Universiteit Leiden
The Netherlands

Pratical Assignment

Evolutionary Algorithms Course, LIACS, 2022-2023

General Info

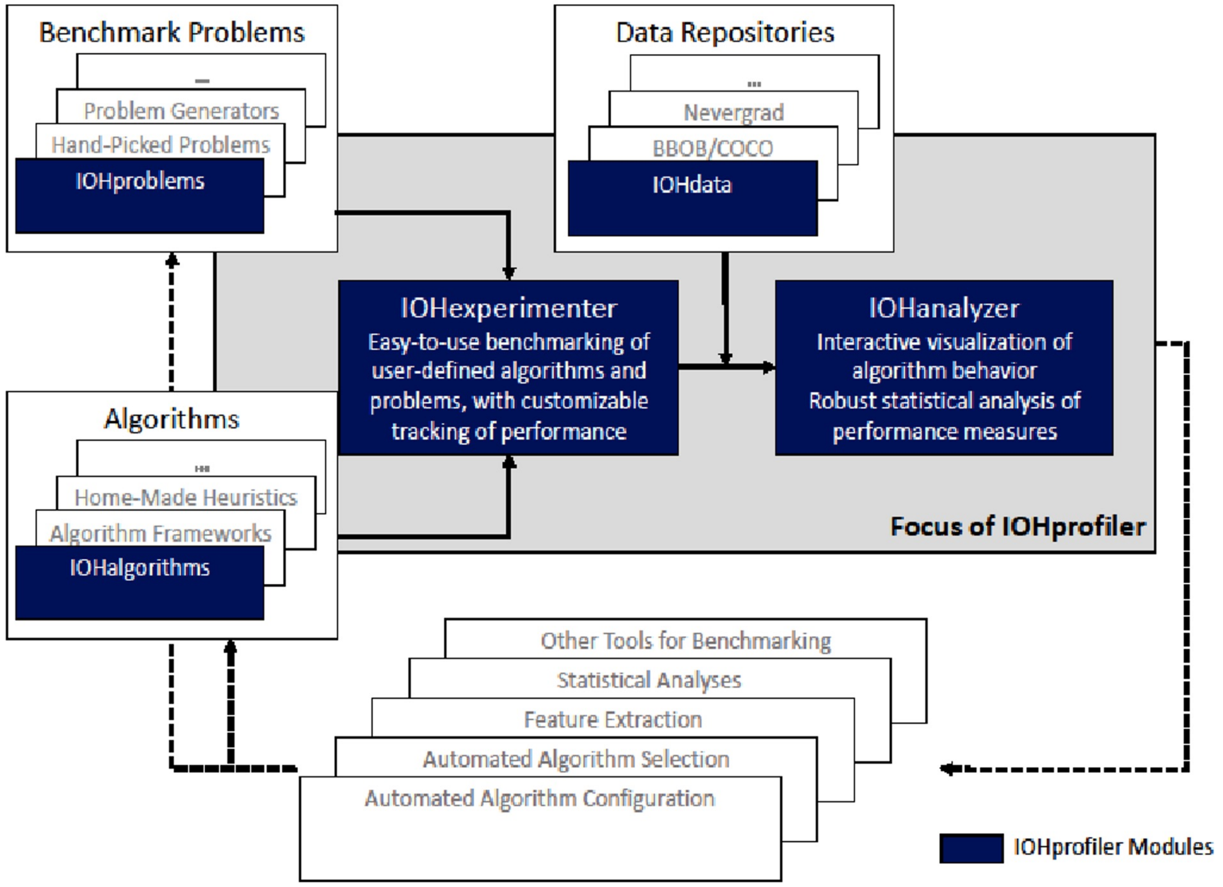
- Work in a team of up to 2.
- Enrol in the same group with your teammate in Brightspace (even if you work individually)

- You need to submit
  - Source code (Python) with requried format
    - ☐ We will run your codes with a script, so please make sure you program is compatible with the requirement.
    - ☐ 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

- Pactical Assignment:
  - PA ddl: Dec. 7, 23:59
  - Every week late: 0.5 pts grade degradation

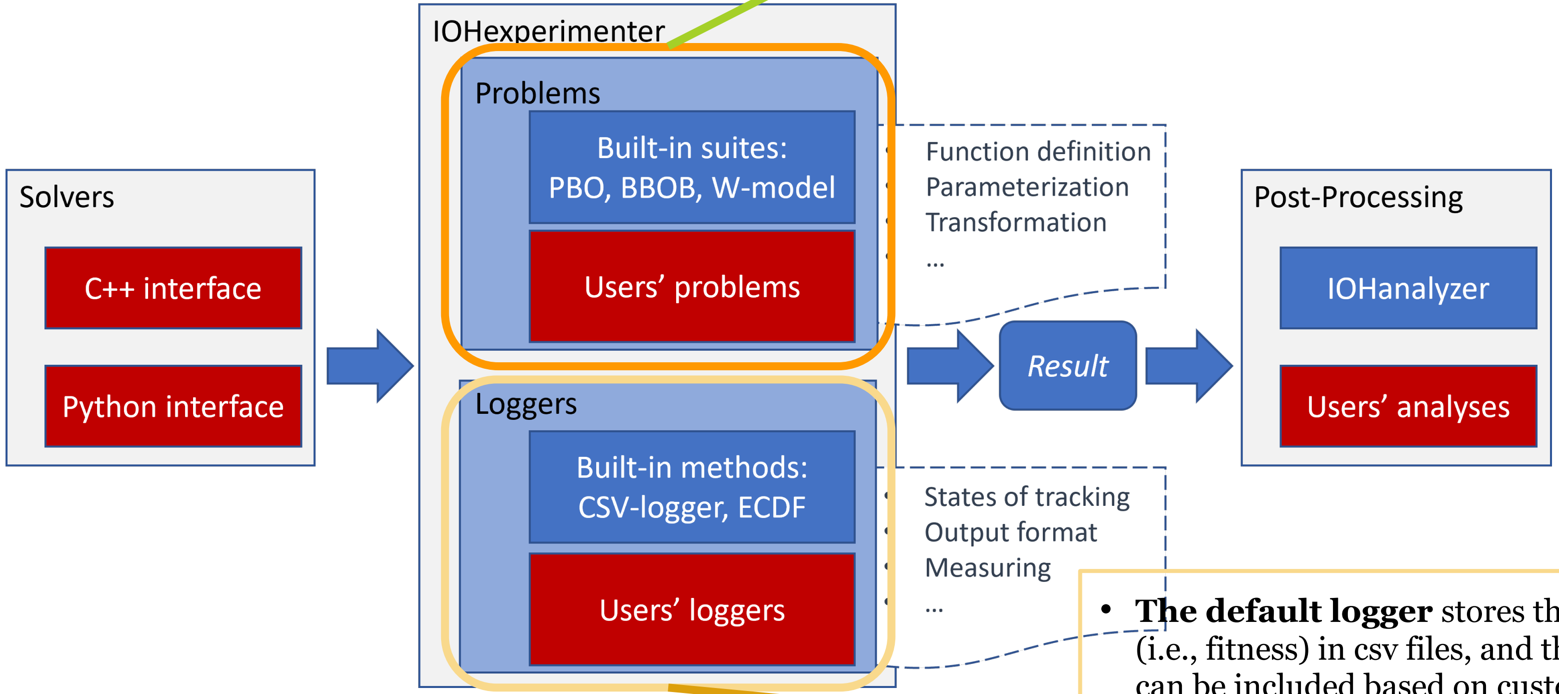
IOHprofiler
A modular benchmarking platform

IOHprofiler Architecture Overview


Benchmark Problems
...
Problem Generators
Hand-Picked Problems
IOHproblems
Data Repositories
...
Nevergrad
BBOB/COCO
IOHdata
IOHexperimenter
Easy-to-use benchmarking of user-defined algorithms and problems, with customizable tracking of performance
IOHanalyzer
Interactive visualization of algorithm behavior
Robust statistical analysis of performance measures
Algorithms
...
Home-Made Heuristics
Algorithm Frameworks
IOHalgorithms
Focus of IOHprofiler
Other Tools for Benchmarking
Statistical Analyses
Feature Extraction
Automated Algorithm Selection
Automated Algorithm Configuration
IOHprofiler Modules

Workflow of IOHexperimenter

- For example, the problem to be solved for the assignment.



