The Journal of Open Source Software

PyXAB - A Python Library for $\mathcal{X}$-Armed Bandit and Online Blackbox Optimization Algorithms

Wenjie Li [1*¶], **Haoze Li**[1*], **Qifan Song**[1], **and Jean Honorio**[2]

1 Department of Statistics, Purdue University, USA **2** Department of Computer Science, Purdue University, USA ¶ Corresponding author * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 04 October 2023
Published: unpublished

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

We introduce a Python open-source library for $\mathcal{X}$-armed bandit and online blackbox optimization named PyXAB. PyXAB contains the implementations of more than 10 $\mathcal{X}$-armed bandit algorithms, such as `Zooming`, `StoSOO`, `HCT`, and the most recent works such as `GPO` and `VHCT`. PyXAB also provides the most commonly-used synthetic objectives to evaluate the performance of different algorithms and the various choices of the hierarchical partitions on the parameter space. The online documentation for PyXAB includes clear instructions for installation, straightforward examples, detailed feature descriptions, and a complete reference of the API. PyXAB is released under the MIT license in order to promote both academic and industrial usage. The library can be directly installed from PyPI with its source code available at https://github.com/WilliamLwj/PyXAB.

Statement of need

Online blackbox optimization has become a heated research topic due to the recent popularity of machine learning models and thus the increasing demand for hyper-parameter tuning algorithms (L. Li et al., 2018; Shang et al., 2019). Other applications, such as neural architecture search, federated learning, and personal investment portfolio designs, also contribute to its prosperity nowadays (W. Li et al., 2022, 2023). Different online blackbox optimization algorithms, e.g., Bayesian Optimization algorithms (Shahriari et al., 2016) and two-point evaluation methods (Duchi et al., 2015; Shamir, 2015) have been proposed.

Apart from the aforementioned works, another very famous line of research is $\mathcal{X}$-armed bandit, also known as Lipschitz bandit, global optimization or bandit-based blackbox optimization (Bartlett et al., 2019; Bubeck et al., 2011; Grill et al., 2015; Kleinberg et al., 2008). In this field, researchers split the parameter domain $\mathcal{X}$ into smaller and smaller sub-domains (commonly known as nodes) hierarchically, and treat each sub-domain to be an un-evaluated arm as in the multi-armed bandit problems (Azar et al., 2014; Bubeck et al., 2011). However, such $\mathcal{X}$-armed bandit problems are much harder than their multi-armed counterparts, since the number of sub-domains increases exponentially as the partition grows, and the hierarchical structure/Lipschitzness assumption implies internal correlations between the "arms". Therefore, directly applying multi-armed bandit algorithms to such problems would infeasible and more complicated algorithms are developed (Grill et al., 2015; W. Li et al., 2023; Shang et al., 2019).

Despite the popularity of this area, most of the algorithms proposed by the researchers are either not open-sourced or implemented in different programming languages in disjoint packages. For example, `StoSOO`(Valko et al., 2013) is implemented in MATLAB and C[1], whereas `HOO` (Bubeck

[1] https://team.inria.fr/sequel/software/

DRAFT

Table 1: Selected examples of $\mathcal{X}$-armed bandit algorithms implemented in our library. *Cumulative*: whether the algorithm focuses on optimizing cumulative regret or simple regret. *Stochastic*: whether the algorithm deals with noisy rewards. *Open-sourced?*: the code availability before the development of PyXAB.

$\mathcal{X}$-Armed Bandit Algorithm	Cumulative	Stochastic	Open-sourced?
HOO	yes	yes	yes (Python)
DOO	no	no	no
StoSOO	no	yes	yes (MATLAB, C)
HCT	yes	yes	no
POO	no	yes	yes (Python, R)
GPO	no	yes	no
SequOOL	no	no	no
StroquOOL	no	yes	no
VROOM	no	no	no
VHCT	yes	yes	no

et al., 2011) is implemented in Python[2]. For most of the other algorithms, no open-sourced implementations could be found on the internet. We believe the lack of such resources results from the following two main reasons.

