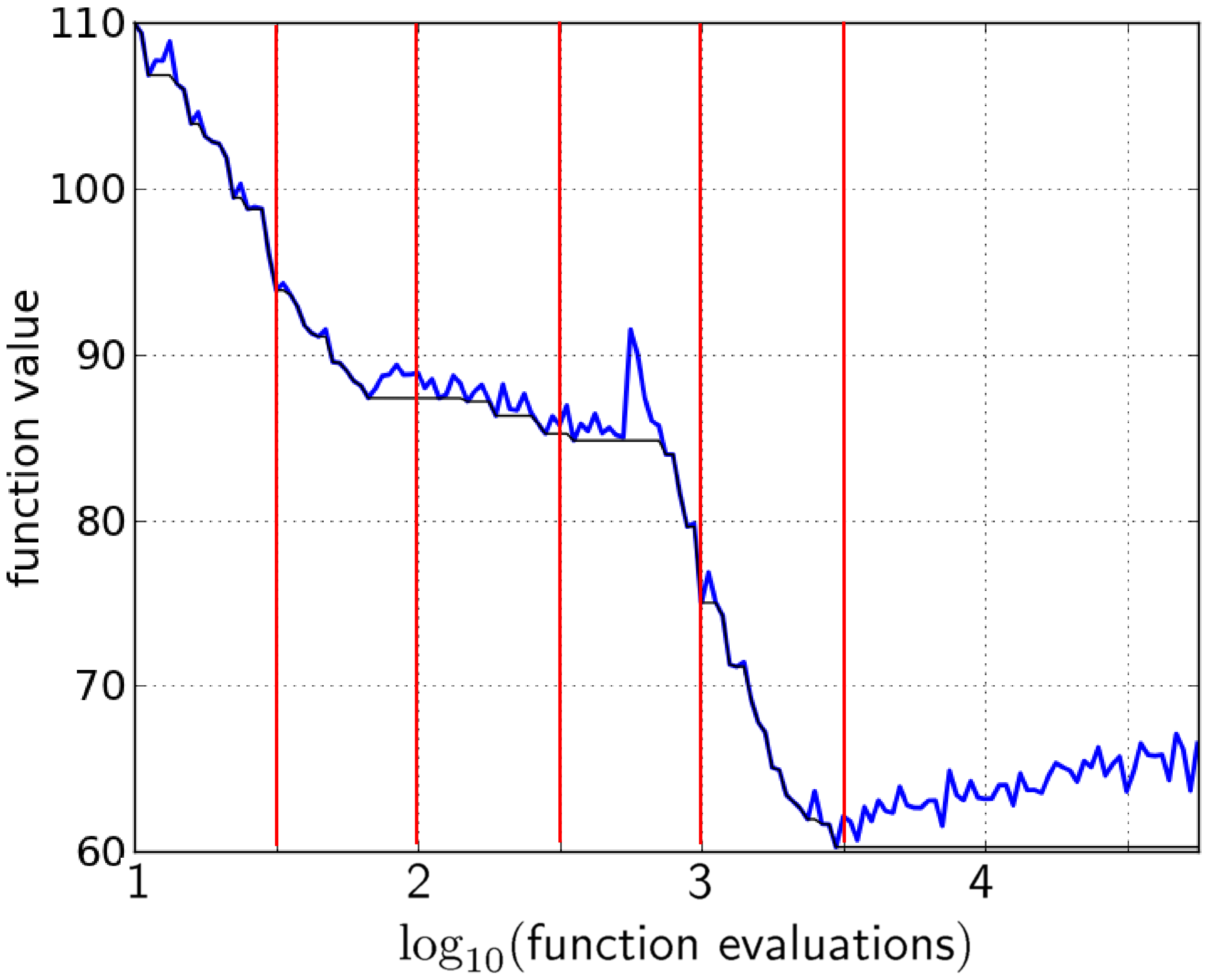
the area over the ECDF curve is the average log runtime (or geometric average runtime) over all targets (difficult and easy) and all runs



- 50 targets from 15 runs integrated in a single graph
- a shift of the empirical CDF to the left means a speed up by a certain factor