- **The default logger** stores the algorithm's performance (i.e., fitness) in csv files, and the algorithm's parameters can be included based on customized settings. The csv files can be uploaded to IOHanalyzer for analyzing and visualizing algorithms' performance.
- **Customized** loggers can be created for tasks such as algorithm configuration.

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

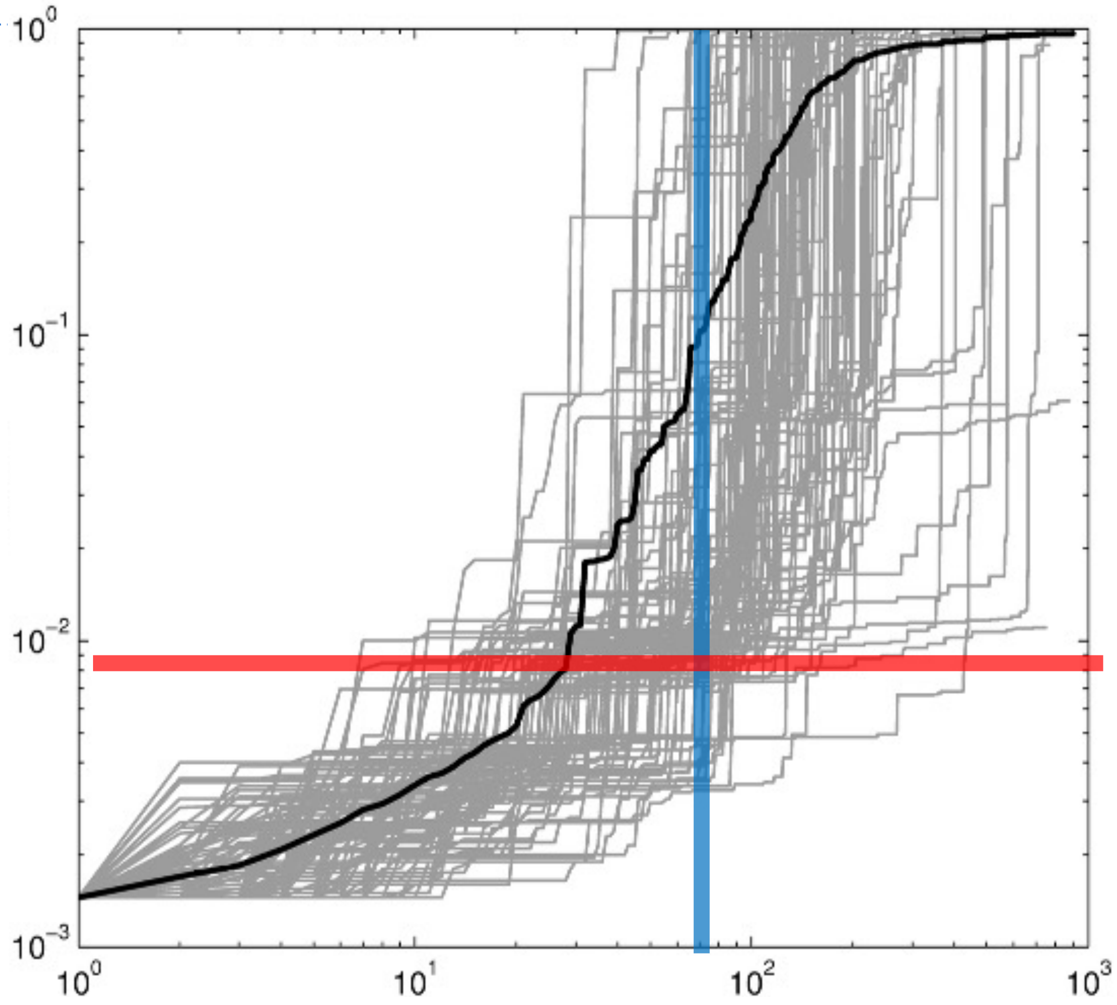
IOHanalyzer overview

- 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



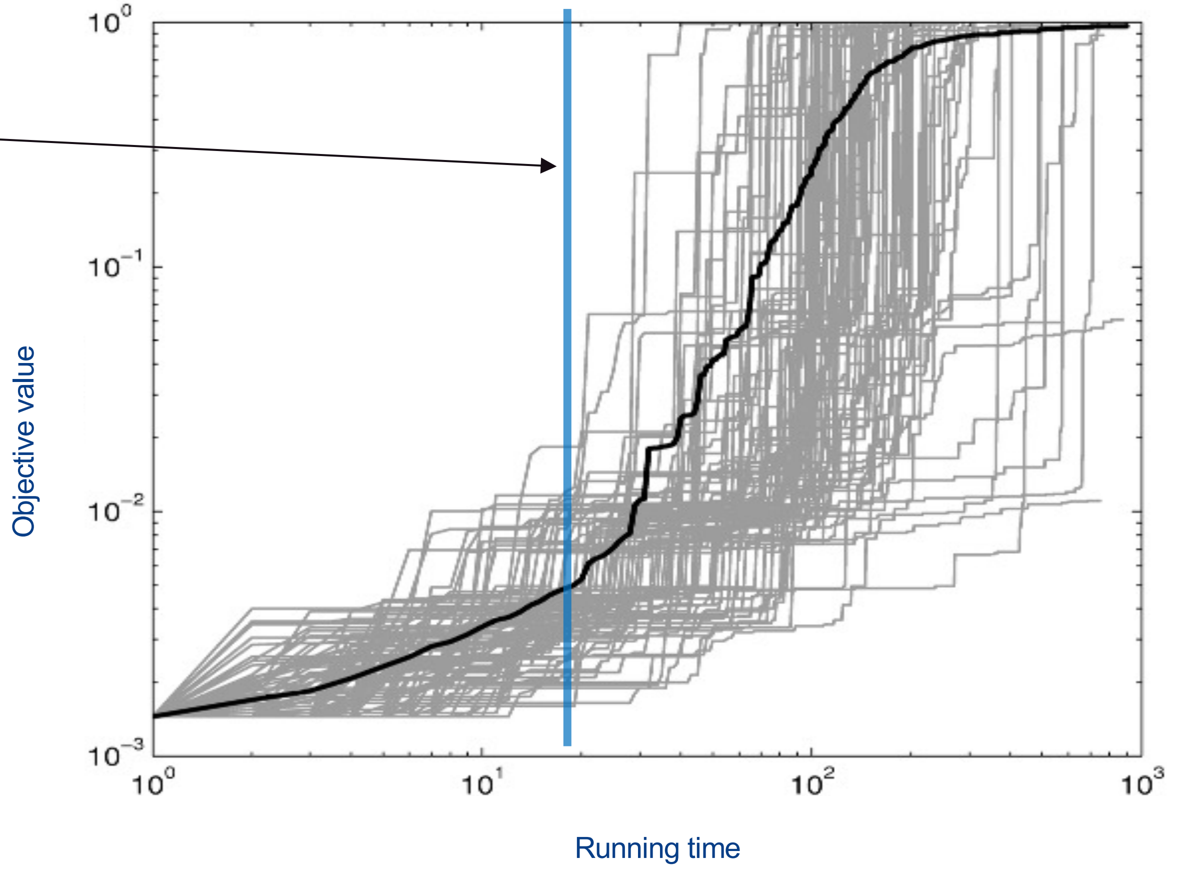
Fixed-budget analysis

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

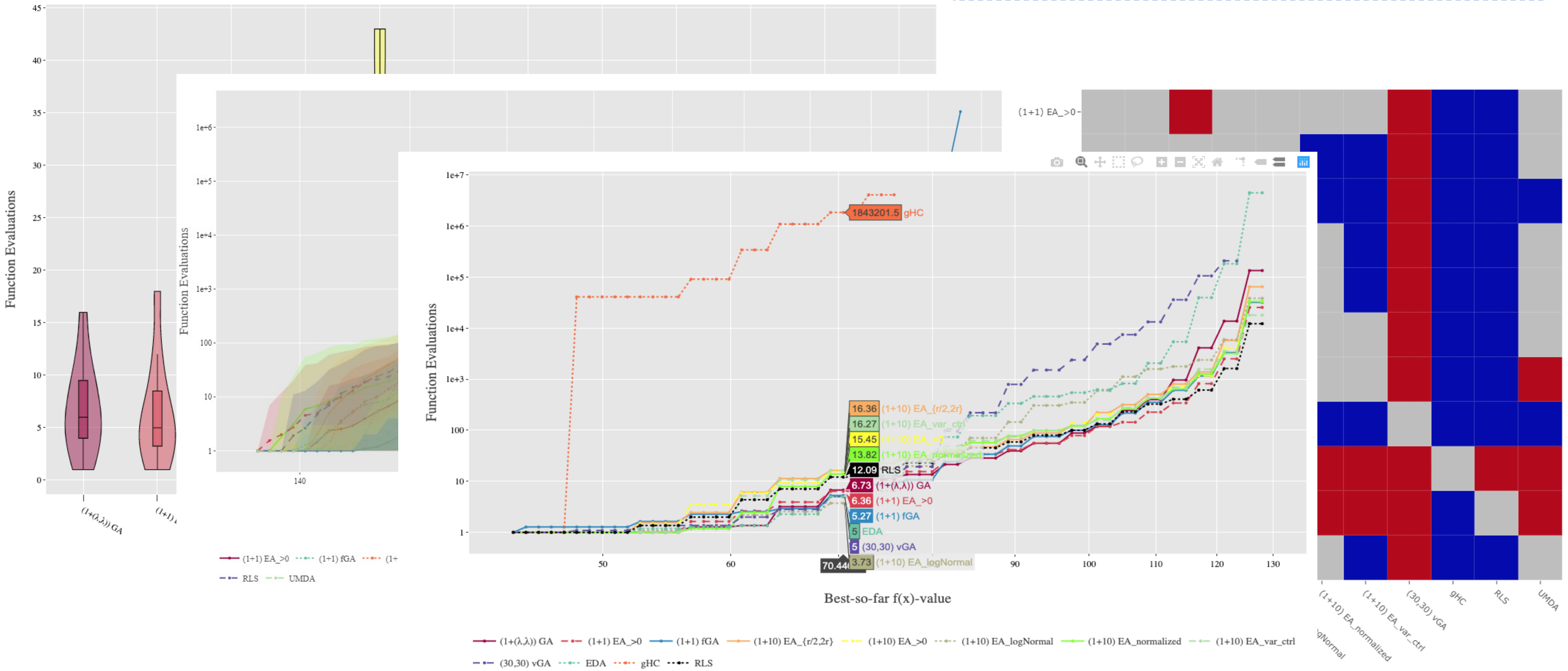
$$V(b) \in \mathbb{R}, \boxed{b} \leq B$$

- The number of runs – *r*

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



Fixed-target Analysis



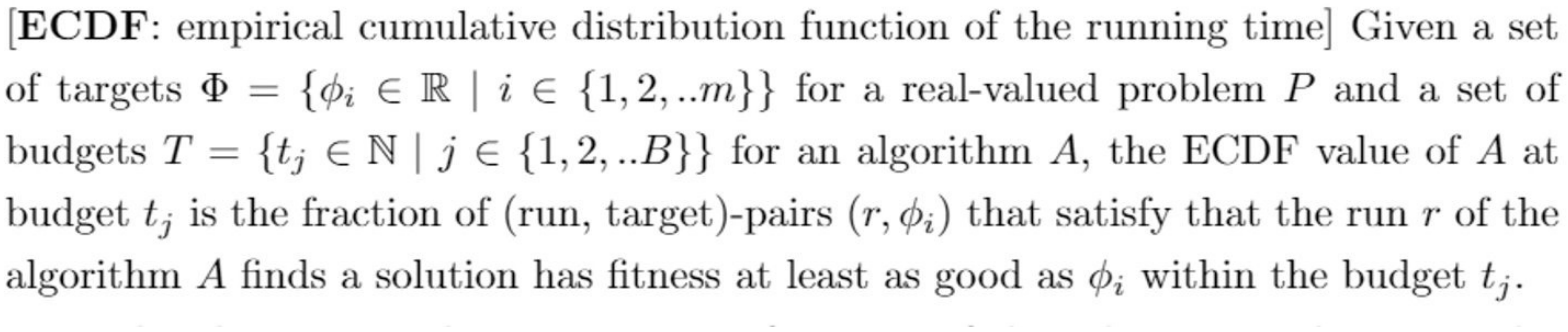
Algorithm Performance Measures

