OpenBox: A Python Toolkit for Generalized Black-box Optimization

Huaijun Jiang\(^1\)\(^*\)
Yu Shen\(^1\)\(^*\)
Yang Li\(^2\)\(^*\)
Beicheng Xu\(^1\)
Sixian Du\(^1\)
Wentao Zhang\(^1\)
Ce Zhang\(^3\)
Bin Cui\(^1\)

1 Key Lab of High Confidence Software Technologies (MOE), School of CS, Peking University, China
2 Department of Data Platform, TEG, Tencent Inc., China
3 Department of Computer Science, ETH Zürich, Switzerland

\(^*\) Equal contribution.

Abstract

Black-box optimization (BBO) has a broad range of applications, including automatic machine learning, experimental design, and database knob tuning. However, users still face challenges when applying BBO methods to their problems at hand with existing software packages in terms of applicability, performance, and efficiency. This paper presents OpenBox, an open-source BBO toolkit with improved usability. It implements user-friendly interfaces and visualization for users to define and manage their tasks. The modular design behind OpenBox facilitates its flexible deployment in existing systems. Experimental results demonstrate the effectiveness and efficiency of OpenBox over existing systems. The source code of OpenBox is available at https://github.com/PKU-DAIR/open-box.

Keywords: Python, Black-box Optimization, Bayesian Optimization, Hyper-parameter Optimization

1. Introduction

Black-box optimization (BBO) (Muñoz et al., 2015) deals with optimizing an objective function under a limited budget for function evaluation. However, since the evaluation of objective function is usually expensive, the goal of BBO is to find a configuration that approaches the optimal configuration as soon as possible. Recently, generalized BBO has attracted great attraction in various areas (Foster et al., 2019; Sun et al., 2022b, a; Meng et al., 2023; Huang et al., 2023), which requires more functionalities than traditional BBO tasks. Specifically, traditional BBO refers to BBO with integer or float input parameters and with only a single objective, while there can be various input types (e.g., ordinal and categorical), multiple objectives, and constraints in generalized BBO. Though many software and platforms have been developed for traditional BBO (Hutter et al., 2011; Bergstra et al., 2011; Golovin et al., 2017; Akiba et al., 2019; Balandat et al., 2020; Liu et al., 2022; Lindauer et al., 2022), so far, there is no platform specifically designed to target generalized...
BBO. Existing platforms suffer from the following limitations when applied to generalized BBO scenarios: 1) Restricted application scope. Many existing BBO platforms cannot support multiple objectives and constraints, and they cannot support categorical parameters. 2) Unstable performance across problems. Many platforms only implement one or very few BBO algorithms. According to the “no free lunch” theorem (Ho and Pepyne, 2001), this would inevitably lead to unstable performance when applied to various problems. 3) Limited scalability and efficiency. Most platforms execute optimization in a sequential manner, which is inefficient and unscalable.

This paper proposes OpenBox to address the above limitations simultaneously. OpenBox has the following crucial features: 1) **Generality.** OpenBox hosts most state-of-the-art optimization algorithms and is capable of handling BBO tasks with different requirements. 2) **Ease of use.** OpenBox provides user-friendly interfaces, visualization, and automatic decisions on algorithms. Users can conveniently define their tasks via either wrapped services or ask-and-tell interfaces. 3) **State-of-the-art performance.** Extensive experiments show that OpenBox outperforms the existing systems on a wide range of BBO tasks.

So far, OpenBox has helped researchers solve various realistic BBO problems like database tuning (Kanellos et al., 2022; Zhang et al., 2022c) and traffic simulation (Liang et al., 2022). It has also performed as the core part of the open-source graph learning system SGL (Zhang et al., 2022a) and database tuning system DBTune (Zhang et al., 2022b). Besides the academic usages, OpenBox won the first place in CIKM 2021 AnalytiCup Track 2 (Jiang et al., 2021), and parts of the toolkit have been successfully deployed in corporations like Tencent (Li et al., 2023) and ByteDance (Shen et al., 2023).