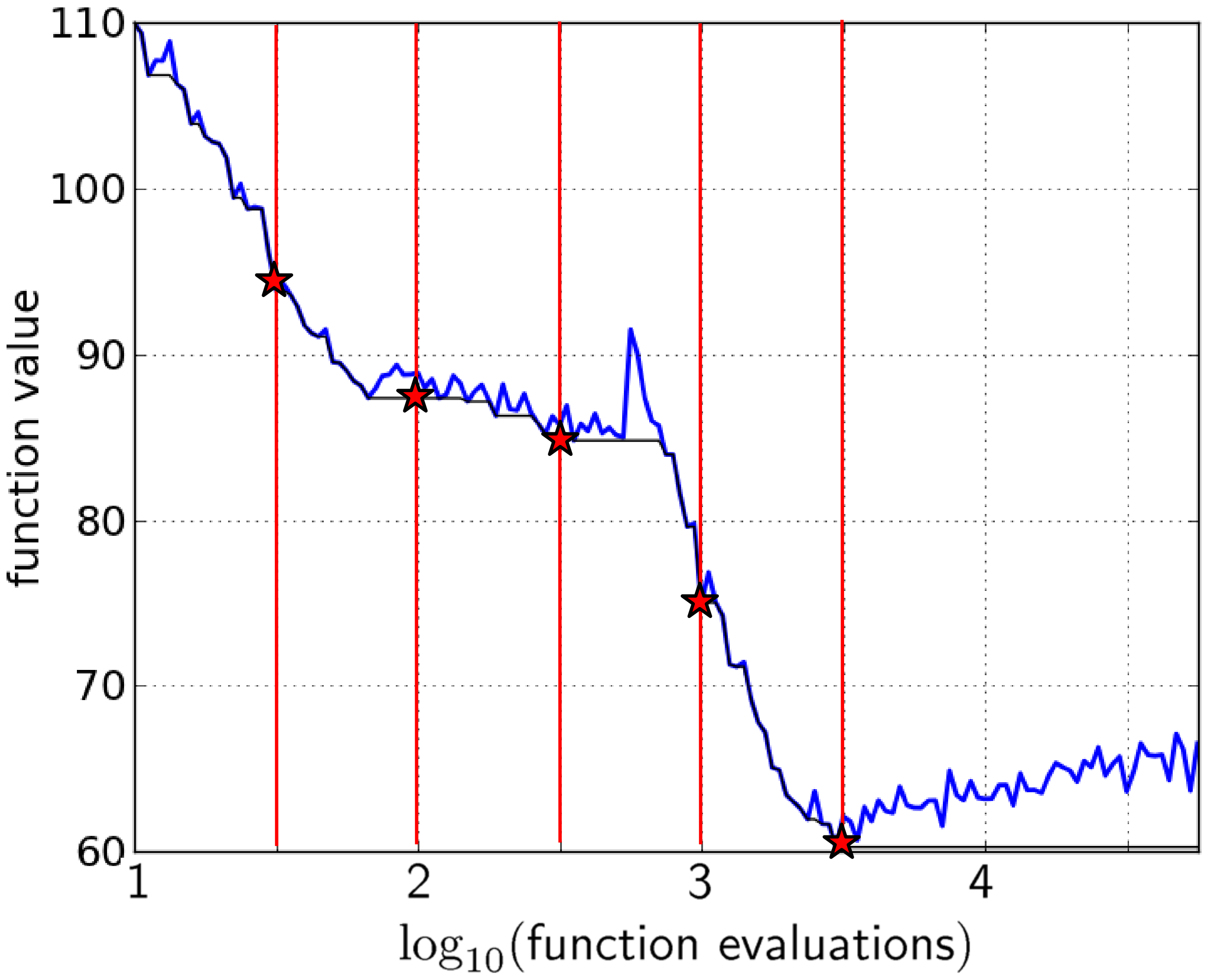
Runlength-based Target Values

Target budgets



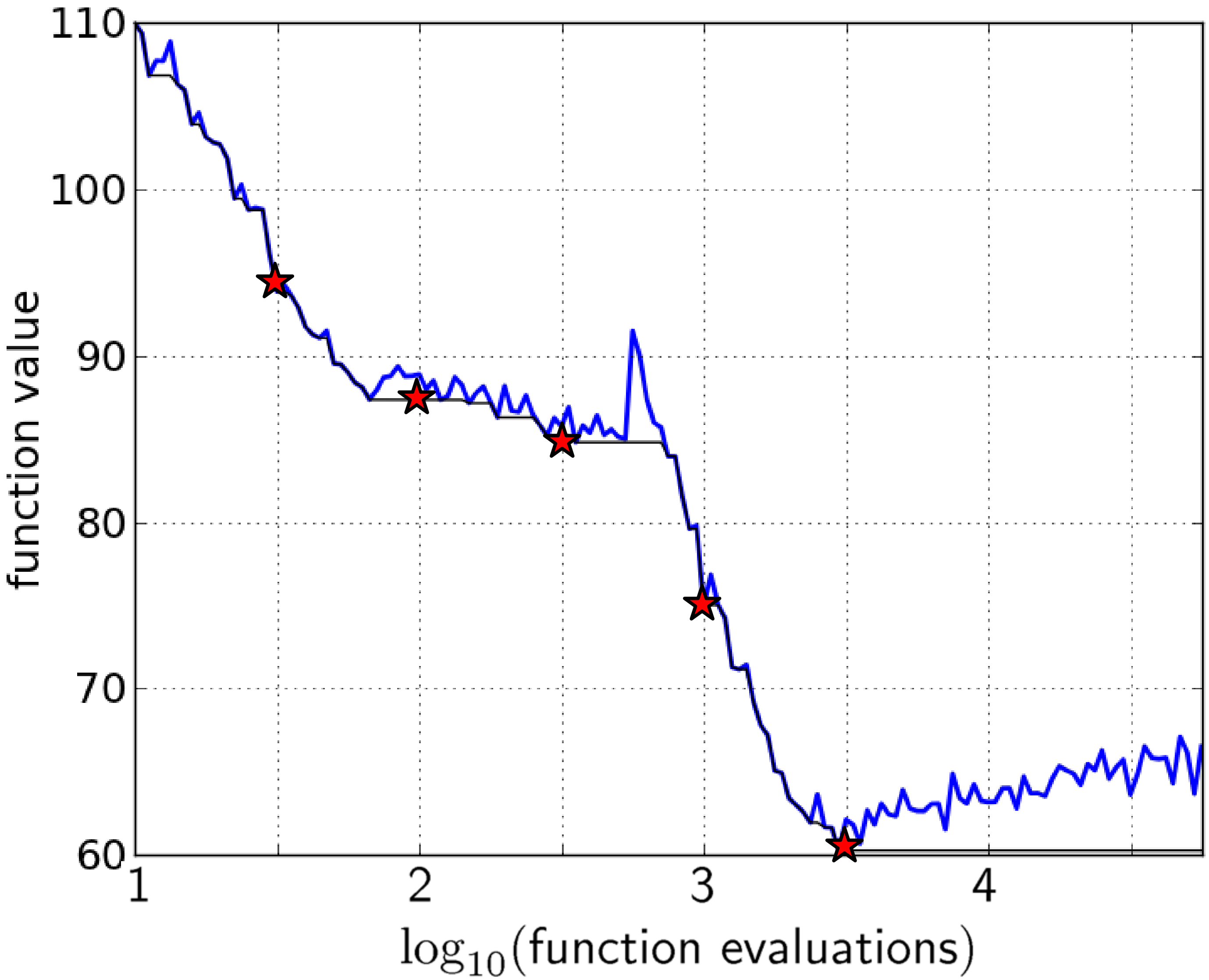
1) define reference target budgets

Target budgets on the reference algorithm



