Optimization Performance Under Outlier Noise Models

Nikolaus Hansen[1], Anne Auger[1], Dimo Brockhoff[1], Alexandre Chotard[2], Oskar Girardin[1], and Tanguy Villain[1]
[1]Inria, CMAP, Ecole Polytechnique, Institut Polytechnique de Paris, France
[2]University Littoral Cote d'Opale, Calais

June 2026

Objective

Evaluation of the performance of off-the-shelf solvers under an outlier noise model

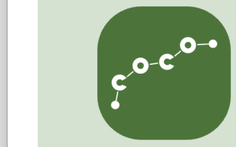
- Based on a standard benchmark: COCO/bbob
- Using an additive Cauchy-like outlier noise modification

  Implemented as a Python wrapper
- Running eight solvers directly accessible from the Python Package Index PyPI

COCO: COmparing Continuous

coco-platform.org

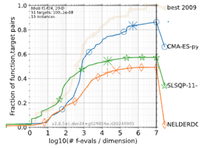
Update

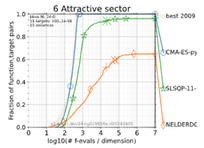


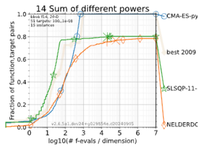
Search

- Home
- Motivation
- Get Started
  - Experiment
  - Postprocess or Display
- Test Suites
  - bbob
  - bbob-biobj
  - bbob-biobj-mixint
  - bbob-boxed
  - bbob-constrained
  - bbob-largescale
  - bbob-mixint
  - bbob-noisy
- Publish Results
- Reference
  - cocopp Python API
  - cocoex Python API
  - Core cocoex C API
- Workshops
  - Latest (BBOB-2025)
  - Previous
- Development
  - coco-experiment
  - coco-postprocess
  - coco-visualize
  - Code sprints
- Misc
  - No Free Lunch
  - Critiques and Rebuttals

COCO: COmparing Continuous Optimizers







COCO is a software platform for a systematic and sound comparison of mainly continuous and mixed optimization algorithms. COCO provides implementations of

- benchmark function testbeds,
- experimentation templates which are easy to parallelize,
- tools for processing and visualization of the data generated by one or several optimizers.

For a general introduction to the COCO software and its underlying concepts of performance assessment see Hansen et al. (2021). For a detailed discussion of the performance assessment methodology see Hansen et al. (2022). For getting started see Getting Started.

On this page

- Related links
- Citation and References
- Data Flow Chart

Related links

- Getting Started
- Data archive of all officially registered benchmark experiments (also accessible via the postprocess module)
- Postprocessed data of these archives for browsing
- How to submit a data set
- How to create and use COCO data archives with the `cocopp.archiving` Python module
- Source code (developer) page on Github
- Get news about COCO by registering here
- To visit the old COCO webpage, see the Internet Archive

Citation and References

Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., & Brockhoff, D. (2021). COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, *36*(1), 114-144.

The bbob Test Suite – <b>CO

coco-platform.org/testsuites/b...

Update

The bbob Test Suite

This is an online-friendly presentation of the `bbob` functions, based on the BBOB 2009 function document (Hansen et al. 2009b, BibTeX entry).

We present here 24 noise-free real-parameter single-objective benchmark functions (see (Hansen et al. 2009a, 2016) for our experimental setup and (Hansen et al. 2022) for our performance assessment methodology). Our intention behind the selection of benchmark functions was to evaluate the performance of algorithms with regard to typical difficulties which we believe occur in continuous domain search. We hope that the function collection reflects, at least to a certain extend and with a few exceptions, a more difficult portion of the problem distribution that will be seen in practice (easy functions are evidently of lesser interest).

We prefer benchmark functions that are comprehensible such that algorithm behaviors can be understood in the topological context. In this way, a desired search behavior can be pictured and deficiencies of algorithms can be profoundly analyzed. Last but not least, this can eventually lead to a systematic improvement of algorithms.

All benchmark functions are scalable with the dimension. Most functions have no

Separable

f_1: Sphere
f_2: Ellipsoid separable
f_3: Rastrigin separable
f_4: Skew Rastrigin-Bueche
f_5: Linear slope

Low or moderate conditioning

f_6: Attractive sector
f_7: Step-ellipsoid
f_8: Rosenbrock original
f_9: Rosenbrock rotated

High conditioning and unimodal

f_{10}: Ellipsoid
f_{11}: Discus
f_{12}: Bent cigar
f_{13}: Sharp ridge
f_{14}: Sum of different powers

Multi-modal with adequate global structure

f_{15}: Rastrigin
f_{16}: Weierstrass
f_{17}: Schaffer F7, condition 10
f_{18}: Schaffer F7, condition 1000
f_{19}: Griewank-Rosenbrock F8F2

Multi-modal with weak global structure

f_{20}: Schwefel x*sin(x)
f_{21}: Gallagher 101 peaks
f_{22}: Gallagher 21 peaks
f_{23}: Katsuura
f_{24}: Lunacek bi-Rastrigin

750 citations

OPTIMIZATION METHODS & SOFTWARE
2021, VOL. 36, NO. 1, 114–144
https://doi.org/10.1080/10556788.2020.1808977

COCO: a platform for comparing continuous optimizers in a black-box setting

Nikolaus Hansen[a,b], Anne Auger[a,b], Raymond Ros[c], Olaf Mersmann[d], Tea Tušar[e] and Dimo Brockhoff[a,b]

[a]Randopt Team, Inria, Palaiseau, France; [b]CMAP, CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, France; [c]LRI, Université Paris-Sud, Orsay, France; [d]Computational Statistics, TU Dortmund University, Dortmund, Germany; [e]Jožef Stefan Institute, Ljubljana, Slovenia

ABSTRACT

We introduce COCO, an open-source platform for Comparing Continuous Optimizers in a black-box setting. COCO aims at automatizing the tedious and repetitive task of benchmarking numerical optimization algorithms to the greatest possible extent. The platform and the underlying methodology allow to benchmark in the same framework deterministic and stochastic solvers for both single and multiobjective optimization. We present the rationals behind the (decade-long) development of the platform as a general proposition for guidelines towards better benchmarking. We detail underlying fundamental concepts of COCO such as the definition of a problem as a function instance, the underlying idea of instances, the use of target values, and runtime defined by the number of function calls as the central performance measure. Finally, we give a quick overview of the basic code structure and the currently available test suites.

ARTICLE HISTORY
Received 17 June 2019
Accepted 8 August 2020

KEYWORDS
Numerical optimization; black-box optimization; derivative-free optimization; benchmarking; performance assessment; test functions; runtime distributions; software

1. Introduction

We consider the continuous black-box optimization or search problem to minimize

$$f : X \subset \mathbb{R}^n \to \mathbb{R}^m \quad n, m \geq 1, \tag{1}$$

where the search domain X is typically a bounded hypercube or the entire continuous space.[1] More specifically, we aim to find, as quickly as possible, one or several solutions $\boldsymbol{x}$ in the search space X with *small* value(s) of $f(\boldsymbol{x}) \in \mathbb{R}^m$.

A continuous optimization algorithm, denoted as *solver*, addresses the above problem. In this paper, we only consider zero-order black-box optimization [19,57,58]: while the search domain $X \subset \mathbb{R}^n$ and its boundaries are accessible, no other prior knowledge about f is available to the solver.[2] That is, f is considered as a black-box, also known as an oracle, and the *only* way the solver can acquire information on f is by querying the value $f(\boldsymbol{x})$ of a solution $\boldsymbol{x} \in X$. Zero-order black-box optimization is thus a derivative-free optimization setting.[3] We generally consider 'time' to be the number of calls to the function f and will define 'runtime' correspondingly.

81 citations

Anytime Performance Assessment in Blackbox Optimization Benchmarking

Nikolaus Hansen, Anne Auger, Dimo Brockhoff, and Tea Tušar

***Abstract*—We present concepts and recipes for the anytime performance assessment when benchmarking optimization algorithms in a blackbox scenario. We consider runtime—oftentimes measured in the number of blackbox evaluations needed to reach a target quality—to be a universally measurable cost for solving a problem. Starting from the graph that depicts the solution quality versus runtime, we argue that runtime is the *only* performance measure with a generic, meaningful, and quantitative interpretation. Hence, our assessment is solely based on runtime measurements. We discuss proper choices for solution quality indicators in single- and multi-objective optimization, as well as in the presence of noise and constraints. We also discuss the choice of the target values, budget-based targets, and the aggregation of runtimes by using simulated restarts, averages, and empirical cumulative distributions which generalize convergence graphs of single runs. The presented performance assessment is to a large extent implemented in the comparing continuous optimizers (COCO) platform freely available at https://github.com/numbbo/coco.**

***Index Terms*—Anytime optimization, benchmarking, blackbox optimization, performance assessment, quality indicator.**

I. Introduction

We present practical concepts and ideas for the performance assessment of optimization algorithms when benchmarked in a blackbox and anytime scenario. Going beyond a simple ranking of algorithms, we aim to provide a *quantitative* and *meaningful* performance assessment, which allows for conclusions like *algorithm A is seven times faster than algorithm B* in solving a given problem or in solving problems with certain characteristics. To achieve this end in a comparative and timeless manner, we argue that we should measure the *number of blackbox evaluations* to reach a predefined *quality indicator* value (a target). More generally, we argue to measure a cost that is defined on a ratio scale and is comparable across publications. We call this measure the *runtime* of the algorithm to reach a given target. Yet, our assessment methodology does not depend on any specific cost measure, as long as the costs are quantitative and comparable.[1]

In this article, we formalize the optimization goal by a so-called quality indicator. Its definition may heavily depend on the optimization scenario, e.g., the number of objectives or constraints. Broadly speaking, a quality indicator is based on the sequence of all so-far visited solutions. In the simplest case, it is the objective function value of the last visited solution.

Runtimes represent the cost of optimization. Compared to the quality indicator, the definition of costs depends to a lesser extent on the specific optimization scenario. To sustain reproducibility and comparability across publications, we recommend against CPU or wall-clock time as cost measure[2] (see also Hooker [22] for a further discussion on the unwanted consequences of benchmarking based on CPU time).

Benchmarking is usually computationally expensive and benchmarking for a *single* budget seems vastly inefficient by 1) addressing only one of many possible budget scenarios (scenarios heavily depend on the software and hardware environment) and 2) throwing away most of the data generated during the experiment. An anytime approach to benchmarking prevents these drawbacks. To allow for a budget-free performance assessment even for non-anytime algorithms that have a maximum or timeout budget as decisive or mandatory *input* parameter (decided by the user), we collect data with an any-budget *experimental procedure* that runs repeated experiments with increasing input budget [31].[3] Non-anytime algorithms that do not take a maximum budget as an input parameter can be accurately assessed only by the time of their final solution proposal.

In this article, we advocate to routinely use (anytime) *empirical runtime distributions* to assess the performance of optimization algorithms. We demonstrate how to directly compare the runtime distributions of algorithms that have

Manuscript received 18 September 2021; revised 16 February 2022 and 16 June 2022; accepted 14 September 2022. Date of publication 29 September 2022; date of current version 1 December 2022. This work was supported in part by the French National Research Agency (NumBBO) under Grant ANR-12-MONU-0009, and in part by the Slovenian Research Agency under Grant P2-0209 and Grant N2-0254. *(Corresponding author: Dimo Brockhoff.)*

Nikolaus Hansen, Anne Auger, and Dimo Brockhoff are with Inria and Ecole Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau, France (e-mail: dimo.brockhoff@inria.fr).

Tea Tušar is with the Department of Intelligent Systems, Jožef Stefan Institute, 1000 Ljubljana, Slovenia.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TEVC.2022.3210897.

[1]We are grateful to the anonymous reviewer pointing this out to us.

