



**Universiteit
Leiden**
The Netherlands

Practical Assignment

Evolutionary Algorithms Course, LIA CS, 2024-2025

Solving the F18 and F23 problems from the *Pseudo-Boolean Optimization* (PBO) problem set using Genetic Algorithms

F18: Low Autocorrelation Binary Sequences (LABS)

F18: Low Autocorrelation Binary Sequences (LABS)

The Low Autocorrelation Binary Sequences (LABS) problem poses a non-linear objective function over a binary sequence space, with the goal to maximize the reciprocal of the sequence's autocorrelation: $x \in \{0,1\}^n$

$$\text{LABS} : x \mapsto \frac{n^2}{2 \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} s_i s_{i+k} \right)^2}, \text{ where } s_i = 2x_i - 1.$$

1. Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O. M., & Bäck, T. (2019, July). Benchmarking discrete optimization heuristics with IOHprofiler. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (pp. 1798-1806).
2. <https://iohprofiler.github.io/IOHproblem/PBO>

Example Problem: LABS

- ▶ Low Autocorrelation of Binary Sequences
- ▶ Autocorrelation function on $\{-1,1\}^n$
- ▶ Important applications
 - ▶ Telecommunications
 - ▶ Radar
 - ▶ Sonar
- ▶ Transformation of variables:
 - ▶ $\{0,1\} \rightarrow \{-1,1\}$

The Objective Function

- ▶ Search space: $\{0,1\}^n$
- ▶ Goal: Find $\mathbf{x} \in \{0,1\}^n$ such that

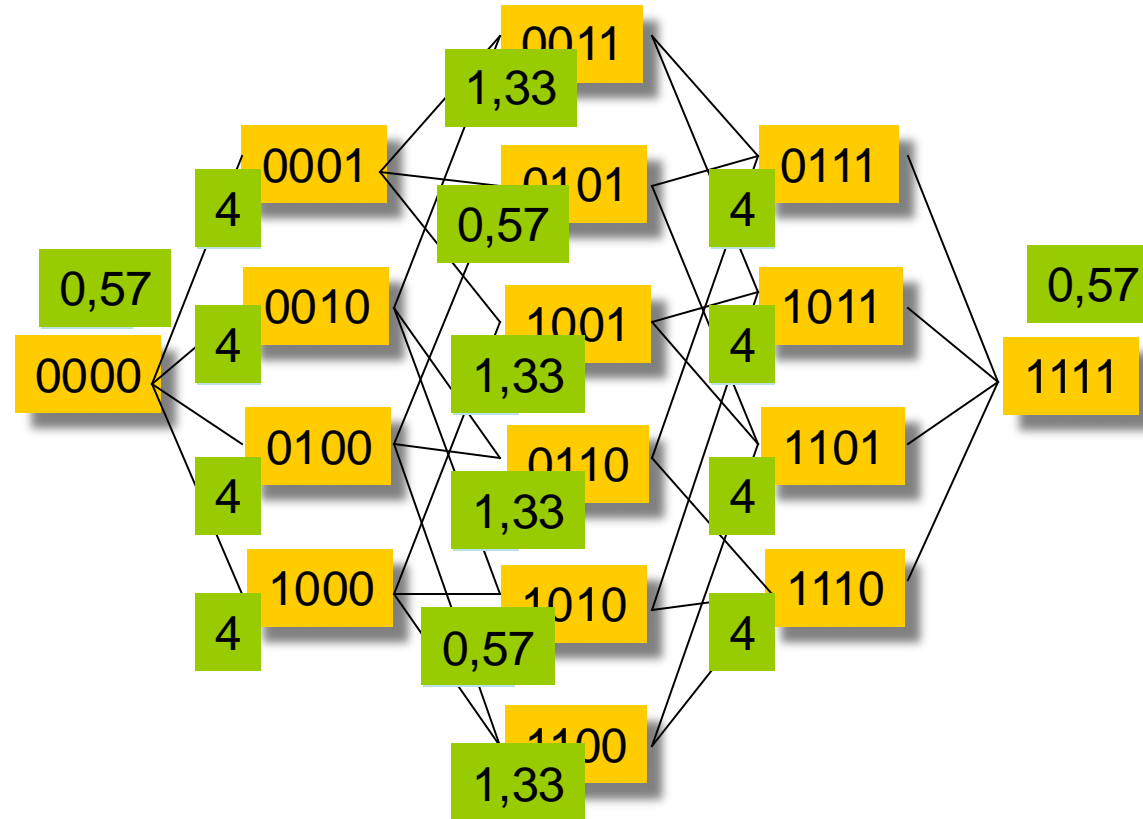
$$E(x) = \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} y_i \cdot y_{i+k} \right)^2 \rightarrow \min$$

$$y_i = 2x_i - 1$$

- ▶ Alternative formulation (merit factor):

$$F(\mathbf{x}) = \frac{n^2}{2E(\mathbf{x})} \rightarrow \max$$

Example: $n=4$



$$E(\mathbf{x}) = (y_1y_2 + y_2y_3 + y_3y_4)^2 + (y_1y_3 + y_2y_4)^2 + (y_1y_4)^2$$

Some Values

- ▶ See the table for some known records
 - ▶ Values in bold are not confirmed to be the best possible
 - ▶ Most optimizers get stuck around 7

n	Best value of f
20	7.6923
50	8.1699
100	8.6505
199	7.5835
200	7.4738
201	7.5263
202	7.3787
203	7.5613
219	7.2122
220	7.0145
221	7.2207
222	7.0426

