Overview

- ✓ **Introduction to *IOHprofiler* – 30 min**
- ✓ **Demo – 10 min**
- ✓ **Hands-on Session – 40 min**
- ✓ **Wrap-up, Q & A – 10 min**

IOHprofiler – Our Vision:

- ✓ a modern and effective tool for benchmarking and analyzing iterative optimization heuristics (IOHs)
- ✓ standardized experimental environment
- ✓ interactive performance analyses and statistics
- ✓ public, cloud-based data repositories for
  - solvers
  - benchmark problems
  - performance data

https://github.com/IOHprofiler

IOHprofiler Philosophy:
Individual benchmarking made possible

User chooses

- ✓ **which benchmark problems and algorithms to use**
  - ✓ easy to add new benchmark functions and algorithms
- ✓ **which performance measures to use**
  - ✓ wide range of performance statistics:
    - ✓ fixed-target runtimes (+distributions)
    - ✓ fixed-budget results (+distributions)
    - ✓ probabilities of success, ECDF curves, etc.
    - ✓ statistical comparisons
- ✓ **how to aggregate results**
  - ✓ highly interactive evaluation tool

https://github.com/IOHprofiler

Availability

Documentation:

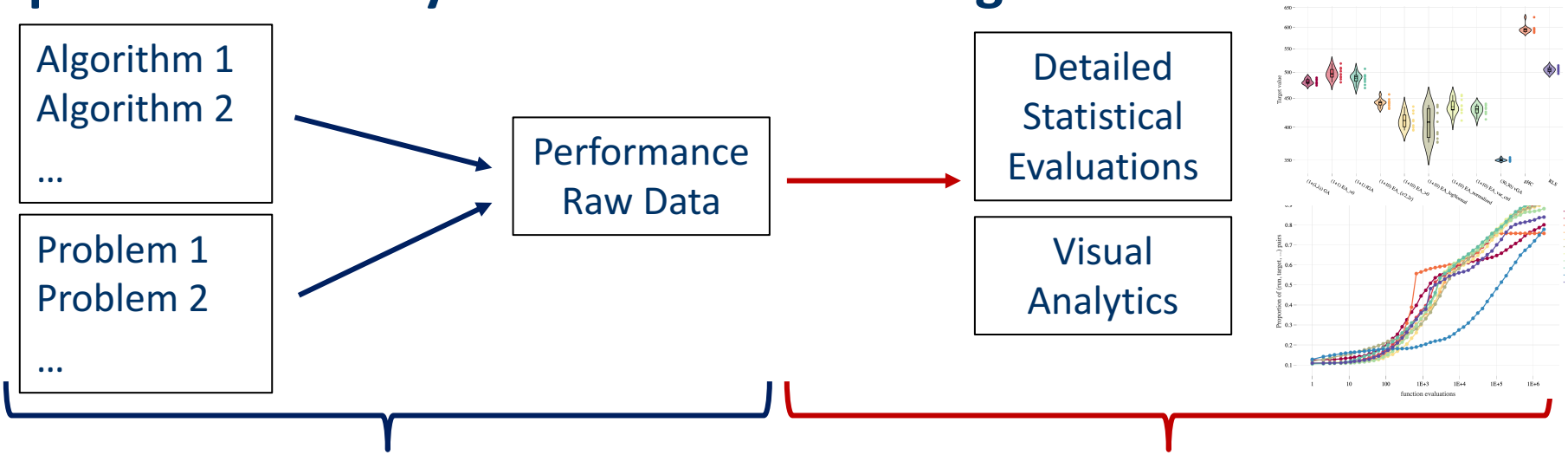
- ✓ IOHprofiler is online at GitHub: https://github.com/IOHprofiler
- ✓ Documentation available on arxiv: https://arxiv.org/abs/1810.05281
- ✓ Also a wiki page: https://iohprofiler.github.io/
- ✓ Web-based evaluation tool: http://iohprofiler.liacs.nl/

Pre-installed problems and algorithms:

- ✓ Implementations and performance data available for
  - ✓ 12 algorithms
  - ✓ 23 *discrete* benchmark functions available → more problems and algorithms are continuously added

Your contributions are very welcome!

IOHprofiler: A Ready-to-Use Benchmarking Suite with 2 Modules



IOHexperimenter (data *generation*)

Selected features:

- easy integration of new benchmark problems
- optional transformations to generate different problem instances
- parameter tracking: *algorithm profiling*
- user-specified granularity of data records

IOHanalyzer (data *evaluation*)

- Standalone tool, written in R
- Generates attractive plots in common formats for your next documentation
- Independent of IOHexperimenter: admits data from
  - IOHexperimenter
  - COCO/BBOB[1]
  - Facebook's nevergrad[2] [in progress]
  - other sources (following some minimalistic format requirements)

https://github.com/IOHprofiler [1]https://github.com/numbbo/coco [2]https://github.com/facebookresearch/nevergrad

IOHexperimenter

--

Standardized data generation to track performance and evolution of crucial parameters

IOHexperimenter

- *General benchmark software*, which could be instantiated to different benchmark suites, according to user's specification
- Wiki: https://iohprofiler.github.io/IOHexperimenter/
- Backbone: implemented in **C++** (1/14)
- Effective wrappers in **R**, **Python** [in progress]

- ***Pseudo-Boolean Optimization*** (PBO) suite: $f: S \to \mathbb{R}$ with finite search space S
  - Currently, **23** built-in combinatorial test-functions
  - OneMax, LeadingOnes *feat.* W-model [Weise, Wu. GECCO'18]
  - Ising models, N-Queens problem, Maximum Independent Set, Low Autocorrelation Binary Sequences
  - Extendible to any test-function following the interface

IOHexperimenter

- *General benchmark software*, which could be instantiated to different benchmark suites, according to user's specification

- ***Integration of Black-Box Optimization*** (PBO) suite: $f: S \to \mathbb{R}$ with finite search space S
  - Currently, **23** built-in combinatorial test-functions
  - OneMax, LeadingOnes *feat.* W-model [Weise, Wu. GECCO'18]
  - Ising models, N-Queens problem, Maximum Independent Set, Low Autocorrelation Binary Sequences
  - Extendible to any test-function following the interface

