Journal of Machine Learning Research 25 (2024) 1-28 Submitted 3/23; Revised 7/24; Published 10/24

PyPop7: A Pure-Python Library for Population-Based Black-Box Optimization

Qiqi Duan[1,2,*] 11749325@mail.sustech.edu.cn
Guochen Zhou[2,*] 12132378@mail.sustech.edu.cn
Chang Shao[3,*] chang.shao@student.uts.edu.au
Zhuowei Wang[4] zhuowei.wang@csiro.au
Mingyang Feng[5] 11856010@mail.sustech.edu.cn
Yuwei Huang[2] 12332473@mail.sustech.edu.cn
Yajing Tan[2] 12332416@mail.sustech.edu.cn
Yijun Yang[6] altmanyang@tencent.com
Qi Zhao[2] zhaoq@sustech.edu.cn
Yuhui Shi[2,†] shiyh@sustech.edu.cn
[1] *Harbin Institute of Technology, Harbin, China*
[2] *Southern University of Science and Technology, Shenzhen, China*
[3] *University of Technology Sydney, Sydney, Australia*
[4] *Space and Astronomy, CSIRO, Marshfield, Australia*
[5] *University of Birmingham, Birmingham, UK*
[6] *Tencent Inc., Shenzhen, China*

Editor: Sebastian Schelter

Abstract

In this paper, we present an open-source pure-Python library called PyPop7 for black-box optimization (BBO). As population-based methods (e.g., evolutionary algorithms, swarm intelligence, and pattern search) become increasingly popular for BBO, the design goal of PyPop7 is to provide a unified API and elegant implementations for them, particularly in challenging high-dimensional scenarios. Since these population-based methods easily suffer from the notorious curse of dimensionality owing to random sampling as one of core operations for most of them, recently various improvements and enhancements have been proposed to alleviate this issue more or less mainly via exploiting possible problem structures: such as, decomposition of search distribution or space, low-memory approximation, low-rank metric learning, variance reduction, ensemble of random subspaces, model self-adaptation, and fitness smoothing. These novel sampling strategies could better exploit different problem structures in high-dimensional search space and therefore they often result in faster rates of convergence and/or better qualities of solution for large-scale BBO. Now PyPop7 has covered many of these important advances on a set of well-established BBO algorithm families and also provided an open-access interface to adding the latest or missed black-box optimizers for further functionality extensions. Its well-designed source

∗. These three authors contributed equally.
†. Corresponding author.

code (under GPL-3.0 license) and full-fledged online documents (under CC-BY 4.0 license) have been freely available at `https://github.com/Evolutionary-Intelligence/pypop` and `https://pypop.readthedocs.io`, respectively.

Keywords: Black-box optimization, Evolutionary computation, Large-scale optimization, Open-source software, Population-based optimization, Swarm intelligence

1. Introduction

An increasing number of population-based randomized optimization methods (Campelo and Aranha, 2023; Aranha et al., 2022; Swan et al., 2022) have been widely applied to a diverse set of real-world black-box problems such as direct search (Moritz et al., 2018) of deep neural network-based policy for reinforcement learning (Salimans et al., 2017). In typical black-box optimization (BBO) scenarios, the lack/unavailability of gradient information severely limits the common usage of powerful gradient-based optimizers such as gradient descent (Amari, 1998) and coordinate descent (Wright, 2015), a problem worsened by noisy objective functions (Arnold and Beyer, 2003). Instead, a variety of black-box (aka zeroth-order or derivative-free or direct search) optimizers from multiple research communities are natural algorithm choices of practical acceptance in these challenging BBO cases (Varelas et al., 2018). Please refer to e.g., the latest *Nature* review (Eiben and Smith, 2015) or the classical *Science* review (Forrest, 1993) for an introduction to population-based (also called evolution/swarm-based) optimization methods e.g., evolutionary algorithms (Miikkulainen and Forrest, 2021; Bäck et al., 1997), swarm intelligence (Kennedy et al., 2001; Bonabeau et al., 1999), and pattern search (Torczon, 1997).

Over the past ten years, rapid developments of deep models (LeCun et al., 2015; Schmidhuber, 2015) and big data have generated a large number of new challenging high-dimensional BBO problems, e.g., direct policy search of deep reinforcement learning (Salimans et al., 2017; Moritz et al., 2018), black-box attacks of deep neural networks (Ilyas et al., 2018), black-box prompt tuning of large language models (Sun et al., 2022), and black-box optimization of complex generative models (Choudhury et al., 2023). These new large-scale BBO problems have greatly urged plenty of researchers from different science and engineering fields to scale up previous black-box optimizers via efficient improvements to existing (mostly random) sampling strategies (Varelas et al., 2018) or to propose novel versions of black-box optimizers targeted for large-scale scenarios, given the fact that random sampling strategies adopted by most of them suffer easily from the notorious curse of dimensionality (Nesterov and Spokoiny, 2017; Bellman, 1961).