Some Values

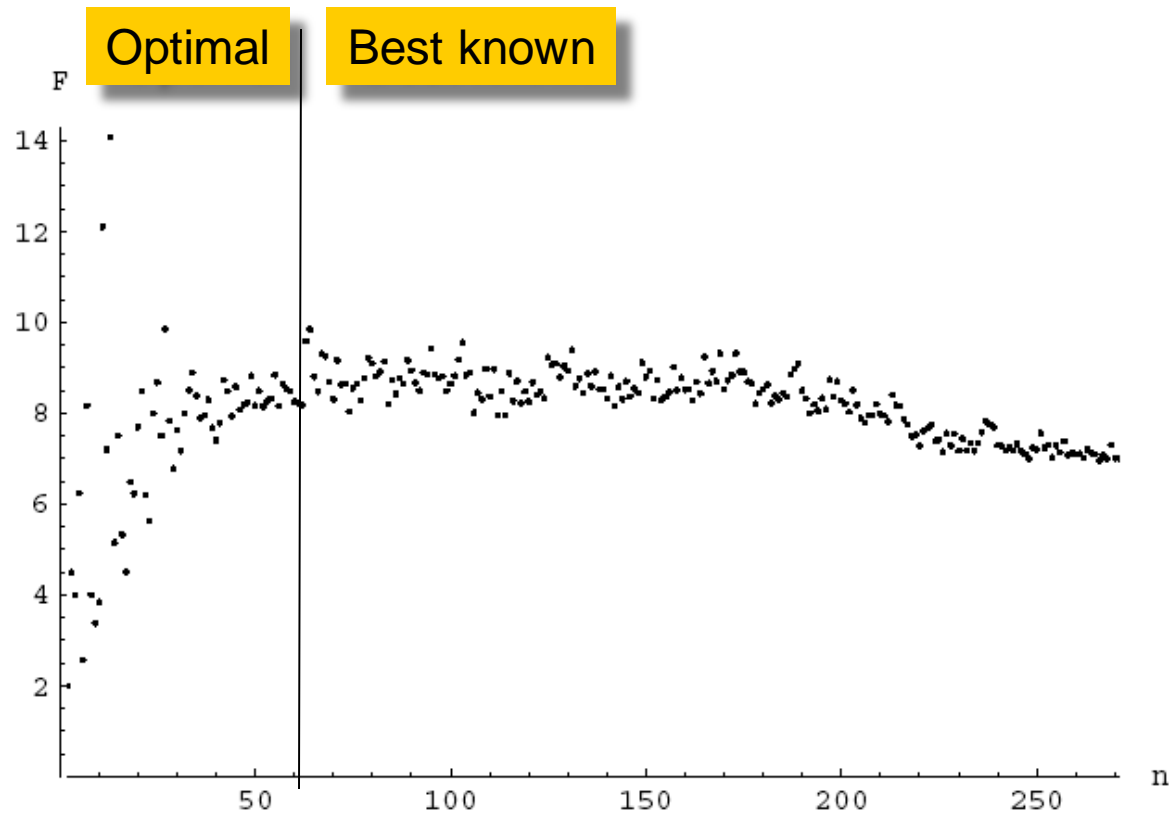
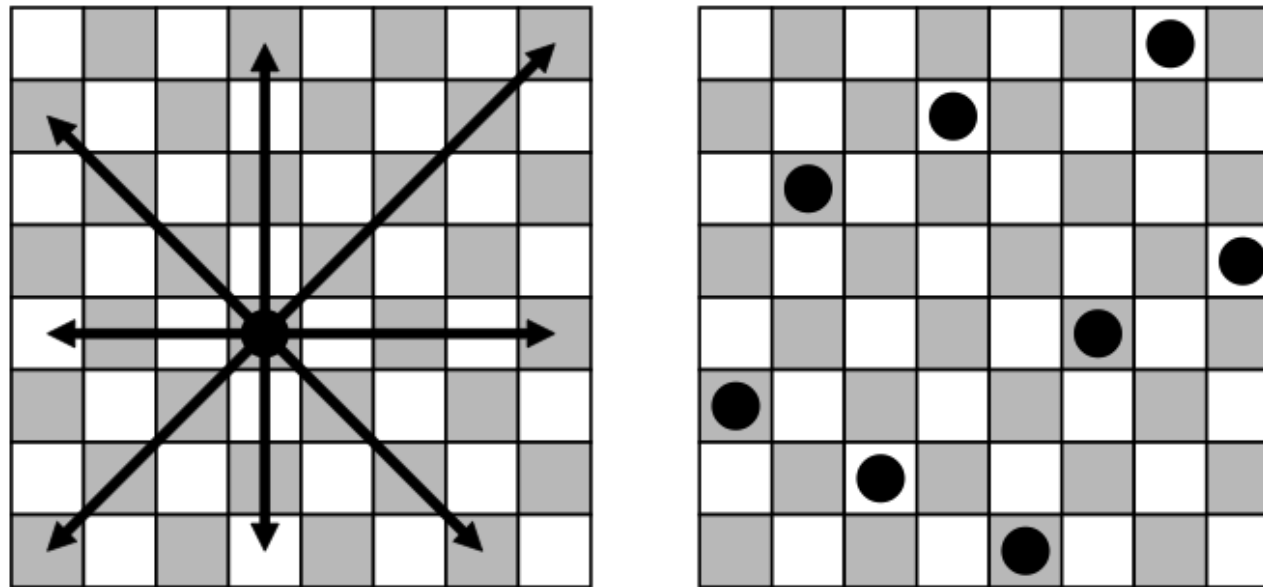


Figure 1: The optimal merit factor (for $2 \leq n \leq 60$) and the best known merit factor (for $61 \leq n \leq 271$) for binary sequences of length n .