- **12** optimization algorithms implemented in C

https://github.com/IOHprofiler

Pseudo-Boolean Optimization (PBO) suite

- F1: OneMax
- F2: LeadingOnes
- F3: Harmonic Linear Function
- F4-F10: W-Model[2] on top of OneMax
  (W-model allows to scale effective dimension, epistasis, neutrality, ruggedness, fitness plateaus, etc.)
- F5-17: W-Model on top of LeadingOnes
- F18: LABS: Low Autocorrelation Binary Sequences
- F19: Ising Model: Ring
- F20: Ising Model: Torus
- F21: Ising Model: Triangular (Isometric 2D Grid)
- F22: MIVS: Maximum Independent Vertex Set
- F23: N-Queens Problem

Thousands of combinations are possible, our implementation covers the whole W model

[2][Weise, Wu: GECCO'18 (BB-DOB workshop paper)]

https://github.com/IOHprofiler

11 Reference Algorithms
(more to come, your contributions most appreciated!)

- gHC: Greedy Hill Climber: flip 1 bit/iteration
- RLS: Randomized Local Search: flips 1 random bit/iteration
- (1+ λ) EAs with standard bit mutation
- (1+ λ) EAs with normalized bit mutation and self-adjusting means [YeDB'19]
- (1+ λ) EAs with normalized bit mutation and self-adjusting means and variance control [YeDB'19]
- (1+ λ) EAs with log-normal self-adaptive mutation rates [BäckS96]
- fast GA from [DoerrLMN GECCO'17]
- self-adjusting 2-rate (1+λ) EA from [DoerrGWY GECCO'17]
- self-adjusting (1+(λ, λ)) GA from [DoerrD Algorithmica'18]
- (μ, λ) "vanilla"/textbook GA

Strong focus on "textbook" algorithms and recent developments from the theory community
→ your contributions are highly welcome!

11 Reference Algorithms
(more to come, your contributions most appreciated!)

- All implemented in C++
- https://github.com/IOHprofiler/IOHalgorithm (master/Algorithms)
- **Python** and **R** interface to those algorithm [in progress]
- Benchmark data on those 11 algorithms
  - https://github.com/IOHprofiler/IOHdata
  - All 23 test problems
  - 11 runs of each algorithms
  - $d = 16, 100, 625$
  - We will use this data set later…

IOHexperimenter – R interface

- Install C++ **boost** library:https://www.boost.org/users/download/

- Linux/MacOS:
  - tar -xzf boost_1_71_0.tar.gz
  - cd boost_1_71_0
  - ./bootstrap.sh --prefix=/some/dir/you/would/like/to/prefix
  - ./b2
  - ./b2 install

- On MacOS:
  - brew install boost
  - sudo port install boost

- Windows: please use the binary installer

IOHexperimenter – R interface

- Wrappers enable modern *look-n-feel* in scripting languages:
- To install the **R** interface

```
R> install.packages('devtools')
R> devtools::install_github('IOHprofiler/IOHexperimenter@R')
R> library('IOHexperimenter')
```

- Compilation might take a while (we are working on pre-compiled binary packages for the common platform)

IOHexperimenter – R interface

- Specify an experiment

```
R> set.seed(42) # to create reproducible results
R> experimenter <- IOHexperimenter(
+   suite = "PBO",
+   dims = c(16, 100),
+   functions = c(1, 2, 19, 23),
+   instances = 1:5, algorithm.name = 'RLS',
+   data.dir = './data'
+ )
```

- ***instances***:
  - Translation
  - Permutation of binary string

$$af(\sigma(\mathbf{x} \oplus \mathbf{z})) + b$$

IOHexperimenter – R interface

- Specify an experiment

```
R> set.seed(42) # to create reproducible results
R> experimenter <- IOHexperimenter(
+   suite = "PBO",
+   dims = c(16, 100),
+   functions = c(1, 2, 19, 23),
+   instances = 1:5, algorithm.name = 'RLS',
+   data.dir = './data'
+ )
```

- ***data.dir***: The path to the folder where (raw) benchmark data are stored

- ***param.track***: A vector containing the names of the parameters which will be tracked during the benchmarking

IOHexperimenter – R interface

- The benchmark object:

```
R> experimenter
IOHexperimenter (5 instances, 4 functions, 2 dimensions)
Dimensions: 16, 100
Functions: 1, 2, 19, 23
Instances: 1, 2, 3, 4, 5
```

- Get test problems:

```
R> problem <- next_problem(experimenter)
R> print(problem)
IOHproblem (Suite PBO: Instance 1 of function OneMax 16D)
```