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

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

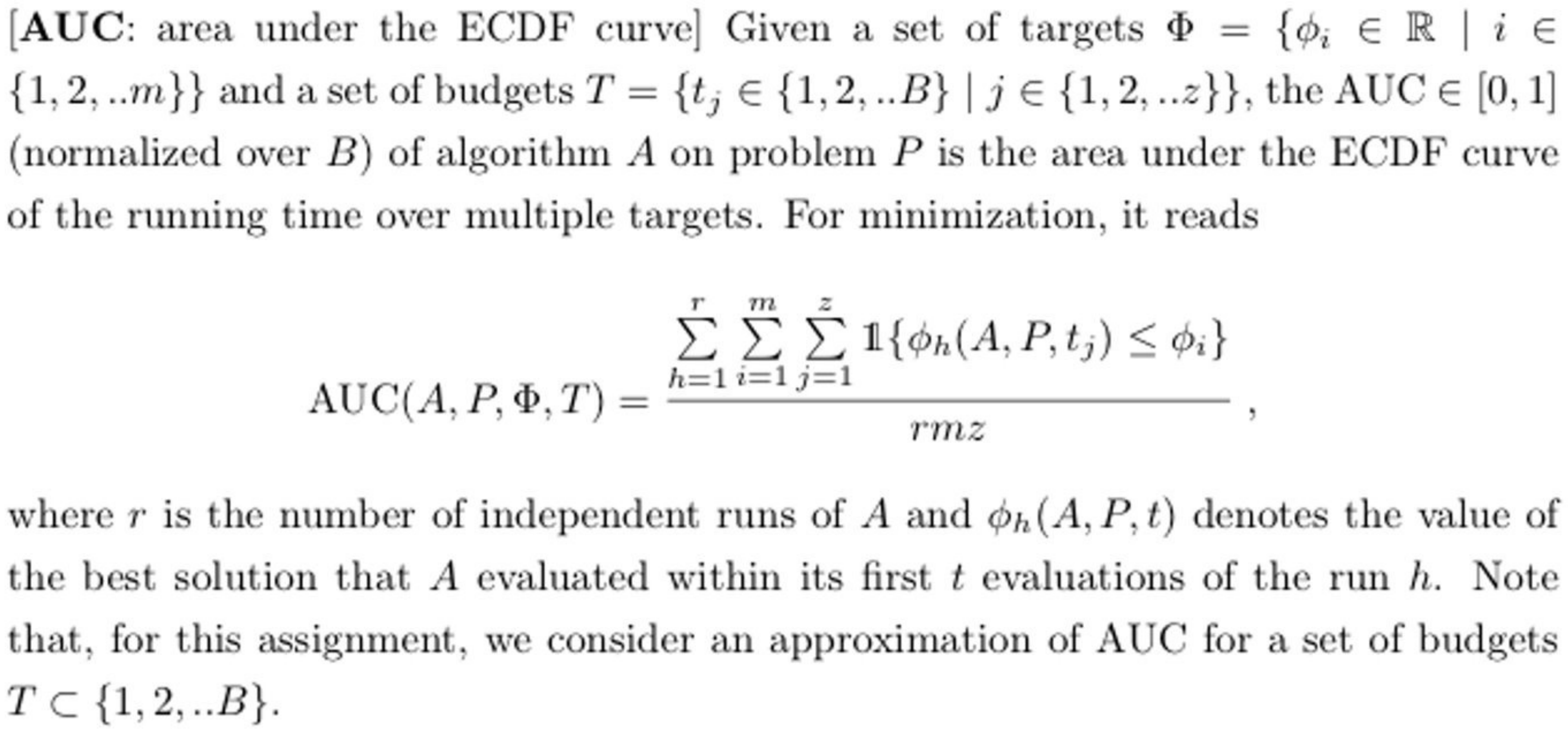
where r is the number of independent runs of A, B is the cutoff time (i.e., the maximal number of solution candidates that algorithm A may evaluate in one run), $t_i(A, P, \phi) \in \mathcal{N} \cup \{\infty\}$ is the running time (for finite values, the running time is the number of function evaluations 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 $\Phi = \{\phi_i \in \mathbb{R} \mid i \in \{1, 2, ..m\}\}$ for a real-valued problem P and a set of budgets $T = \{t_j \in \mathbb{N} \mid j \in \{1, 2, ..B\}\}$ 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 $\Phi = \{\phi_i \in \mathbb{R} \mid i \in \{1, 2, ..m\}\}$ and a set of budgets $T = \{t_j \in \{1, 2, ..B\} \mid j \in \{1, 2, ..z\}\}$, the 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