1) define reference target budgets
2) compute best function value achieved by a reference algorithm

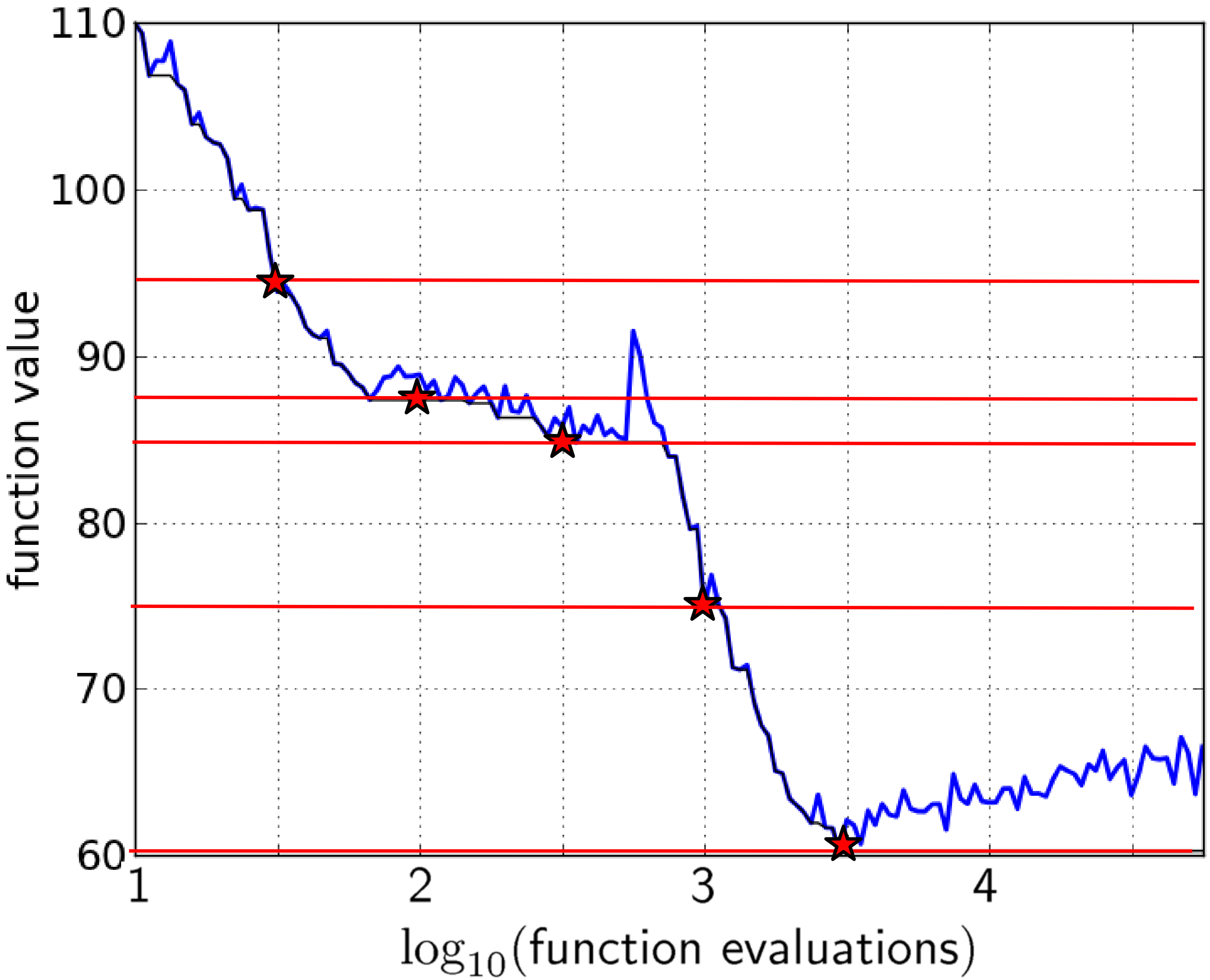
Target budgets on the reference algorithm