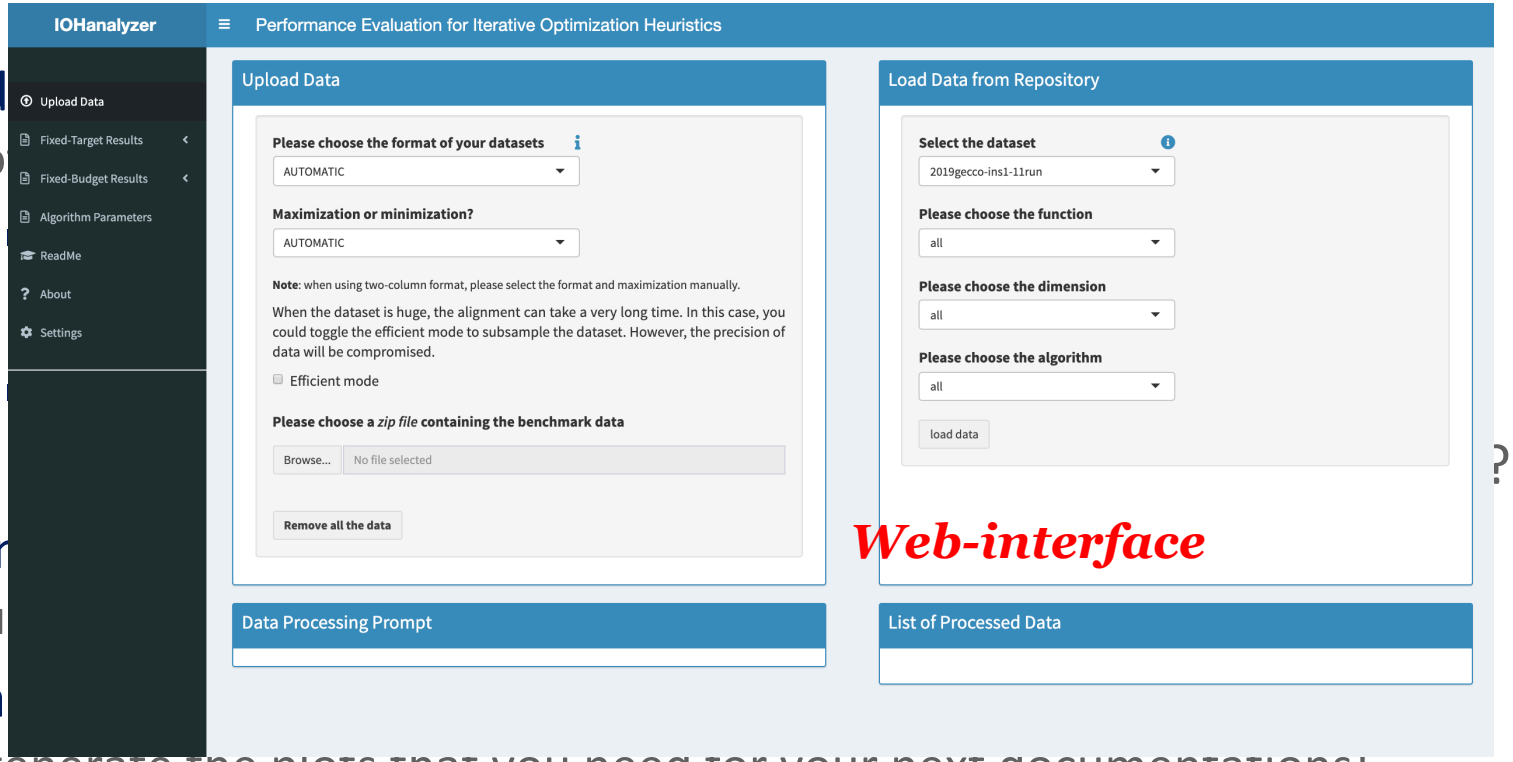
IOHanalyzer

--

Performance evaluation and visualization

IOHanalyzer – Vision

- ✓ d
  o
- ✓ ir
  u
- ✓ a



?

generate the plots that you need for your next documentations!

- ✓

```
"algId","target","mean","median","sd","2%","5%","10%","25%","50%","75%","90%","95%","98%","ERT","runs","ps"
"(1+(λ,λ)) GA",600,2378,2360,209.07,1988,1988,1988,2250,2360,2461,2651,2651,2690,2378,11,1
"(1+1) EA_>0",600,2114.27,2116,213.32,1797,1797,1797,1907,2116,2275,2330,2330,2457,2114.27,11,1
"gHC",600,575.45,574,6.65,566,566,566,569,574,579,583,583,587,575.45,11,1
"(1+10) EA_{r/2,2r}",600,4964.91,4956,442.67,4086,4086,4086,4714,4956,5084,5362,5362,5832,4964.91,11,1
"(1+10) EA_>0",600,3391.55,3396,155.01,3076,3076,3076,3266,3396,3483,3557,3557,3582,3391.55,11,1
"(1+10) EA_logNormal",600,4251.82,4093,643.14,3489,3489,3489,3804,4093,4464,4998,4998,5539,4251.82,11,1
"(1+10) EA_normalized",600,3162.18,3173,179.59,2827,2827,2827,3036,3173,3310,3369,3369,3374,3162.18,11,1
"(1+10) EA_var_ctrl",600,3080.91,3096,131.21,2913,2913,2913,2963,3096,3141,3221,3221,3342,3080.91,11,1
"(1+1) fGA",600,2933.18,2935,306.38,2435,2435,2435,2728,2935,3036,3078,3078,3653,2933.18,11,1
"(30,30) vGA",600,358746.91,347996,31706.41,322774,322774,322774,332515,347996,380811,405370,405370,409037,358746.91,11,1
"RLS",600,1566.27,1600,109.04,1392,1392,1392,1415,1600,1642,1653,1653,1700,1566.27,11,1
```

CSV

Performance Measures

- Fixed-target and fixed-budget perspectives
- Sample mean, median, quantiles, standard deviation, and success rate
- Expected Running Time (ERT)
- ECDF curves of the running time and function value

Performance Measures

- Fixed-target and fixed-budget view
- ECDF curves
- Averages (mean), median and other quantiles, standard deviation, probability of success