In this paper, we design an open-source pure-Python software library called PyPop7, in order to cover a large number of population-based black-box optimizers, especially their large-scale variants/versions owing to their practical potential for BBO problems of interest. Specifically, our goal is to provide a unified (API) interface and elegant implementations for them, in order to promote research repeatability (Sonnenburg et al., 2007), systematic benchmarking of BBO (Hansen et al., 2021; Meunier et al., 2022), and most importantly their real-world applications. By product, we have also provided an open-access (API) interface to add the latest or missed black-box optimizers as further functionality extensions of this open-source library. Please refer to Figure 1 for its core conceptual framework, which is mainly consisting of 6 basic components (computing engines, a family of black-box

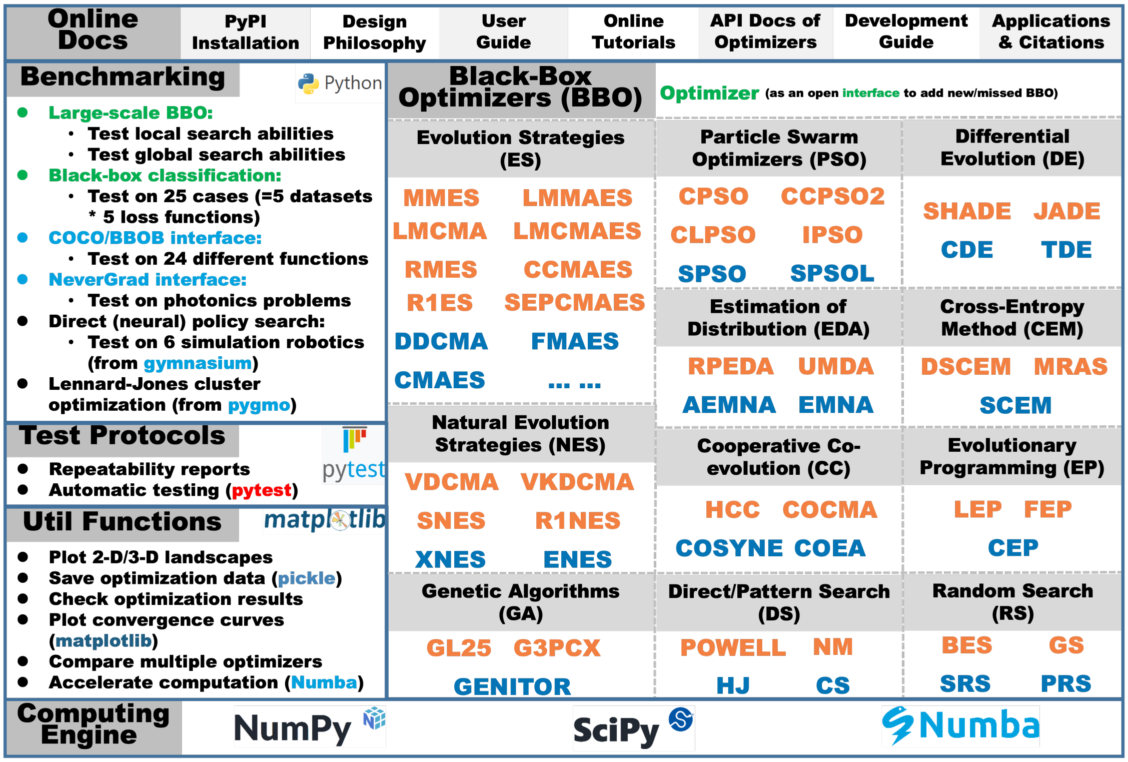

Figure 1: A conceptual framework of PyPop7 for black-box optimization (BBO), where all optimizers colored in orange are mainly designed for large-scale BBO.

optimizers, a set of util functions, two test protocols, a series of benchmarking, and full-fledged documentations). For details to each of them, please refer to Section 3 and Section 4 or its open-source repository (available at GitHub) and its online documentations (available at readthedocs.io).

2. Related Work

In this section, we will introduce some existing Python libraries including population-based optimizers for BBO (e.g., evolutionary algorithms and swarm intelligence) and compare/highlight main differences between our work and them, as presented below.

