Benchmarking and analyzing iterative optimization heuristics with IOHprofiler

Carola Doerr, Hao Wang, Diederick Vermetten, Thomas Bäck, Jacob de Nobel, Furong Ye

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA
© 2022 Copyright is held by the owner/author(s).
ACM ISBN 978-1-4503-9268-6/22/07.
https://doi.org/10.1145/3520304.3533665

The most up-to-date version of these slides can be found at: https://github.com/IOHprofiler/GECCO-Tutorial

Introduction

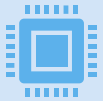
- Benchmarking is a key component in the field of optimization algorithms
- Need data to judge effectiveness of a new algorithm relative to state-of-the-art
- But benchmarking is not just a number showing algorithm A is better than algorithm B!

Benchmarking

Benchmarking can be used to gain important insights about algorithms and problems

Highlights interplay between problem and algorithm

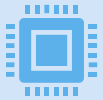
Differences in performance show potential for new developments



Data-driven studies of algorithm selection, configuration, dynamic switching...

Requirements for benchmarking

Ease of access

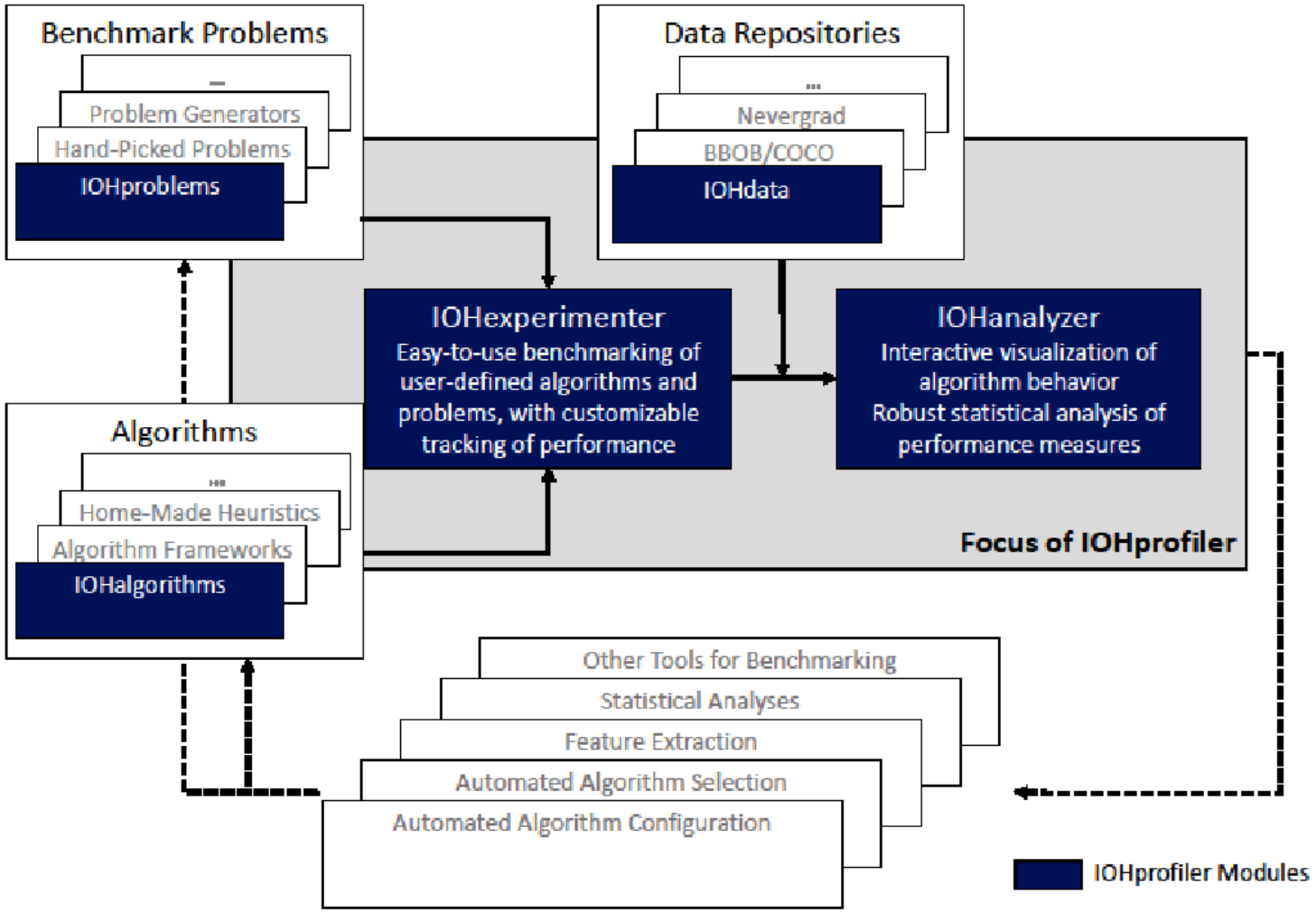


Flexible to the specific requirements of the user

Interoperability with other common tools

Standardized to ensure reproducibility

IOHprofiler Architecture Overview

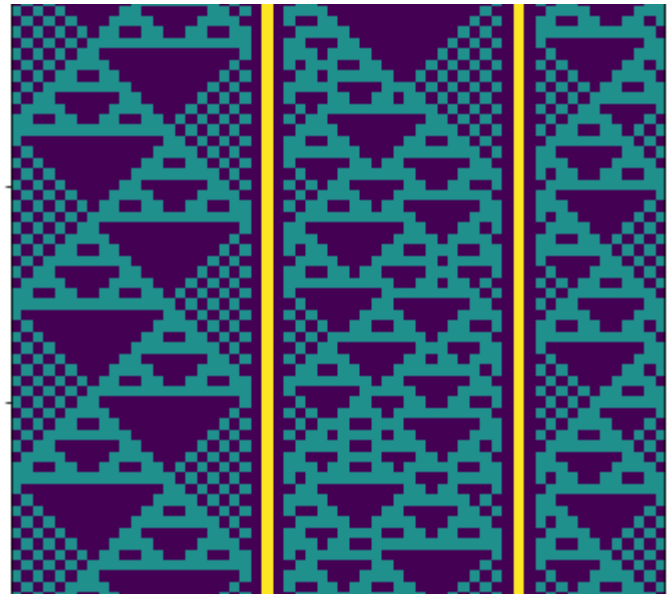


Teaching

- Benchmarking is a core component of any study of optimization algorithms
- So, should be accessible to students
- Introducing benchmarking concepts to students can be challenging
- IOHexperimenter helps by providing a fixed framework with pre-existing examples
- Interactivity of IOHanalyzer enables non-expert programmers to contribute to understanding of algorithm performance