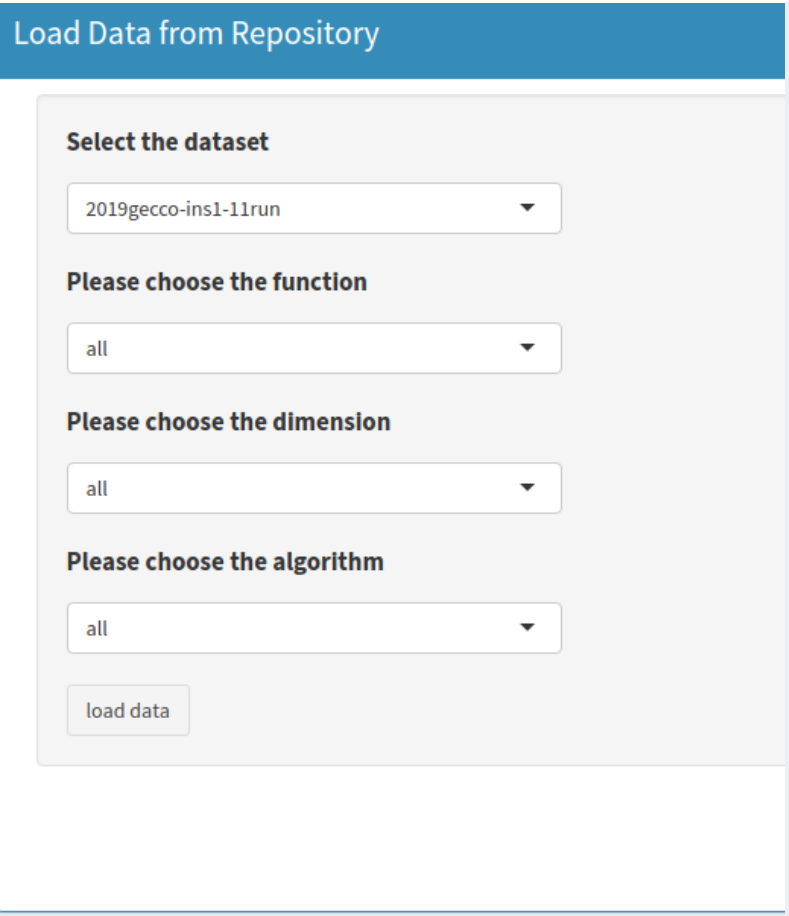
	algId	target	mean	median	sd	2%	5%	10%	25%	50%	75%	90%	95%	98%
	All	Al	All	All	A									
1	(1+(λ,λ)) GA	600	2378	2360	209.07	1988	1988	1988	2250	2360	2461	2651	2651	269
2	(1+1) EA_>0	600	2114.27	2116	213.32	1797	1797	1797	1907	2116	2275	2330	2330	245
3	gHC	600	575.45	574	6.65	566	566	566	569	574	579	583	583	587
4	(1+10) EA_{r/2,2r}	600	4964.91	4956	442.67	4086	4086	4086	4714	4956	5084	5362	5362	583
5	(1+10) EA_>0	600	3391.55	3396	155.01	3076	3076	3076	3266	3396	3483	3557	3557	358
6	(1+10) EA_logNormal	600	4251.82	4093	643.14	3489	3489	3489	3804	4093	4464	4998	4998	553
7	(1+10) EA_normalized	600												
8	(1+10) EA_var_ctrl	600												
9	(1+1) fGA	600												
10	(30,30) vGA	600												

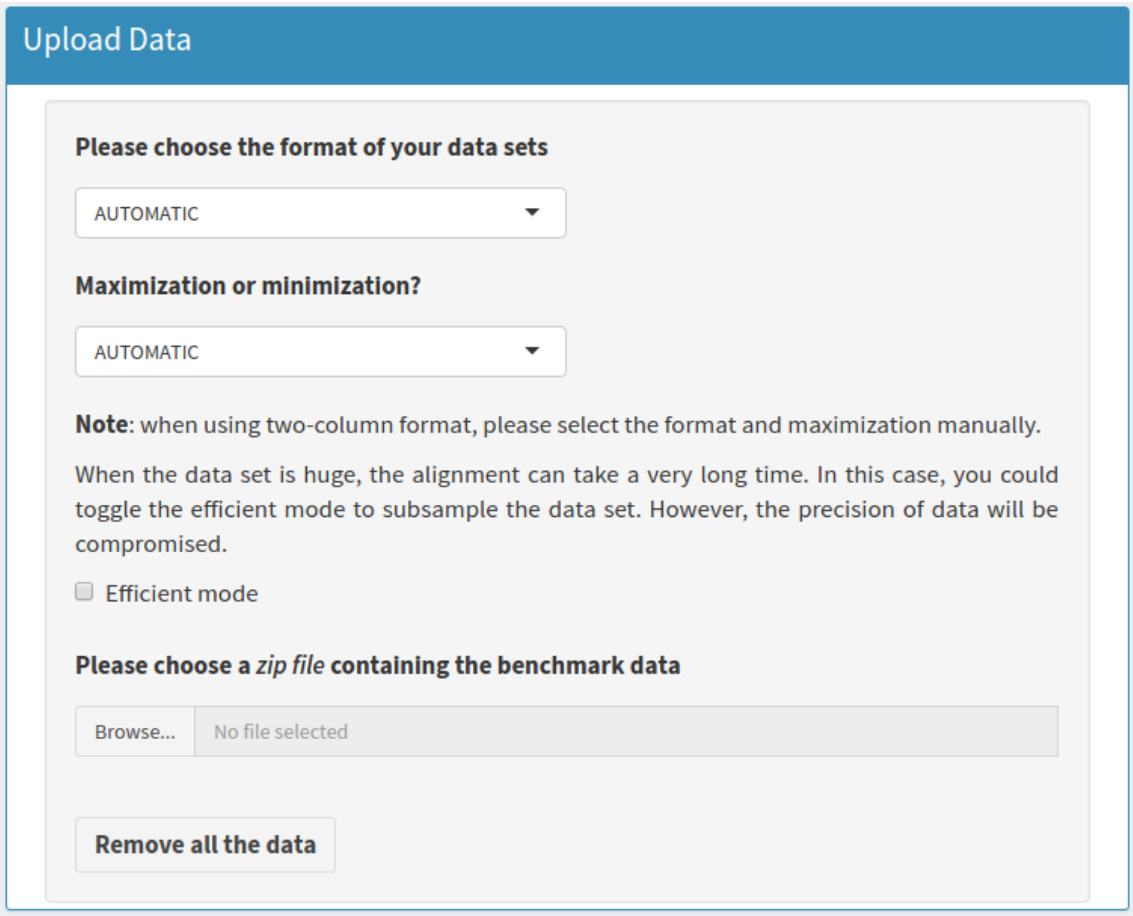
Showing 1 to 10 of 11 entries


```
"algId","target","mean","median","sd","2%","5%","10%","25%","50%","75%","90%","95%","98%","ERT","runs","ps"
"(1+(λ,λ)) GA",600,2378,2360,209.07,1988,1988,1988,2250,2360,2461,2651,2651,2690,2378,11,1
"(1+1) EA_>0",600,2114.27,2116,213.32,1797,1797,1797,1907,2116,2275,2330,2330,2457,2114.27,11,1
"gHC",600,575.45,574,6.65,566,566,566,569,574,579,583,583,587,575.45,11,1
"(1+10) EA_{r/2,2r}",600,4964.91,4956,442.67,4086,4086,4086,4714,4956,5084,5362,5362,5832,4964.91,11,1
"(1+10) EA_>0",600,3391.55,3396,155.01,3076,3076,3076,3266,3396,3483,3557,3557,3582,3391.55,11,1
"(1+10) EA_logNormal",600,4251.82,4093,643.14,3489,3489,3489,3804,4093,4464,4998,4998,5539,4251.82,11,1
"(1+10) EA_normalized",600,3162.18,3173,179.59,2827,2827,2827,3036,3173,3310,3369,3369,3374,3162.18,11,1
"(1+10) EA_var_ctrl",600,3080.91,3096,131.21,2913,2913,2913,2963,3096,3141,3221,3221,3342,3080.91,11,1
"(1+1) fGA",600,2933.18,2935,306.38,2435,2435,2435,2728,2935,3036,3078,3078,3653,2933.18,11,1
"(30,30) vGA",600,358746.91,347996,31706.41,322774,322774,322774,332515,347996,380811,405370,405370,409037,358746.91,11,1
"RLS",600,1566.27,1600,109.04,1392,1392,1392,1415,1600,1642,1653,1653,1700,1566.27,11,1
```