1) define reference target budgets

2) compute best function value achieved by a reference algorithm

Run-length based target f-values

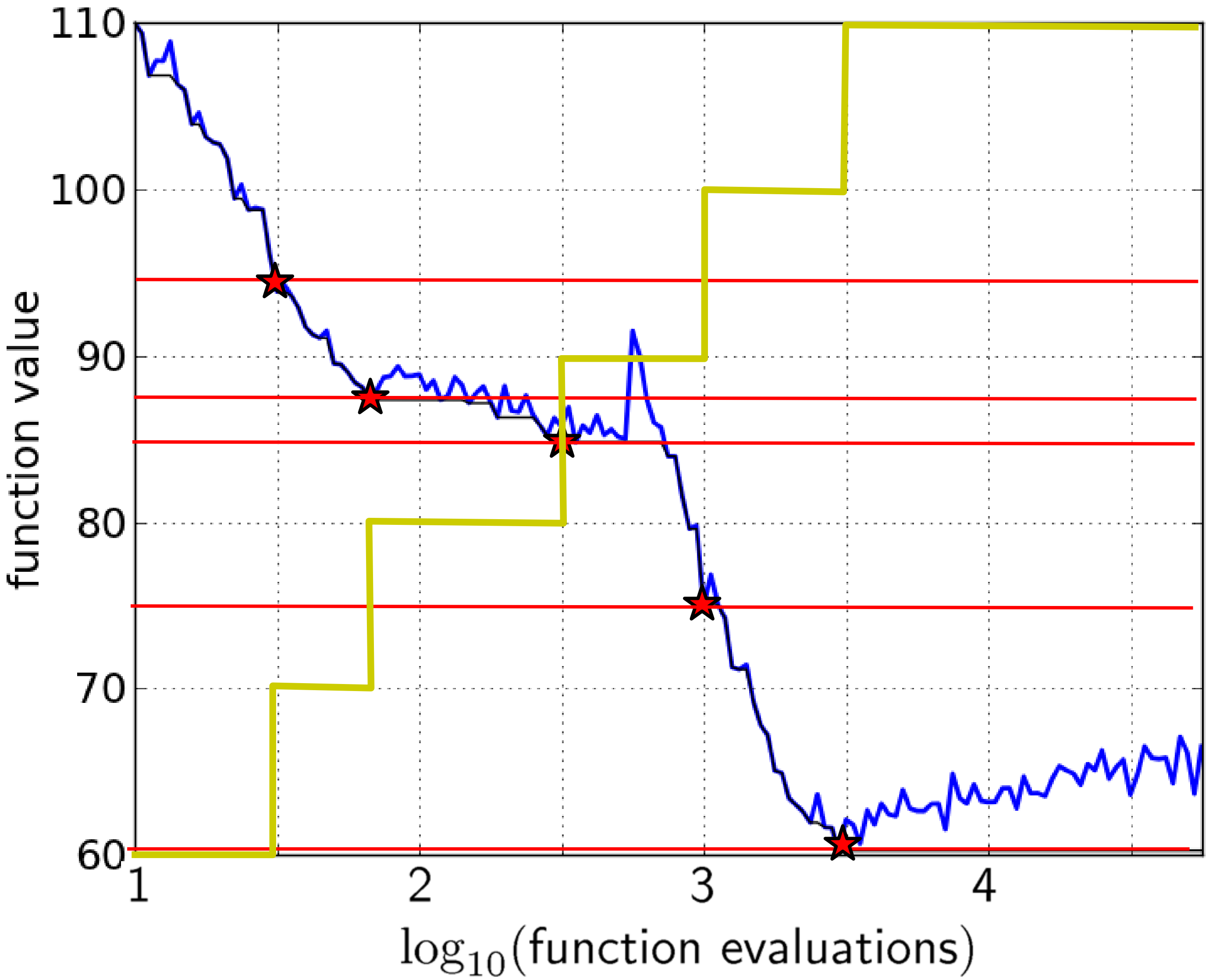


1) define reference target budgets

2) compute best function value achieved by a reference algorithm

=> set of target function values

Run-length based target f-values



1) define reference target budgets

2) compute best function value achieved by a reference algorithm

=> set of target function values

.

"Expensive" setting

- Changed target values compared to the original setting
- Target values depend on the function and the dimension
- Computation is based on the run length of the GECCO BBOB 2009 best result ==> runlength-based target values

ECDF: Summary

Empirical Cumulative Distribution Functions

- recover a single convergence graph (and generalize)
- can aggregate over any set of functions and target values