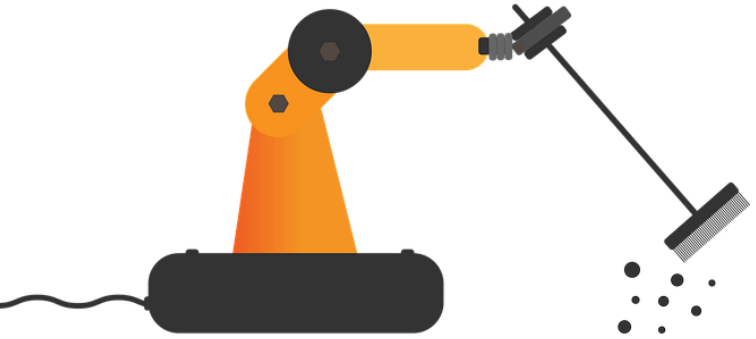
Teaching

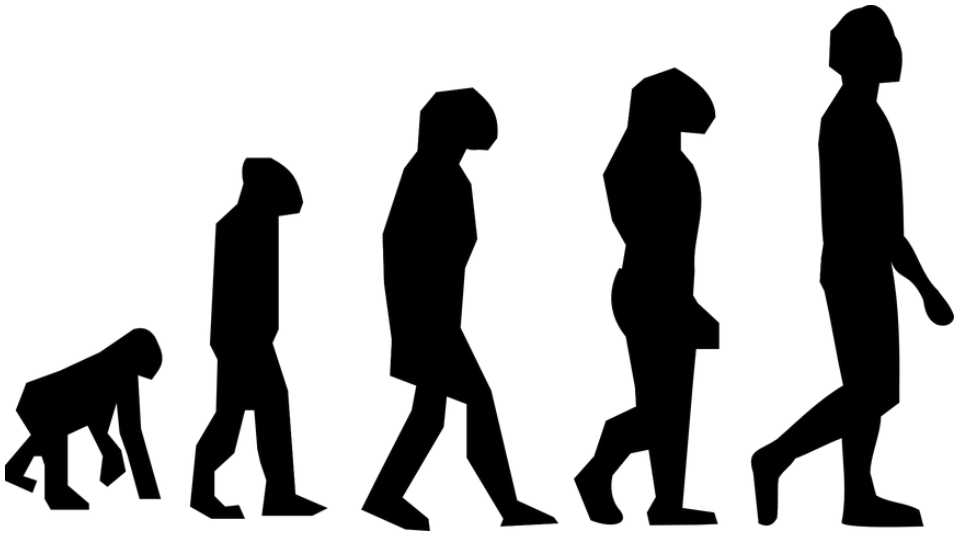
- Some examples of benefits we found while teaching:
  - Easily extending objective functions enables more wide variety of types of problems (e.g. inverting rules of cellular automata for a bachelor course on natural computing)
  - The common data format and many-algorithm analysis options enables for some degree of competitive algorithm design (e.g. a bonus point for the top 10% algorithms based on area under aggregated ECDF)
  - Common structure allows easy reproducibility of submissions + enables the creation of simple test-functions so students can check their algorithm for minimal acceptable performance



Student Projects

- Within our group, IOHprofiler is commonly used in both Bsc. and Msc. thesis projects
- Significantly reduces the overhead of setting up correct benchmarking practices and common visualizations
- Makes transitioning to scientific papers easier, since the benchmarking pipeline is correct



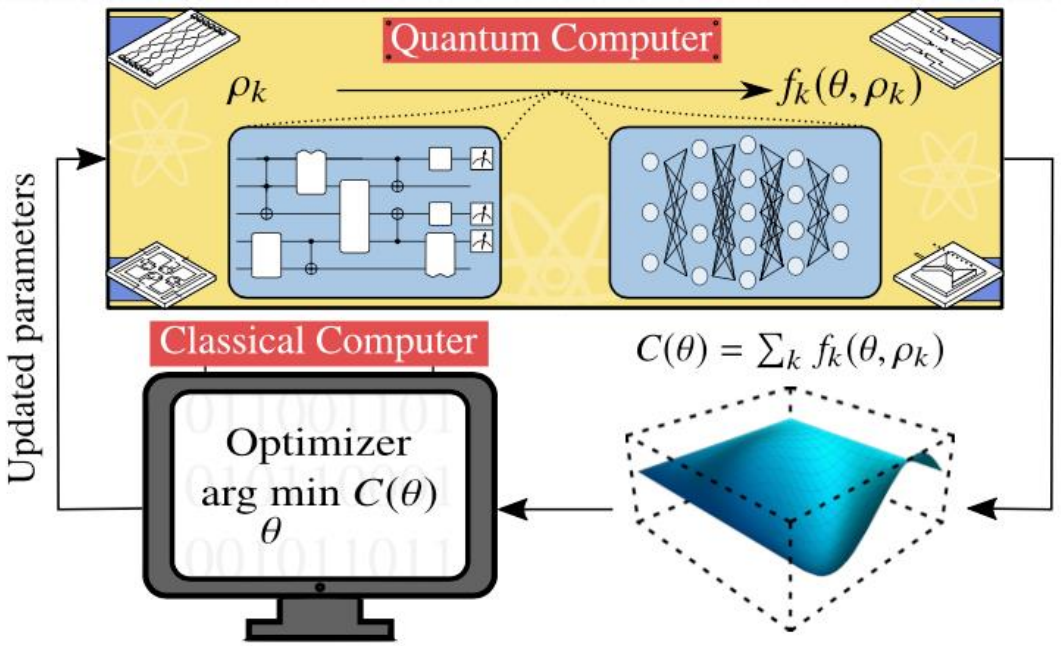


Ontology

- Benchmark data contains a lot of information
- But not always easy to extract
- Collaboration with OPTION: data ontology for iterative optimization
- Annotated data supports wide variety of queries
- Current prototype interface enables loading data by study name
- Significant potential to implement many more easy to use query types

