IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics

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

IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics

Jacob de Nobel* j.p.de.nobel@liacs.leidenuniv.nl
Furong Ye* f.ye@liacs.leidenuniv.nl
Diederick Vermetten d.l.vermetten@liacs.leidenuniv.nl
Hao Wang h.wang@liacs.leidenuniv.nl
LIACS, Leiden University, the Netherlands
Carola Doerr Carola.Doerr@lip6.fr
Sorbonne Université, CNRS, LIP6, Paris, France
Thomas Bäck t.h.w.baeck@liacs.leidenuniv.nl
LIACS, Leiden University, the Netherlands

Abstract

We present IOHexperimenter, the experimentation module of the IOHprofiler project. IOHexperimenter aims at providing an easy-to-use and customizable toolbox for benchmarking iterative optimization heuristics such as local search, evolutionary and genetic algorithms, and Bayesian optimization techniques. IOHexperimenter can be used as a stand-alone tool or as part of a benchmarking pipeline that uses other modules of the IOHprofiler environment.

IOHexperimenter provides an efficient interface between optimization problems and their solvers while allowing for granular logging of the optimization process. Its logs are fully compatible with existing tools for interactive data analysis, which significantly speeds up the deployment of a benchmarking pipeline. The main components of IOHexperimenter are the environment to build customized problem suites and the various logging options that allow users to steer the granularity of the data records.

Keywords

Iterative Optimization Heuristics, Benchmarking, Algorithm Comparison

1 Introduction

In order to compare and to improve upon state-of-the-art optimization algorithms, it is important to gain insights into their search behavior on a wide range of problems. To do so systematically, a robust benchmarking setup has to be created that allows for rigorous testing of algorithms. Numerous benchmark problems have been proposed within the evolutionary computation community, and these are often implemented many times over, without an overarching structure or proper maintenance (Li et al., 2013; Liang et al., 2013; Suganthan et al., 2020). The importance of using overarching frameworks to facilitate the benchmarking process has been gaining increasing traction within the community in the last decade when seminal works (Hansen et al., 2010) showed the benefits that these kinds of tools can provide. Since then, two of the most popular benchmarking tools have been COCO (Hansen et al., 2021) and Nevergrad (Rapin and Teytaud, 2018). While these tools enable users to benchmark their algorithms with relative ease, their overall design has some drawbacks.

* These authors contributed equally to this work.

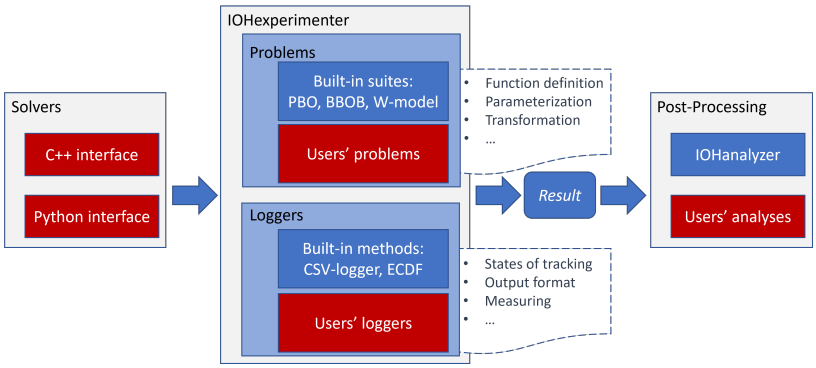

Figure 1: Workflow of IOHexperimenter

In the case of COCO, the enforced design of a suite-based structure allows for very robust benchmarking on problems made available by the developers. However, this simultaneously restricts users to using only that set of available problems and adds a complexity barrier for benchmarking algorithms on other problems. In addition, the logging of performance data follows a fixed framework, and extending it, e.g. to keep track of dynamic algorithm parameters is not straightforward. Nevergrad, in contrast, offers great flexibility with respect to adding new benchmark problems but is severely limited in terms of the information that is tracked about algorithm performance and behavior. It essentially only stores the final solution quality after exhausting a user-defined optimization budget.

With IOHexperimenter, we offer a benchmarking module that emphasizes extendability and customizability, allowing users to easily add new problems while providing a comprehensive set of built-in defaults. The logging of performance data is flexible and allows users to customize the content and frequency of the data collected. To improve ease of use, several out-of-the-box storage structures are made available, one of which can be used to collect the same type of data as COCO.

IOHexperimenter is a part of the overarching IOHprofiler project (Doerr et al., 2018), which connects algorithm frameworks, problem suites, interactive data analysis, and performance repositories in an extendable benchmarking pipeline. Within this pipeline, IOHexperimenter can be considered the interface between algorithms and problems, allowing consistent collection of performance data and algorithmic data such as the evolution of control parameters that change during the optimization process. To perform the benchmarking, three components interact with each other: *problems*, *loggers*, and *algorithms*. Within IOHexperimenter, an interface is provided to ensure that any of these components can be modified without impacting the behavior of the others, in the sense that any changes to their setup will be compatible with the other components of the benchmarking pipeline.