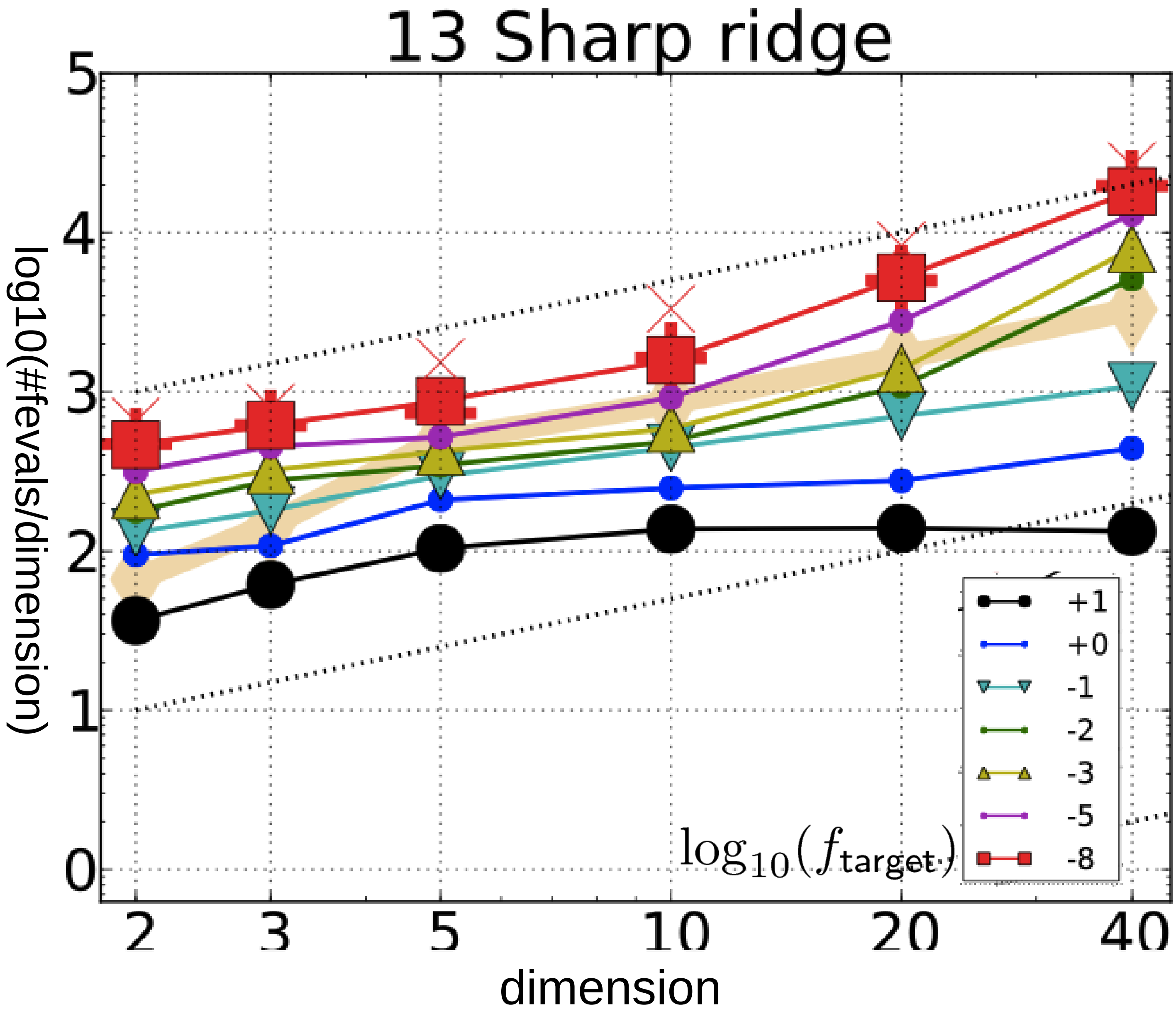
they display a set of run lengths or runtimes (RT)

- for RT on a single problem (function & target value) allow to estimate any statistics of interest from them, like median, expectation (ERT),… in a meaningful way
- AKA data profile [Moré&Wild 2009]
- Performance profile [Dolan&Moré 2002]: ECDFs of run lengths divided by the smallest observed run length

Questions?