---

**Figure 1:** Simplified overview of supported scenarios and built-in algorithms in OpenBox.

### 2. Functionality Overview

OpenBox implements a wide range of state-of-the-art algorithms for various BBO problems (Figure 1). Concretely, OpenBox is capable of handling BBO problems with different numbers of objectives (Belakaria and Deshwal, 2019; Belakaria et al., 2020) and constraints (Sui et al., 2015), and the optimization procedure can either be sequential or parallel (González et al., 2016). To ensure stable performance in different scenarios, OpenBox implements variants of Bayesian optimization (Hutter et al., 2011; Bergstra et al., 2011; Snoek et al., 2012; Hou and Behdinan, 2022), evolutionary algorithms (Storn and Price, 1997; Deb et al., 2002; Jin, 2011), and other competitive methods like CMA-ES (Hansen et al., 2003).
While it may be challenging for users to choose the proper algorithm for different scenarios, **OpenBox** also provides automatic decisions on algorithms and settings according to the characteristics of the incoming task, including the search space, number of objectives, and number of constraints. The decisions are based on experimental analysis (Eggensperger et al., 2013) or practical experience. For example, if there are over ten parameters in the input space, or the number of trials exceeds 300, we choose Probabilistic Random Forest (PRF) (Hutter et al., 2011) instead of Gaussian Process (GP) as the surrogate to avoid incompatibility or high computational complexity in Bayesian optimization.

### 3. System Design

In this section, we will introduce the system design of **OpenBox**. As shown in the upper left corner of Figure 2, the core of the optimization framework is **Optimizer**, which takes an objective function and a search space as inputs, executes the optimization process, and outputs the final results. **Optimizer** consists of three main components: **Advisor**, **Executor**, and **Visualizer**. **Advisor** implements the optimization algorithms described in Section 2 and suggests new configurations based on the provided search space and stored optimization history. **Executor** then runs the objective function on the recommended configurations and returns the results, i.e., observations. To monitor the optimization process, **Visualizer** provides comprehensive visualization APIs for users, including the convergence curve, parameter importance analysis based on SHAP (Lundberg and Lee, 2017), etc. For multi-objective problems, Pareto front and Hypervolume charts are also available. An example of the convergence curve is provided on the bottom left of Figure 2, where the best-observed configurations during optimization are connected by a blue line.
For ease of use, OpenBox also provides the ask-and-tell interface using Advisor. The example code is shown on the right side of Figure 2. Users first build an Advisor based on the search space (Line 4-5). Next, users run optimization by interacting with the Advisor (Line 8-16). Concretely, users get a configuration suggestion from Advisor (Line 10) and evaluate the performance of the configuration by themselves (Line 12). Users then build an Observation that contains the evaluation result (Line 14) and update the Observation to Advisor (Line 16). Finally, an HTML page for visualization is generated based on the optimization history obtained from Advisor (Line 19-20). This ask-and-tell interface is compatible with various types of optimization algorithms and enables users to make flexible modifications to the optimization process.

4. Benchmark
To demonstrate the generality and efficiency of OpenBox, we conduct experiments on the constrained multi-objective benchmark CONSTR and the LightGBM (Ke et al., 2017) tuning task on 24 OpenML datasets (Feurer et al., 2021). We report the Hypervolume difference from the optimum in the CONSTR benchmark and the performance rank of the best-achieved accuracy in the LightGBM tuning task. In Figure 3(a), we observe that OpenBox outperforms the other baselines in terms of convergence speed and stability. In Figure 3(b), we observe that OpenBox outperforms the other competitive systems, achieves a median rank of 1.25, and ranks first in 12 out of 24 datasets.

5. Conclusion
This paper presents OpenBox, an open-source system for solving generalized BBO tasks. OpenBox hosts a wide range of state-of-the-art optimization algorithms and provides user-friendly interfaces along with comprehensive visualization functions. Evaluations showcase the excellent performance of OpenBox over existing systems. The recently released version 0.8.3 has been tested on Linux, macOS, and Windows and can be installed easily via PyPI by ‘pip install openbox’. The source code of OpenBox is now available at https://github.com/PKU-DAIR/open-box. More detailed examples, APIs, and advanced usages can be found in our documentation.

1. https://open-box.readthedocs.io/
Acknowledgments

We thank all contributors to this project. This work is supported by the National Natural Science Foundation of China (No. U23B2048 and U22B2037), Beijing Municipal Science and Technology Project (No. Z231100010323002), and High-performance Computing Platform of Peking University. Bin Cui is the corresponding author.

References