Recently, Hansen et al. (2021) released a well-documented benchmarking platform called COCO for comparing continuous black-box optimizers, after experiencing more than 10-years developments. COCO, however, focuses on the systematic design of benchmarking functions and does not provide any optimization algorithms up to now. Another similar work is the popular NeverGrad platform from Facebook Research, which covers a relatively limited number of large-scale algorithm versions (Rapin and Teytaud, 2018). Therefore, our pure-Python library, PyPop7, can be seen as their complement particularly for large-scale BBO. In our online tutorials, we have shown how to connect black-box optimizers from our library with these two well-designed benchmarking platforms for BBO.

In the past, DEAP (Fortin et al., 2012) provided a Python platform for rapid prototyping of population-based optimizers, but leaves the challenging performance-tuning task to the end-users. This is obviously different from our library wherein performance-tuning is attributed to the developers except the coding of the fitness function to be optimized. Although PyBrain (Schaul et al., 2010) mainly provided a class of natural evolution strategies (NES), now it seems to be not maintained anymore and does not cover many other BBO versions in our library. The well-designed PaGMO (Biscani and Izzo, 2020) library for parallel population-based optimizers has been actively maintained for more than 10 years. However, its current focus turns to multi-objective optimization rather than large-scale BBO, which is the focus of our paper.

Overall, our Python library (called PyPop7) have provided a large set of rich and powerful optimizers for BBO from multiple research communities (e.g., artificial intelligence, machine learning, evolutionary computation, meta-heuristics, swarm intelligence, operations research, mathematical optimization, statistics, automatic control, and etc.).

3. A Modular Coding Framework of PyPop7

In this section, we will introduce the unified interface of PyPop7 (via objective-oriented programming), testing protocols for pytest-based automatic checking and artificially-designed repeatability reporting, its computational efficiency (via comparing PyPop7 with one popular counterpart), and benchmarking on modern ML tasks for large-scale BBO.

3.1 A Unified API for Black-Box Optimizers

For simplicity, extensibility, and maintainability (arguably three desirable properties for any software), PyPop7 has provided a unified API for a large set of black-box optimizer versions/variants within the modular coding structures based on powerful objective-oriented programming (OOP) (Lutz, 2013). At first glance, its main organization framework is briefly summarized in Figure 1, wherein two levels of inheritance are employed via OOP for any instantiated optimizers in order to maximize reuse and unify the design of APIs. For computational efficiency (crucial for large-scale BBO), our library depends mainly on four open-source high-performance scientific/numeric computing libraries: *NumPy* (Harris et al., 2020), *SciPy* (Virtanen et al., 2020), *Scikit-Learn* (Pedregosa et al., 2011) and *Numba* as underlying computing engines.

In our library, currently all of these black-box optimizers have been roughly classified into a total of 13 optimization algorithm classes, as presented below. To gain insights into their application cases, we have built an online website to specifically collect their applications, which have been published on many (though not all) top-tier journals and conferences (such as, Nature, Science, PNAS, PRL, JACS, PIEEE, Cell, JMLR, etc.)[1].

- Evolution Strategies: ES (Akimoto et al., 2022; Vicol et al., 2021; Ollivier et al., 2017; Diouane et al., 2015; Bäck et al., 2013; Rudolph, 2012; Beyer and Schwefel, 2002; Hansen and Ostermeier, 2001; Schwefel, 1984; Rechenberg, 1984),