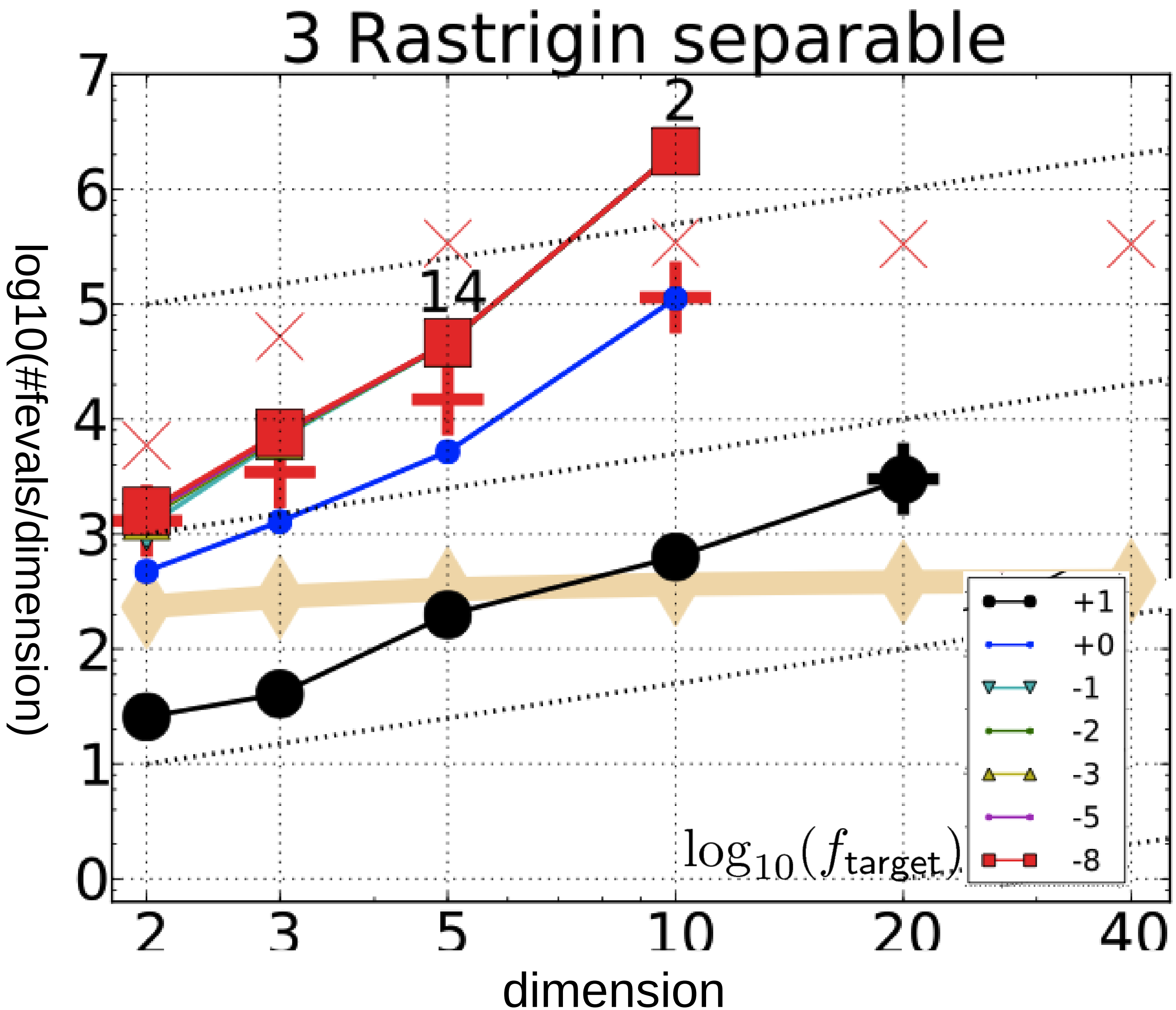
Different Displays of Runtimes

Scaling Behaviour with Dimension



- slanted grid lines: quadratic scaling
- horizontal lines: linear scaling
- light brown: artificial best 2009