https://github.com/IOHprofiler

IOHanalyzer – Data Upload





- Loading the data set from our repository (***left***) and uploading your data set (***right***)

https://github.com/IOHprofiler

IOHanalyzer – Data Summary

Runtime Statistics at Chosen Target Values

Set the range and the granularity of the results. The table will show fixed-target runtimes for evenly spaced target values.

$f_{\min}$: **Smallest target value**

274.00

$f_{\max}$: **Largest target value**

625.00

Δf : **Granularity (step size)**

39.00

$f_{\min} = f_{\max}$? Once toggled, only $f_{\min}$ is used to generate the table on the right.

Algorithms

(1+(λ,λ)) GA_1

Format

csv

Save this table

This table summarizes for each algorithm and each target value chosen on the left:

- runs: the number of runs that have found at least one solution of the required target quality $f(x)$,
- mean: the average number of function evaluations needed to find a solution of function value at least $f(x)$
- median, $2\%, 5\%, \ldots, 98\%$: the quantiles of these first-hitting times

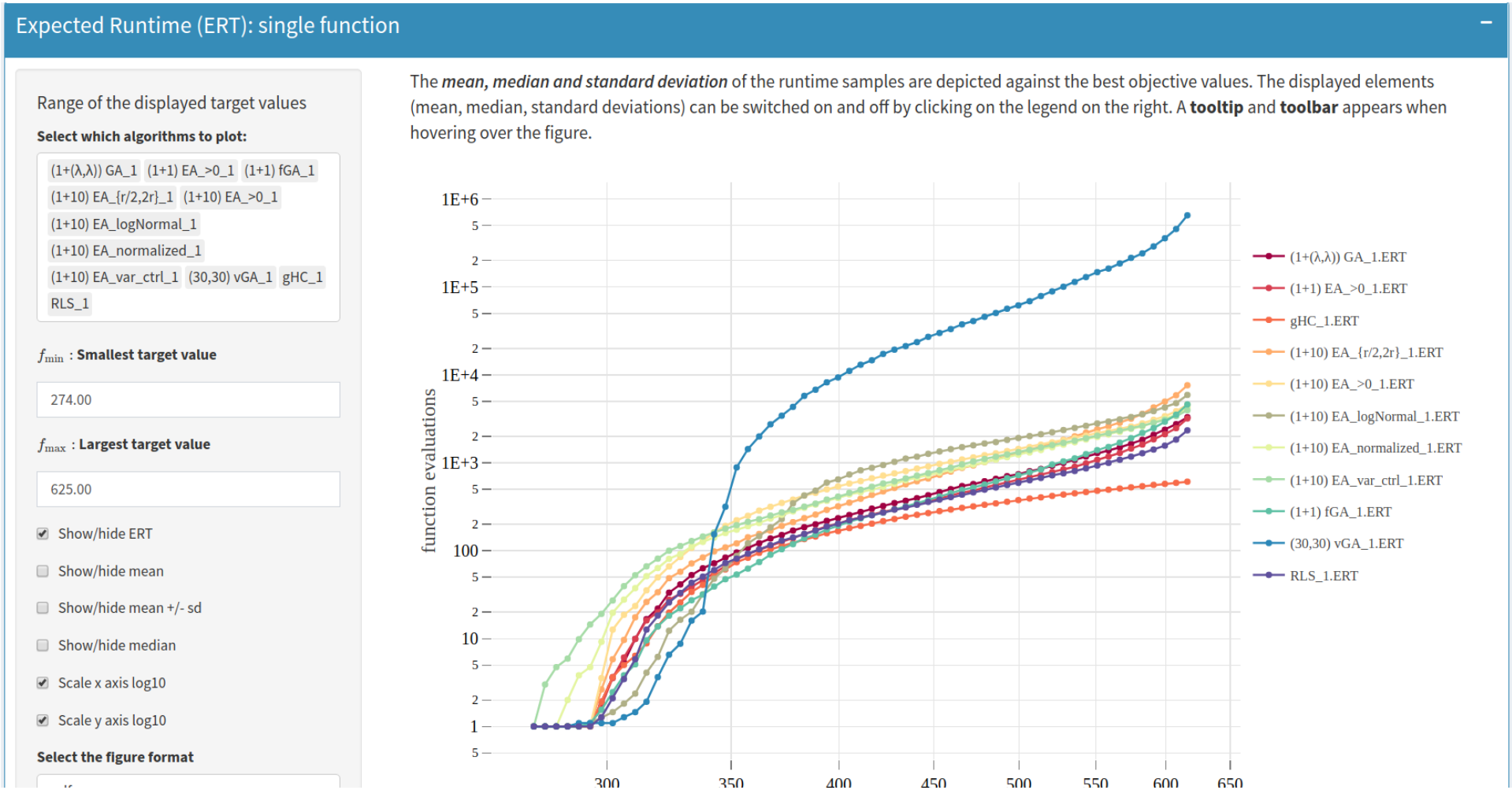
When not all runs managed to find the target value, the statistics hold only for those runs that did. That is, the mean value is the mean of the successful runs. Same for the quantiles. An alternative version with simulated restarts is currently in preparation.