1. It is a long-term open project, which is still actively updated in the near future.

- Natural Evolution Strategies: NES (Hüttenrauch and Neumann, 2024; Wei et al., 2022; Wierstra et al., 2014; Yi et al., 2009; Wierstra et al., 2008),
- Estimation of Distribution Algorithms: EDA (Zheng and Doerr, 2023; Brookes et al., 2020; Larrañaga, 2002; Baluja, 1996; Baluja and Caruana, 1995),
- Cross-Entropy Methods: CEM (Wang and Ba, 2020; Amos and Yarats, 2020; Hu et al., 2007; Rubinstein and Kroese, 2004; Mannor et al., 2003),
- Differential Evolution: DE (Koob et al., 2023; Higgins et al., 2023; Li et al., 2022a; Laganowsky et al., 2014; Storn and Price, 1997),
- Particle Swarm Optimizers: PSO (Melis et al., 2024; Bungert et al., 2024; Huang et al., 2024; Bolte et al., 2024; Cipriani et al., 2022; Fornasier et al., 2021; Tang et al., 2019; Kennedy and Eberhart, 1995),
- Cooperative Coevolution: CC (Gomez et al., 2008; Panait et al., 2008; Schmidhuber et al., 2007; Fan et al., 2003; Potter and Jong, 2000; Gomez et al., 1999; Moriarty and Mikkulainen, 1996; Moriarty and Miikkulainen, 1995; Potter and Jong, 1994),
- Simulated Annealing[2]: SA (Samyak and Palacios, 2024; Bouttier and Gavra, 2019; Siarry et al., 1997; Bertsimas and Tsitsiklis, 1993; Corana et al., 1987; Kirkpatrick et al., 1983; Hastings, 1970; Metropolis et al., 1953),
- Genetic Algorithms: GA (Chen et al., 2020; Whitley, 2019; Goldberg, 1994; Forrest, 1993; Mitchell et al., 1993; Goldberg and Holland, 1988; Holland, 1962),
- Evolutionary Programming: EP (Cui et al., 2006; Yao et al., 1999; Fogel, 1999; Fogel and Fogel, 1995; Fogel, 1994; Fogel et al., 1965),
- Pattern/Direct Search: PS/DS (Kolda et al., 2003; Lagarias et al., 1998; Wright, 1996; Nelder and Mead, 1965; Powell, 1964; Kaupe, 1963; Hooke and Jeeves, 1961; Fermi, 1952),
- Random Search: RS (Nesterov and Spokoiny, 2017; Stich, 2014; Bergstra and Bengio, 2012; Schmidhuber et al., 2001; Rosenstein and Barto, 2001; Solis and Wets, 1981; Schumer and Steiglitz, 1968; Rastrigin, 1963; Brooks, 1958), and
- Bayesian Optimization: BO (Wang et al., 2020; Shahriari et al., 2016; Jones et al., 1998).

To alleviate their curse of dimensionality (Bellman, 1957) for large-scale BBO, different kinds of sophisticated strategies have been employed to enhance these black-box optimizers, as presented in the following:

1) Decomposition of search distribution (Akimoto and Hansen, 2020; Bäck et al., 2013; Schaul et al., 2011; Ros and Hansen, 2008) or search space (Panait et al., 2008; Gomez and Schmidhuber, 2005; Siarry et al., 1997; Corana et al., 1987),

2. Note that SA is an individual-based rather than population-based optimization method.

2) Recursive spatial partitioning, e.g., via Monte Carlo tree search (Wang et al., 2020),

3) Low-memory approximation for covariance matrix adaptation (He et al., 2021; Loshchilov et al., 2019; Loshchilov, 2017; Krause et al., 2016),

4) Low-rank metric learning (Li and Zhang, 2018; Sun et al., 2013),

5) Variance-reduction (Gao and Sener, 2022; Brockhoff et al., 2010),

6) Ensemble of random subspaces constructed via random matrix theory (Demo et al., 2021; Kabán et al., 2016),

7) Meta-model self-adaptation (Akimoto and Hansen, 2016; Lee and Yao, 2004),

8) Smoothing of fitness expectation (Hüttenrauch and Neumann, 2024; Gao and Sener, 2022; Nesterov and Spokoiny, 2017),

9) Smoothing of sampling operation (Bungert et al., 2024; Amos and Yarats, 2020; Deb et al., 2002), and

10) Efficient allocation of computational resources (García-Martínez et al., 2008).

In this new Python library PyPop7, we aim to provide high-quality open-source implementations to many of these advanced techniques on population-based optimizers for large-scale BBO in a unified way (which have been summarized in Figure 1).

3.2 Testing Protocols

Importantly, in order to ensure the coding correctness of black-box optimizers, we have provided an open-access code-based repeatability report for each black-box optimizer. Specifically, for each black-box optimizer, all experimental details are given in a specific folder (corresponding to a hyperlink in the Examples section of its online API documentation) and main results generated for it are compared to reported results in its original literature. For all optimizers with repeatability reports unavailable owing to specific reasons, their Python3-based implementations have been checked carefully by three authors (and perhaps other users) to avoid trivial bugs and errors. For any failed repeatability experiment, we try our best to reach an agreement regarding some possible reason(s), which is also finally described in its repeatability report. All repeatability code/results are summarized in Table 1, wherein each hyperlink is used to navigate the used Python code or generated results.

Following the standard workflow practice of open-source software, we have used the popular pytest tool and the free circleci service to automate all light-weighted testing processes.

For any randomized black-box optimizer, properly controlling its random sampling process is very important to repeat its entire optimization experiments. In our library, the random seed for each black-box optimizer should be explicitly set in order to ensure maximal repeatability, according to the newest suggestion from *NumPy* for random sampling.