[2]An exploratory CPU timing experiment to get an estimate of the internal time complexity of the algorithm is still advisable, like it is prescribed in the comparing continuous optimizer (COCO) platform [20].

[3]We can stop the procedure when the last budget was not fully exhausted. Increasing the budget each time by a factor of $r > 1$ adds to the overall computational costs for the experimentation less than $\sum_{i=0}^{\infty} 1/r^i = r/(r-1)$ times the last consumed budget. For the performance assessment, always the data from the smallest eligible budget is used. The performance assessment will be too optimistic by tacitly assuming that the budget can be set properly without additional costs. On the other hand, runtimes may be overestimated (by less than a factor of r). A code example is provided in

An Outlier Noise Model

$$x \mapsto f(x) + B(x)N(x)$$

where

- $B = 1$ with probability p and 0 otherwise
- N may be either
  - additive heavy tail (bad outliers, $N \geq 0$) or
  - subtractive heavy tail (good outliers, $N \leq 0$)

Specifically

$$N \sim \pm \frac{\pi}{2} \left| \frac{\mathcal{N}(0,1)}{\mathcal{N}(0,1)} \right| \text{ which implies}$$

$$P(|N| > \alpha) \approx 1/\alpha \quad \text{when } \alpha \geq 5$$

Solvers

Python implementations from the Python Package Index PyPI accessible via `pip install ...` [`module` (downloads per week)]:

- `pdfo` (1.2K/w): Powell's Derivative-Free Optimization 5 solvers UOBYQA, NEWUOA, BOBYQA, LINCOA, and COBYLA

- `cma` (460K/w): CMA-ES (Covariance Matrix Adaptation Evolution Strategy), second order model, function-value-free and randomized (sampling based)

- `scipy.optimize` (87M/w): Nelder-Mead and SLSQP (Sequential Least Square Quadratic Programming)

Calling codes

- `pdfo` solvers: COBYLA, UOBYQA, BOBYQA, LINCOA, NEWUOA

```
import pdfo
methods = ['cobyla', 'uobyqa', 'bobyqa', 'lincoa', 'newuoa']
for method in methods:
    finaltrustradius = 1e-15
    res = pdfo.pdfo(problem, problem.initial_solution_proposal(),
            method=method,
            options={'radius_final':finaltrustradius,
            'maxfev':budget_multiplier*problem.dimension})
```

- `scipy.optimize`: NelderMead

```
from scipy import optimize as opt

res = opt.fmin(problem, problem.initial_solution_proposal(),
        xtol = 1e-10, ftol = 1e-11,
        maxfun = problem.dimension * budget_multiplier,
        full_output = True)
```

- `scipy.optimize`: slsqp

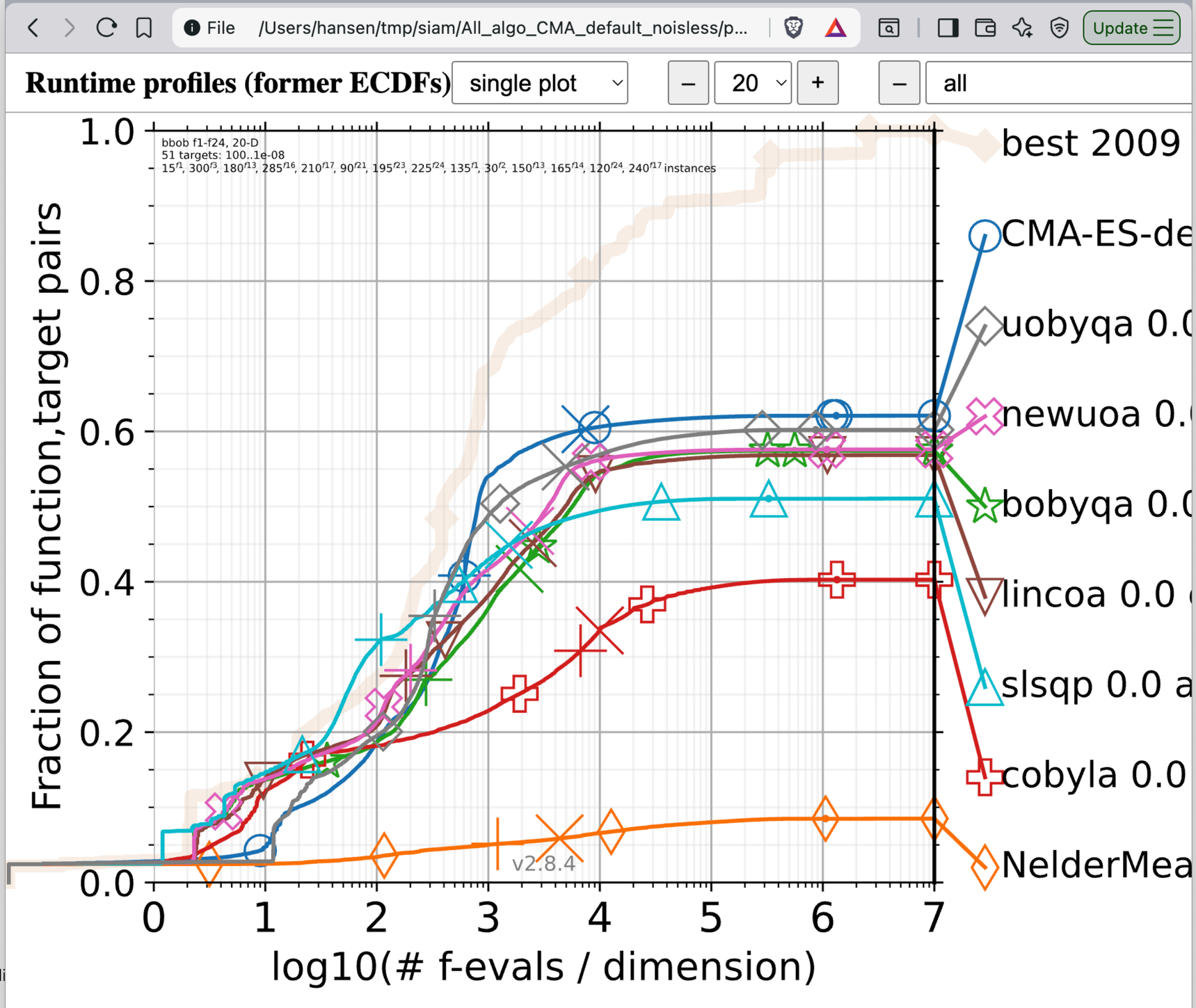
```
from scipy import optimize as opt

res = opt.fmin_slsqp(problem, problem.initial_solution_propo
        acc=1e-11,
        full_output=True, iprint = -1)
```

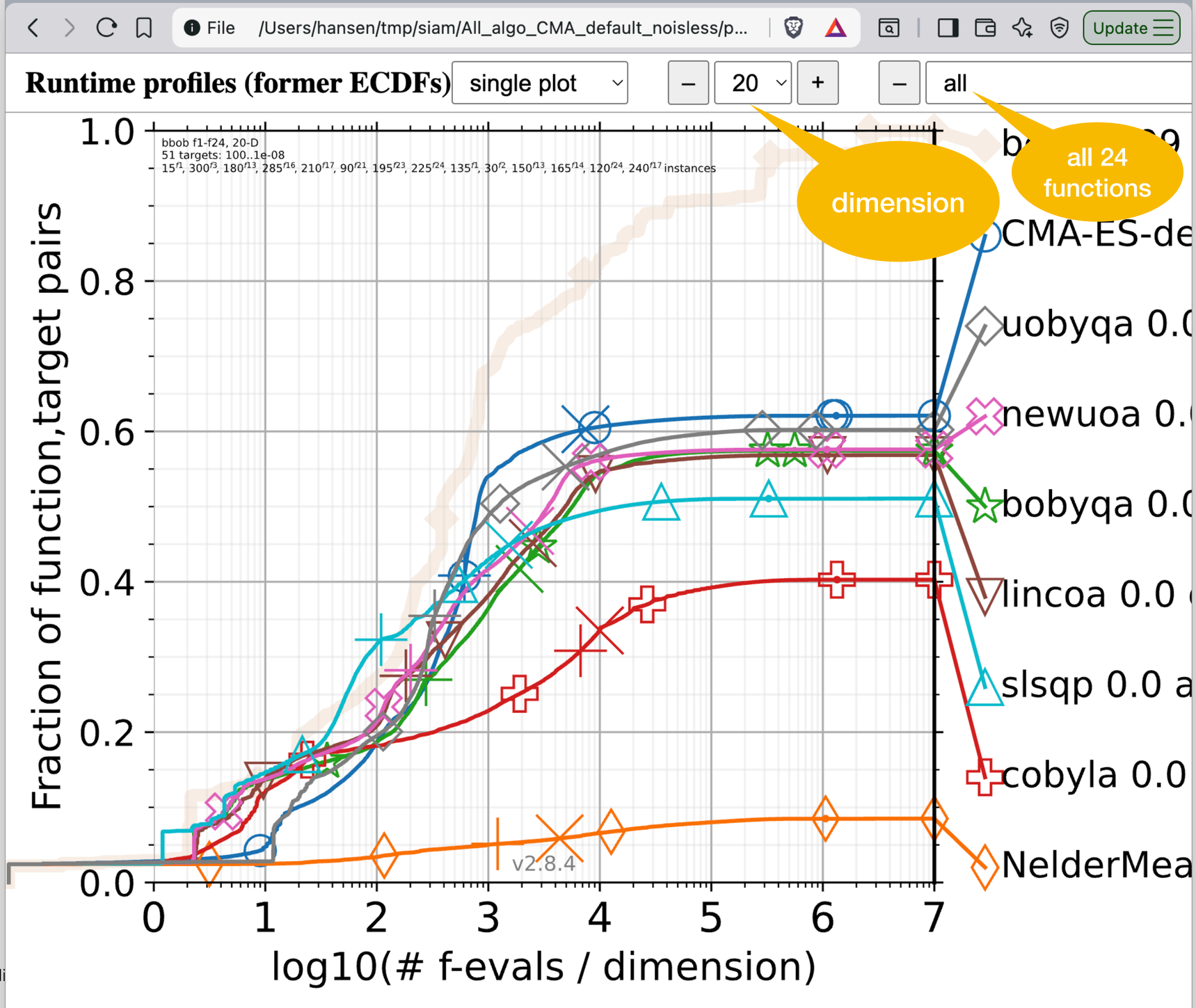
- `pycma`: CMA-ES

```
from cma import fmin2 as fmin

xopt, es = fmin(problem, problem.initial_solution_proposal,
        2, options={
        'maxfevals': 8 * problem.dimension * budget_multipli
        'termination_callback': lambda es: problem.final_tar
        'conditioncov_alleviate': None})
```