F23: N-Queens Problem

The N-queens problem (NQP) is defined as the task to place N queens on an $N \times N$ chessboard in such a way that they cannot attack each other. The figure below provides an illustration for the 8-queens problem. Notably, the NQP is actually an instance of the MIVS problem– when considering a graph on which all possible queen-attacks are defined as edge.



1. Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O. M., & Bäck, T. (2019, July). Benchmarking discrete optimization heuristics with IOHprofiler. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (pp. 1798-1806).
2. <https://iohprofiler.github.io/IOHproblem/PBO>

Assignment Task

- ▶ Work in a team of up to 2.
- ▶ Enrol in the same group with your teammate on Brightspace.
- ▶ You need to submit
 - ▶ Source code (Python) with the required format
 - Python template will be provided.
 - We will run your codes, so please make sure your program meets the requirements.
 - Please submit the version which is consistent with the result in your report.
 - ▶ A Scientific report
 - We provide the template.
 - Exercise for writing scientific articles.
- ▶ Practical Assignment:
 - ▶ PA deadline: Dec. 1, 23:59
 - ▶ Every week late: 0.5 pts grade degradation



Requirements and Details

Requirements

- ▶ Implement a Genetic Algorithm (GA) to solve the F18 and F23 problems.
 - ▶ We set the search dimension to 50 for F18 and 49 for F23.
- ▶ Tune the hyper-parameters of your implementation with **1,000,000 function evaluations on both problems**
 - ▶ The hyperparameter setting determined by your tuning procedure has to work on both problems.
 - ▶ Using different hyperparameter settings on each problem is NOT allowed.
 - ▶ You can use ANY tuning method; think about it.
- ▶ Submit the code of GA, '*studentnumber1_studentnumber2_GA.py*'. For a group of three, '*studentnumber1_studentnumber2_studentnumber3_GA.py*'.
- ▶ Submit the code for the tuning procedure, '*studentnumber1_studentnumber2_tuning.py*'. For a group of three, '*studentnumber1_studentnumber2_studentnumber3_tuning.py*'.
 - ▶ The tuning code should make a function call to the GA implementation.

Requirements

- ▶ Additional files of other functions are allowed. However, we will only execute the GA and the tuning codes.
- ▶ Submit a report introducing your algorithms and presenting the experimental results. We will upload an example.
- ▶ Set a fixed random seed in the implementation to obtain the same results as those in the report by running the submitted codes.

Tuning Hyperparameters

- ▶ Goal: we want to get good performance upon **5,000 function evaluations on each problem**.
- ▶ GA is random: we will execute it for **20** independent runs and take an aggregated performance value.
- ▶ Now, I give you a much larger tuning budget, **1,000,000** function evaluations.
- ▶ You can use any tuning method you can find or come up with.
- ▶ The tuning method consumes maximally **1,000,000** function evaluations and outputs a hyperparameter setting, which makes the GA perform well on both problems with a budget of 5,000 function evaluations.

Think about..

- ▶ Which variations (i.e., mutation and crossover) and selection operators will you use?
- ▶ How to encode/decode the phenotype of the N-Queens problem?
- ▶ How to tune the hyper-parameters (e.g., population size, mutation rate, etc.)?
- ▶ How do you allocate the tuning budget?
 - ▶ Remember we want to make the GA work well at a fixed budget of **5,000 function evaluations**.
 - ▶ GA is random... meaning one single run is not representative of the quality of a hyperparameter setting
 - ▶ How do I tune for two different problems at the same time?

General Info

▶ How to evaluate your PA?

▶ Following the guidelines (10%)

- ▶ You will get a full score if you follow all the guidelines

▶ Experimental Results (45%)

- ▶ If your code reproduces the results in the report.

▶ Report (45%)

- ▶ Based on the presentation of the design of algorithms, experimental settings, and discussion about the results.

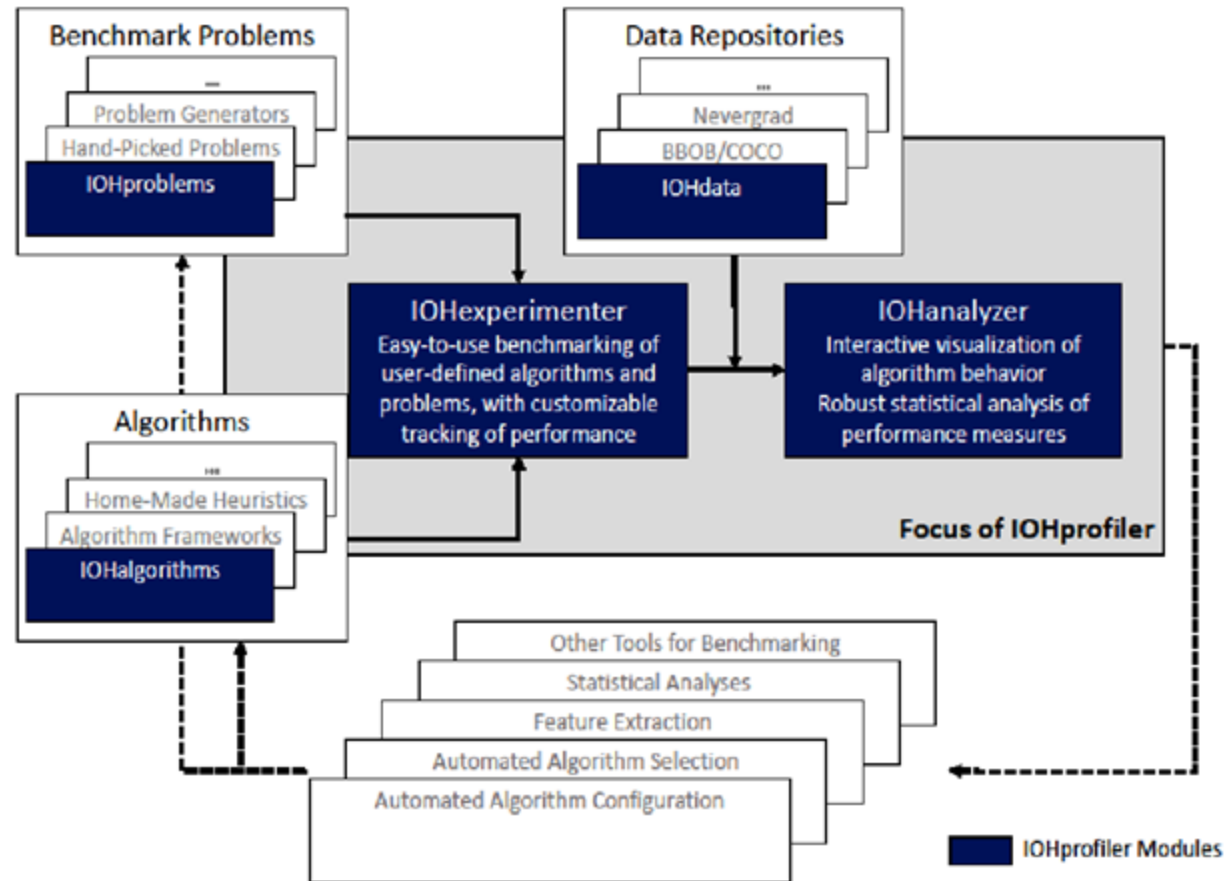
▶ Other:

- ▶ Plagiarism check: if the report copies more than 30%, the PA grade is 0.
- ▶ If the results in your report do not match the results we obtain from using your codes, the PA grade is 0.
- ▶ If the results of your algorithms rank top 2 among all teams, you will get a 0.5 bonus for the **final grade**.

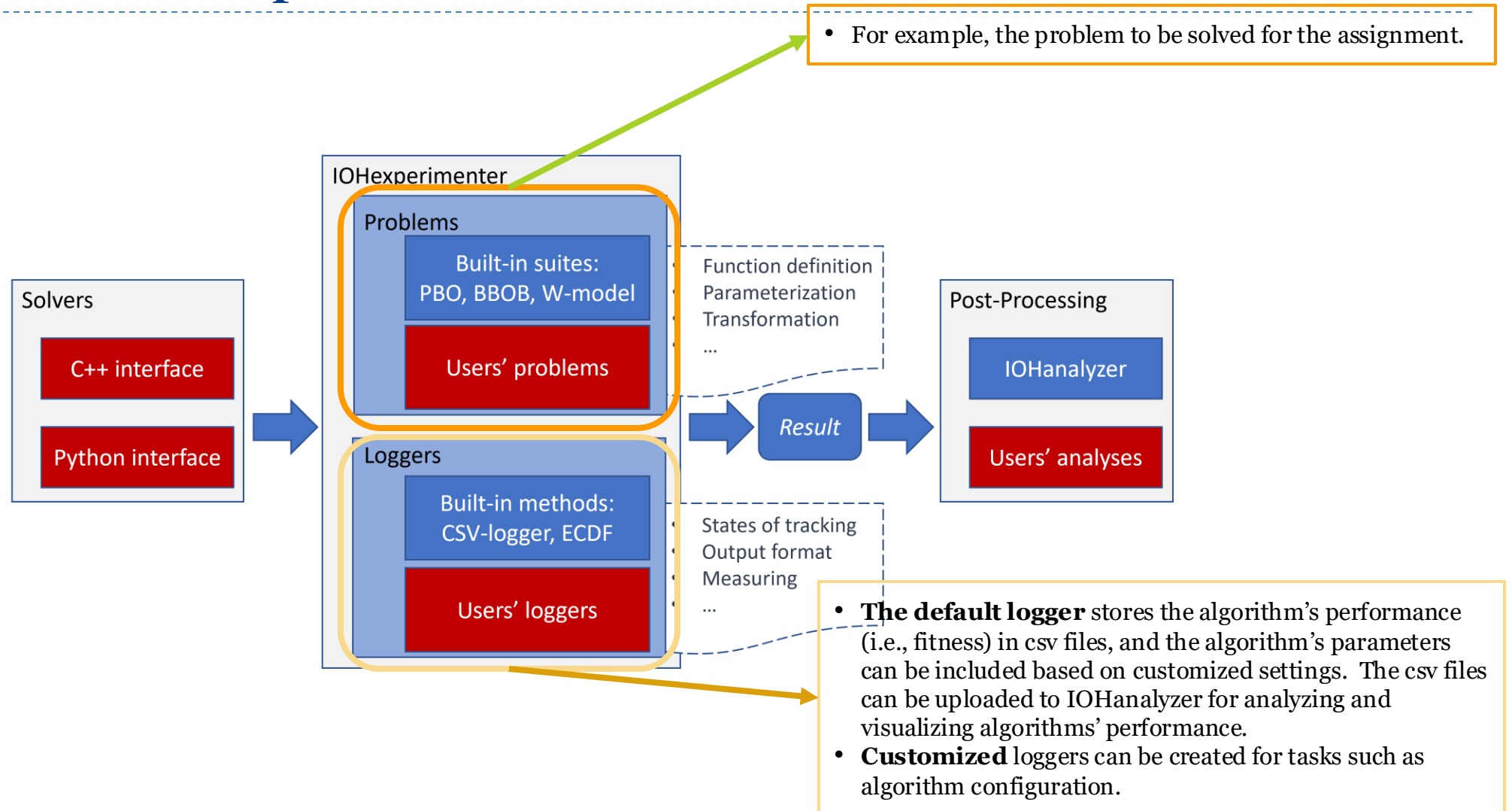


Visualize the empirical performance yourself with
IOHprofiler

IOHprofiler Architecture Overview



Workflow of IOHexperimenter



Usage of IOHexperimenter

- ▶ Installation: pip package is available at <https://pypi.org/project/ioh>
- ▶ A tutorial is available at <https://github.com/IOHprofiler/IOHexperimenter/blob/master/example/tutorial.ipynb>
- ▶ A full documentation is available at <https://iohprofiler.github.io/IOHexperimenter/python>