Table 1: Repeatability Reports of All Black-Box Optimizers from PyPop7

Optimizer	Repeatability Code	Results	Success	Optimizer	Repeatability Code	Results	Success
MMES	_repeat_mmes.py	figures	YES	FCMAES	_repeat_fcmaes.py	figures	YES
LMMAES	_repeat_lmmaes.py	figures	YES	LMCMA	_repeat_lmcma.py	figures	YES
LMCMAES	_repeat_lmcmaes.py	data	YES	RMES	_repeat_rmes.py	figures	YES
R1ES	_repeat_r1es.py	figures	YES	VKDCMA	_repeat_vkdcma.py	data	YES
VDCMA	_repeat_vdcma.py	data	YES	CCMAES2016	_repeat_ccmaes2016.py	figures	YES
OPOA2015	_repeat_opoa2015.py	figures	YES	OPOA2010	_repeat_opoa2010.py	figures	YES
CCMAES2009	_repeat_ccmaes2009.py	figures	YES	OPOC2009	_repeat_opoc2009.py	figures	YES
OPOC2006	_repeat_opoc2006.py	figures	YES	SEPCMAES	_repeat_sepcmaes.py	data	YES
DDCMA	_repeat_ddcma.py	data	YES	MAES	_repeat_maes.py	figures	YES
FMAES	_repeat_fmaes.py	figures	YES	CMAES	_repeat_cmaes.py	data	YES
SAMAES	_repeat_samaes.py	figure	YES	SAES	_repeat_saes.py	data	YES
CSAES	_repeat_csaes.py	figure	YES	DSAES	_repeat_dsaes.py	figure	YES
SSAES	_repeat_ssaes.py	figure	YES	RES	_repeat_res.py	figure	YES
R1NES	_repeat_r1nes.py	data	YES	SNES	_repeat_snes.py	data	YES
XNES	_repeat_xnes.py	data	YES	ENES	_repeat_enes.py	data	YES
ONES	_repeat_ones.py	data	YES	SGES	_repeat_sges.py	data	YES
RPEDA	_repeat_rpeda.py	data	YES	UMDA	_repeat_umda.py	data	YES
AEMNA	_repeat_aemna.py	data	YES	EMNA	_repeat_emna.py	data	YES
DCEM	_repeat_dcem.py	data	YES	DSCEM	_repeat_dscem.py	data	YES
MRAS	_repeat_mras.py	data	YES	SCEM	_repeat_scem.py	data	YES
SHADE	_repeat_shade.py	data	YES	JADE	_repeat_jade.py	data	YES
CODE	_repeat_code.py	data	YES	TDE	_repeat_tde.py	figures	YES
CDE	_repeat_cde.py	data	YES	CCPSO2	_repeat_ccpso2.py	data	YES
IPSO	_repeat_ipso.py	data	YES	CLPSO	_repeat_clpso.py	data	YES
CPSO	_repeat_cpso.py	data	YES	SPSOL	_repeat_spsol.py	data	YES
SPSO	_repeat_spso.py	data	YES	HCC	N/A	N/A	N/A
COCMA	N/A	N/A	N/A	COEA	_repeat_coea.py	figures	YES
COSYNE	_repeat_cosyne.py	data	YES	ESA	_repeat_esa.py	data	N/A
CSA	_repeat_csa.py	data	YES	NSA	N/A	N/A	N/A
ASGA	_repeat_asga.py	data	YES	GL25	_repeat_gl25.py	data	YES
G3PCX	_repeat_g3pcx.py	figure	YES	GENITOR	N/A	N/A	N/A
LEP	_repeat_lep.py	data	YES	FEP	_repeat_fep.py	data	NI
CEP	_repeat_cep.py	data	YES	POWELL	_repeat_powell.py	data	YES
GPS	N/A	N/A	N/A	NM	_repeat_nm.py	data	YES
HJ	_repeat_hj.py	data	YES	CS	N/A	N/A	N/A
BES	_repeat_bes.py	figures	YES	GS	_repeat_gs.py	figures	YES
SRS	N/A	N/A	N/A	ARHC	_repeat_arhc.py	data	YES
RHC	_repeat_rhc.py	data	YES	PRS	_repeat_prs.py	figure	YES

NI : Need to be Improved.

3.3 Comparisons of Computational Efficiency

In this subsection, we will analyze the runtime efficiency (in the form of *number of function evaluations*) of our implementations via empirically comparing them with those from one widely-used BBO library (called DEAP). Note that DEAP (which was published in 2012) mainly provided several (limited) baseline versions and has not covered the latest large-scale variants comprehensively, till now.

The test-bed is one high-dimensional (2000-d) yet light-weighted test function (named sphere), since using a light-weighted test function could make us focusing on the algorithm implementation itself rather than the external fitness function provided by the end-users. We postpone more benchmarking experiments in the following two subsections.