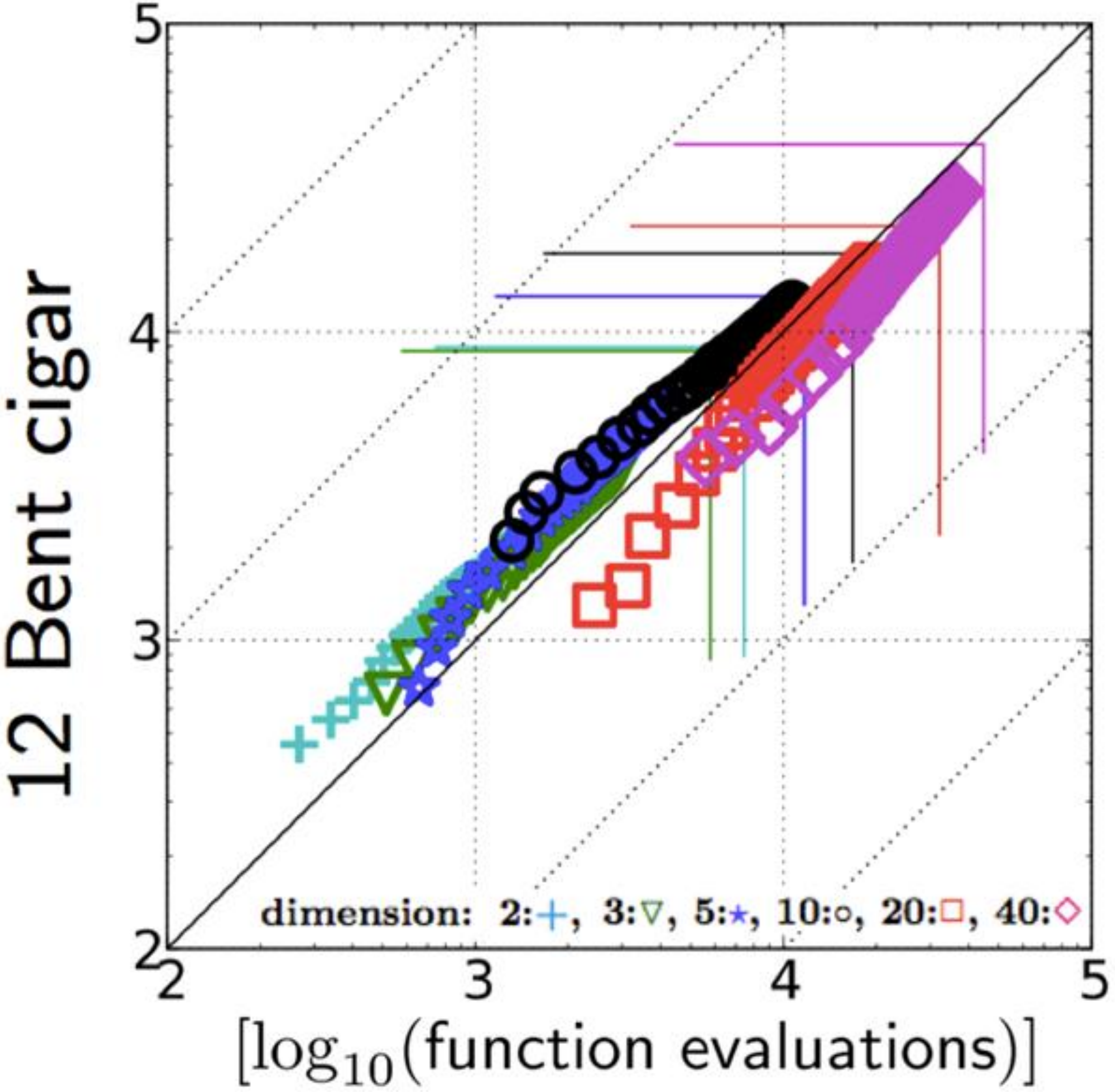
Example: Scaling Behaviour



- slanted grid lines: quadratic scaling
- horizontal lines: linear scaling
- light brown: artificial best 2009

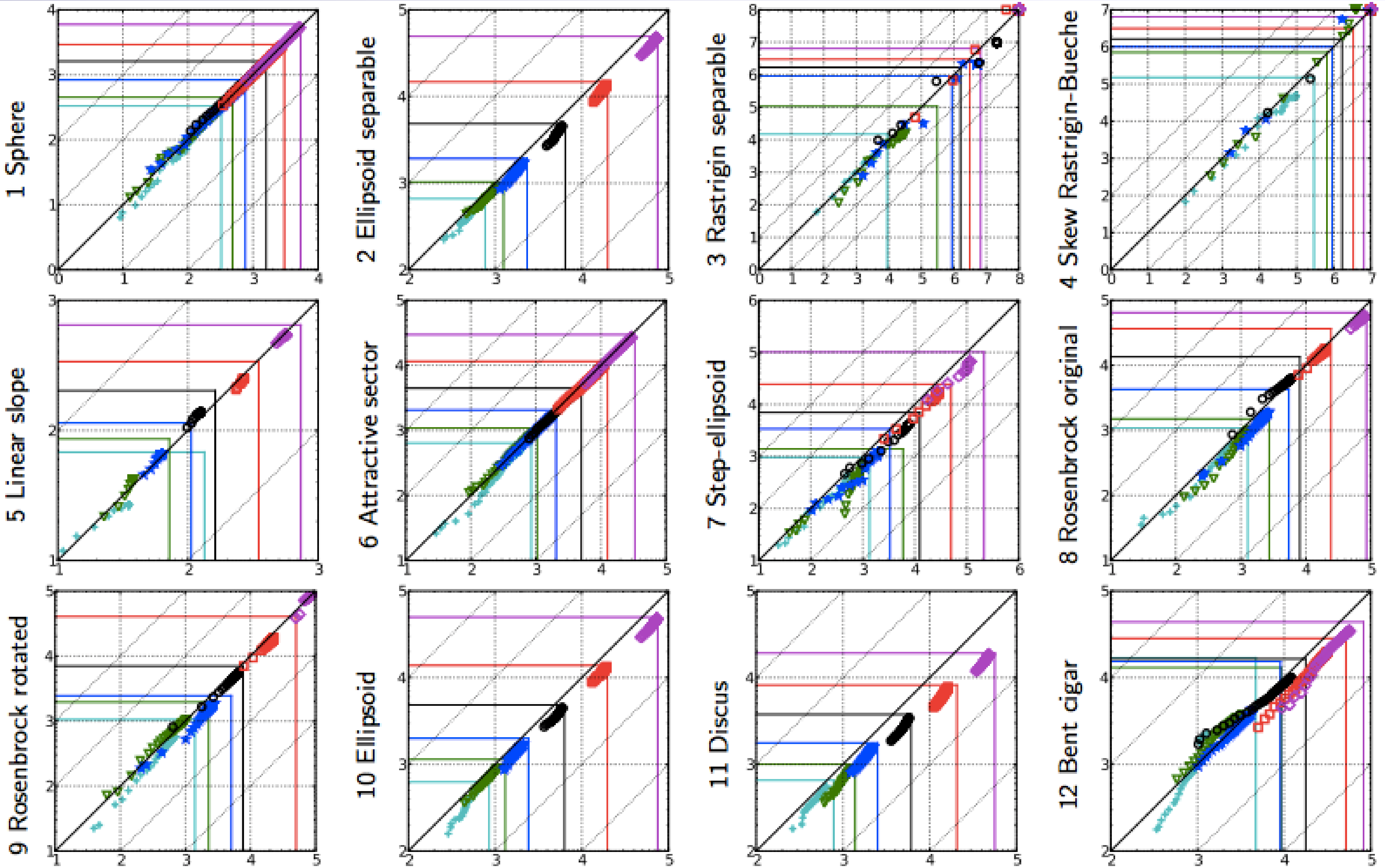
$\Rightarrow$ Experiments in >40-D are more often than not virtually superfluous