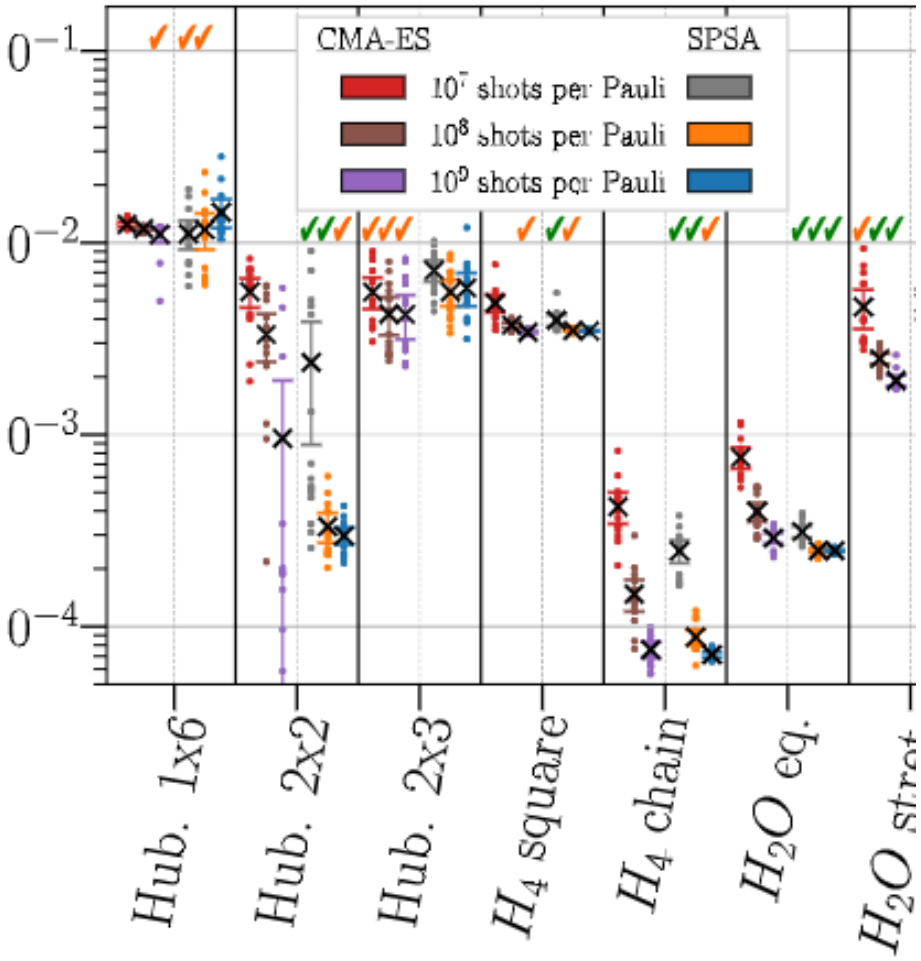
Quantum Computing

- **Variational quantum algorithms** (VQAs) need classical optimization algorithms
  1. Prepare a quantum state using a parameterized quantum circuit
  2. Measure the cost value of the quantum state
  3. Optimize the parameter of the circuit using classical optimizers



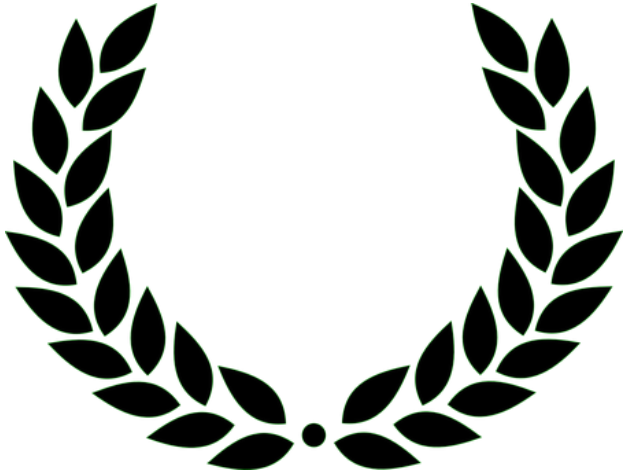
Quantum Computing

- Lack of knowledge on which optimizer we should use for VQAs; some commonly used:
  - CMA-ES
  - COBYLA
  - SPSA (Simultaneous perturbation stochastic approximation)
  - Adam
  - …
- Benchmarking and performance analysis via IOHprofiler



IOHanalyzer overview

- Performance can not be captured in a single number
- Analysis of performance on a benchmark can be very context-dependent
- To a large extent influenced by the specific requirements of the user
- As such, flexibility is key



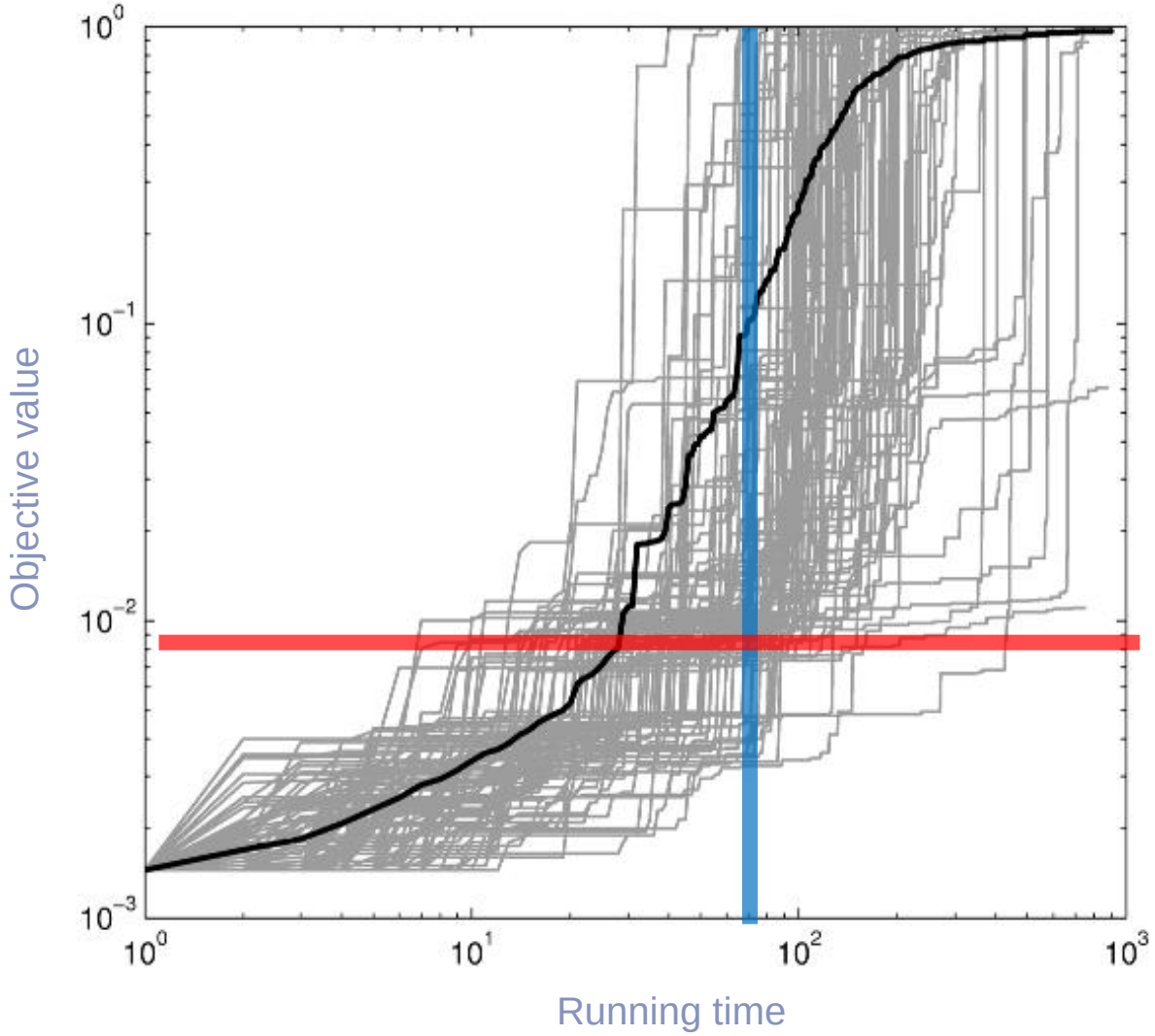
IOHanalyzer overview

- What perspective to consider?