As we can see from Figures 2, 3, and 4, our algorithm implementations are always better than DEAP's corresponding implementations, from both the *speedup of function evaluations* and the *quality of final solutions* perspectives, given the same maximal runtime (=3 hours). After carefully inspecting their own Python source code, we can conclude that different ways of storing and operating the population between two libraries (PyPop7 vs. DEAP) result in such a significant gap on computational efficiency. For DEAP naive data types such as list are used to store and operate the population (slowly) while for PyPop7 the highly-optimized data type ndarray from *NumPy* is used as the base of population initialization and evolution, along with other high-performance scientific computing libraries such as *SciPy*, *Scikit-Learn*, and *Numba*. Computational efficiency is one main goal of our open-source library, that is, developers rather than end-users are responsible for performance optimization except the customized fitness function provided by the end-user. This design practice can significantly reduce the programming and experimental overheads of end-users for large-scale BBO.

3.4 Benchmarking on Computationally-Expensive Functions

To design a set of 20 computationally-expensive test functions, the standard benchmarking practice has been used here, that is, the input vector of each test functions have been rotated and shifted/transformed before fitness evaluations. For benchmarking on this large set of 2000-dimensional and computationally-expensive test functions, some of these large-scale versions from our library obtain the best solution quality on nearly all test functions under the same runtime limit (=3 hours) and the same fitness threshold (=1e-10). Please refer to Figures 5, 6, 7, and 8 for detailed convergence curves of different algorithm classes on different test functions. For example, for the PSO family, four large-scale variants (CLPSO, CCPSO2, CPSO, and IPSO) obtained the best quality of solution on 9, 6, 3, and 2 test functions, respectively.

3.5 Benchmarking on Block-Box Classifications

In this subsection, we choose one modern ML task (known as black-box classifications) as the base of benchmarking functions. Following currently common practices of black-box classifications, five loss functions (Bollapragada and Wild, 2023; Li et al., 2022b; Ruan et al., 2020; Xu et al., 2020; Liu et al., 2019; Bollapragada et al., 2018; Liu et al., 2018) with different landscape features are selected in our numerical experiments. Furthermore, five

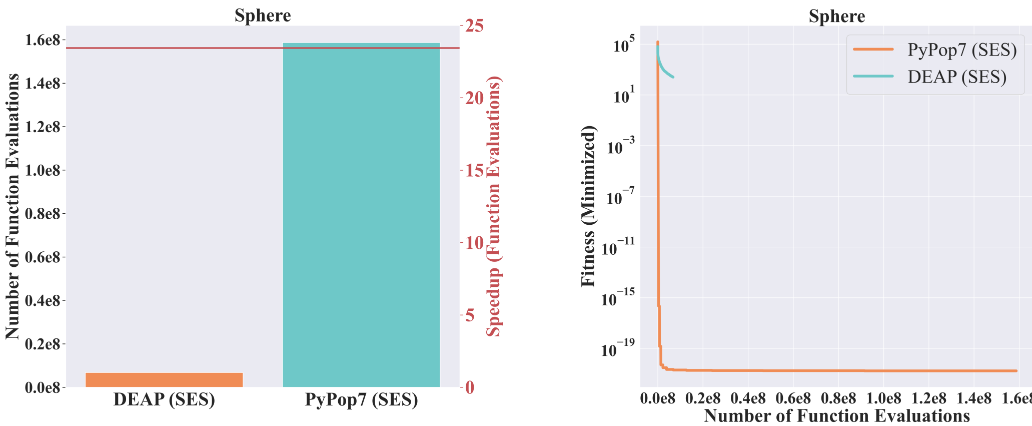

Figure 2: Median comparisons of function evaluations and solution qualities of one baseline version SES of evolution strategies from our library and the widely-used DEAP library (under the same runtime for a fair comparison). Note that each of these two implementation versions is independently run 10 times on this 2000-dimensional, light-weighted test function *sphere*. Here we do not use the standard rotation-shift operations, different from the following computationally-expensive benchmarking process (of quadratic complexity), in order to generate light-weighted function evaluations (of only linear complexity) even in high dimensions.

datasets from different fields are used for data diversity: **Parkinson's disease**, **Semeion handwritten digit**, **CNAE-9**, **Madelon**, and **QSAR androgen receptor**, all of which are now available at the UCI Machine Learning Repository. A combination of these 5 loss functions and 5 datasets leads to a total of 25 test functions for black-box classifications with up to > 1000 dimensions.

In our numerical experiments, we choose a total of 15 black-box optimizers from different algorithm families, each of which is independently run 14 times on every test function. The maximum of runtime to be allowed is set to 3 hours (Duan et al., 2023) and the threshold of fitness is set to 1e-10 to avoid excessive accuracy optimization for all optimizers on each test function.