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

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 $T \subset \{1, 2, ..B\}$.

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

Neural Architecture Search Using Evolutionary Algorithms

Neural Architecture Search (NAS)

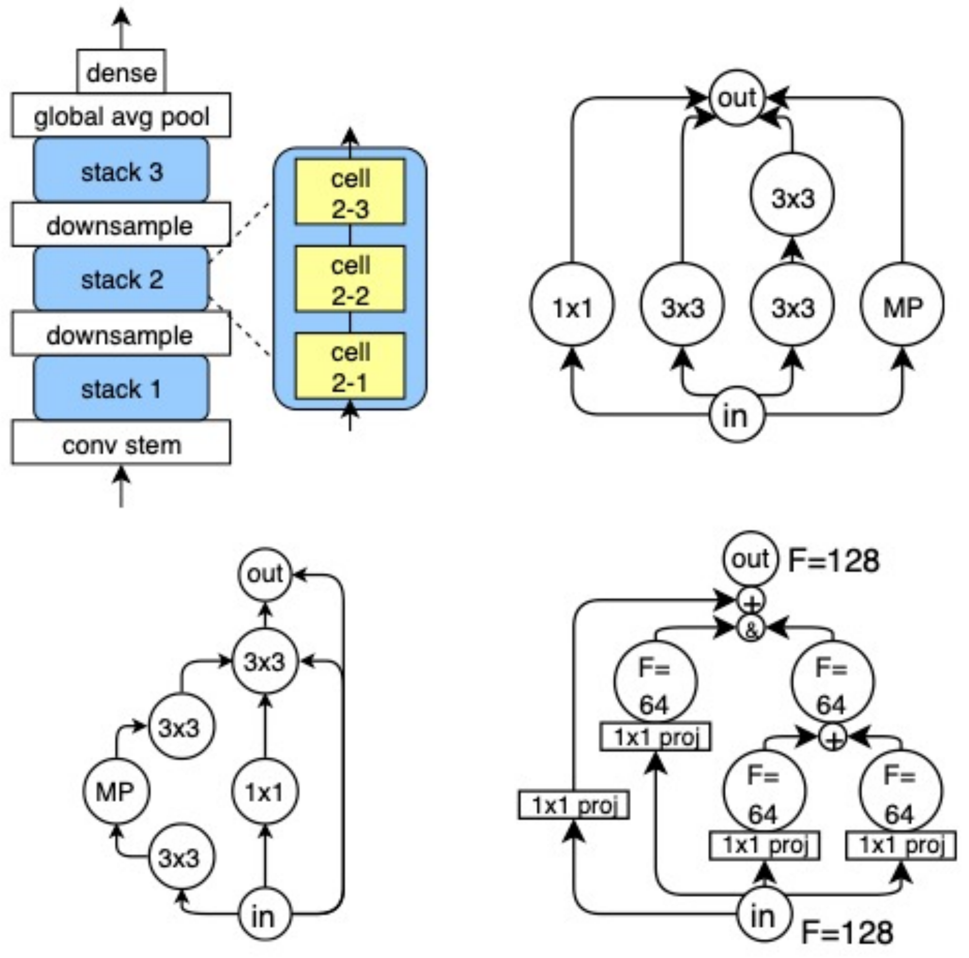
- Neural network has achieved significant success in applications of various domains, and this success requires novel and proper architecture designs.
- Neural Architecture Search (NAS) address the technique of automatically learning and constructing a well-performing architecture for specific tasks.

- *The NAS-Bench-101 provides a dataset mapping neural architectures to their training and evaluation metrics. It allows us to evaluate various neural architectures quickly and test NAS algorithms within a limited time.*

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., & Hutter, F. (2019, May). Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning* (pp. 7105-7114). PMLR.

NAS-Bench-101

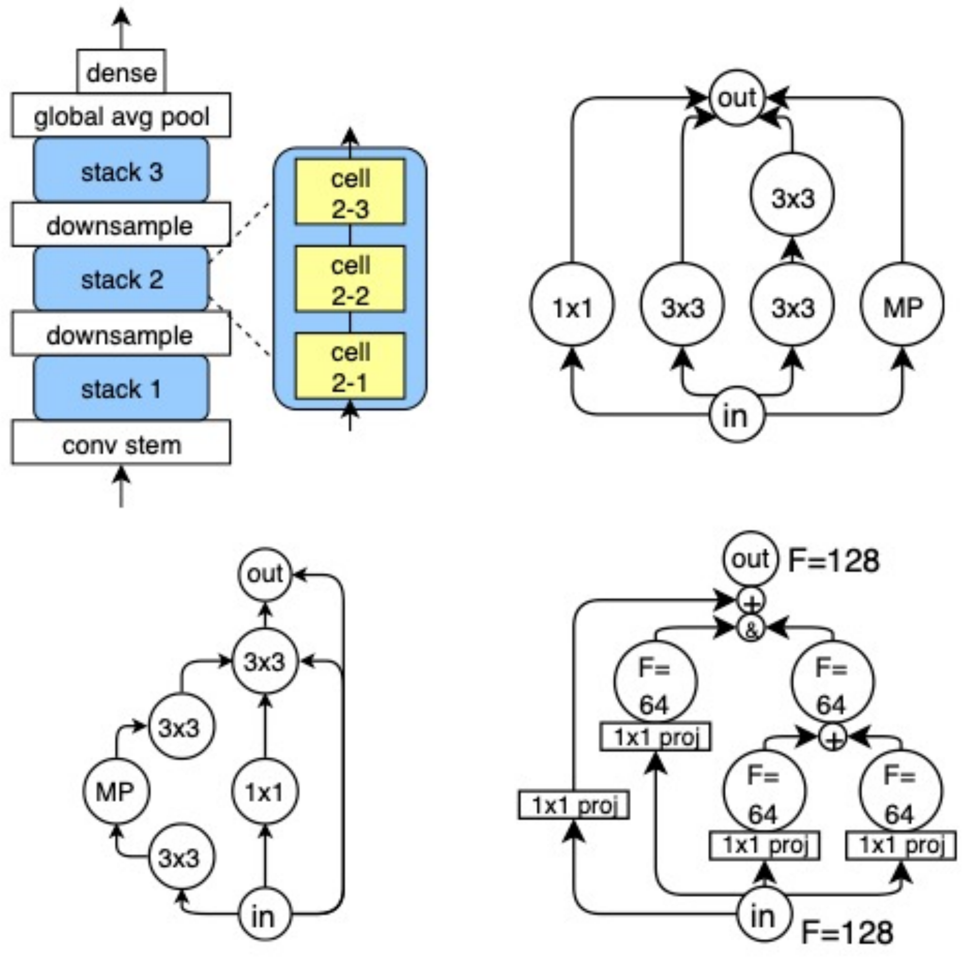
- NAS-bench-101 restricts the search for neural net topologies to the space of small feedforward structures, usually called *cells*. It stacks each cell 3 times, followed by a downsampling layer, in which the image height and width are halved via max-pooling and the channel count is doubled. It repeats this pattern 3 times, followed by global average pooling and a final dense softmax layer. The initial layer of the model is a *stem* consisting of one 3 × 3 convolution with 128 output channels



Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., & Hutter, F. (2019, May). Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning* (pp. 7105-7114). PMLR.

NAS-Bench-101

- The space of cell architectures consists of all possible directed acyclic graphs on V nodes, where each possible node has one of L labels, representing the corresponding operation. Two of the vertices are specially labelled as operation IN and OUT, representing the input and output tensors to the cell, respectively.
- It sets $L = 3$, using only the following operations:
  - – 3×3 convolution
    – 1×1 convolution
    – 3×3max-pool
- It limits $V \leq 7$.
- It limits the maximum number of edges to 9.



Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., & Hutter, F. (2019, May). Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning* (pp. 7105-7114). PMLR.

Problem Presentation

- We replace the encoding of the NAS problem in this assignment by a set of discrete variables. In practice, this assignment asks to solve a 26 dimensional discrete optimization problem, of which the first 21 variables $x_i \in \{0,1\}, i \in [1..21]$ represent the upper-triangular binary matrix of NAS, and the last 5 variables $x_i \in \{0,1,2\}, i \in [22..26]$ represent the operations of the 5 intermediate vertices of NAS.
- A solution $x = \{x_1, \dots, x_{26}\}$ maps to a structure of NAS as

$$\text{Adjacency matrix} = \begin{bmatrix} 0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 0 & 0 & x_7 & x_8 & x_9 & x_{10} & x_{11} \\ 0 & 0 & 0 & x_{12} & x_{13} & x_{14} & x_{15} \\ 0 & 0 & 0 & 0 & x_{16} & x_{17} & x_{18} \\ 0 & 0 & 0 & 0 & 0 & x_{19} & x_{20} \\ 0 & 0 & 0 & 0 & 0 & 0 & x_{21} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\text{Operations at the 7vertices} = [\text{INPUT}, x_{22}, x_{23}, x_{24}, x_{25}, x_{26}, \text{OUTPUT}],$$

- 3×3 convolution, 1×1 convolution, and 3×3 max-pool will be operated at the corresponding vertex when $x_i = 0, 1,$ and $2, i \in [22..26]$, respectively
- The fitness value is the accuracy rate of the corresponding neural architecture structure, ranging from 0 to 1.

Implementation Details

Requirements

- Implement a genetic algorithm solving the NAS problem

- Implement an evolution strategy solving the NAS problem

- Submit at least two files ‘*sstudentnumber1_sstudentnumber2_GA.py*’ and ‘*sstudentnumber1_sstudentnumber2_ES.py*’.
- Additional files of other functions are allowed. Please make sure we can get results by running ‘*python *_GA.py*’ and ‘*python *_ES.py*’ without additional arguments.

- Submit a report introducing your algorithms and presenting the experimental results.

Task 1: Genetic Algorithm

- Solving the NAS problem $f: \mathbb{Z}^n \rightarrow \mathbb{R}$ to be maximized.
- The problem can be defined as (in dimension $n = 26$):
  - $f(\mathbf{z}) \rightarrow \max, z_i \in \{0,1\}$ for $i = 1, \dots, 21,$ and $z_i \in \{0,1,2\}$ for $i = 22, \dots, 26$
  - We know that $f \in [0,1]$
  - We denote $f(z) = 0$ when z represents an invalid neural architecture module

- How to handle the discrete variables $z \in \{0,1,2\}$?
- Which variators (i.e., mutation and crossover) and selection operators will you use?
- What's your suggestion for the parameter settings (e.g., population size, mutation rate, etc.)?

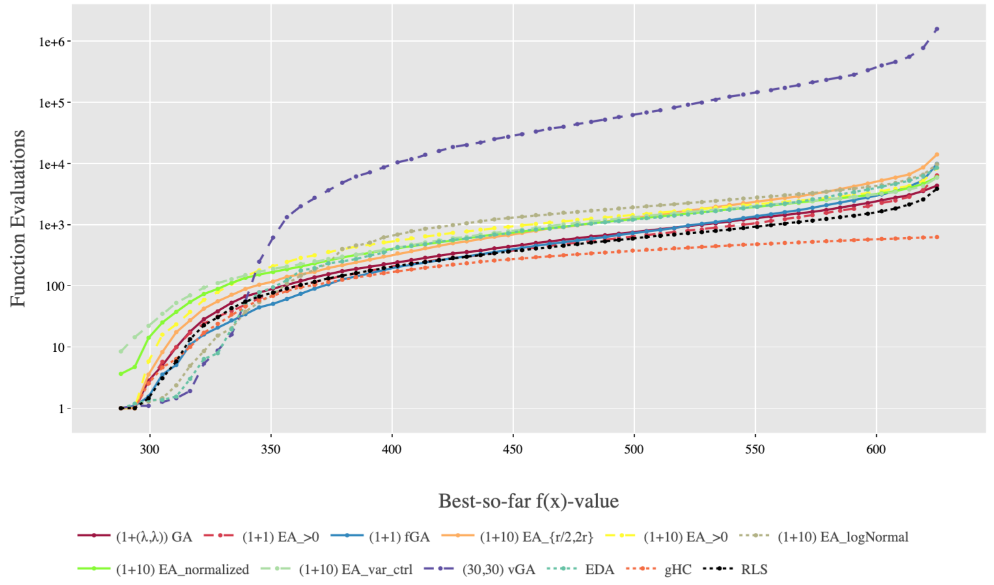
Task 2: Evolution Strategy

- Solving the NAS problem $f: \mathbb{Z}^n \rightarrow \mathbb{R}$ to be maximized.
- The problem can be defined as (in dimension $n = 26$):
  - $f(\boldsymbol{z}) \rightarrow \max, z_i \in \{0,1\}$ for $i = 1, \dots, 21$, and $z_i \in \{0,1,2\}$ for $i = 22, \dots, 26$
  - We know that $f \in [0,1]$
  - We denote $f(z) = 0$ when z represents an invalid neural architecture module

- How to handle the representation of discrete variables while applying evolution strategies?
- What encoding and decoding methods will you use?
- Which variators (e.g., one-σ mutation, correlated mutation, discrete recombination, etc) and selection operators will you use?
- What's your suggestion for the parameter settings (e.g., population size, step size, etc.)?