- The algorithms are long and intrinsically hard to implement due to the heavy usage of hierarchical partitions, node sampling, and the exploration-exploitation strategies that involve building, maintaining, and expanding complicated tree structures. It is hence time-consuming to implement and test one single algorithm.
- The problem settings for the algorithms could be slightly different. Some algorithms such as HOO (Bubeck et al., 2011) and HCT (Azar et al., 2014) are designed for the setting where the function evaluations can be noisy, while SequOOL (Bartlett et al., 2019) is proposed for the noiseless case. Some algorithms focus on cumulative-regret optimization whereas some only care about the last-point regret or the simple regret[3]. Therefore, experimental comparisons often focus on a small subset of algorithms, see e.g., (Azar et al., 2014), (Bartlett et al., 2019). The unavailability of a general package only deteriorates the situation.

In Table 1, we provide the comparison among some of the algorithms implemented in PyXAB, including HOO (Bubeck et al., 2011), DOO (Munos, 2011), StoSOO (Valko et al., 2013), HCT (Azar et al., 2014), POO (Grill et al., 2015) GPO (Shang et al., 2019), SequOOL (Bartlett et al., 2019), StroquOOL (Bartlett et al., 2019), VROOM (Ammar et al., 2020). and VHCT (W. Li et al., 2023).

To remove the barriers for future research in this area, we have developed PyXAB, a Python library of the existing popular $\mathcal{X}$-armed bandit algorithms. To the best of our knowledge, this is the first comprehensive library for $\mathcal{X}$-armed bandit, with clear documentations and user-friendly API references.

[2] https://github.com/ardaegeunlu/X-armed-Bandits

[3] A more detailed discussion on simple regret and cumulative regret can be found in [@bubeck2011X]

DRAFT

The Journal of Open Source Software

Library Design and Usage

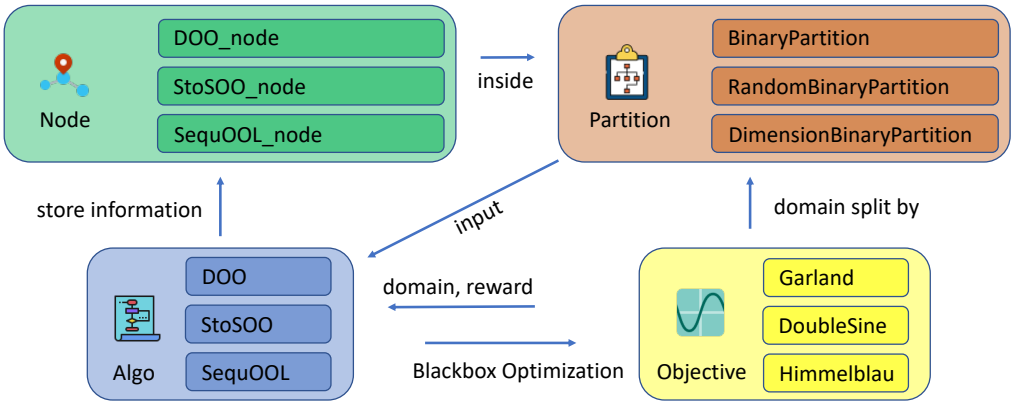

Figure 1: An overview of the PyXAB library structure.

The API of PyXAB is designed to follow the $\mathcal{X}$-armed bandit learning paradigm and to allow the maximum freedom of usage. We provide an overview of the library in Figure Figure 1.

Algorithm. All the algorithms inherit the abstract class `Algorithm`. Each algorithm will implement two kinds of actions via: (1) a `pull()` function that returns the chosen point to be evaluated by the objective; (2) a `receive_reward()` function to collect the evaluation result and update the algorithm behavior.

Partition. Given any parameter domain, the user is able to choose any partition of the domain as part of the input of the optimization algorithm. All implemented partitions inherit the `Partition` class, which has useful base functions such as deepen and `get_node_list`. Each specific partition class needs to implement a unique `make_children()` function that split one parent node into the children nodes and maintain the tree structure. For implementation convenience, the package also provides a few built-in choices such as `BinaryPartition` and `RandomBinaryPartition`.

Node. The base node class used in any partition is `P_node`, which contains useful helper functions to store domain information and maintain the partition structure. However, we allow the algorithms to overwrite the node choices in any partition so that node-wise operations are allowed. For example, the `StoSOO` algorithm needs to compute and store the $b_{h,i}$-value for each node (Valko et al., 2013). The `StroquOOL` algorithm needs to record the number of times a node is opened (Bartlett et al., 2019). Therefore, different node classes are implemented for these algorithms.

Objective. For all the objectives implemented in this package, they all inherit the `Objective` class and all have a function `f()` that returns the evaluation result of a given point. We also provide the commonly used synthetic objectives which are used to evaluate the performance of $\mathcal{X}$-armed bandit algorithms in research papers, such as `Garland`, `DoubleSine`, and `Himmelblau`.