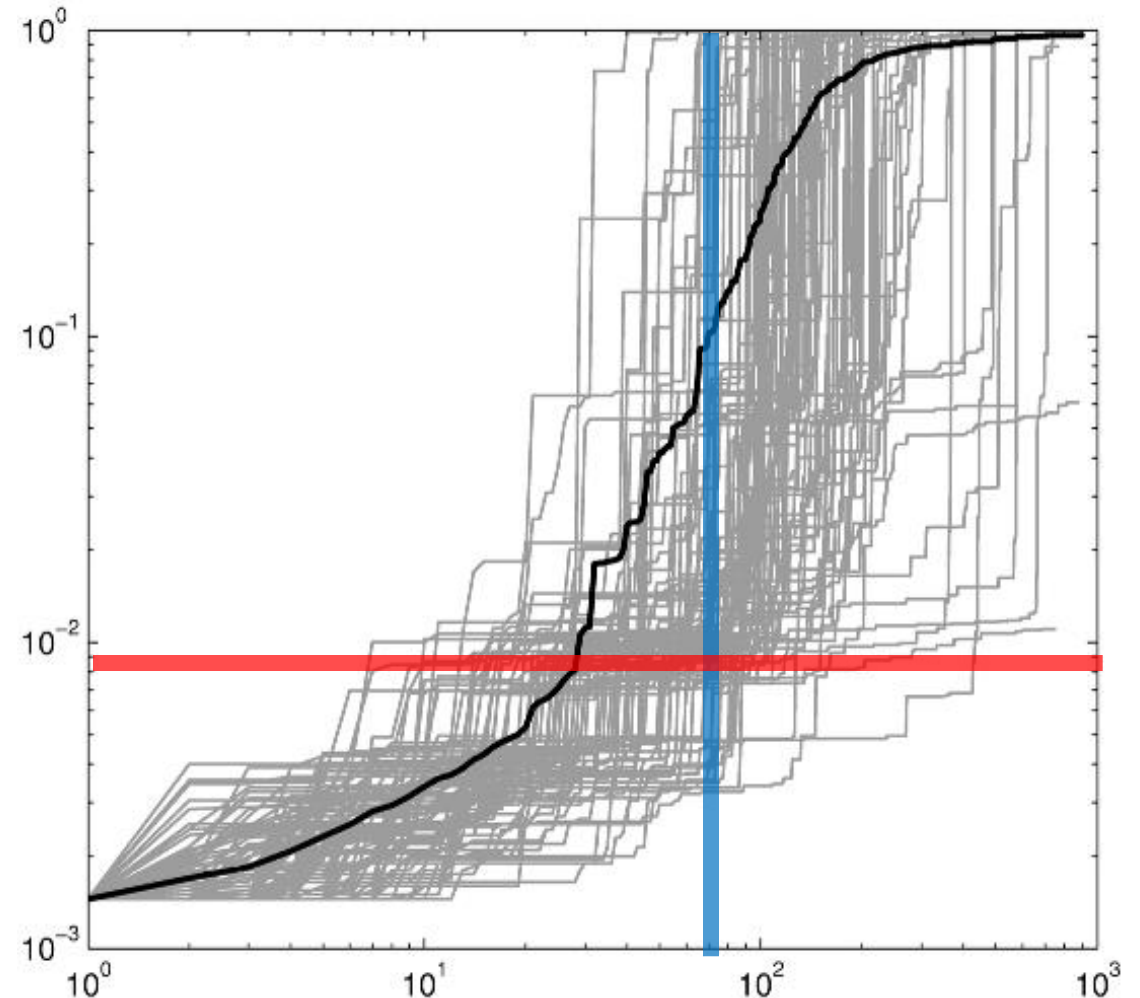
Performance Assessment with IOHanalyzer

- ▶ What perspective to consider?