Show entries Search:

algId	target	mean	median	sd	2%	5%	10%	25%	50%	75%	90%	95%	98%	ERT
(1+(λ,λ)) GA_1	274.00	1.00000	1	0.00000	1	1	1	1	1	1	1	1	1	1.00000
(1+(λ,λ)) GA_1	313.00	12.00000	11	11.89117	1	1	1	1	11	18	28	28	32	12.00000
(1+(λ,λ)) GA_1	352.00	92.09091	101	30.31321	33	33	33	75	101	116	124	124	126	92.09091
(1+(λ,λ)) GA_1	391.00	205.09091	206	28.46561	152	152	152	179	206	227	238	238	241	205.09091
(1+(λ,λ)) GA_1	430.00	352.18182	353	26.35458	312	312	312	328	353	364	381	381	398	352.18182
(1+(λ,λ)) GA_1	469.00	556.63636	561	32.09135	506	506	506	518	561	573	591	591	596	556.63636
(1+(λ,λ)) GA_1	508.00	807.54545	797	51.48468	734	734	734	752	797	841	860	860	893	807.54545

- Descriptive statistics (e.g., mean, median, quantiles…) of running time sample from algorithms

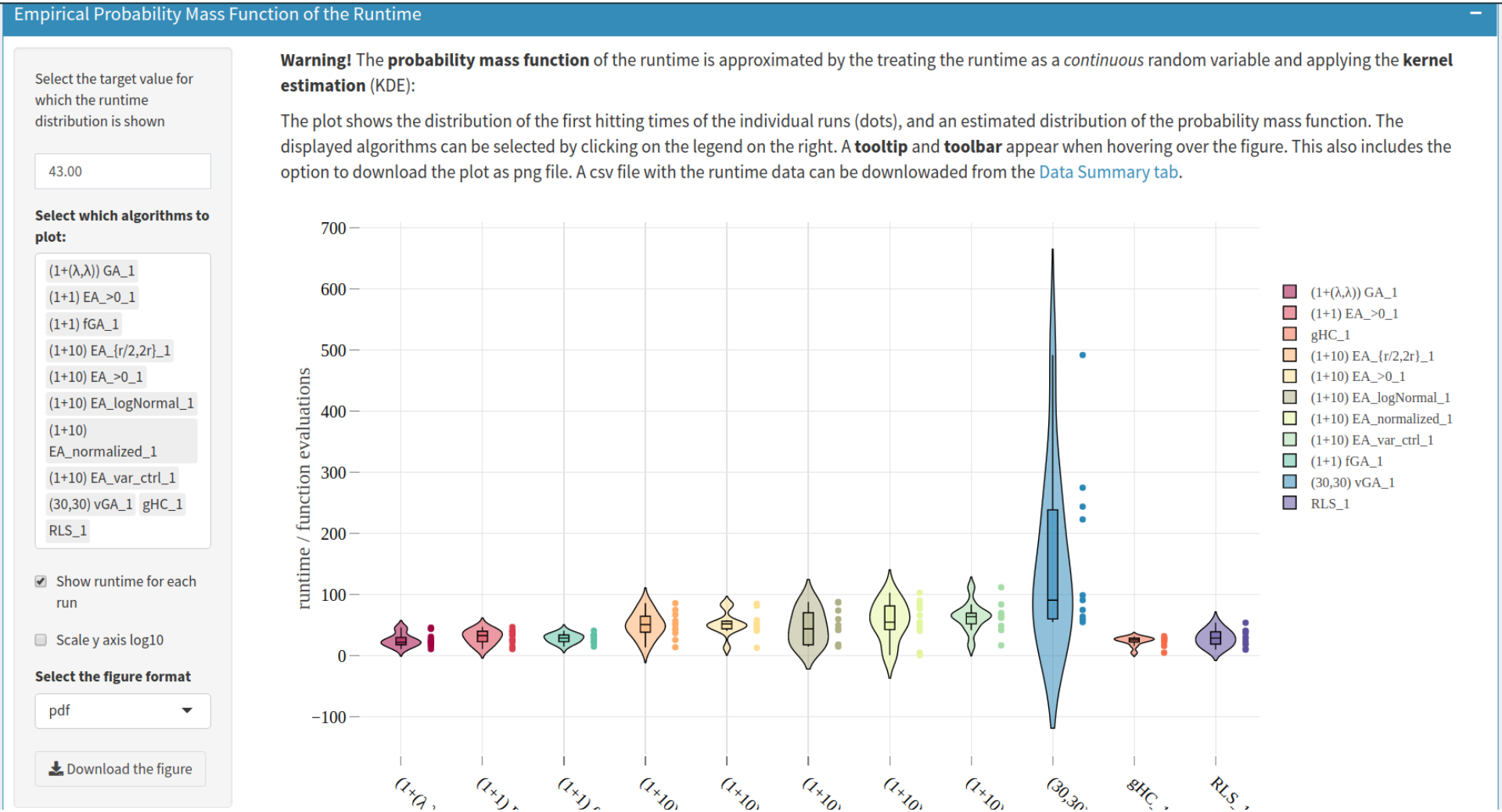
IOHanalyzer – Interactive Visualization



- The ERT (expected running time) as a function of target value reached by algorithms