The usage of the PyXAB library is rather straightforward. Given the number of rounds, the objective function, and the parameter domain, the learner would choose the partition of the parameter space and the bandit algorithm. Then in each round, the learner obtains one point from the algorithm, evaluate it on the objective, and return the reward to the algorithm. The following snippet of code provides an example of optimizing the Garland synthetic objective on the domain $[[0, 1]]$ by running the `HCT` algorithm with `BinaryPartition` for 1000 iterations. As can be observed, only about ten lines of code are needed for the optimization process apart from the import statements.

DRAFT

```python
from PyXAB.synthetic_obj.Garland import Garland
from PyXAB.algos.HCT import HCT

# Define the number of rounds, target, domain, and algorithm
T = 1000
target = Garland()
domain = [[0, 1]]
algo = HCT(domain=domain)

# Run the algorithm HCT
for t in range(1, T+1):
    point = algo.pull(t)
    reward = target.f(point)
    algo.receive_reward(t, reward)
```

Code Quality and Documentations

In order to ensure high code quality, we follow the PEP8 style and format all of our code using the `black` package[4]. We use the `pytest` package to test our implementations with different corner cases. More than 99% of our code is covered by the tests and Github workflows automatically generate a coverage report upon each push or pull request on the main branch[5].

We provide detailed API documentations for each of the implemented classes and functions through numpy docstrings. The documentation is fully available online on ReadTheDocs[6]. On the same website, we also provide installation guides, algorithm introductions, both elementary and advanced examples of using our package, as well as detailed contributing instructions and new feature implementation examples to encourage future contributions.

References

Ammar, H., Gabillon, V., Tutunov, R., & Valko, M. (2020). Derivative-free order-robust optimisation. In S. Chiappa & R. Calandra (Eds.), *Proceedings of the twenty third international conference on artificial intelligence and statistics* (Vol. 108, pp. 2293–2303). PMLR.

Azar, M. G., Lazaric, A., & Brunskill, E. (2014). Online stochastic optimization under correlated bandit feedback. *International Conference on Machine Learning*, 1557–1565.

Bartlett, P. L., Gabillon, V., & Valko, M. (2019). A simple parameter-free and adaptive approach to optimization under a minimal local smoothness assumption. *30th International Conference on Algorithmic Learning Theory*.

Bubeck, S., Munos, R., Stoltz, G., & Szepesvári, C. (2011). χ-armed bandits. *Journal of Machine Learning Research*, *12*(46), 1655–1695.

Duchi, J. C., Jordan, M. I., Wainwright, M. J., & Wibisono, A. (2015). Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, *61*(5), 2788–2806. https://doi.org/10.1109/TIT.2015.2409256

Grill, J.-B., Valko, M., Munos, R., & Munos, R. (2015). Black-box optimization of noisy functions with unknown smoothness. *Advances in Neural Information Processing Systems*.

Kleinberg, R., Slivkins, A., & Upfal, E. (2008). Multi-armed bandits in metric spaces. *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, 681–690.

[4] https://github.com/psf/black
[5] https://github.com/WilliamLwj/PyXAB
[6] https://pyxab.readthedocs.io/

DRAFT

https://doi.org/10.1145/1374376.1374475

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, *18*(185), 1–52.

Li, W., Song, Q., Honorio, J., & Lin, G. (2022). *Federated x-armed bandit*. arXiv. https://doi.org/10.48550/ARXIV.2205.15268

Li, W., Wang, C.-H., Cheng, G., & Song, Q. (2023). Optimum-statistical collaboration towards general and efficient black-box optimization. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=ClIcmwdlxn

Munos, R. (2011). Optimistic optimization of a deterministic function without the knowledge of its smoothness. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 24). Curran Associates, Inc.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & Freitas, N. de. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, *104*(1), 148–175.

Shamir, O. (2015). An optimal algorithm for bandit and zero-order convex optimization with two-point feedback. *Journal of Machine Learning Research*, *18*.

Shang, X., Kaufmann, E., & Valko, M. (2019). General parallel optimization a without metric. *Algorithmic Learning Theory*, 762–788.

Valko, M., Carpentier, A., & Munos, R. (2013). Stochastic simultaneous optimistic optimization. *Proceedings of the 30th International Conference on Machine Learning*, *28*, 19–27.

DRAFT