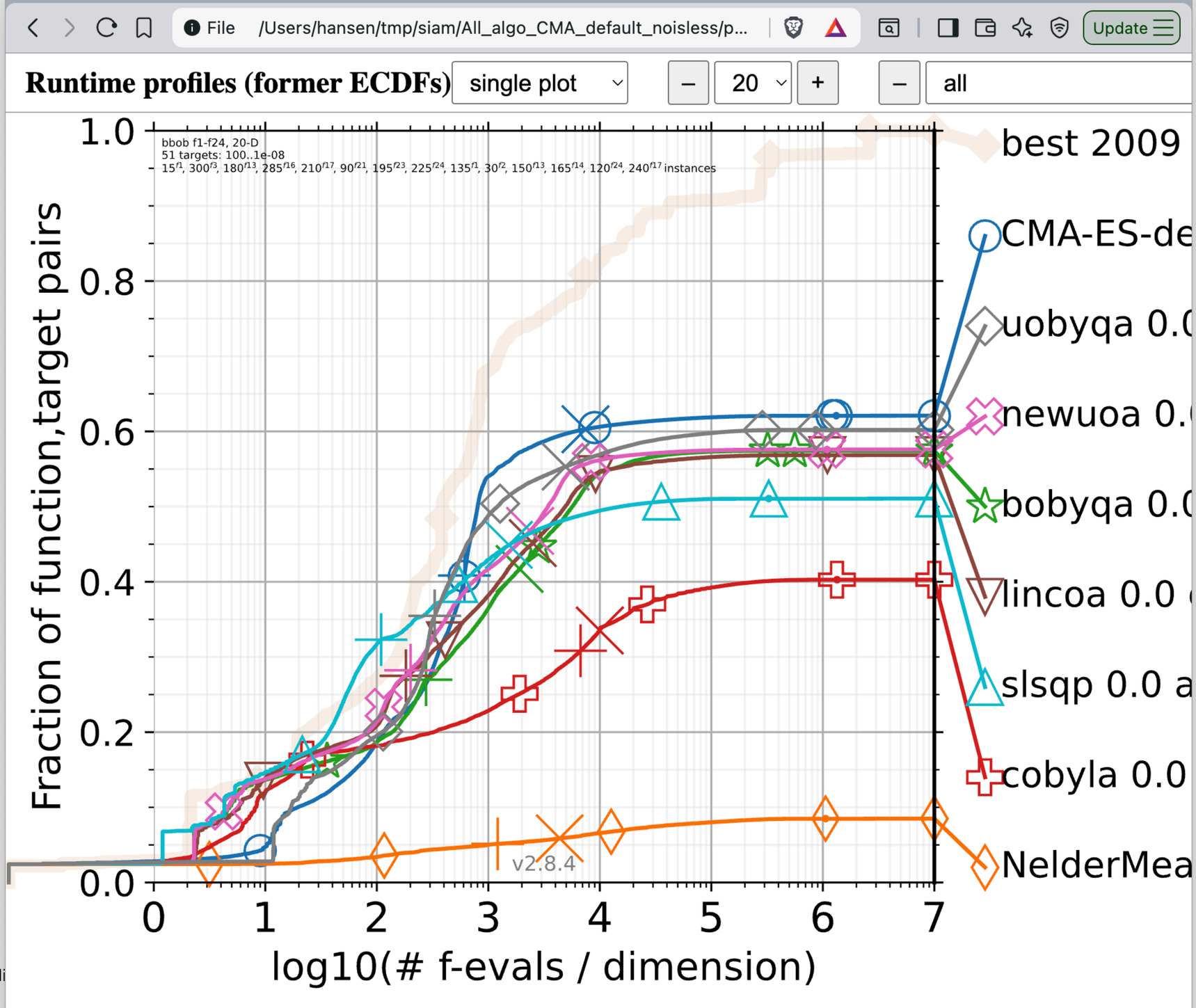


Akin to "data profiles", we show the empirical cumulative distribution function (ECDF) of the **number of evaluations** of f (**runtimes**) to reach *a variety* of selected target f-values



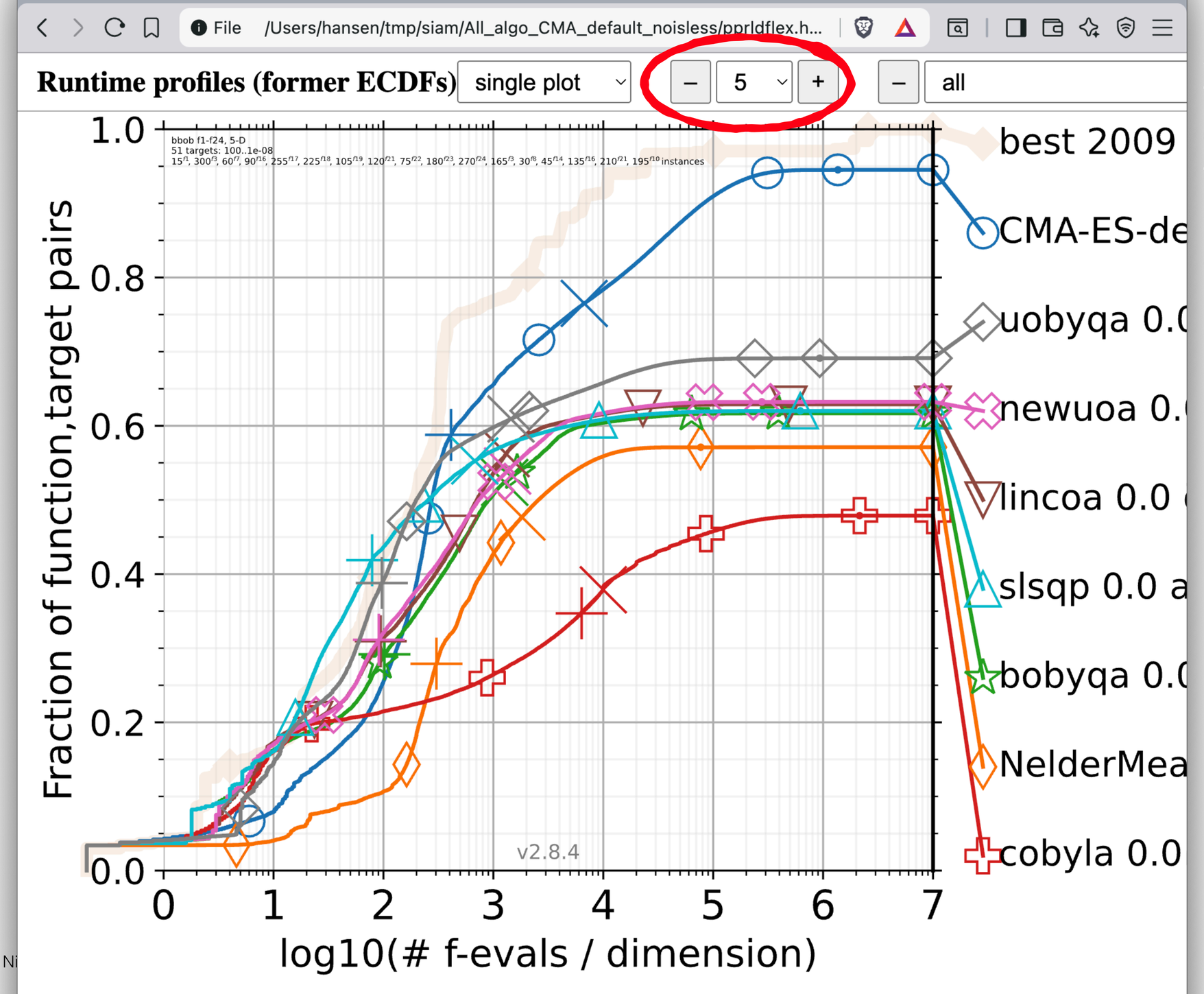
Akin to "data profiles", we show the empirical cumulative distribution function (ECDF) of the **number of evaluations** of f (**runtimes**) to reach *a variety* of selected target f-values

- horizontal + crosses indicate the 90%tile of single runs
- × crosses indicated the 90%tile of expected runtime by instance with repetitions (restarts)
- data to the right of horizontal + crosses represent a restarted solver scheme



Akin to "data profiles", we show the empirical cumulative distribution function (ECDF) of the **number of evaluations** of f (**runtimes**) to reach *a variety* of selected target f-values

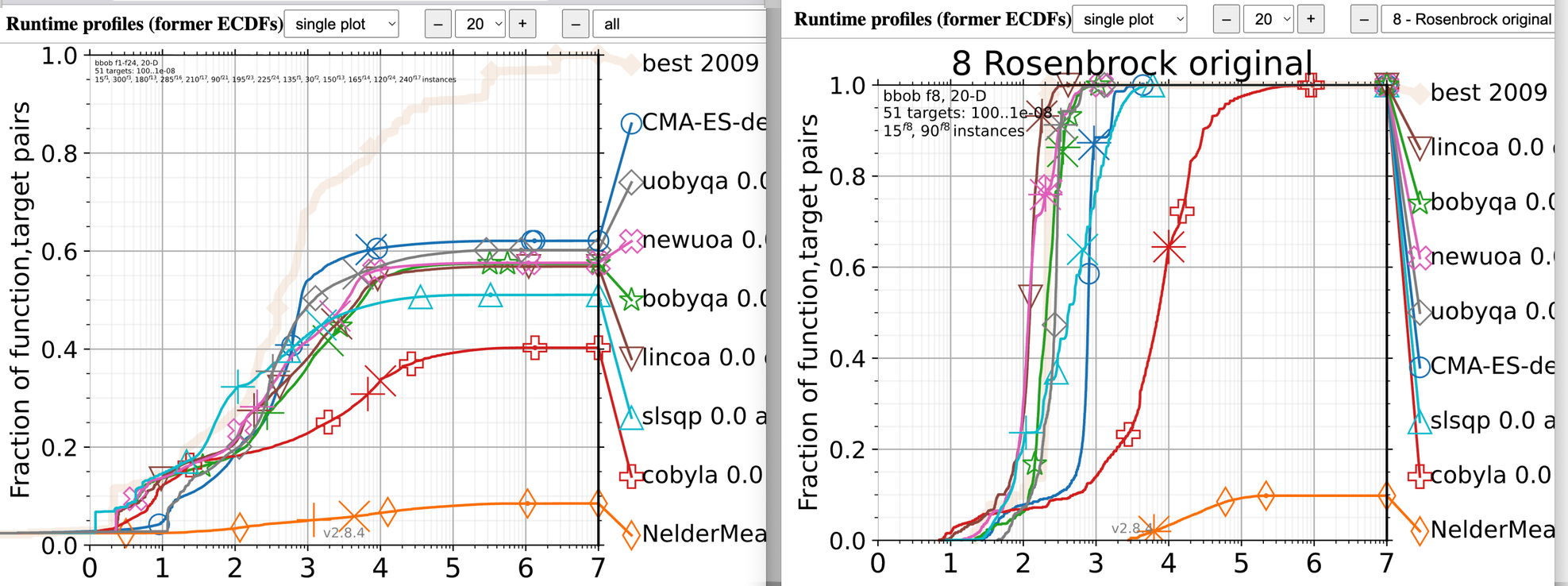
- Nelder-Mead is not at all competitive when $D > 5$
- COBYLA performs inferior (linear approximation is insufficient)
- SLSQP performs up to 4 times faster up to $300D$ evaluations
- UOBYQA and CMA-ES are slower in the beginning and faster on the 20% most difficult problems they solve



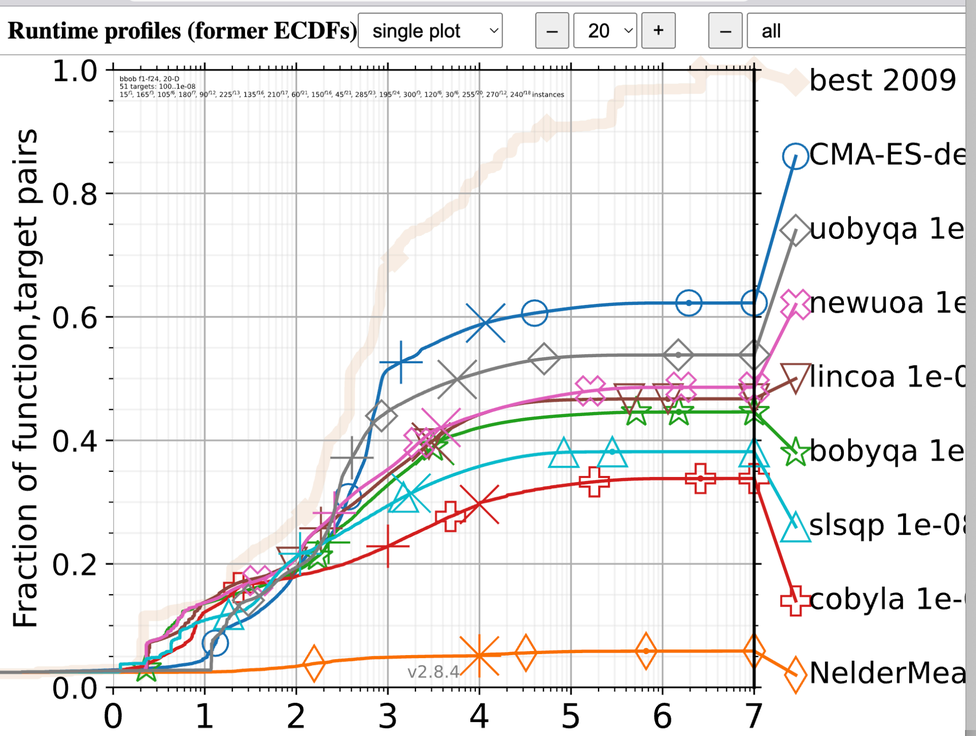
Akin to "data profiles", we show the empirical cumulative distribution function (ECDF) of the **number of evaluations** of f (**runtimes**) to reach *a variety* of selected target f-values