Fixed-budget -
objective samples given runtime budget

Fixed-target - runtime samples given target value

success rate: some runs might not hit this line



Fixed-target Analysis

- Running time – random variable
  - Parameterized by ***a target value***

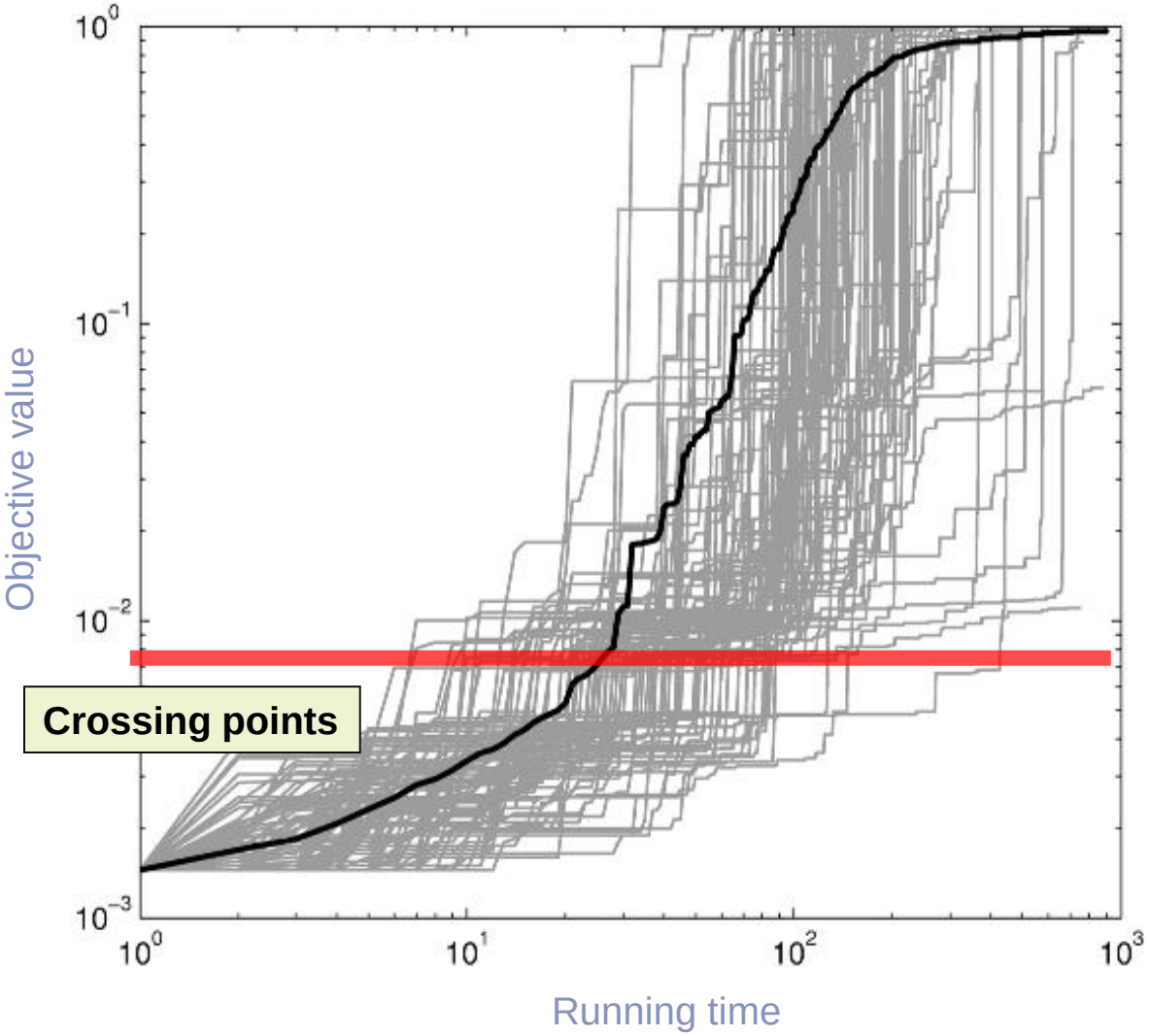
$$T(f_{\text{target}}) \in \mathbb{N}_{>0}$$