Fixed-budget -
objective samples given
runtime budget

Fixed-target - runtime samples
given target value

success rate: some cruns might
not hit this line



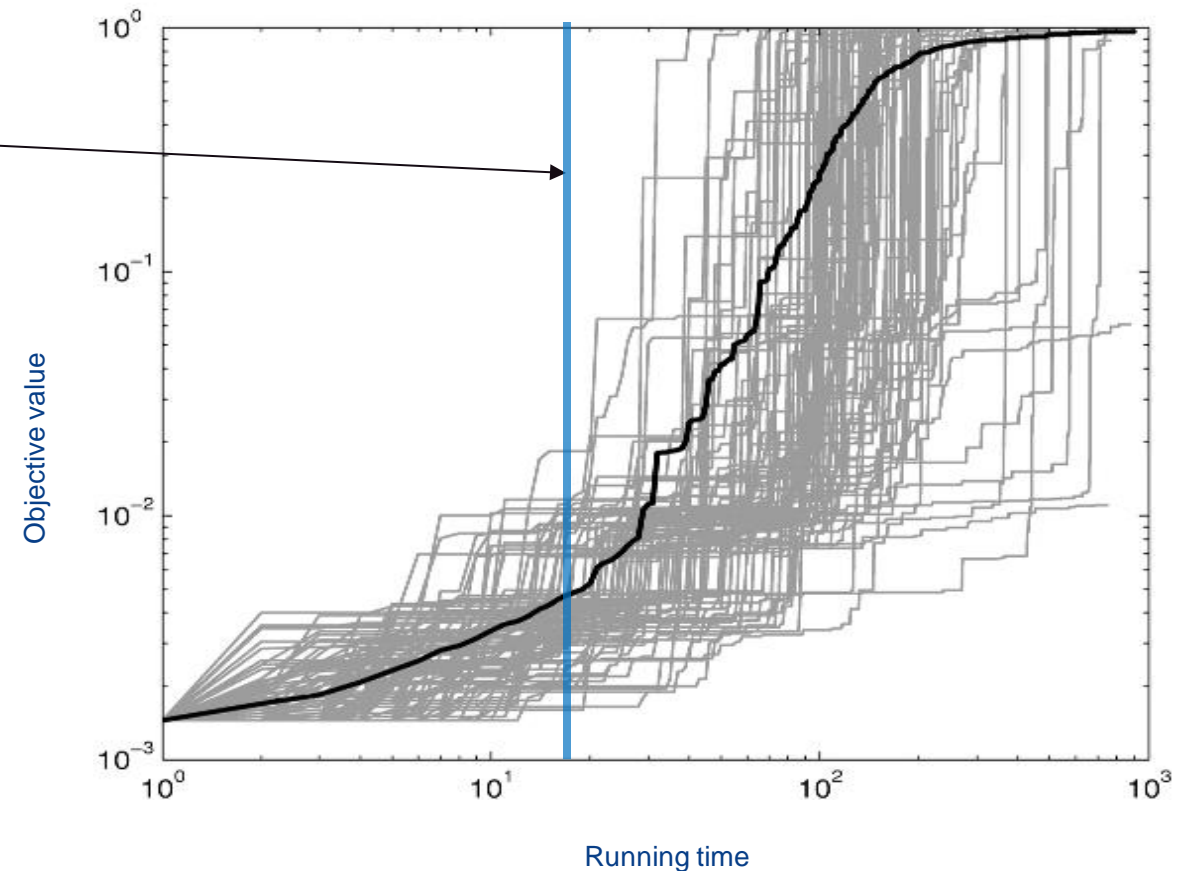
Fixed-budget analysis

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

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

- The number of runs – r

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





Algorithm Performance Measures

[ERT: Expected Running Time] Given a target value ϕ for a problem P , the ERT of algorithm A for hitting ϕ is

$$\text{ERT}(A, P, \phi) = \frac{\sum_{i=1}^r \min\{t_i(A, P, \phi), B\}}{\sum_{i=1}^r \mathbb{1}\{t_i(A, P, \phi) < \infty\}},$$

where r is the number of independent runs of A , B is the cutoff time (i.e., the maximal number of solution candidates that algorithm A may evaluate in one run), $t_i(A, P, \phi) \in \mathcal{N} \cup \{\infty\}$ is the running time (for finite values, the running time is the number of function evaluations that the i -th run of A on problem P uses to hit the target ϕ and $t_i(A, P, \phi) = \infty$ is used if none of the solutions is better than ϕ), and $\mathbb{1}(\mathcal{E})$ is the indicator function returning 1 if event \mathcal{E} is true and 0, otherwise. If the algorithm hits the target ϕ in all r runs, the ERT is equal to the average hitting time (AHT).

Algorithm Performance Measures

[**ECDF**: empirical cumulative distribution function of the running time] Given a set of targets $\Phi = \{\phi_i \in \mathbb{R} \mid i \in \{1, 2, \dots, m\}\}$ for a real-valued problem P and a set of budgets $T = \{t_j \in \mathbb{N} \mid j \in \{1, 2, \dots, B\}\}$ for an algorithm A , the ECDF value of A at budget t_j is the fraction of (run, target)-pairs (r, ϕ_i) that satisfy that the run r of the algorithm A finds a solution has fitness at least as good as ϕ_i within the budget t_j .

Algorithm Performance Measures

[**AUC**: area under the ECDF curve] Given a set of targets $\Phi = \{\phi_i \in \mathbb{R} \mid i \in \{1, 2, \dots, m\}\}$ and a set of budgets $T = \{t_j \in \{1, 2, \dots, B\} \mid j \in \{1, 2, \dots, z\}\}$, the $\text{AUC} \in [0, 1]$ (normalized over B) of algorithm A on problem P is the area under the ECDF curve of the running time over multiple targets. For minimization, it reads

$$\text{AUC}(A, P, \Phi, T) = \frac{\sum_{h=1}^r \sum_{i=1}^m \sum_{j=1}^z \mathbb{1}\{\phi_h(A, P, t_j) \leq \phi_i\}}{rmz},$$