ERT scatter plots, all dimensions&targets



- estimated Expected Run Time (ERT), two algorithms
- 2-10 D: first algorithm “dominates”
- 20 & 40 D: second algorithm “dominates”

ERT scatter plots, all dimensions&targets



Single Function Table

Table 6: 20-D, running time excess ERT/ERT$_{\text{best}}$ on f_6, in italics is given the median final function value and the median number of function evaluations to reach this value divided by dimension

6 Attractive sector

Δftarget	1e+03	1e+02	1e+01	1e+00	1e-01	1e-02	1e-03	1e-04	1e-05	1e-07	Δftarget
ERT$_{\text{best}}$/D	4.03	26	64.7	87.2	123	152	184	219	248	309	ERT$_{\text{best}}$/D
ALPS	59	25	34	54	64	78	100	150	370	*14e-7/2e5*	ALPS [17]
AMaLGaM IDEA	26	22	19	22	21	22	22	21	22	22	AMaLGaM IDEA [4]
avg NEWUOA	**2.3**	**1.1**	1	**1**	**1**	**1**	1	1	**1**	1	avg NEWUOA [31]
BayEDAcG	46	41	*60e+0/2e3*	.	.	.	.	.	.	.	BayEDAcG [10]
BFGS	**2.2**	**2.7**	3.6	4.7	4.7	4.9	5	4.8	4.9	61	BFGS [30]
Cauchy EDA	6200	1500	1e3	1700	*17e-1/5e4*	.	.	.	.	.	Cauchy EDA [24]
BIPOP-CMA-ES	**2.9**	**2.2**	**1.5**	**1.7**	**1.6**	**1.6**	**1.6**	**1.5**	**1.6**	**1.6**	BIPOP-CMA-ES [15]
(1+1)-CMA-ES	**1.0**	4.5	13	180	1200	*13e-1/1e4*	.	.	.	.	(1+1)-CMA-ES [2]
DASA	12	6.8	9.9	19	25	33	49	58	63	74	DASA [19]
DEPSO	11	7.5	12	64	*13e-1/2e3*	.	.	.	.	.	DEPSO [12]
DIRECT	18	31	*40e+0/5e3*	.	.	.	.	.	.	.	DIRECT [25]
EDA-PSO	27	46	40	45	44	44	44	44	44	44	EDA-PSO [6]
full NEWUOA	5	**1.9**	**1.5**	**1.4**	**1.4**	**1.4**	**1.4**	**1.4**	**1.4**	**1.4**	full NEWUOA [31]
G3-PCX	4.1	**1.4**	**1.4**	**2**	**2.1**	**2.1**	**2.2**	**2.2**	**2.3**	**2.4**	G3-PCX [26]
simple GA	320	130	2e3	*11e+0/1e5*	.	.	.	.	.	.	simple GA [22]
GLOBAL	5	**2.0**	3.6	4.9	8.5	*42e-3/2e3*	.	.	.	.	GLOBAL [23]
iAMaLGaM IDEA	5.1	5.6	5.4	6.8	7.1	7.7	7.8	7.7	8	8.3	iAMaLGaM IDEA [4]
LSfminbnd	9	31	160	760	1100	960	*72e-1/1e4*	.	.	.	LSfminbnd [28]
LSstep	140	260	2300	*59e+0/1e4*	.	.	.	.	.	.	LSstep [28]
MA-LS-Chain	11	4.9	7.5	8.9	8	7.7	7.2	6.7	6.5	6	MA-LS-Chain [21]
MCS (Neum)	**1.8**	33	*42e+0/4e3*	.	.	.	.	.	.	.	MCS (Neum) [18]
NELDER (Han)	**2.2**	**2.4**	**2.7**	3.3	3.2	3.5	3.5	3.5	4	7.4	NELDER (Han) [16]
NELDER (Doe)	**1.5**	**2.3**	9.1	20	28	65	110	430	*46e-5/2e4*	.	NELDER (Doe) [5]
NEWUOA	**1**	**1**	1	**1.3**	**1.4**	**1.5**	**1.6**	**1.6**	**1.7**	**1.7**	NEWUOA [31]
(1+1)-ES	**2**	**2.2**	**2.1**	**2.8**	3.9	5.2	6.1	6.5	6.4	6.7	(1+1)-ES [1]
POEMS	89	26	31	37	36	36	36	35	36	37	POEMS [20]
PSO	6.4	280	1100	1400	980	820	710	620	570	790	PSO [7]
PSO_Bounds	9.5	45	120	150	140	140	140	130	160	220	PSO_Bounds [8]
Monte Carlo	2.4e5	*48e+1/1e6*	.	.	.	.	.	.	.	.	Monte Carlo [3]
Rosenbrock	**2.1**	3.9	31	76	210	230	810	*21e-2/1e4*	.	.	Rosenbrock [27]
IPOP-SEP-CMA-ES	3.2	**2.1**	**1.7**	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	**2**	**2**	IPOP-SEP-CMA-ES [29]
VNS (Garcia)	5	**2.8**	**1.9**	**1.9**	**1.7**	**1.7**	**1.7**	**1.6**	**1.6**	**1.6**	VNS (Garcia) [11]

Questions?