- The number of runs – r

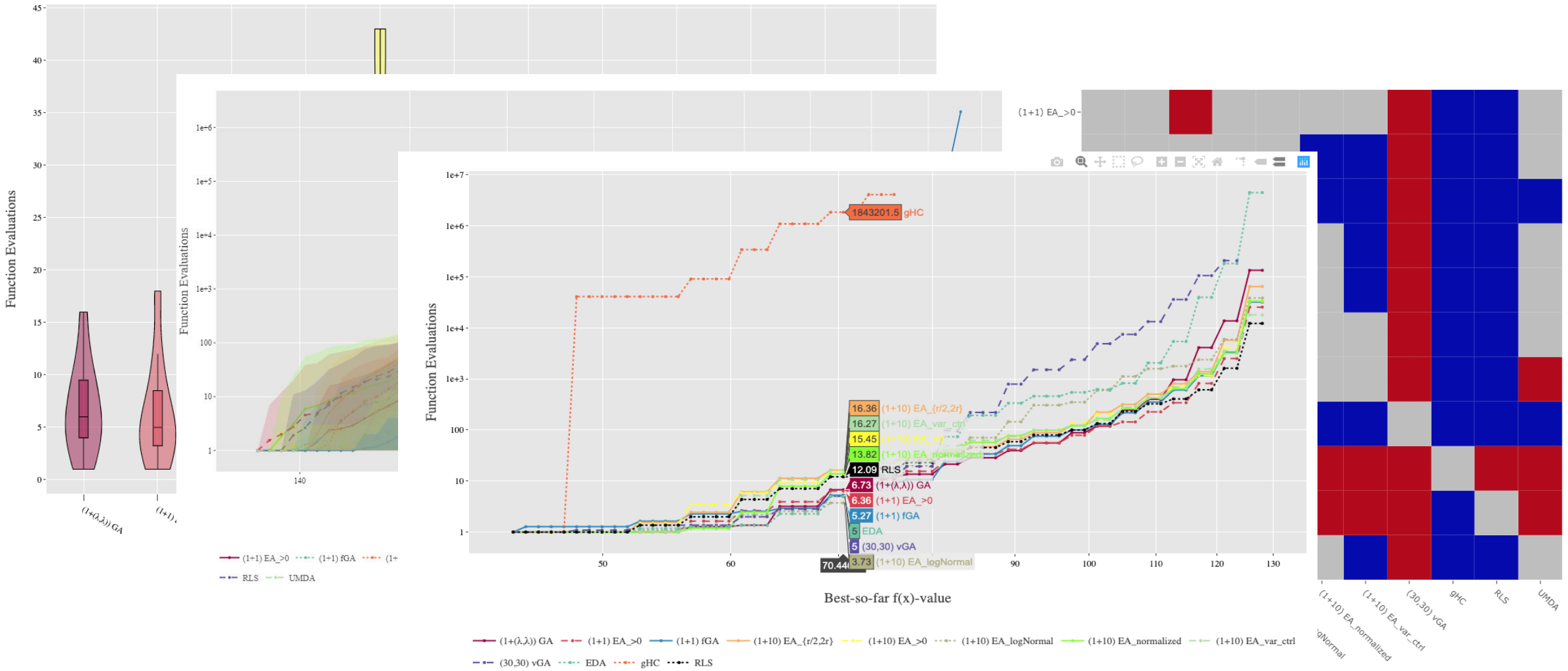
$$\{t_1(f_{\text{target}}), \ldots, t_r(f_{\text{target}})\}$$

- Successful? $t_i(f_{\text{target}}) < \infty$
- Unsuccessful? $t_i(f_{\text{target}}) = \infty$
- Number of successes $N_{\text{succ}} = \sum_{i=1}^{r} \mathbf{1}(t_i(f_{\text{target}}) < \infty)$

- Performance measures:
  - Expected Running Time
  - Penalized Average Running Time
  - Average Hitting time
  - Confidence intervals of hitting time
  - ...