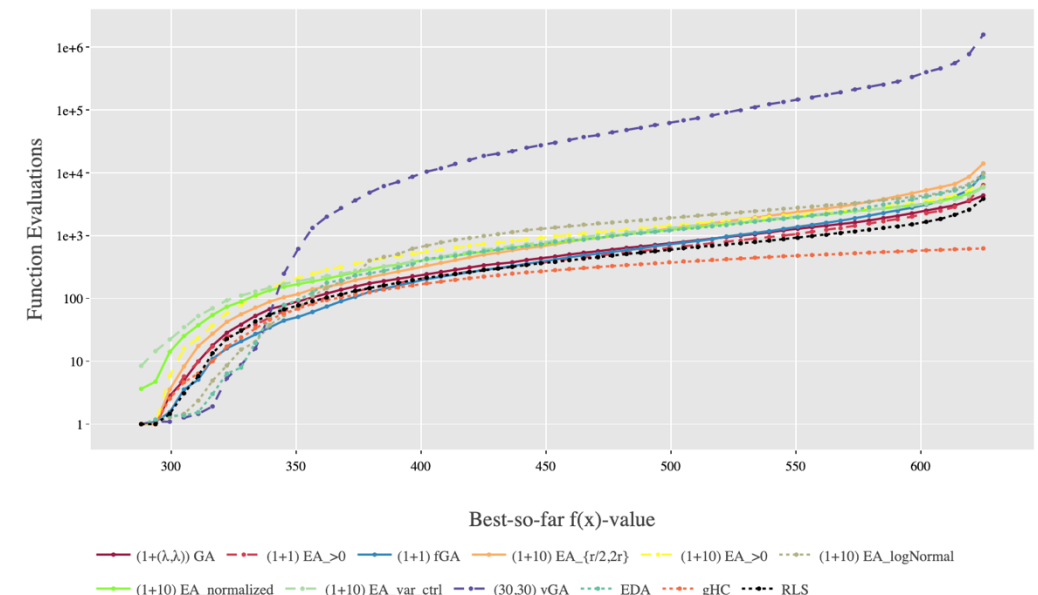
where r is the number of independent runs of A and $\phi_h(A, P, t)$ denotes the value of the best solution that A evaluated within its first t evaluations of the run h . Note that, for this assignment, we consider an approximation of AUC for a set of budgets $T \subset \{1, 2, \dots, B\}$.

Usage of IOHanalyzer

- ▶ GUI: <https://iohanalyzer.liacs.nl>
- ▶ GitHub project: <https://github.com/IOHprofiler/IOHanalyzer> . You can also install a local version following the tutorial on the git page.
- ▶ Paper: <https://dl.acm.org/doi/full/10.1145/3510426>

What to report

- ▶ Description of the Genetic Algorithm
 - ▶ Introduce the operators and the parameter setting that you suggest.
 - ▶ Introduce the approach to the hyper-parameter process.
- ▶ Report the average best-found fitness $f(x)$ of **20 independent runs**
- ▶ Report the AUC values
- ▶ The empirical analysis of your algorithms' performance using ERT and ECDF curves
- ▶ Any other details/results you wish to include



IOHanalyzer

Upload Data

General Overview

Fixed-Target Results

Fixed-Budget Results

Single Function

Data Summary

Expected Target Value

Probability Density Function

Cumulative Distribution

Algorithm Parameters

Statistics

Multiple Functions

About

Settings

Function ID:

2309

Dimension:

26

Expected Target Value (per function)

Range of the displayed budget values

B_{\min} : Smallest budget value

1

B_{\max} : Largest budget value

13

Select which IDs to include:

random-search

☒ Show/hide (arithmetic) mean

☐ Show/hide Geometric mean

☐ Show/hide median

☐ Show/hide mean +/- sd

☐ Show/hide interquartile range

☒ Scale x axis \log_{10}

☐ Scale y axis \log_{10}

☐ Show individual runs

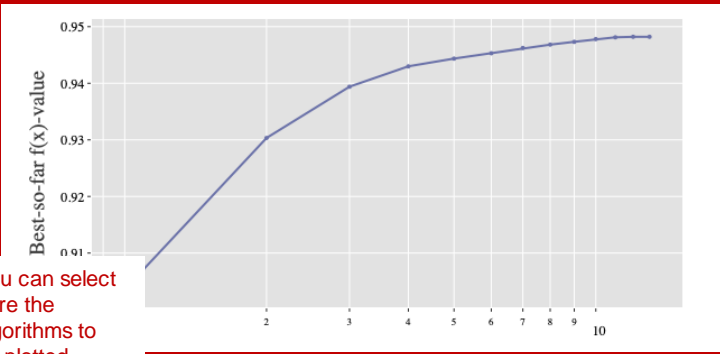
Select the figure format

pdf

Download the figure

The *mean, median and standard deviation* of the best function values found with a fixed-budget of evaluations are depicted against the budget. The displayed elements can be switched on and off by clicking on the legend on the right. A **tooltip** and **toolbar** appears when hovering over the figure.

Best-so-far $f(x)$ -value



random-search

Function Evaluations

You can select here the algorithms to be plotted

How to access the best-found fitness using IOHanalyzer.

The figure presents the convergence process of your algorithm while the function evaluation increases.

Useful information, e.g., best-found fitness, the average of the best-found fitness, etc., can be obtained in the plotted table.

IOHanalyzer

Upload Data

General Overview

Fixed-Target Results

Fixed-Budget Results

Single Function

Data Summary

Expected Runtime

Probability Mass Function

Cumulative Distribution

Algorithm Parameters

Statistics

Multiple Functions

About

Settings

Data Overview

Select which IDs to include:

random-search

File-format

csv

Save this table

This table provides an overview on function values for all algorithms chosen on the left:

- worst recorded: the worst $f(x)$ value ever recorded across *all* iterations,
- worst reached: the worst $f(x)$ value reached in the *last* iterations,
- best reached: the best $f(x)$ value reached in the *last* iterations,
- mean reached: the mean $f(x)$ value reached in the *last* iterations,
- median reached: the median $f(x)$ value reached in the *last* iterations,
- succ: the number of runs which successfully hit the best reached $f(x)$.