https://github.com/IOHprofiler

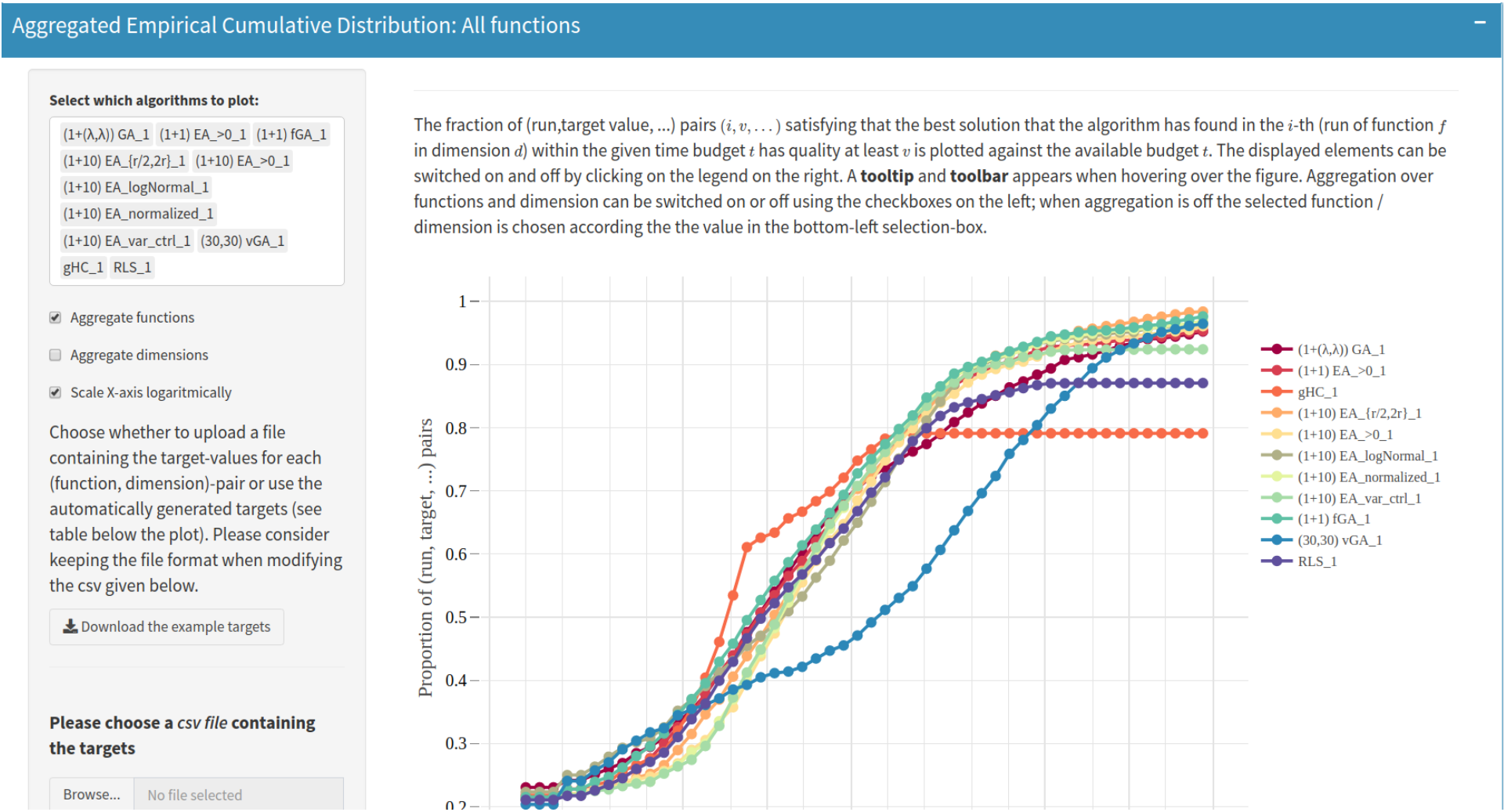
IOHanalyzer – Interactive Visualization



- Empirical probability density function of the running time at a given target value

https://github.com/IOHprofiler

IOHanalyzer – Interactive Visualization



- Cumulative distribution function of running time aggregated over multiple targets and functions

https://github.com/IOHprofiler

IOHanalyzer – Examples of Use-Cases

CEC full papers:

1. Furong Ye, Carola Doerr, and Thomas Bäck. Interpolating Local and Global Search by Controlling the Variance of Standard Bit Mutation. CEC'19 [Wednesday, 5:00 pm in Special Session CEC-21: Theory of Bio-Inspired Computation]

GECCO full papers:

1. Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, Thomas Bäck. Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: profiling (1 + λ) EA variants on OneMax and LeadingOnes. GECCO'18
2. Naama Horesh, Thomas Bäck, Ofer M. Shir. Predict or Screen Your Expensive Assay? DoE vs. Surrogates in Experimental Combinatorial Optimization. GECCO'19
3. Nguyen Dang, Carola Doerr. Hyper-Parameter Tuning for the (1+(λ, λ)) GA. GECCO'19

GECCO workshop papers:

1. Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M. Shir, and Thomas Bäck. Benchmarking Discrete Optimization Heuristics with IOHprofiler. GECCO'19
2. Borja Calvo, Ofer M. Shir, Josu Ceberio, Carola Doerr, Hao Wang, Thomas Bäck, and Jose A. Lozano. Bayesian Performance Analysis for Black-Box Optimization Benchmarking. GECCO'19
3. Ivan Ignashov, Arina Buzdalova, Maxim Buzdalov, and Carola Doerr. Illustrating the Trade-Off between Time, Quality, and Success Probability in Heuristic Search. GECCO'19
4. Nathan Buskulic and Carola Doerr. Maximizing Drift is Not Optimal for Solving OneMax. GECCO'19

Work in progress

- ✓ ***Bayesian analysis*** for comparing algorithms' performance data
- ✓ Extension of benchmark problems and reference algorithms
- ✓ Constant improvements of the User-Experience (UX)
  - ✓ plots/graphics
  - ✓ Efficiency
  - ✓ Report generation [in progress]
- ✓ Supporting more programming languages, e.g., **Scala**, **Java**, etc.

Suggestions and contributions are very welcome!