As is clearly shown in Figure 9, no single black-box optimizer could entirely dominate the top-ranking w.r.t. convergence curves, though some of different large-scale variants obtained the best quality of solution on different test functions. For example, COCMA (Mei et al., 2016; Potter and Jong, 1994) ranked the top on a total of 9 test functions. This may be due to that it could well exploit the sparse problem structure on these functions particularly after dataset normalization. Following it, VKDCMA (Akimoto and Hansen, 2016) and CLPSO (Liang et al., 2006) obtained the best solution on 3 and 3 test functions, respectively. Then, each of 5 black-box optimizers (MAES (Beyer and Sendhoff, 2017), SEPCMAES (Ros and Hansen, 2008), LMCMA (Loshchilov, 2017), LMMAES (Loshchilov et al., 2019), and R1NES (Sun et al., 2013)) showed the best on 2 test functions independently. Here this ranking diversity on optimizers may empirically demonstrate the necessity to include

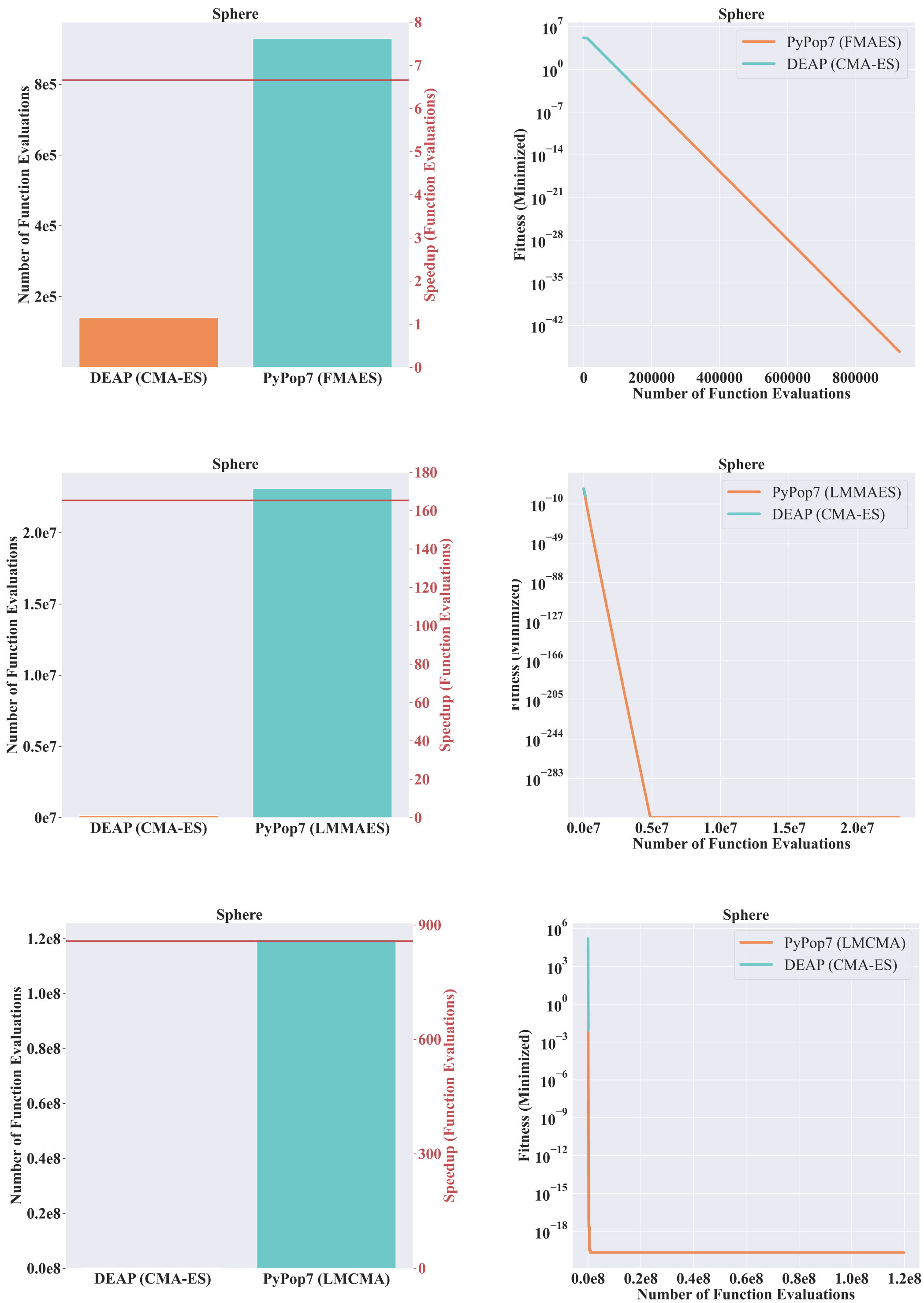

Figure 3: Median comparisons of function evaluations and solution qualities between three large-scale ES versions of our library and DEAP's CMA-ES. The experimental settings are the same as Figure 2 (given the maximal runtime: 3 hours).