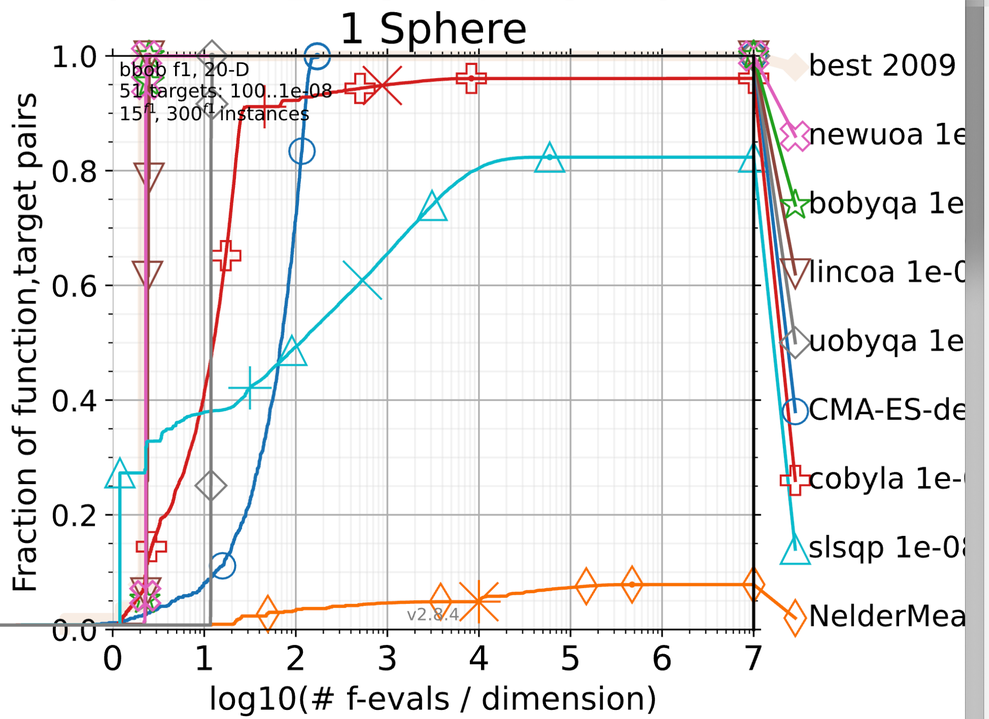
- Nelder-Mead is almost competitive when $D = 5$
- COBYLA performs inferior (linear approximation is insufficient)
- SLSQP performs best up to $200D$ evaluations
- CMA-ES solves ultimately the most problems beyond a budget of 500D evaluations, but is 5 times slower than SLSQP in the beginning

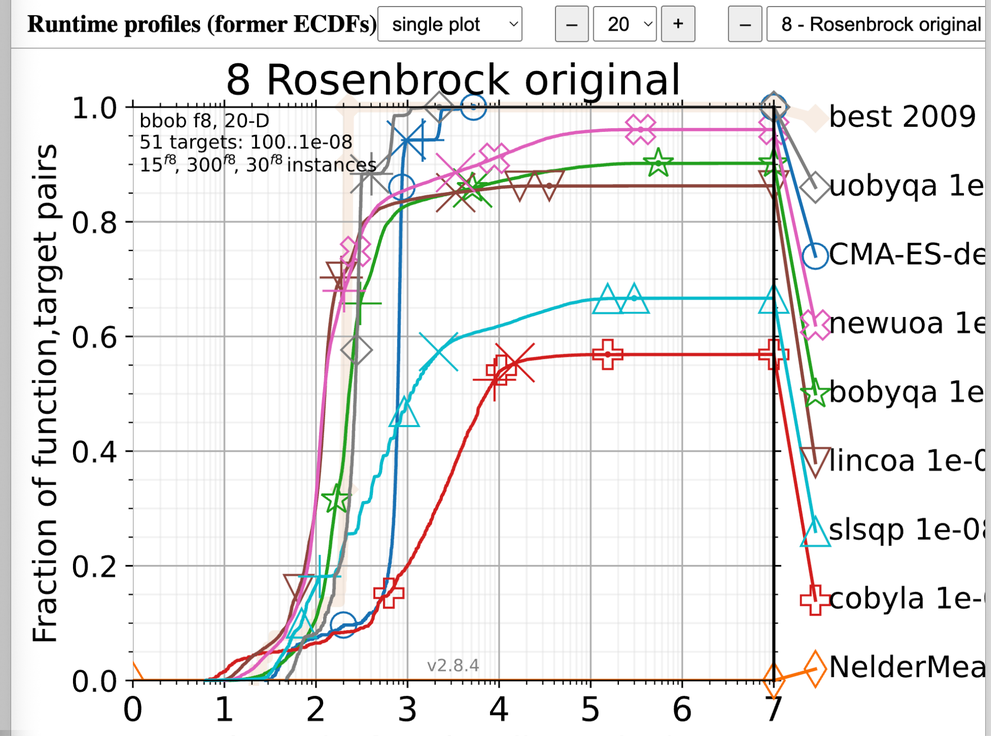
Additive Gaussian noise, 20-D, varying variance