What to report

- Description of the genetic algorithm and the evolution strategy
  - Introduce the operators and the parameter setting that you suggest. Though you might have tested different configurations, please make it clear which setting you choose for the submission
- 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
- ……



General Info

- How to evaluate your PA?
  - Following the guidelines (10%)
    - You will get full score if you follow all guidelines
  - Experimental Results (45%)
    - Based on the rank of your algorithm result among algorithms of all groups.
  - Presentation (45%)
    - Based on the presentation of the design of algorithms, experimental settings, and discussion about the results.
- Other:
  - Plagiarism check: if report copies more than 30%, PA grade is 0.
  - If the results in your report do not match the results we obtain from using your codes, PA grade is 0.
  - If the results of your algorithms rank top 2, you will get a 0.5 bonus for the final grade.

Evaluation

- The submissions will be ranked based on the *averaged best-found fitness* $f(x)$ and AUC *values* obtained using the genetic algorithm and the evolution strategy (there will be four rank lists).
- For each list (i.e., f_ga, f_es, auc_ga, and auc_es), your score will be calculated as

$$10 - \frac{s - s_{best}}{s_{best} - s_{last}} \times 3,$$

where s is the performance value obtained by your algorithm, s_{best} is the best value obtained by all the submissions, and s_{last} is the lowest ranked value of all the submissions. Your submission will be graded basde on the averaged score of the four criteria for the part of the experimental results (45%).

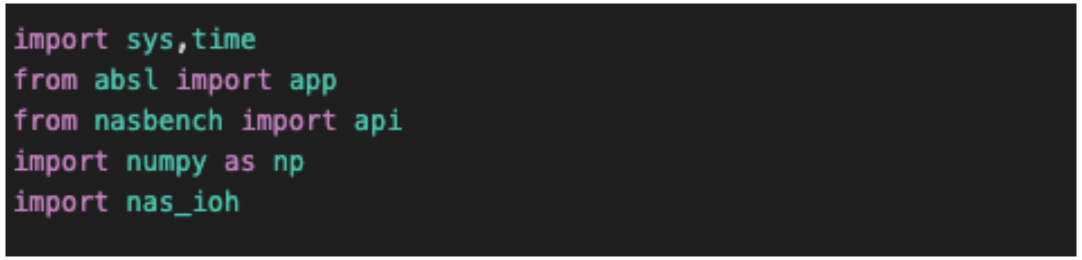
- The top two teams of the experimental results will obtain a 0.5 bonus adding to the final grade of the course.
- All submissions whose codes produce the identical results in the report can obtain at least 0.7*4.5 for the part of the experimental results. When the results are not identical or the submitted codes are not executable, this part will be scored as 0.6*4.5 after convincing arguments from the team, or the practical assignment will be scored as 0.

Toturial

- 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.
- Install the nas-bench-101 package
  - Git clone the project: https://github.com/FurongYe/nasbench
  - Download the dataset (https://storage.googleapis.com/nasbench/nasbench_only108.tfrecord) and put it in the folder of the downloaded project.
  - Install the package: “pip install -e .” at the path of the downloaded folder.
  - Run the random search example: ‘python rs.py’
  - Compress and upload the generated folder ‘sstudentnumber1_sstudentnumber2’ to iohanalyzer.liacs.nl
  - Check the performance of the random search using IOHanalyzer.
- Additional information is available at https://github.com/google-research/nasbench . Note that the version on Brightspace is modified to work with the new version of TensorFlow

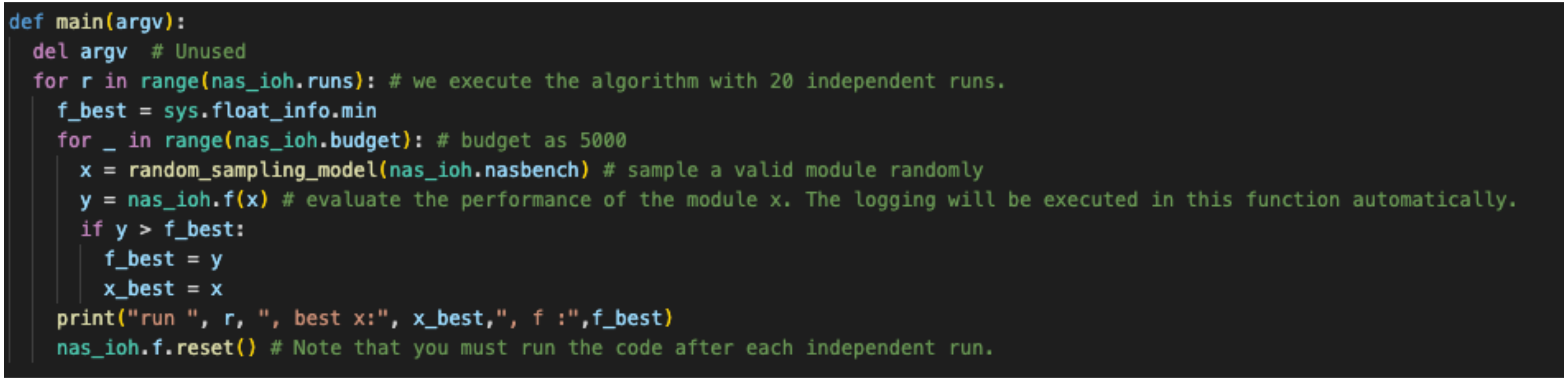
An example of random search (rs.py)

- Necessary packages



```python
import sys,time
from absl import app
from nasbench import api
import numpy as np
import nas_ioh
```

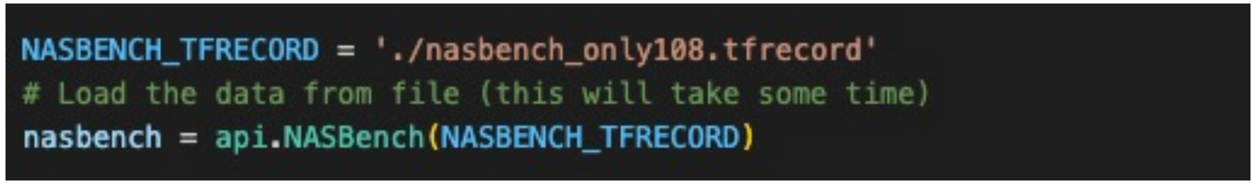
- Random search



```python
def main(argv):
  del argv  # Unused
  for r in range(nas_ioh.runs): # we execute the algorithm with 20 independent runs.
    f_best = sys.float_info.min
    for _ in range(nas_ioh.budget): # budget as 5000
      x = random_sampling_model(nas_ioh.nasbench) # sample a valid module randomly
      y = nas_ioh.f(x) # evaluate the performance of the module x. The logging will be executed in this function automatically.
      if y > f_best:
        f_best = y
        x_best = x
    print("run ", r, ", best x:", x_best,", f :",f_best)
    nas_ioh.f.reset() # Note that you must run the code after each independent run.
```