2 Functionality

At its core, IOHexperimenter provides a standard interface towards expandable benchmark *problems* and several *loggers* to track the performance and the behavior (internal parameters and states) of *algorithms* during the optimization process. The logger is integrated into a wide range of existing tools for benchmarking, including *problem* suites such as PBO (Doerr et al., 2020) and the W-model (Weise et al., 2020) for discrete optimization, COCO's noiseless real-valued single-objective BBOB problems (Hansen et al., 2021) for the continuous case, and submodular problems for constraint opti-

mization (Neumann et al., 2021). On the *algorithms* side, IOHexperimenter has been connected to several algorithm frameworks, including ParadisEO (Aziz-Alaoui et al., 2021), a modular genetic algorithm (Ye et al., 2021), a modular CMA-ES (de Nobel et al., 2021), and the optimizers in Nevergrad (Rapin and Teytaud, 2018). The output generated by the included *loggers* is compatible with the IOHanalyzer module (Wang et al., 2022) for interactive performance analysis. In (Long et al., 2022; Kostovska et al., 2022), the flexibility of IOHexperimenter was demonstrated by generating interfaces between two aforementioned benchmarking tools to execute algorithms from the Nevergrad framework on the BBOB problems from COCO.

Figure 1 shows the way IOHexperimenter can be placed in a typical benchmarking workflow. The key factor here is the flexibility of its design. IOHexperimenter can be used with any user-provided solvers and problems given a minimal overhead. It also ensures that the output of experimental results follows conventional standards. Because of this, the data produced by IOHexperimenter is compatible with post-processing frameworks like IOHanalyzer (Wang et al., 2022), enabling an efficient path from algorithm design to performance analysis. In addition to the built-in interfaces to existing software, IOHexperimenter aims at providing a user-friendly, easily accessible way to customize the benchmarking setup. IOHexperimenter is built in C++, with an interface to Python. In this paper, we describe the functionality of the package on a high level, without going into implementation details. [1] In the following, we introduce the typical usage of IOHexperimenter, as well as how it can be customized to fit different benchmarking scenarios.

2.1 Problems

Single-Objective Optimization. IOHexperimenter is developed with a focus on single-objective optimization problems, i.e., instances defined as $F = T_y \circ f \circ T_x$, in which $f\colon X \to \mathbb{R}$ is a benchmark problem (e.g., for ONEMAX $X = \{0,1\}^n$ and the sphere function $X = \mathbb{R}^n$), and T_x and T_y are automorphisms supported on X and $\mathbb{R}$, respectively, representing transformations in the problem's domain and range (e.g., translations and rotations for $X = \mathbb{R}^n$). To generate a problem instance, one needs to specify a tuple of a problem f, an instance identifier $i \in \mathbb{N}_{>0}$, and the dimension n of the problem. Any problem instances that reconcile with this definition of F, can easily be integrated into IOHexperimenter, using the C++ core or the Python interface.

The transformation methods are particularly important for robust benchmarking, as they allow for the creation of multiple problem instances from the same base function. They also allow the user to check algorithm invariance to transformations in search and objective space. Built-in transformations are available for pseudo-Boolean functions (Doerr et al., 2018) and for continuous optimization, implementing the transformations used by (Hansen et al., 2021). Problems can be combined in a *suite*, which allows the user to easily run solvers on collections of selected problem instances.

Constrained Optimization. Similar to benchmark problems, constraints are defined as free functions that compute a value on an evaluated solution, i.e.; $C\colon X \to \mathbb{R}$, that is non-zero in the case the constraint is violated. IOHexperimenter supports both hard constraints C_h and soft constraints C_s, of which multiple can be added to any given problem. The single-objective constrained problems are defined by $F_c = F \circ C_h \circ C_s$, which evaluates to ∞ when one of the hard constraints C_h is violated. Otherwise,

[1] Technical documentation, a getting-started, and several use-cases are available for both C++ and Python on the IOHexperimenter docs at `https://iohprofiler.github.io/IOHexperimenter/`.

$F_c = F + \sum_{i=0}^{k} w_i (C_s^i)^{\alpha_i}$, where k is the number of soft constraints. The weight w_i and exponent α_i of a constraint C_s^i can be used by the user to customize a penalty for a constraint violation. In this fashion, arbitrary functions can be added as constraints to the benchmark problems in IOHexperimenter, allowing the conversion of existing unconstrained problems into constrained problems.

2.2 Data Logging

IOHexperimenter provides *loggers* to track the performance of algorithms during the optimization process. These *loggers* can be tightly coupled with the problems: when evaluating a solution, the attached loggers will be triggered to store relevant information. Information about solution quality is always recorded, while the algorithm's control parameters are included only if specified by the user. The events that trigger a data record are customized by the user; e.g., via specifying a frequency at which information is stored, or by choosing quality thresholds that trigger a data record when met for the first time.