Fixed-target Analysis



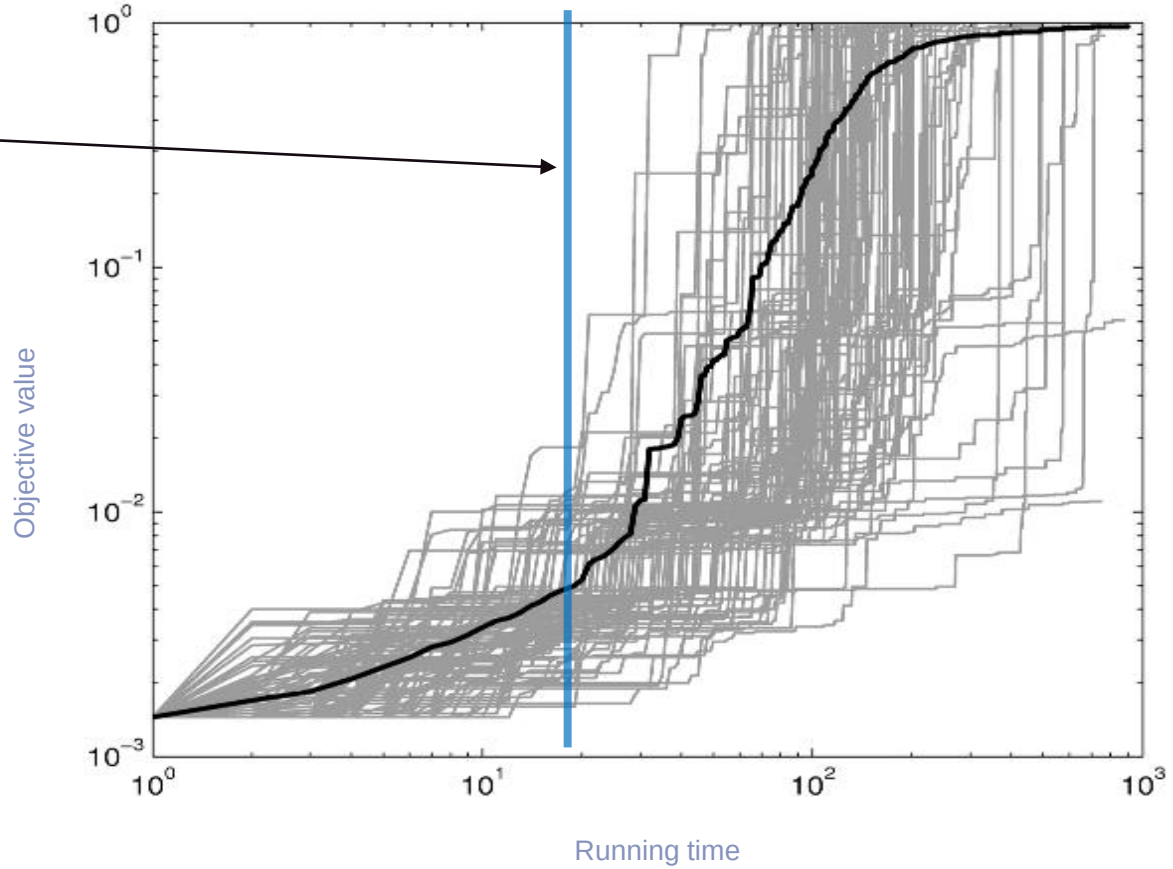
Fixed-budget analysis

- Function value – random variable
  - Parameterized by ***a budget value***

$$V(b) \in \mathbb{R}, b \leq B$$

- The number of runs – r

$$\{v_1(b), \ldots, t_r(b)\}$$



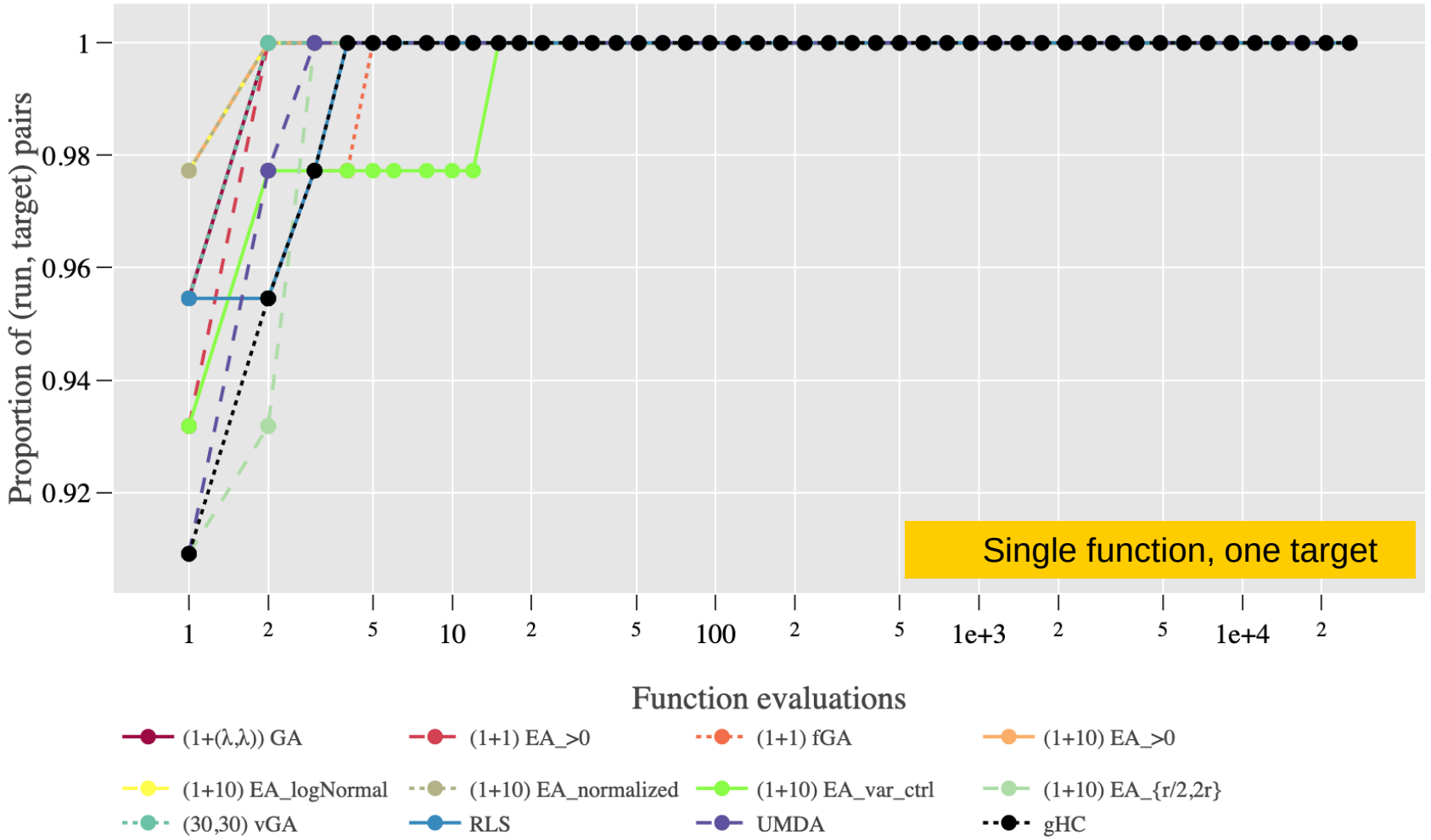
Single-function Analysis

	ID	runtime	runs	mean	median	sd	2%	5%	10%	25%	50%	75%	90%	95%	98%
	All	All	All	All	All	All	All	All	All	All	All	All	All	All	All
1	(1+1) EA_>0	100000	11	312	312	0	312	312	312	312	312	312	312	312	312
2	gHC	100000	11	312	312	0	312	312	312	312	312	312	312	312	312
3	(1+10) EA_{r/2,2r}	100000	11	312	312	0	312	312	312	312	312	312	312	312	312
4	(1+10) EA_>0	100000	11	312	312	0	312	312	312	312	312	312	312	312	312
5	(1+10) EA_logNormal	100000	11	312	312	0	312	312	312	312	312	312	312	312	312
6	(1+10) EA_normalized	100000	11	312	312	0	312	312	312	312	312	312	312	312	312
7	(1+10) EA_var_ctrl	100000	11	312	312	0	312	312	312	312	312	312	312	312	312
8	UMDA	100000	11	312	312	0	312	312	312	312	312	312	312	312	312
9	(1+1) fGA	100000	11	312	312	0	312	312	312	312	312	312	312	312	312
10	(30,30) vGA	100000	11	293	293	3.81	287	288	289	291	293	294	297	298	299
11	RLS	100000	11	312	312	0	312	312	312	312	312	312	312	312	312