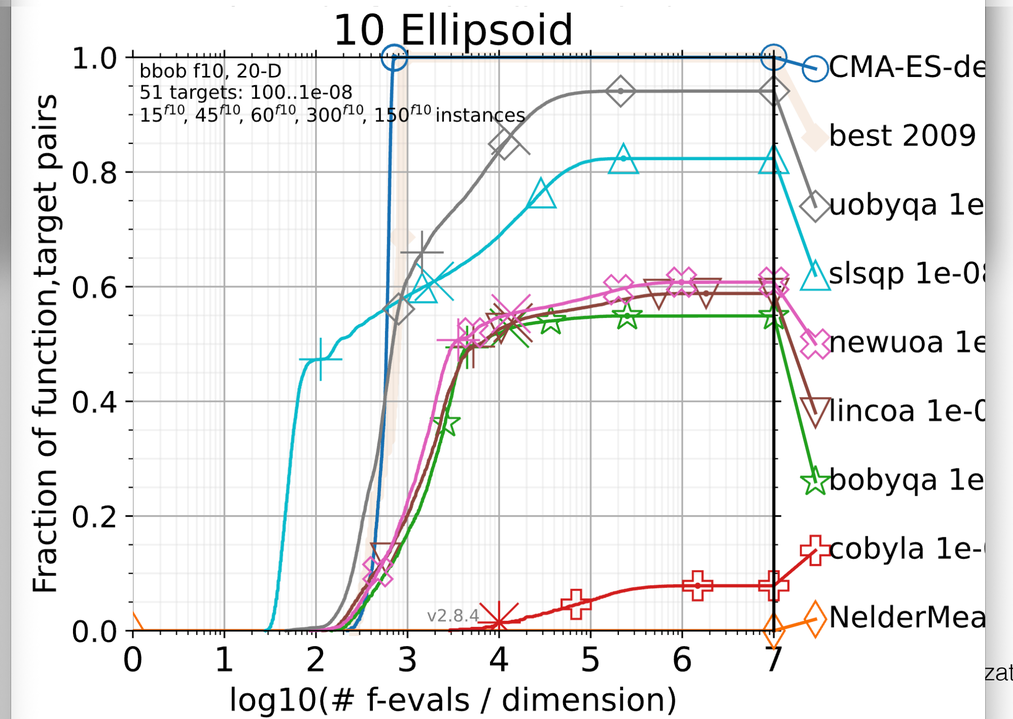


$$f + 0 \times \mathcal{N}(0,1)$$

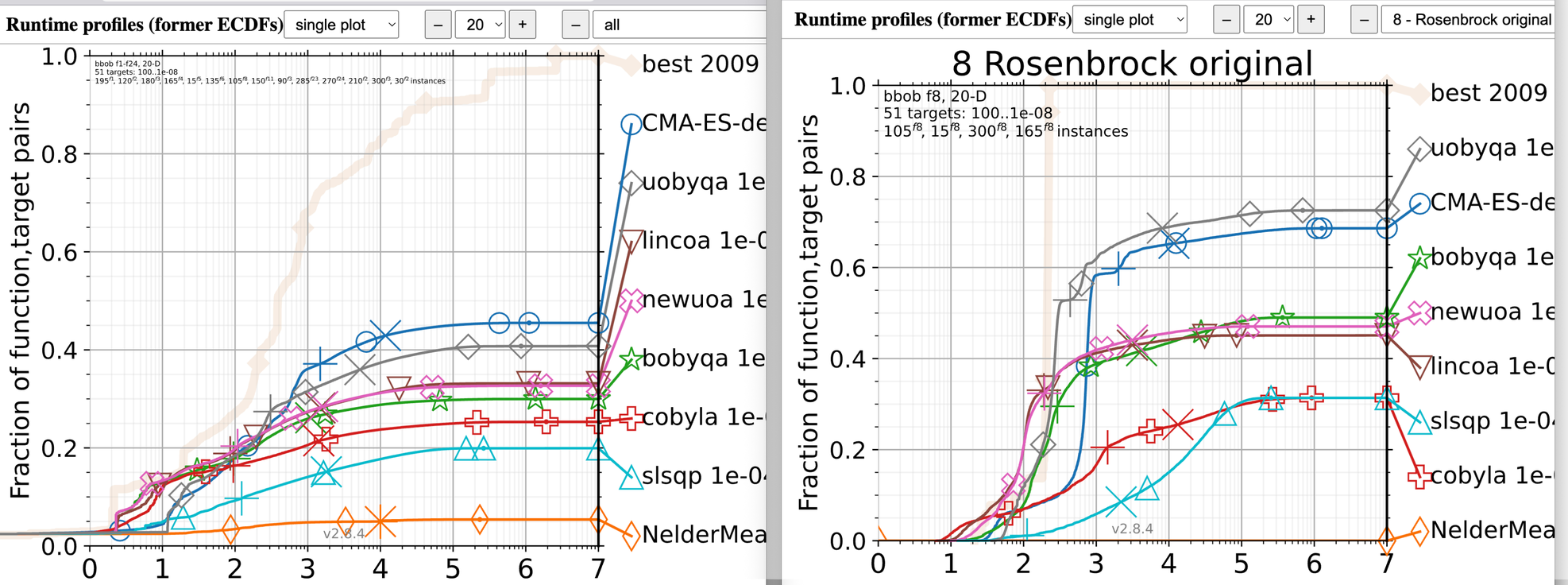


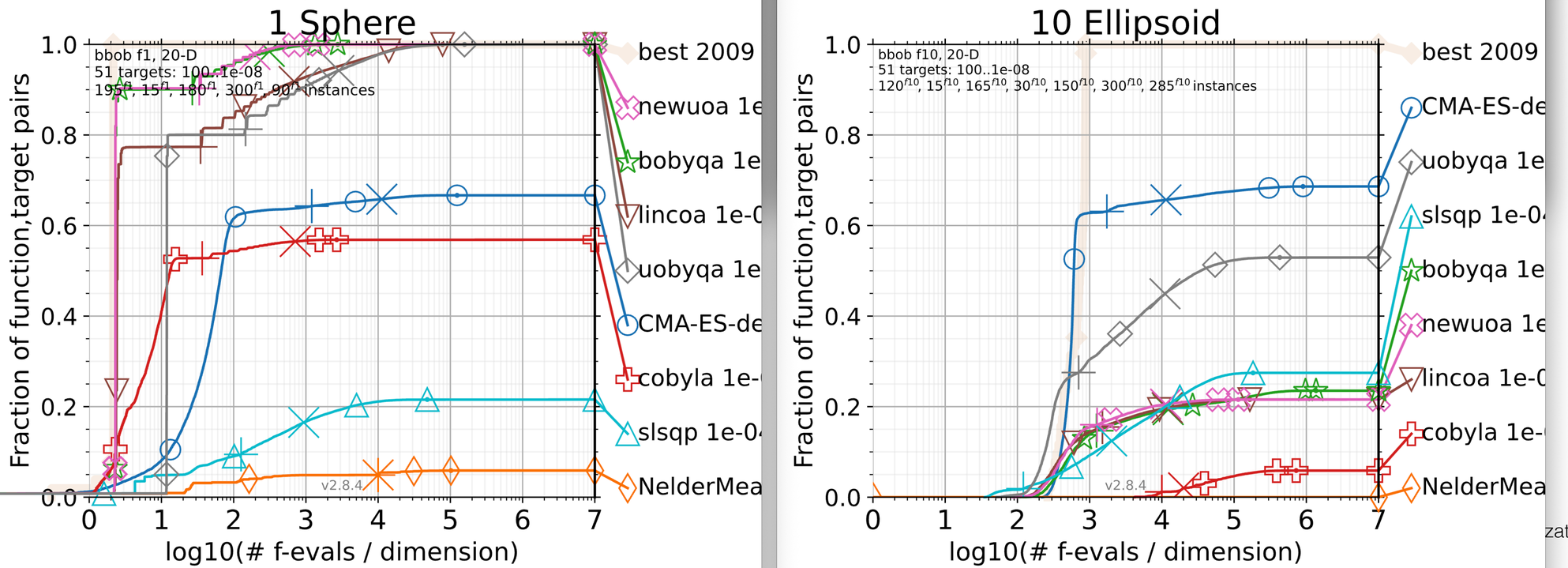






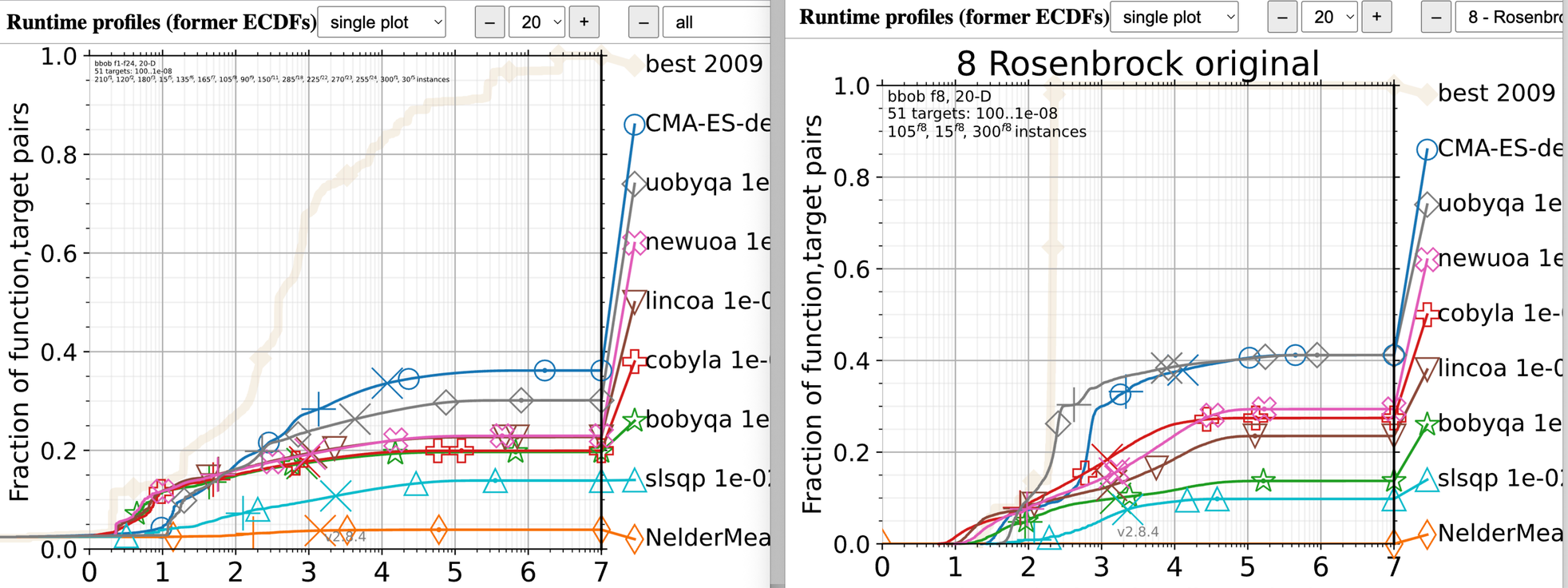
$$f + 10^{-8}\mathcal{N}(0,1)$$

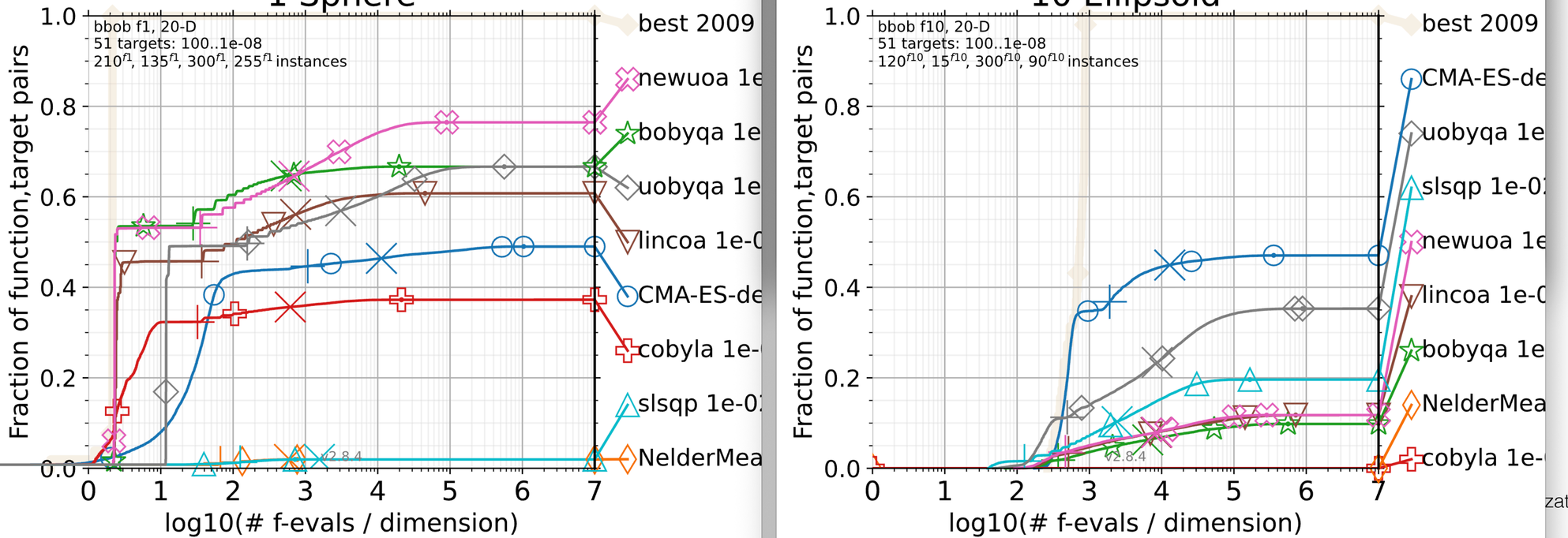




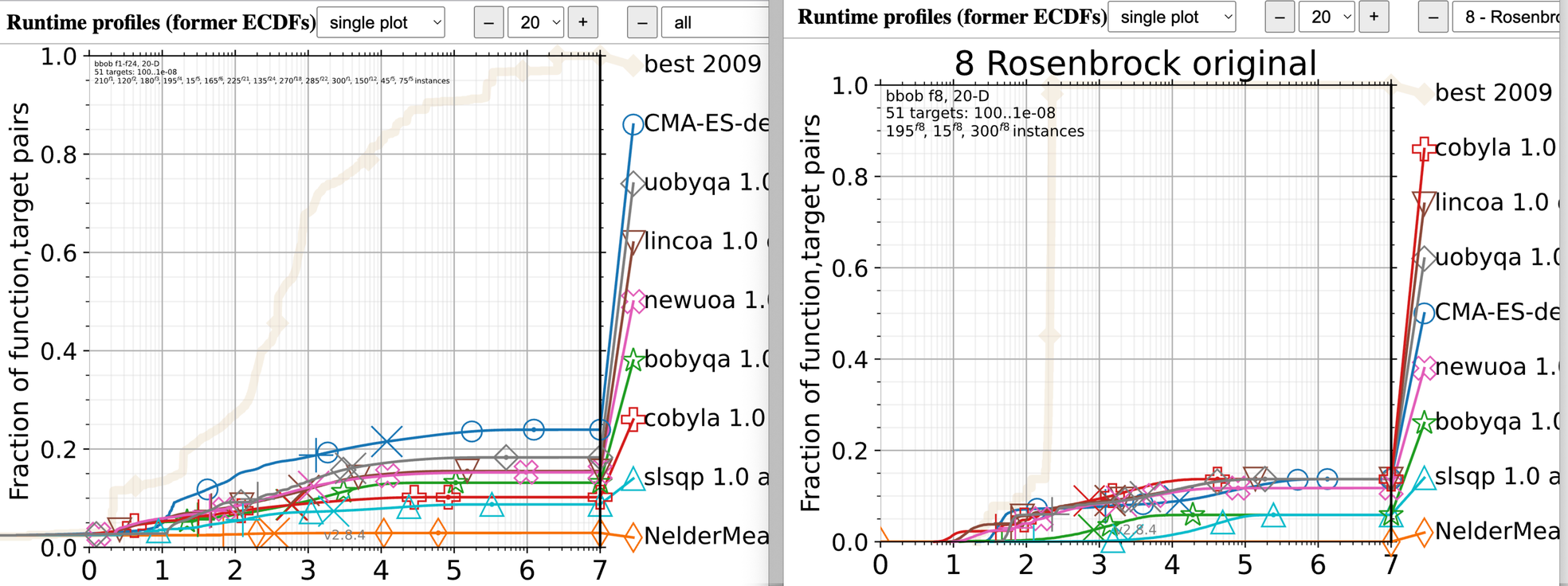
$$f + 10^{-4}\mathcal{N}(0,1)$$

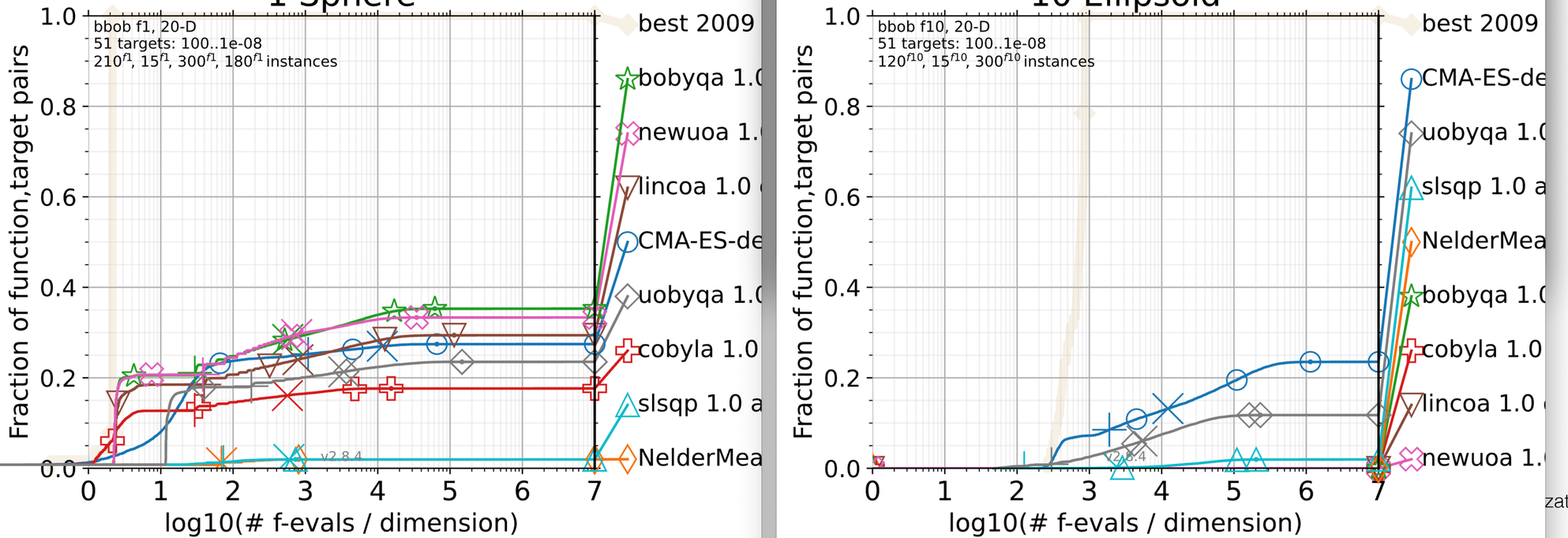
zation Performance Under Outlier Noise Models