A default logger makes use of a two-part data format: meta-information such as function id, instance, and dimension, written to `.json`-files, and the performance data that gets written to space-separated `.dat`-files. A full specification of this format can be found in Wang et al. (2022). Additional loggers to store the data in memory or use different file structures are available. In addition to the built-in loggers, users can also create their own custom logging functionalities. For example, a logger storing only the final calculated performance measure was created for algorithm configuration tasks (Aziz-Alaoui et al., 2021).

3 Conclusions and Future Work

IOHexperimenter is a tool for benchmarking iterative optimization heuristics. It aims at making rigorous benchmarking more approachable by providing a structured benchmarking pipeline that can be adapted to fit a comprehensive range of scenarios. The combination of a clear output format and common interface across both Python and C++ makes IOHexperimenter a useful component for reproducible algorithm comparison. IOHexperimenter can be slotted into a benchmarking pipeline by generating output data for the IOHanalyzer module, which provides an interactive analysis of algorithms performance. New benchmark problems can be easily integrated with IOHexperimenter, which makes the tool suitable for teaching and hosting competitions. IOHexperimenter currently supports single-objective, noiseless optimization, with support for arbitrary constraints. The focus on flexibility makes the extension to other types of problems, such as noisy, multi-objective, and mixed-integer problems, natural next steps.

Contributing to IOHexperimenter: The IOHprofiler project welcomes contributions of problems from various domains with different perspectives. We appreciate feedback and comments through GitHub[2] or via iohprofiler@liacs.leidenuniv.nl.

Acknowledgments: We acknowledge financial support through CNRS INS2I project RandSearch and through ANR T-ERC project VARIATION (ANR-22-ERCS-0003-01).

References

Aziz-Alaoui, A., Doerr, C., and Dréo, J. (2021). Towards large scale automated algorithm design by integrating modular benchmarking frameworks. In *Proc. of GECCO'21*, pages 1365–1374. ACM. Full version: `https://arxiv.org/abs/2102.06435`.

[2] `https://github.com/IOHprofiler/IOHexperimenter/issues`

de Nobel, J., Vermetten, D., Wang, H., Doerr, C., and Bäck, T. (2021). Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules. In *Proc. of GECCO'21*, pages 1375–1384. ACM.

Doerr, C., Wang, H., Ye, F., van Rijn, S., and Bäck, T. (2018). IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *CoRR*, abs/1810.05281.

Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O. M., and Bäck, T. (2020). Benchmarking discrete optimization heuristics with IOHprofiler. *Applied Soft Computing*, 88:106027.

Hansen, N., Auger, A., Ros, R., Finck, S., and Pošík, P. (2010). Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 1689–1696.

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

Kostovska, A., Jankovic, A., Vermetten, D., de Nobel, J., Wang, H., Eftimov, T., and Doerr, C. (2022). Per-run algorithm selection with warm-starting using trajectory-based features. In Rudolph, G., Kononova, A. V., Aguirre, H. E., Kerschke, P., Ochoa, G., and Tusar, T., editors, *Parallel Problem Solving from Nature - PPSN XVII - 17th International Conference, PPSN 2022, Dortmund, Germany, September 10-14, 2022, Proceedings, Part I*, volume 13398 of *Lecture Notes in Computer Science*, pages 46–60. Springer.

Li, X., Tang, K., Omidvar, M. N., Yang, Z., Qin, K., and China, H. (2013). Benchmark functions for the cec 2013 special session and competition on large-scale global optimization. *gene*, 7(33):8.

Liang, J. J., Qu, B. Y., and Suganthan, P. N. (2013). Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, 635:490.

Long, F. X., Vermetten, D., van Stein, B., and Kononova, A. V. (2022). BBOB instance analysis: Landscape properties and algorithm performance across problem instances. *CoRR*, abs/2211.16318.

Neumann, A., Neumann, F., and Qian, C. (2021). Evolutionary submodular optimisation. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO'21, Companion)*, pages 918–940.

Rapin, J. and Teytaud, O. (2018). Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`.

Suganthan, P. N., Ali, M., Liang, J. J., Qu, B. Y., Yue, C. T., and Price, K. (2020). Competition on single objective bound constrained numerical optimization. `https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark`.

Wang, H., Vermetten, D., Ye, F., Doerr, C., and Bäck, T. (2022). IOHanalyzer: Detailed performance analyses for iterative optimization heuristics. *ACM Transactions on Evolutionary Learning and Optimization*.

Weise, T., Chen, Y., Li, X., and Wu, Z. (2020). Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms. *Applied Soft Computing*, 92:106269.

Ye, F., Doerr, C., Wang, H., and Bäck, T. (2021). Automated configuration of genetic algorithms by tuning for anytime performance. *CoRR*, abs/2106.06304.