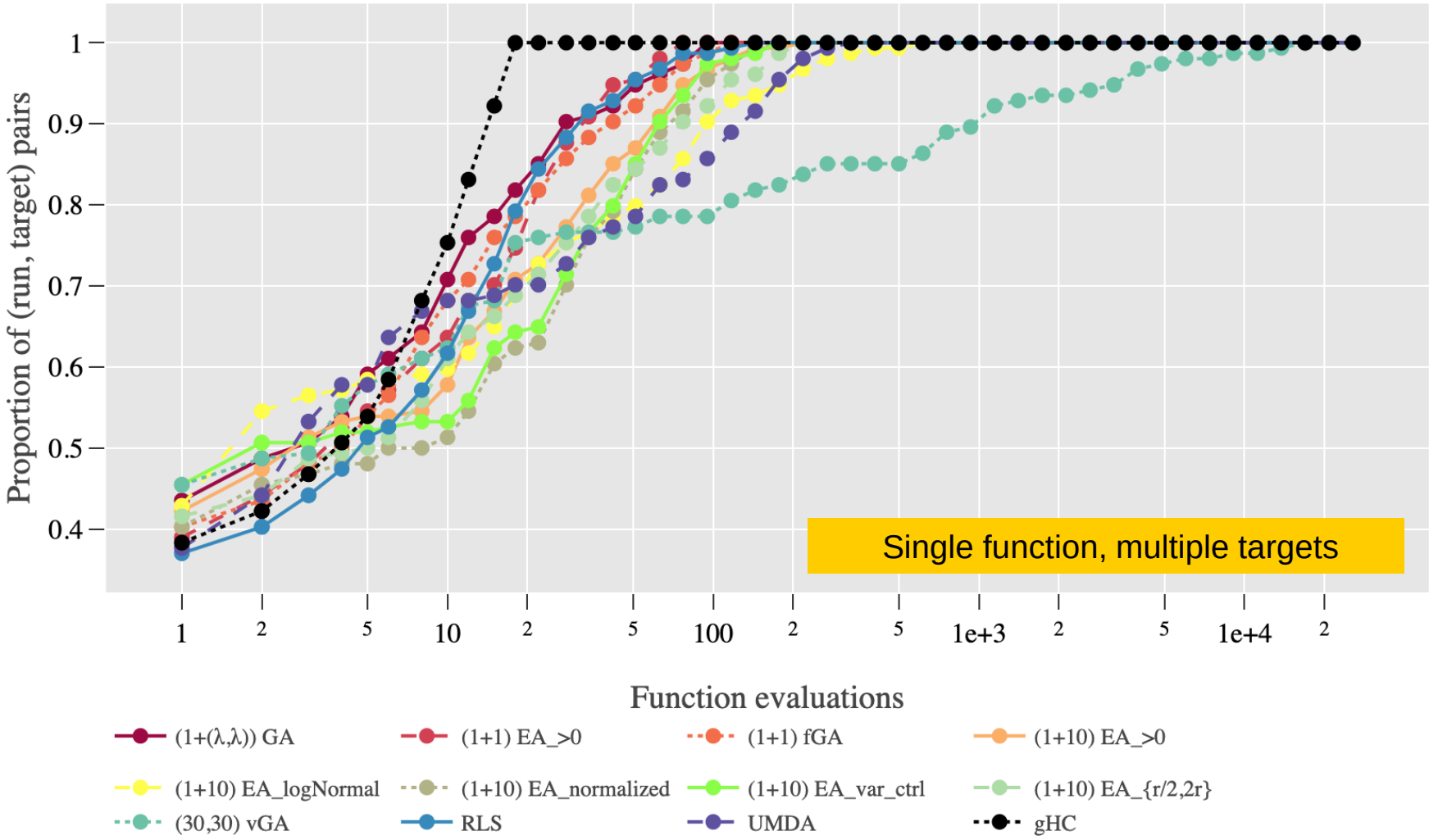
Showing 1 to 11 of 11 entries

Previous 1 Next

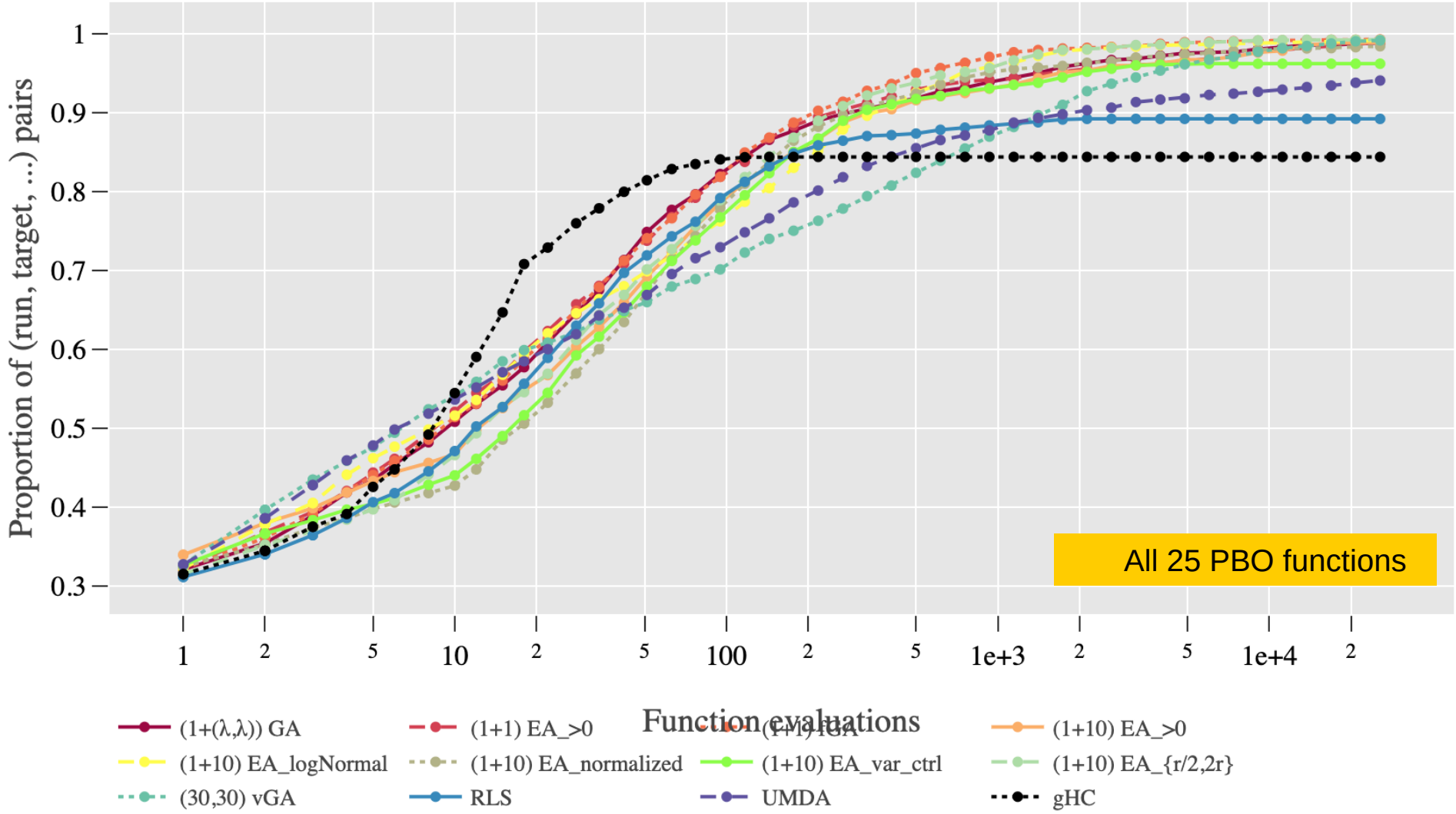
Empirical Cumulative Density Functions



Empirical Cumulative Density Functions

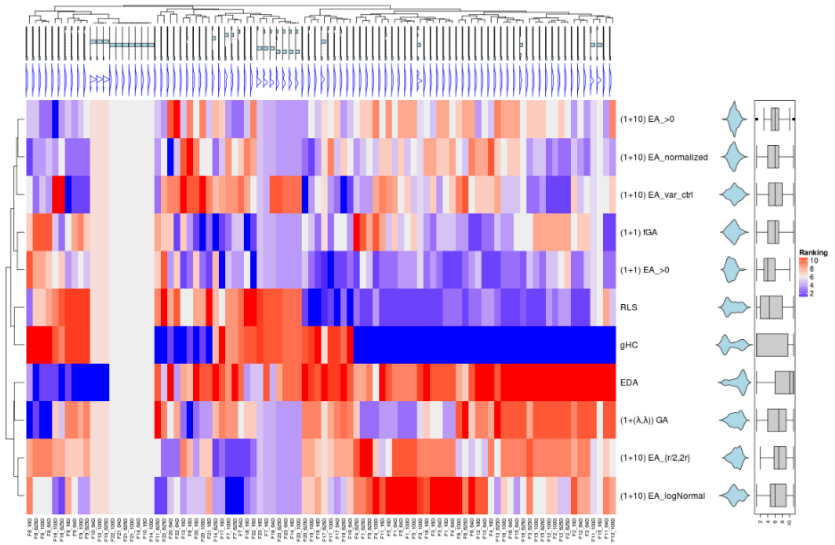


Empirical Cumulative Density Functions

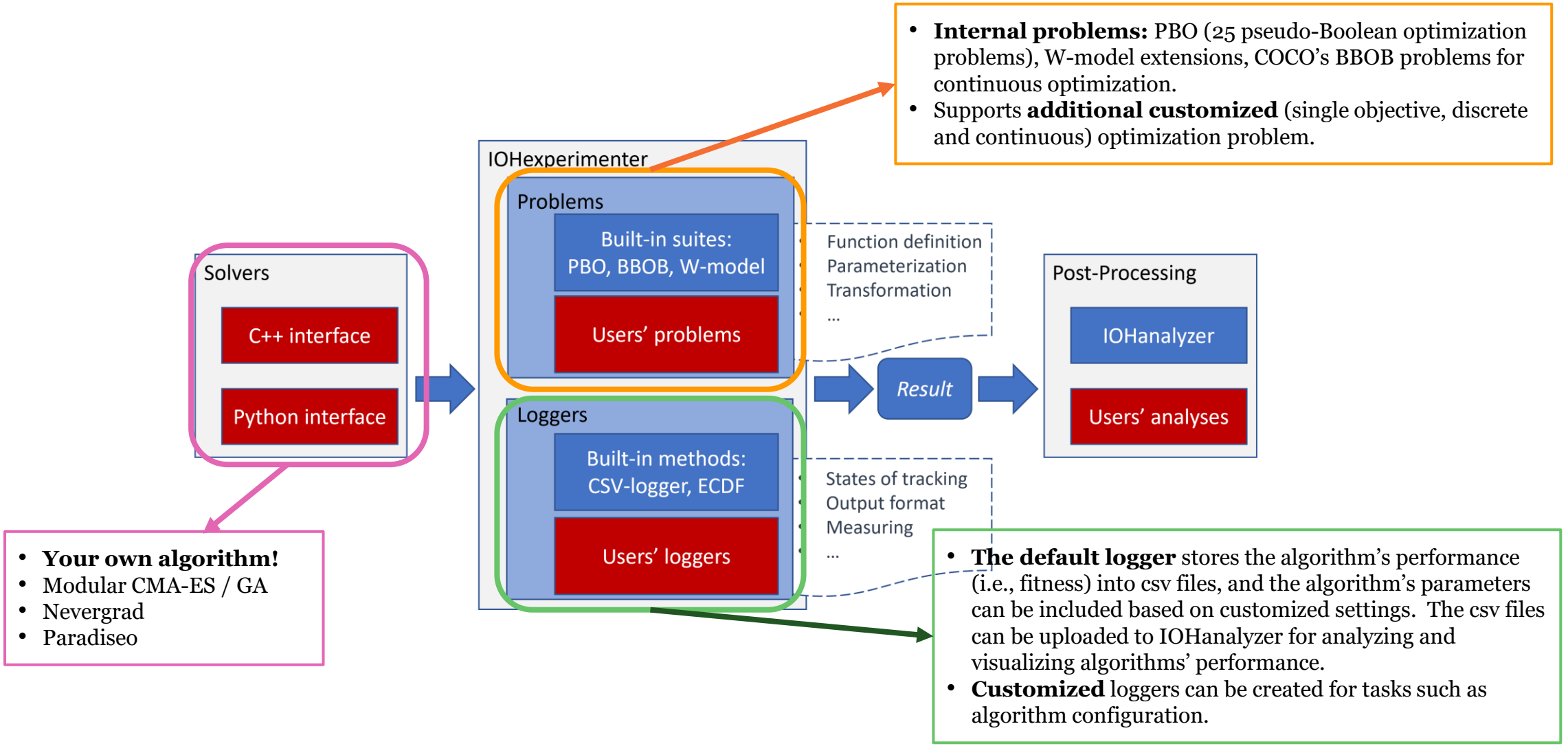


Other functionality

- Aggregated rankings
  - Based on better mean per function
  - Glicko2 ranking based on per-run comparisons
- Portfolio contributions (Based on combined ECDF)
- Deep Statistical Comparison
- Parameter Analysis



Overview of IOHexperimenter



Benchmark Problems

- 25 Pseudo-Boolean optimization problems
  - OneMax
  - LeadingOnes
  - ....
- 24 BBOB continuous optimization problems
  - Sphere
  - ...
- W-model extensions (collaboration with Thomas Weisse)
  - Discrete optimization problems can be configured with customized properties.
- Submodular Problems (collaboration with Frank Neumann et al.)
- MK-Landscape (collaboration with Tobias Driesse)
- Wrapper supporting any discrete or continuous problems

Logger

- Logger generating data that can be processed by IOHanalyzer

- Modular design
  - Customized trigger to determine when to store the data
  - Tracking arbitrary parameters of the algorithm

- CSV logger
- In-memory
- Loggers computing statistics on the fly

What can I do with IOHexperimenter?

- I want to test my algorithms on IOHproblems without modifying much of the current implementation.
  - Include the IOH header files for C++ or import the IOH package for Python and replace your fitness functions with the ones defined in IOHexperimenter.