$$f + 10^{-2}\mathcal{N}(0,1)$$

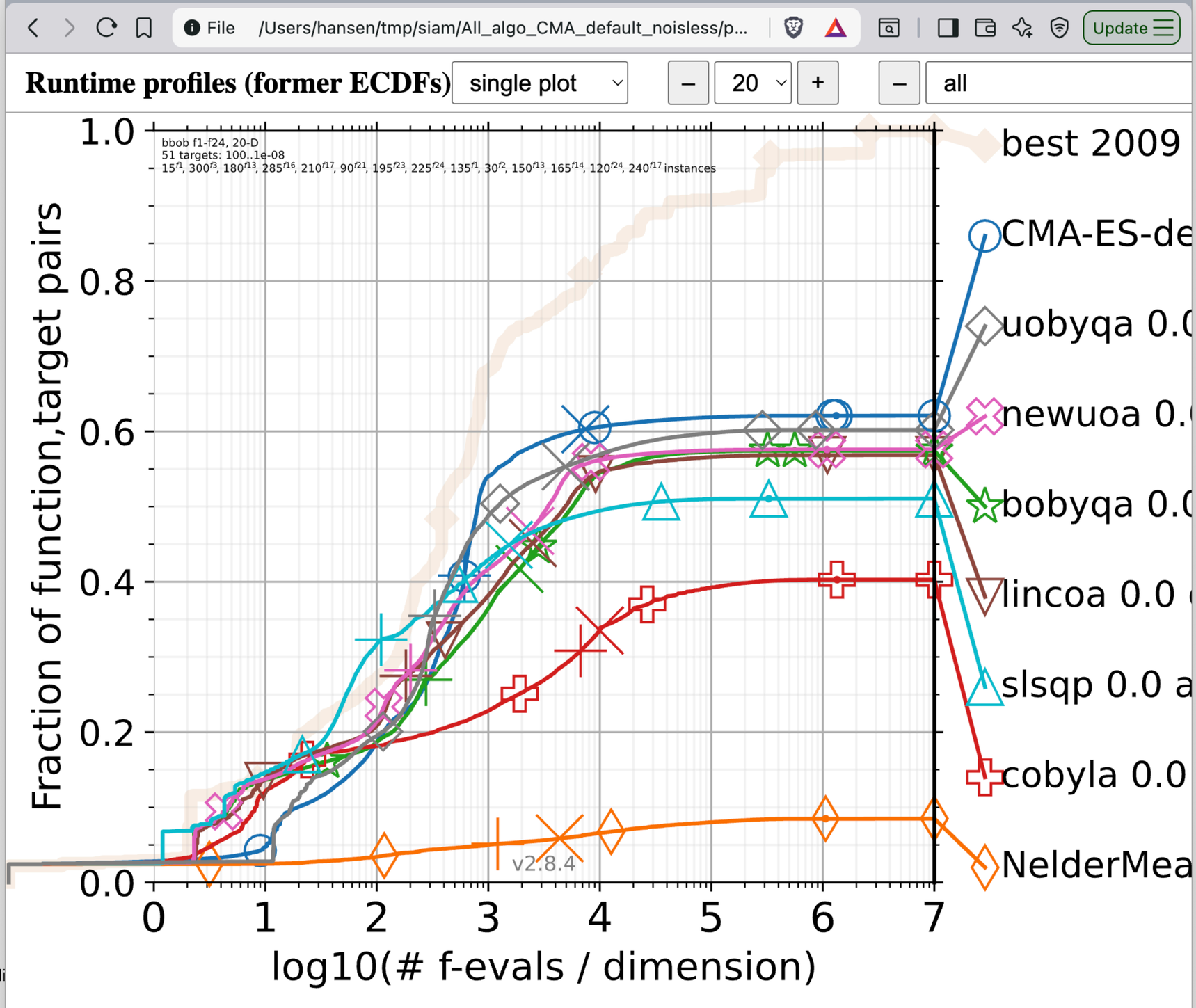


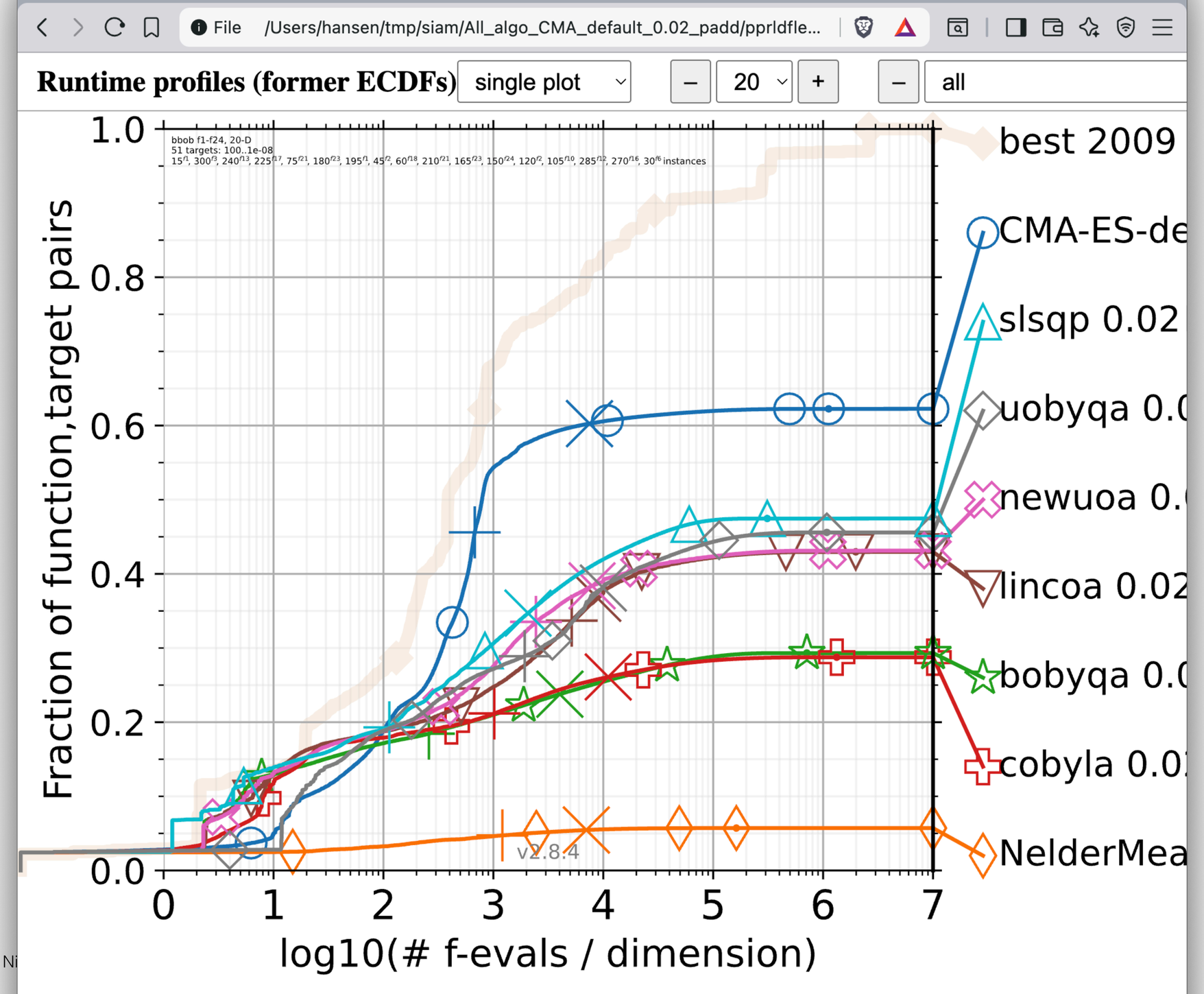


$$f + \mathcal{N}(0,1)$$

Bad outliers, 2%, 10%, 40%, 80%, 20-D

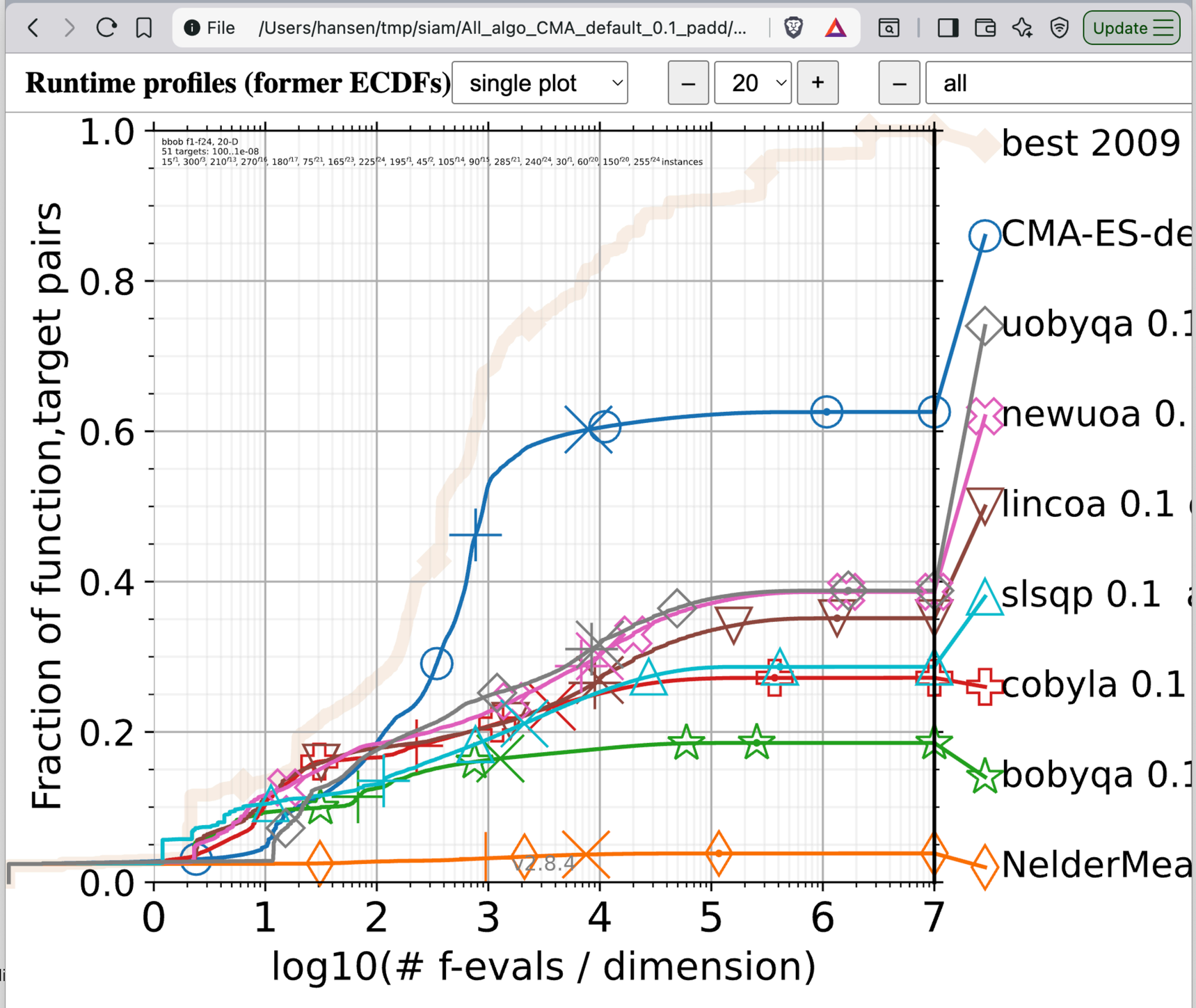
Noiseless


File /Users/hansen/tmp/siam/All_algo_CMA_default_noisless/p...
Update
Runtime profiles (former ECDFs)
single plot
20
all
bbob f1-f24, 20-D
51 targets: 100..1e-08
15^f1, 300^f3, 180^f13, 285^f16, 210^f17, 90^f21, 195^f23, 225^f24, 135^f1, 30^f2, 150^f13, 165^f14, 120^f24, 240^f17 instances
Fraction of function,target pairs
log10(# f-evals / dimension)
v2.8.4
best 2009
CMA-ES-de
uobyqa 0.0
newuoa 0.0