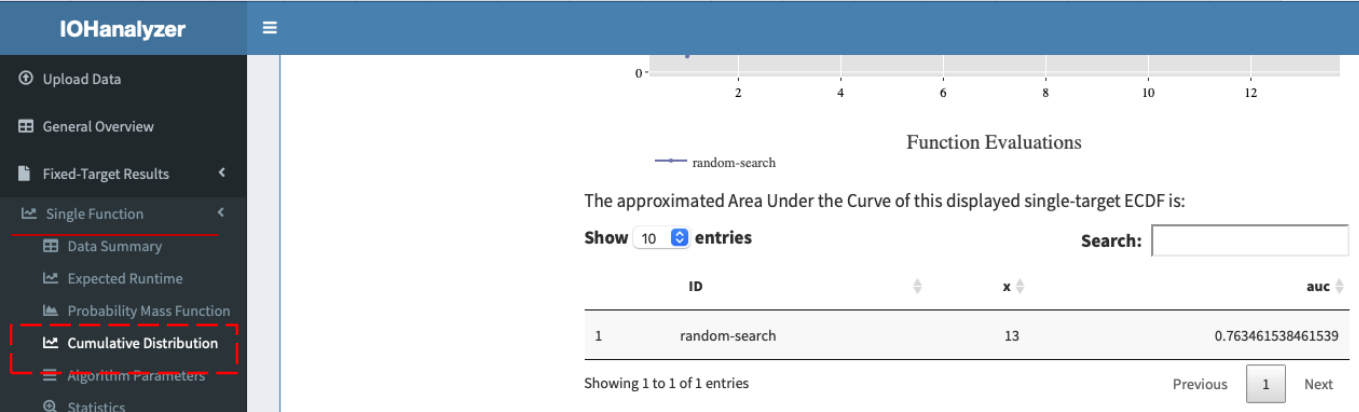
Show 15 entries

ID	DIM	funcId	worst recorded	worst reached	best reached	mean reached	median reached	runs
1	random-search	26	2309	0.82	0.95	0.95	0.95	20

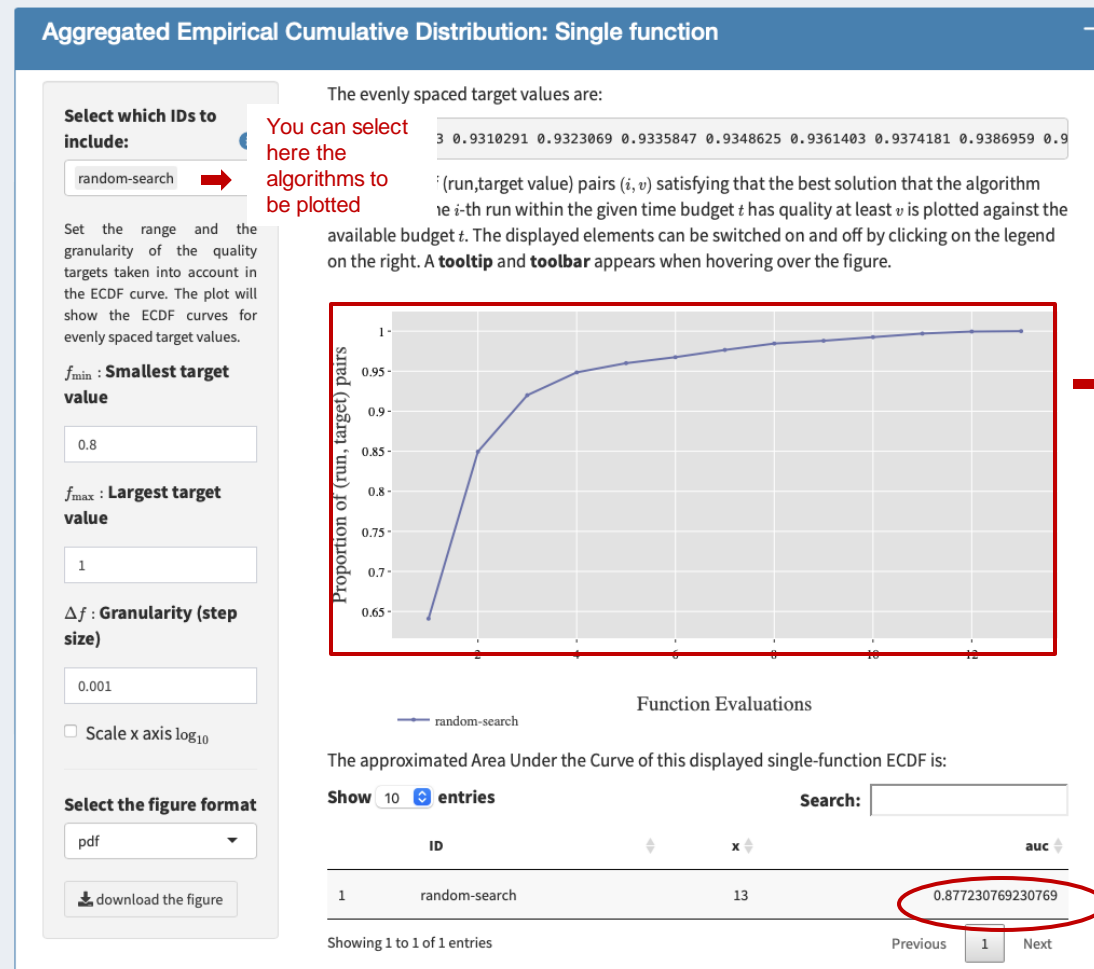
Showing 1 to 1 of 1 entries

Dr. Hao Wang | LIACS Natural Computing Group | Evolutionary algorithms

29



How to access the ECDF curves and the AUC values using IOHanalyzer.



The figure shall be downloaded and presented in your submission. Please explain your observations on the figure.

This is the approximation of the AUC value you shall report.

Tutorial

- ▶ Install ioh package
 - ▶ In terminal: “python -m pip install ioh”
 - ▶ You can follow the tutorial (<https://github.com/IOHprofiler/IOHexperimenter/blob/master/example/tutorial.ipynb>) if you are interested with the additional functionalities of the tool.