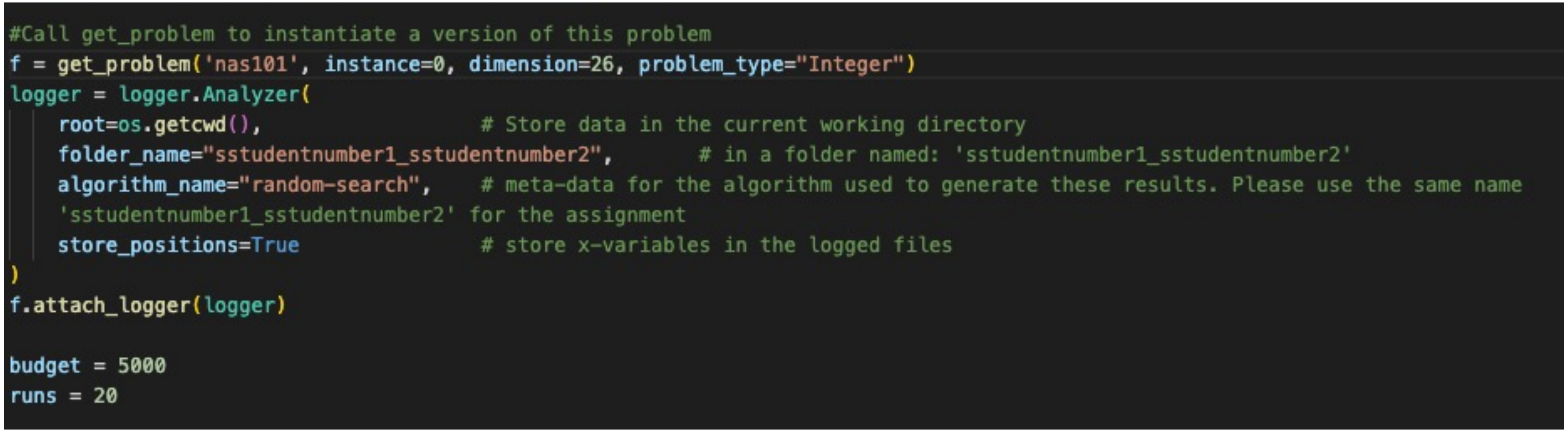
The basic setting (nas_ioh.py)

- Make sure that the nasbench data file locates at the correct path



```
NASBENCH_TFRECORD = './nasbench_only108.tfrecord'
# Load the data from file (this will take some time)
nasbench = api.NASBench(NASBENCH_TFRECORD)
```

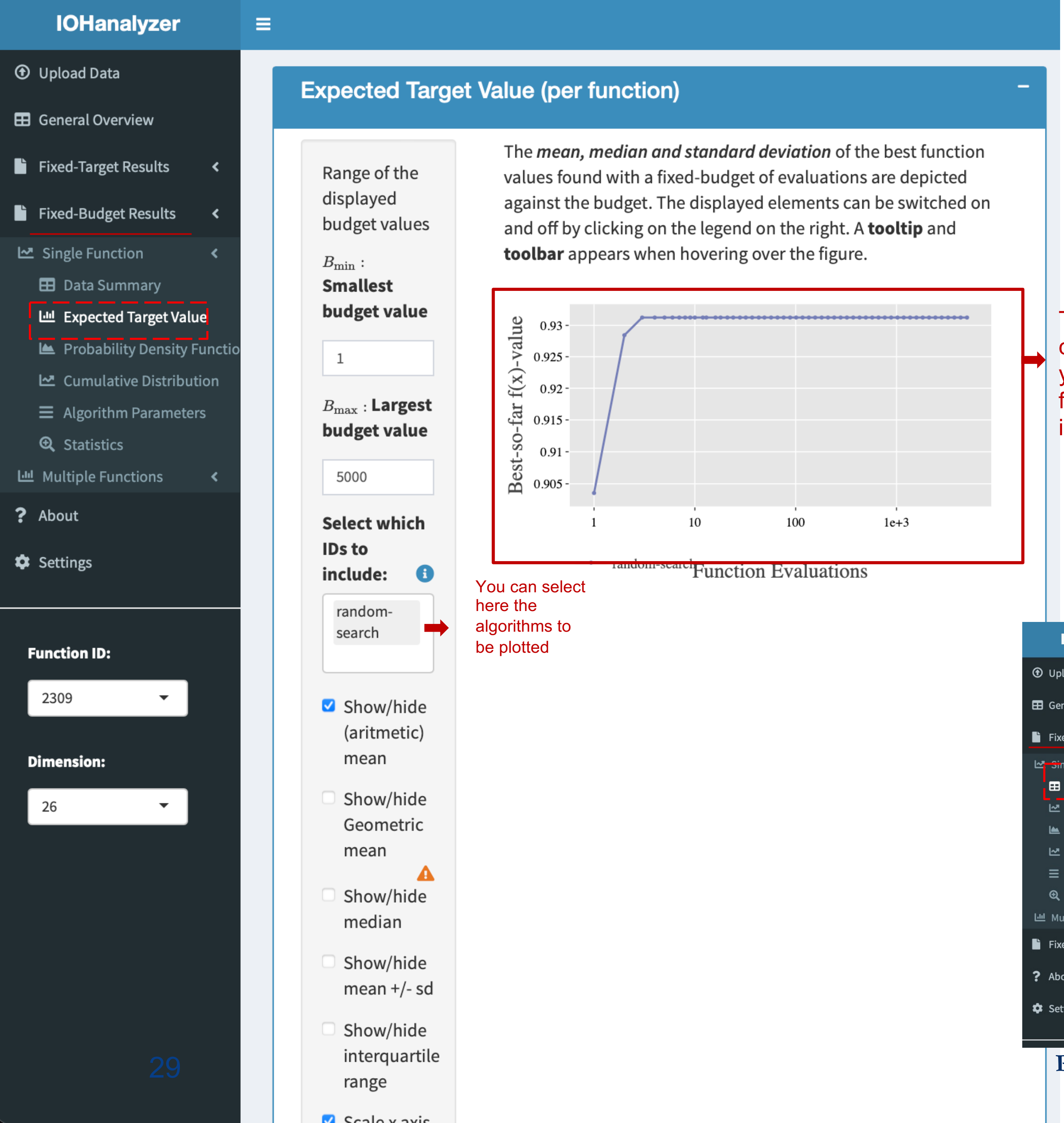
- Useful variables



```
#Call get_problem to instantiate a version of this problem
f = get_problem('nas101', instance=0, dimension=26, problem_type="Integer")
logger = logger.Analyzer(
    root=os.getcwd(),                  # Store data in the current working directory
    folder_name="sstudentnumber1_sstudentnumber2",       # in a folder named: 'sstudentnumber1_sstudentnumber2'
    algorithm_name="random-search",    # meta-data for the algorithm used to generate these results. Please use the same name
    'sstudentnumber1_sstudentnumber2' for the assignment
    store_positions=True               # store x-variables in the logged files
)
f.attach_logger(logger)

budget = 5000
runs = 20
```

How to access the best-found fitness using IOHanalyzer.



IOHanalyzer

- Upload Data
- General Overview
- Fixed-Target Results
- Fixed-Budget Results
  - Single Function
    - Data Summary
    - Expected Target Value
    - Probability Density Functio
    - 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: 5000

Select which IDs to include: random-search

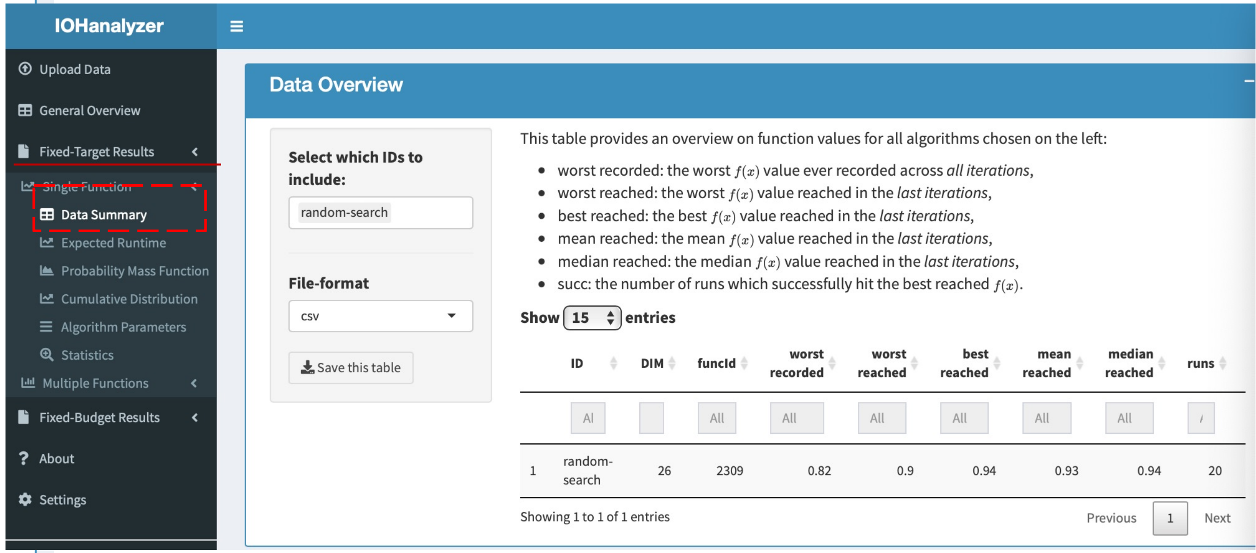
- [✓] Show/hide (aritmetic) mean
- [ ] Show/hide Geometric mean
- [ ] Show/hide median
- [ ] Show/hide mean +/- sd
- [ ] Show/hide interquartile range
- [✓] Scale x axis

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.