bobyqa 0.0
lincoa 0.0
slsqp 0.0 a
cobyla 0.0
NelderMea



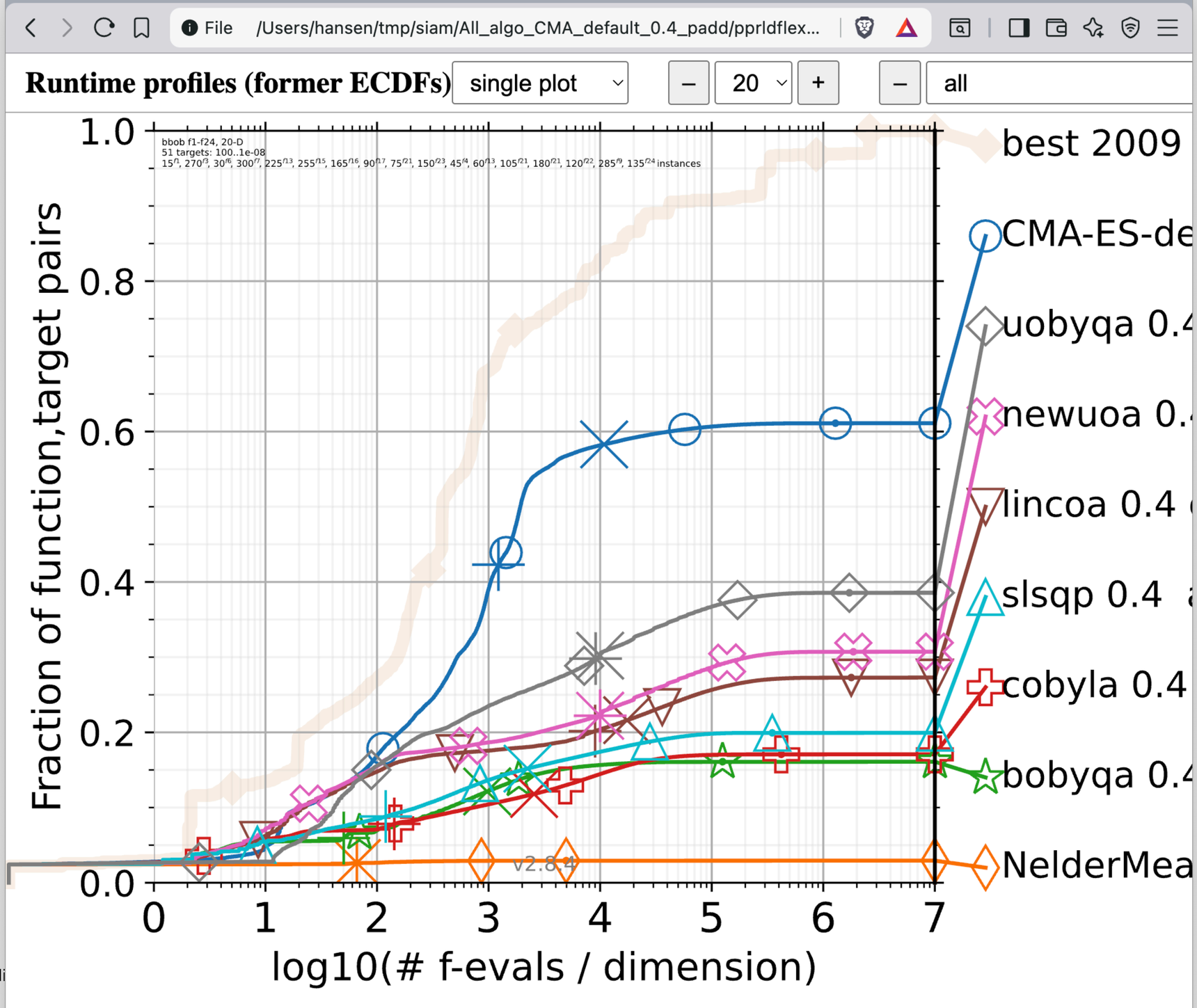
2% bad outliers

- only CMA-ES is unaffected
- all other competitive solvers are notably affected
- SLSQP consistently outperforms all solvers but CMA-ES



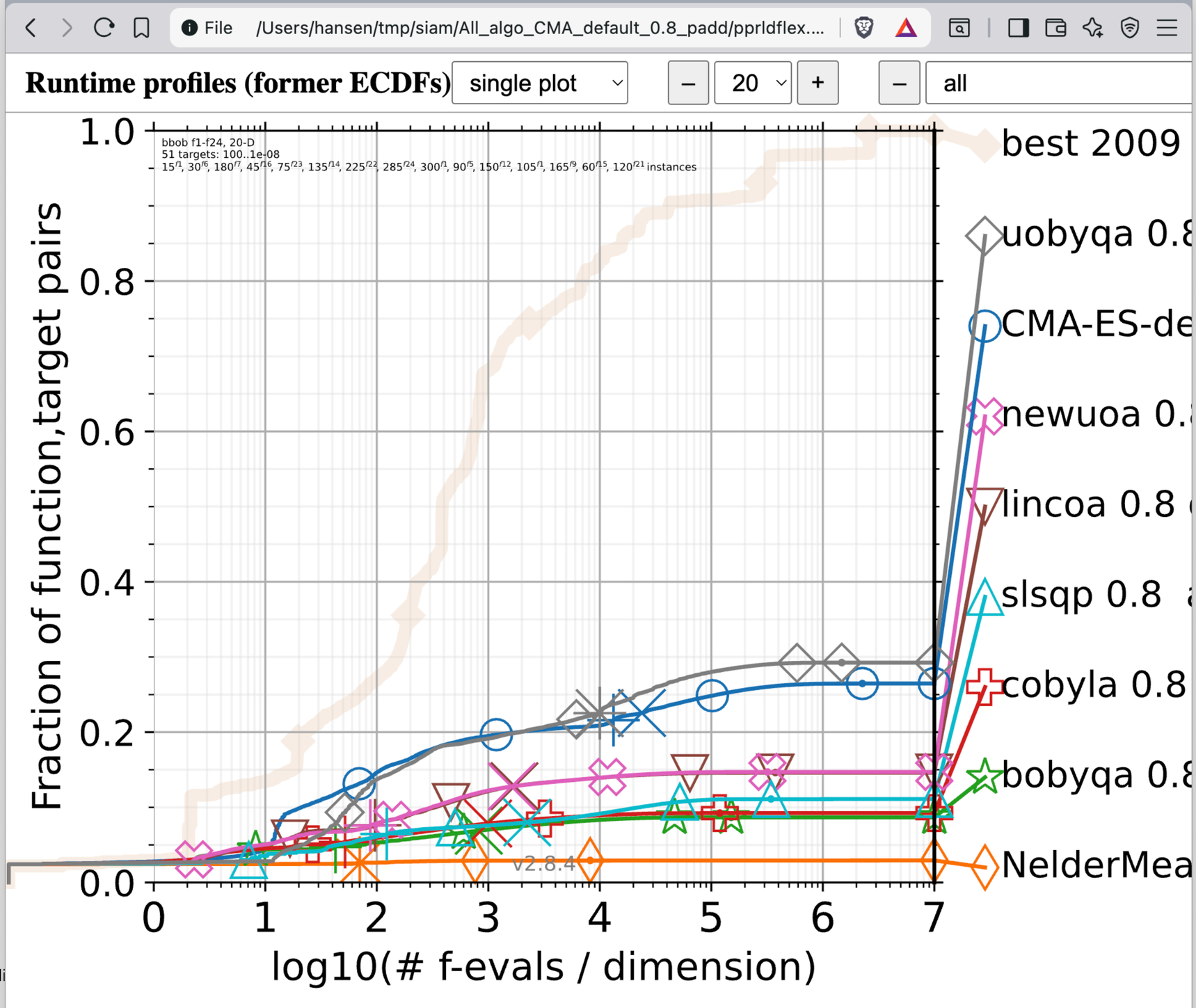
10% bad outliers

- only CMA-ES is virtually unaffected
- all other competitive solvers are notably affected



40% bad outliers

- CMA-ES gets about 2 times slower on the more difficult problems
- UOBYQA is the least affected of the remaining solvers
- The performance difference between 2% and 40% bad outliers is, generally, surprisingly small