https://github.com/IOHprofiler

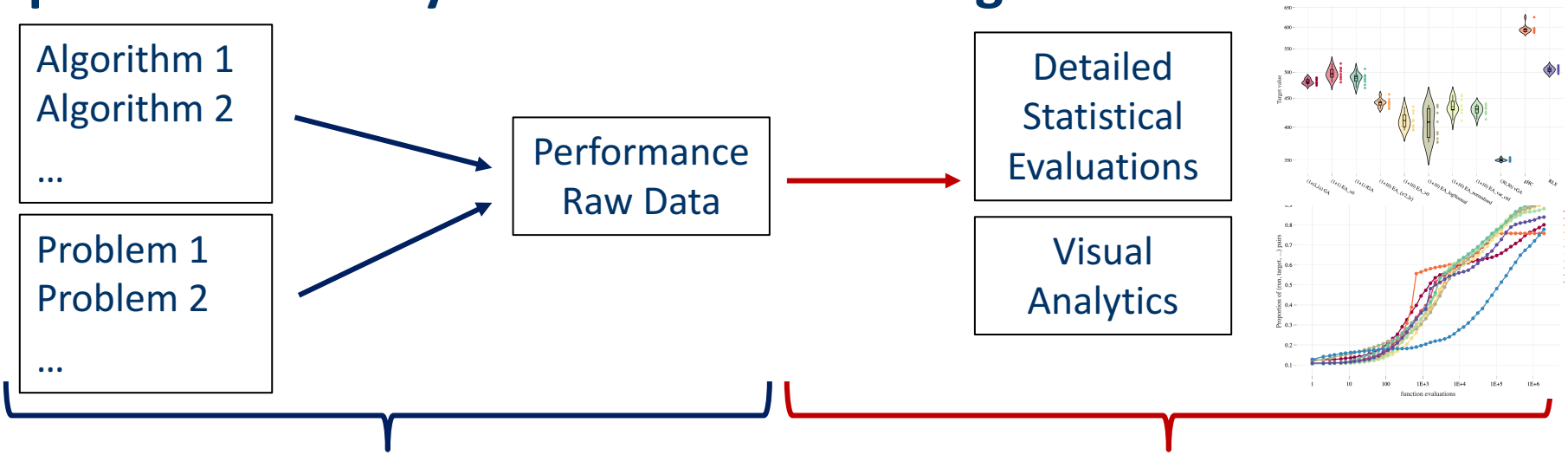
Hands-on Session

- ✓ The best way to get familiar with a software
- ✓ Online version for this tutorial: http://34.90.145.180/
- ✓ Practical question can be downloaded via https://github.com/wangronin/CA15140

We will go through those questions together!

WRAP-UP
(after hands-on)

IOHprofiler: A Ready-to-Use Benchmarking Suite with 2 Modules



IOHexperimenter (data *generation*)

Originally built on COCO/BBOB[1] modules now re-designed to enable interactive functionalities

Selected features:

- easy integration of new benchmark problems
- optional transformations to generate different problem instances
- parameter tracking: *algorithm profiling*
- user-specified granularity of data records

IOHanalyzer (data *evaluation*)

- Standalone tool, written in R
- Generates attractive plots in common formats for your next documentation
- Independent of IOHexperimenter: admits data from
  - IOHexperimenter
  - COCO/BBOB[1]
  - Facebook's nevergrad[2] [in progress]
  - other sources (following some minimalistic format requirements)

https://github.com/IOHprofiler [1]https://github.com/numbbo/coco [2]https://github.com/facebookresearch/nevergrad

Want to get involved?

You can contribute by providing (or suggesting)

- benchmark problems
- reference algorithms
- performance statistics
- ideas how to combine performance analysis with landscape analysis
- other features that would improve IOHprofiler

… and by simply using IOHprofiler, exporting its output for your presentation/manuscript, and sharing your user-experience!

Acknowledgments

- We thank the following colleagues for contributions and feedback on IOHprofiler and/or discussions around the topic of benchmarking IOHs:
  - Furong Ye, Diederick Vermetten, Sander van Rijn from Leiden University, NL
  - Arina Buzdalova, Maxim Buzdalov, and students from ITMO University, Russia
  - Borja Calvo, Josu Ceberio, and Jose A. Lozano from San Sebastián, Spain
  - Thomas Weise from Heifei University, China
  - Naama Horesh and Assaf Israeli from Migal Institute, Israel
  - Dirk Sudholt from Sheffield University, UK
  - Anne Auger, Dimo Brockhoff, and Niko Hansen from INRIA, France

- We acknowledge financial support from
  - Chinese scholarship council (CSC No. 201706310143)
  - ANR-11-LABX-0056-LMH,
  - Paris Ile-de-France Region
  - COST Action CA15140 on Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO)

- PPSN 2020 conference
- Leiden, The Netherlands
- September 5 – 9, 2020
- http://ppsn2020.liacs.leidenuniv.nl/

APPENDIX

Few Details about Algorithms

1. **gHC:** A (1+1) greedy hill climber, which goes through the string from left to right, flipping exactly one bit per each iteration, and accepting the offspring if it is at least as good as its parent.
2. **RLS:** Randomized Local Search, the elitist (1+1) strategy flipping one uniformly chosen bit in each iteration. I.e., RLS and gHC differ only in the choice of the bit which is flipped. While RLS is unbiased in the sense of Sec. 3.2, gHC is not permutation-invariant and thus biased.
3. $(1+1)$ **EA:**[4] The (1+1) EA with static mutation rate $p = 1/n$. This algorithm differs from RLS in that the number of uniformly chosen, pairwise different bits to be flipped is sampled from the conditional binomial distribution $\mathrm{Bin}_{>0}(n,p)$.
4. **fGA:** The "fast GA" proposed in [DLMN17] with $\beta = 1.5$. While in the (1+1) EA the *mutation strength* (i.e., the number of bits flipped in each iteration) follows a (conditional) binomial distribution [PD17, DYvR+18], its distribution follows a power-law in the fGA.
5. $(1+10)$ **EA:** The (1+10) EA with static $p = 1/n$, which differs from the (1+1) EA only in that 10 offspring are sampled (independently) per each iteration. Only the best one of these (ties broken at random) replaces the parent, and only if it is as least as good.
6. $(1+10)$ $\mathbf{EA}_{r/2,2r}$**:** The two-rate EA with self-adjusting mutation rates suggested and analyzed in [DGWY17].
7. $(1+10)$ $\mathbf{EA}_{\text{norm.}}$**:** a variant of the $(1+10)$ EA sampling the mutation strength from a normal distribution $N(pn, pn(1-p))$ with self-adjusting choice of p; was recently suggested in [YDB19].
8. $(1+10)$ $\mathbf{EA}_{\text{var.}}$**:** The $(1+10)$ $\mathrm{EA}_{\text{norm.}}$ with an adaptive choice of the variance in the normal distribution from which the mutation strengths are sampled. Also from [DYvR+18].
9. $(1+10)$ $\mathbf{EA}_{\text{log-n.}}$ The (1+10) EA with log-normal self-adaptation of the mutation rate proposed in [BS96].
10. $(1+(\lambda,\lambda))$ **GA:** A binary (i.e., crossover-based) EA originally suggested in [DDE15]. We use the variant with self-adjusting λ analyzed in [DD18].
11. **vGA:** A $(30,30)$ "vanilla" GA (following the so-called traditional GA, as described in e.g., [Gol89, Bäc96]).

https://github.com/IOHprofiler