You can select here the algorithms to be plotted

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
  - Single Function
    - Data Summary
    - Expected Runtime
    - Probability Mass Function
    - Cumulative Distribution
    - Algorithm Parameters
    - Statistics
  - Multiple Functions
- Fixed-Budget Results
- 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

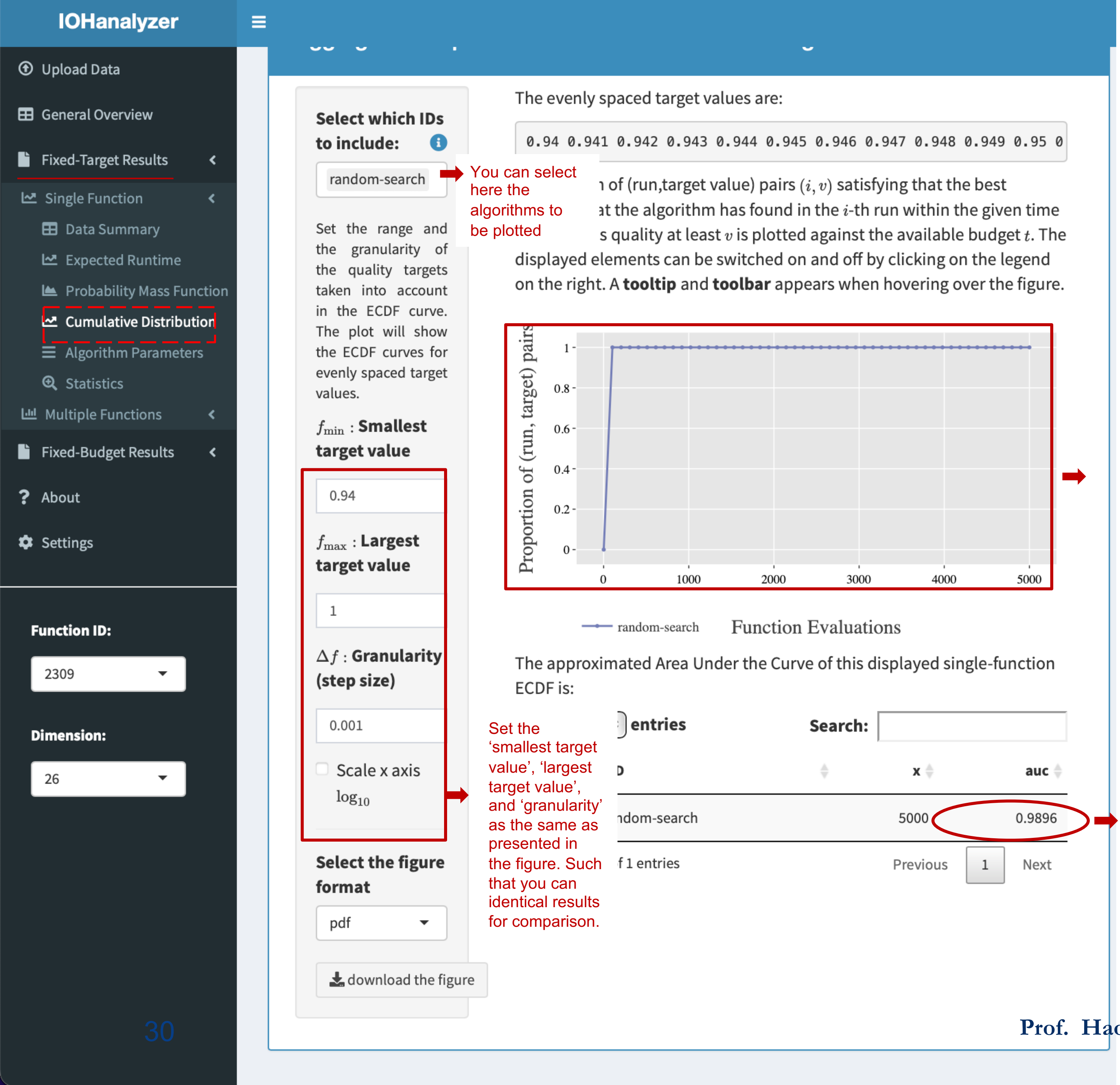
	ID	DIM	funcId	worst recorded	worst reached	best reached	mean reached	median reached	runs
	Al		All	All	All	All	All	All	/
1	random-search	26	2309	0.82	0.9	0.94	0.93	0.94	20

Showing 1 to 1 of 1 entries

Previous 1 Next

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

On the page, set the ‘smallest target value ‘largest target value’, and ‘granularity (step size)’ as 0.94, 1, and 0.001, respectively. ***Not*** *Scale x-axis log10.*



Function ID: 2309

Dimension: 26

Select which IDs to include: random-search

Set the range and the granularity of the quality targets taken into account in the ECDF curve. The plot will show the ECDF curves for evenly spaced target values.

$f_{\min}$: Smallest target value: 0.94

$f_{\max}$: Largest target value: 1

Δf : Granularity (step size): 0.001

Scale x axis $\log_{10}$

Select the figure format: pdf

The evenly spaced target values are:

```
0.94 0.941 0.942 0.943 0.944 0.945 0.946 0.947 0.948 0.949 0.95 0
```

n of (run,target value) pairs (i, v) satisfying that the best at the algorithm has found in the i-th run within the given time s quality at least v is plotted against the available budget t. 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.

The approximated Area Under the Curve of this displayed single-function ECDF is:

D	x	auc
ndom-search	5000	0.9896

f 1 entries

You can select here the algorithms to be plotted

Set the ‘smallest target value’, ‘largest target value’, and ‘granularity’ as the same as presented in the figure. Such that you can identical results for comparison.

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. And for your implementation, the value of ‘x’ shall be 5,000 because the assignment requires the given budget of 5,000

Universiteit Leiden
The Netherlands

Tips

- Test the algorithm in **20** independent runs (nas_ioh.runs)
- The budget is **5,000** for each run (nas_ioh.budget). The budget is the maximal evaluation times of using 'ioh_nas.f(x)'. The ioh package records this automatically, so no way to cheat!
- Remember to execute 'nas_ioh.f.reset()' after each independent run of your algorithm. This will reset the records (i.e., best_found fitness, uesd evaluation times, etc.) of the ioh package.
- 'ioh_nas.f(x)' will return 0 and cost budget if x is invalid (an infeasible solution). You can test if x is valid using 'ioh_nas.is_valid(x)'
- Set a fixed random seed in the implementation so that we can obtain the same results as those in the report by running the submitted codes.
- Set the algorithm_name as 'sstudentnumber1_sstudentnumber2'
- Submit files seperately. **DO NOT** submit zip files.