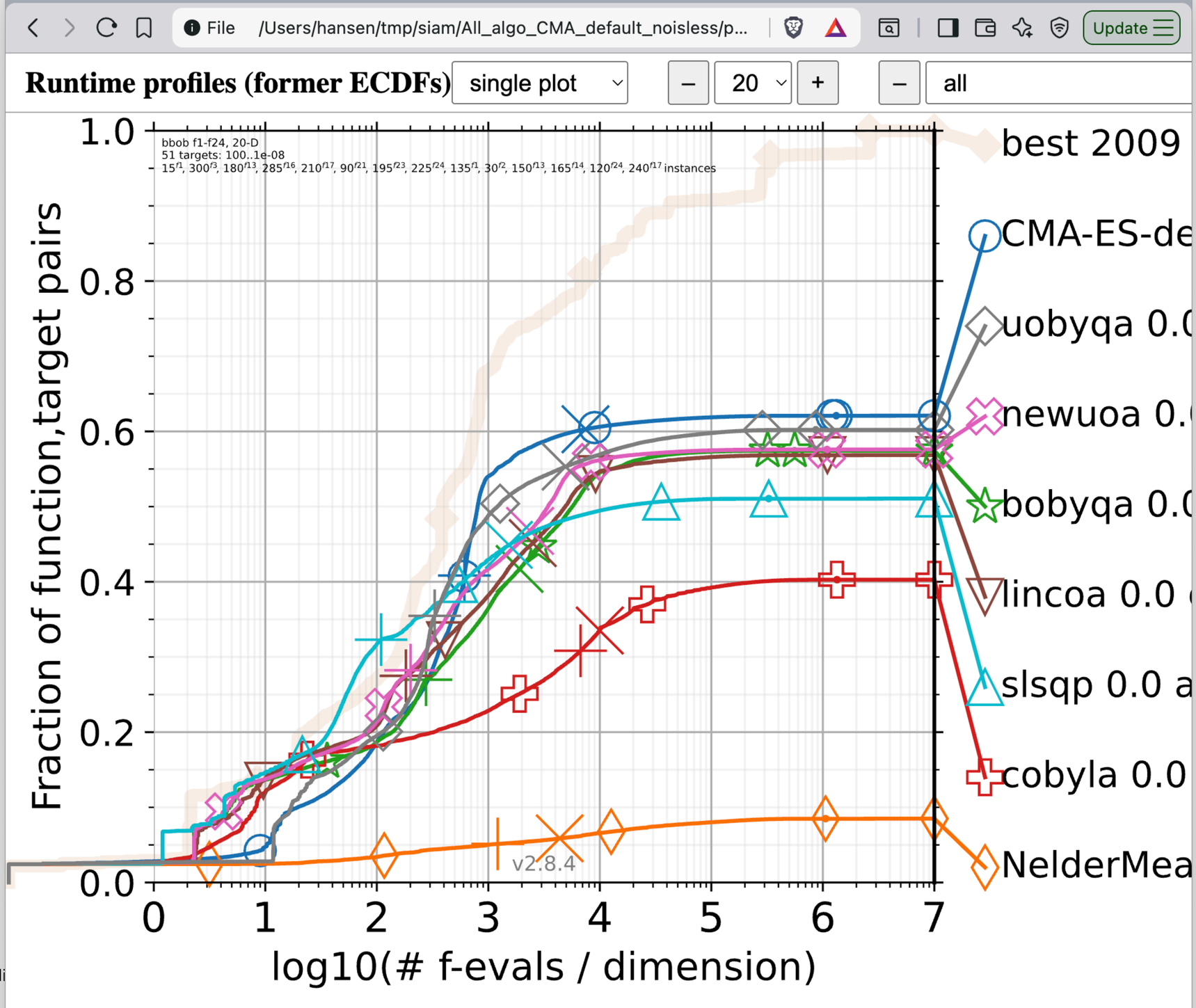


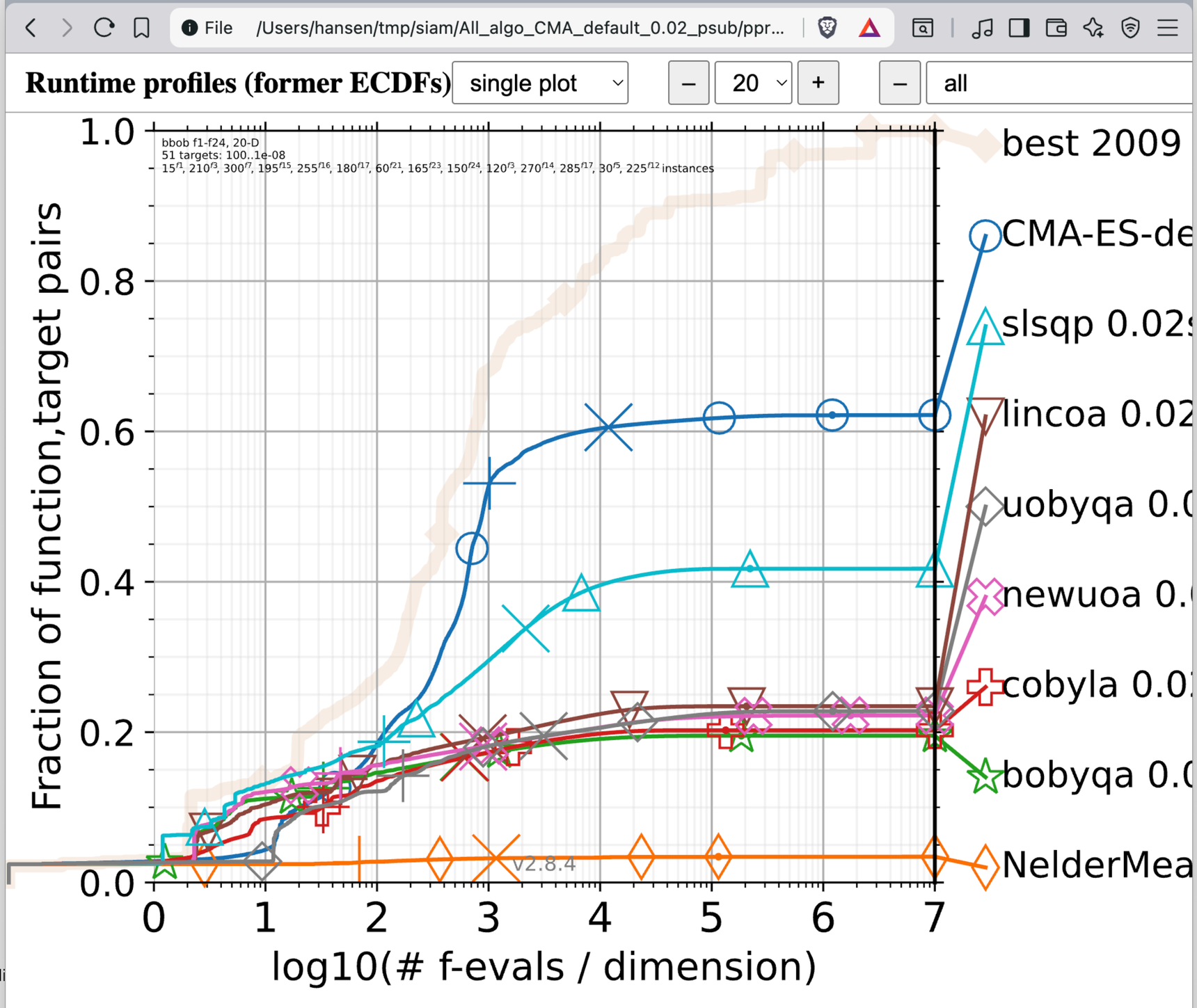
80% bad outliers

- all solvers are heavily affected, the order remains roughly intact

Good outliers, 2%, 10%, 20-D

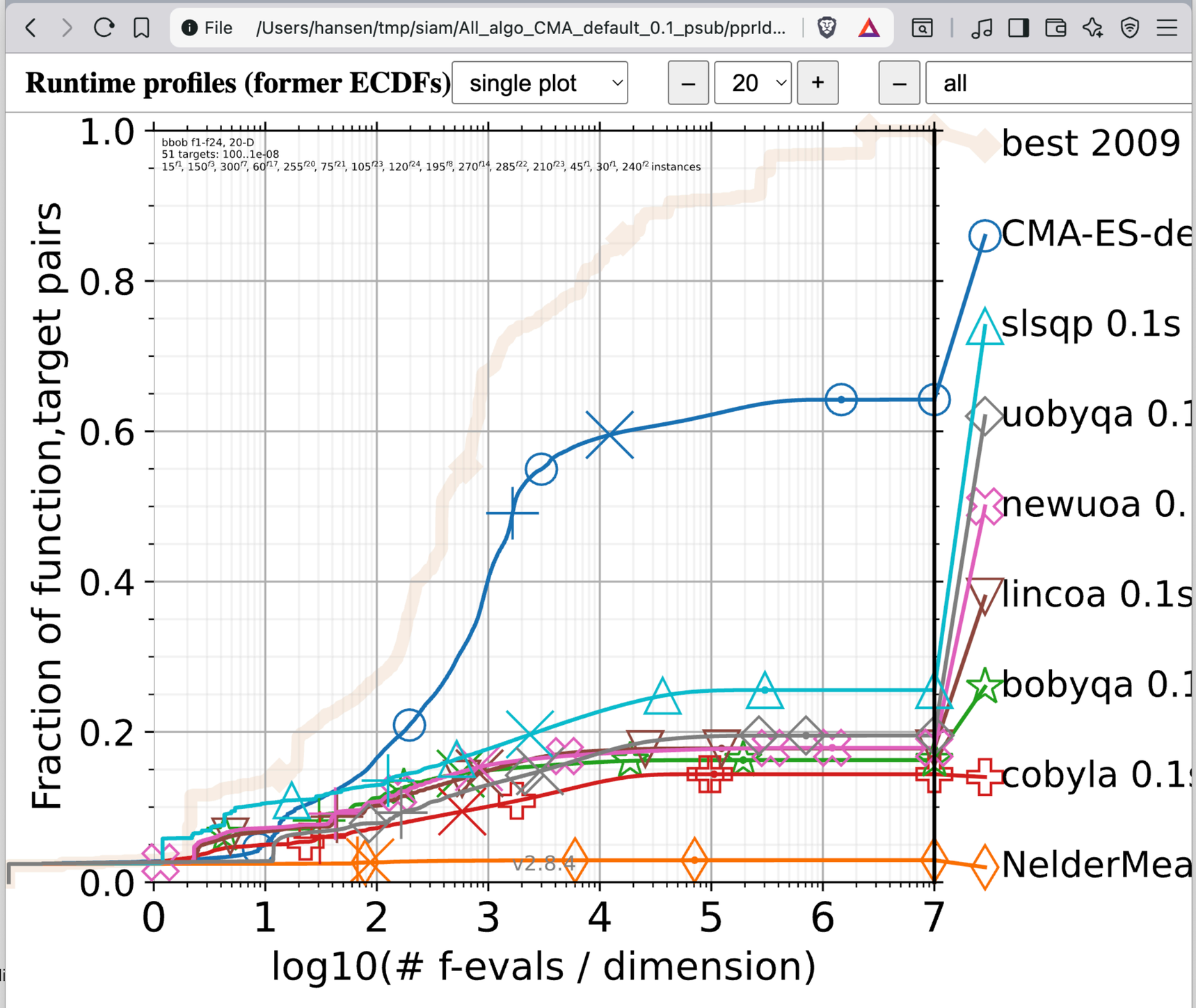
Noiseless


File /Users/hansen/tmp/siam/All_algo_CMA_default_noisless/p...
Update
Runtime profiles (former ECDFs)
single plot
20
all
bbob f1-f24, 20-D
51 targets: 100..1e-08
15^f1, 300^f3, 180^f13, 285^f16, 210^f17, 90^f21, 195^f23, 225^f24, 135^f1, 30^f2, 150^f13, 165^f14, 120^f24, 240^f17 instances
Fraction of function,target pairs
log10(# f-evals / dimension)
v2.8.4
best 2009
CMA-ES-de
uobyqa 0.0
newuoa 0.0
bobyqa 0.0
lincoa 0.0
slsqp 0.0 a
cobyla 0.0
NelderMea



2% good outliers

- CMA-ES is virtually unaffected
- SLSQP is as affected as with bad outliers
- The remaining solvers are *more* affected by good outliers than by bad outliers



10% good outliers

- CMA-ES gets about 1.5-2 times slower
- SLSQP is as affected as with bad outliers
- The remaining solvers are *more* affected by good outliers than by bad outliers

Summary of Observations (20-D)

- SLSQP is the most affected by noise, bad and good outliers have similar effect
- CMA-ES is the **least affected by noise**, in particular with outlier noise, and the least with bad outliers
- UOBYQA, when compared to the other `pdfo` methods, is
  - generally, slower for easier and faster for more difficult problems
  - somewhat less affected by bad outliers and Gaussian noise