IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics

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
LIP6, Sorbonne Université, 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, which aims at providing an easy-to-use and highly customizable toolbox for benchmarking iterative optimization heuristics such as local search, evolutionary and genetic algorithms, Bayesian optimization techniques, etc. IOHexperimenter can be used as a stand-alone tool or as part of a benchmarking pipeline that uses other components of IOHprofiler such as IOHanalyzer, the module for interactive performance analysis and visualization.

IOHexperimenter provides an efficient interface between optimization problems and their solvers while allowing for granular logging of the optimization process. These 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 be able to gain insights into their search behavior on wide ranges of problems. To do so systematically, a robust benchmarking setup has to be created that allows for rigorous testing of algorithms. While many benchmarking projects exist (Bartz-Beielstein et al., 2020; Aziz-Alaoui et al., 2021), they are often created for specific research problems and hard to extend beyond their original scope. To address this limitation, we have designed the benchmarking environment, IOHexperimenter, which strongly emphasizes extensibility and customizability, allowing users to add new problems or to build interfaces with other benchmarking software. IOHexperimenter supports customized logging of algorithm performance. The extendability of IOHexperimenter distinguishes it from other well-known benchmark projects such as COCO (Hansen et al., 2021), which cannot be easily extended to include new benchmark problems, and Nevergrad (Rapin and Teytaud, 2018), which provides a platform for comparing

* These authors contributed equally to this work.

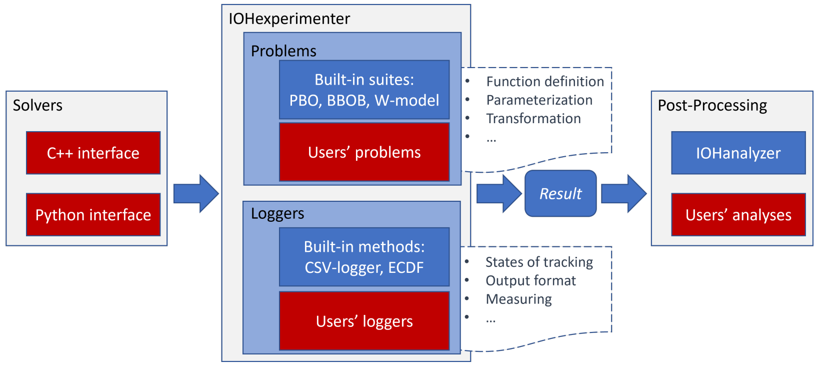

Figure 1: Workflow of IOHexperimenter

built-in optimizers but not customized algorithms and which does not provide logging functionality beyond fixed-budget simple regret.

IOHexperimenter is a part of the overarching IOHprofiler project, which connects algorithm frameworks, problem suites, interactive data analysis, and performance repositories together in an extendable benchmarking pipeline. Within this pipeline, IOHexperimenter can be considered the interface between algorithms and problems, where it allows consistent collection of performance data and of algorithmic data such as the evolution of control parameters that change during the optimization process.

To perform the benchmarking, three component 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 and COCO's noiseless real-valued single-objective BBOB problems (Hansen et al., 2021) for the continuous case. 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.

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, and ensures output of experimental results which follow 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 the ways in which it can be customized to fit different benchmarking scenarios.

2.1 Problems

In IOHexperimenter, a problem instance is 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 for 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.[2]

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 allow the user to check algorithm invariance to search space transformations, such as scaling, rotation, and translation. Built-in transformations for pseudo-Boolean functions are available (Doerr et al., 2018), as well as transformation methods for continuous optimization used by (Hansen et al., 2021). Additionally, problems can be combined in a *suite*, which allows the solver to easily run on the selected problem instances.

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 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 they are met for the first time.

The default logger makes use of a two-part data format: meta-information such as function id, instance, and dimension, which gets written to `.info`-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). Data in this format can be used directly with the IOHanalyzer module.

In addition to the built-in loggers, custom logging functionality can be created within IOHexperimenter as well. 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 bench-

[1] Technical documentation for both C++ and Python can be found on the IOHprofiler wiki at `https://iohprofiler.github.io/`, which provides a getting-started and several use-cases.

[2] Note that multi-objective problems do not follow this structure, and are not yet supported within IOHexperimenter. Integration of both noisy and mixed-variable type objective functions is in development.

marking pipeline which can be adapted to fit a wide range of scenarios. The combination of a clear output format and common interface across both Python and C++ makes IOHexperimenter a useful component towards reproducible algorithm comparison.
While currently IOHexperimenter only supports single-objective and noiseless optimization, an extension to other types of problems is desirable, allowing for more general usage of the IOHexperimenter. Additionally, support for arbitrary combinations of variable types would enable the creation of benchmark suites in the mixed-integer optimization domain.
IOHexperimenter can be slotted into a benchmarking pipeline by generating output data for the IOHanalyzer module, which provides a highly interactive analysis of algorithms performance. Its customized logging functionality allows IOHexperimenter to be used in machine learning scenarios such as algorithm configuration.
The IOHprofiler project welcomes contributions of problems from various domains and loggers with different perspectives. We appreciate feedback and comments via iohprofiler@liacs.leidenuniv.nl.

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`.

Bartz-Beielstein, T., Doerr, C., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., López-Ibáñez, M., Malan, K. M., Moore, J. H., Naujoks, B., Orzechowski, P., Volz, V., Wagner, M., and Weise, T. (2020). Benchmarking in Optimization: Best Practice and Open Issues. *CoRR*, abs/2007.03488.

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., 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.

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

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.