- I want to check how IOHalgorithms perform on my own benchmark problems.
  - Wrapper functions are available for C++ and Python. Define your problems following the examples and apply the existing algorithm implementations for the new problems by replacing the function evaluating fitness.
- I want to compare my algorithms to other state-of-the-art algorithms.
  - Implement your algorithms integrating with the *problem* and *logger* classes of IOHexperimenter and upload the generated data to IOHanalyzer, where data of many other algorithms are accessible for comparison.
- I want to use IOHanalyzer to analyze the performance of my algorithms.
  - Apply the logger of IOHexperimenter to track the performance of your algorithms by attaching the *logger* to the *problem* class. If you plan to test on your own problems, we suggest wrapping the problems into IOHexperimenter, which can be easily done.
- How do I contribute my work to IOHexperimenter?
  - Any contributions, e.g., reporting bugs, suggesting new problems, customized loggers, etc., are appreciated. A PR workflow is available on Github.

DEMO

IOHexperimenter

Link to used material: https://github.com/jacobdenobel/ioh-tutorial

DEMO

IOHanalyzer

Link to GUI: iohanalyzer.liacs.nl

Discussion

- What features would make IOHprofiler fit your use-case better?
- How can we collaborate to make benchmarking as easy and useful as possible?
- How can the community assure that minimum standards for comparing optimization algorithms and reporting their results are adhered to?
- How can we support the comparison of stochastic algorithms in general (going beyond optimization algorithms)?