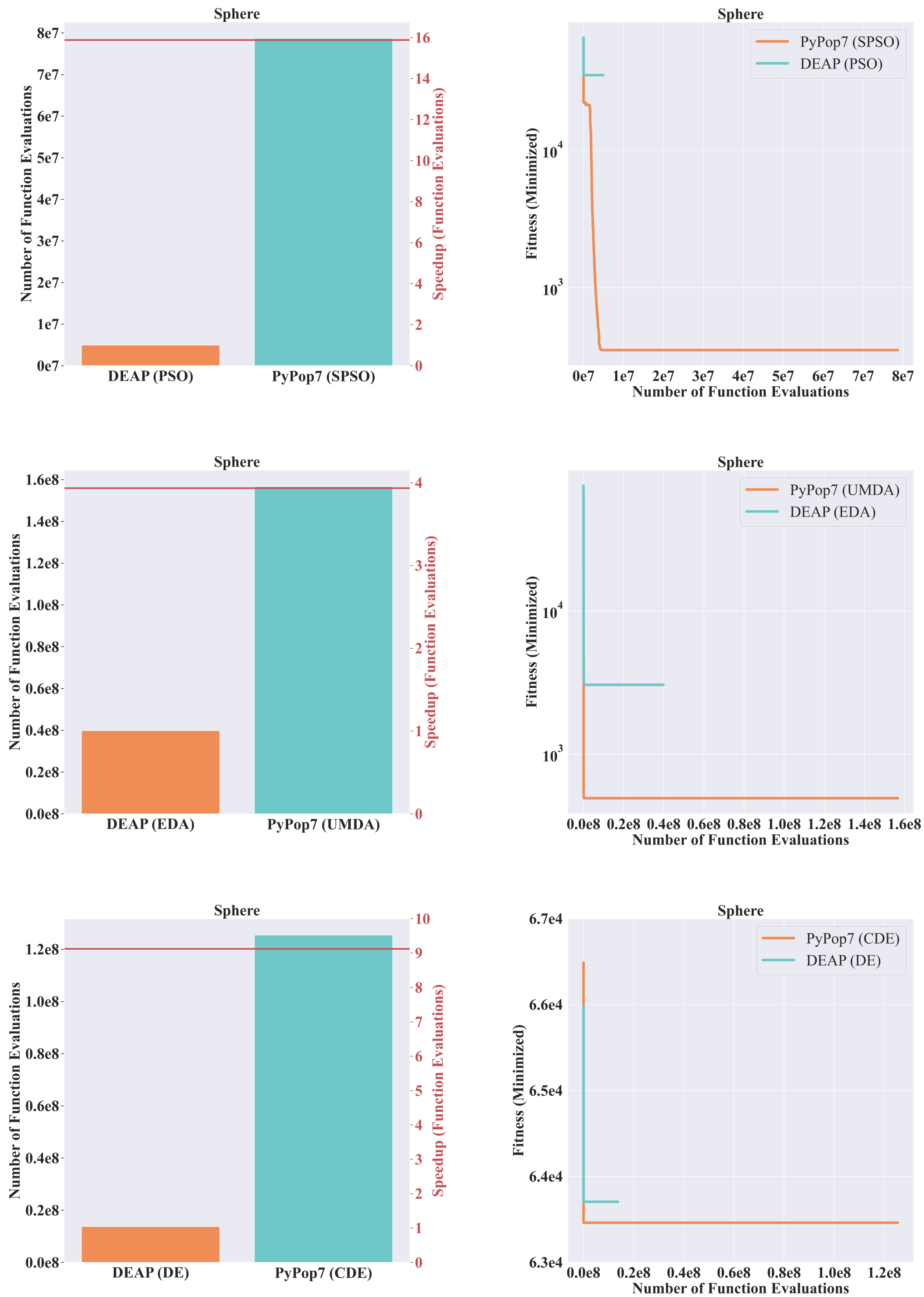

Figure 4: Median comparisons of function evaluations and solution qualities of PSO, EDA, and DE between our library and the widely-used DEAP library. The experimental settings are the same as Figure 2 (given the maximal runtime: 3 hours).

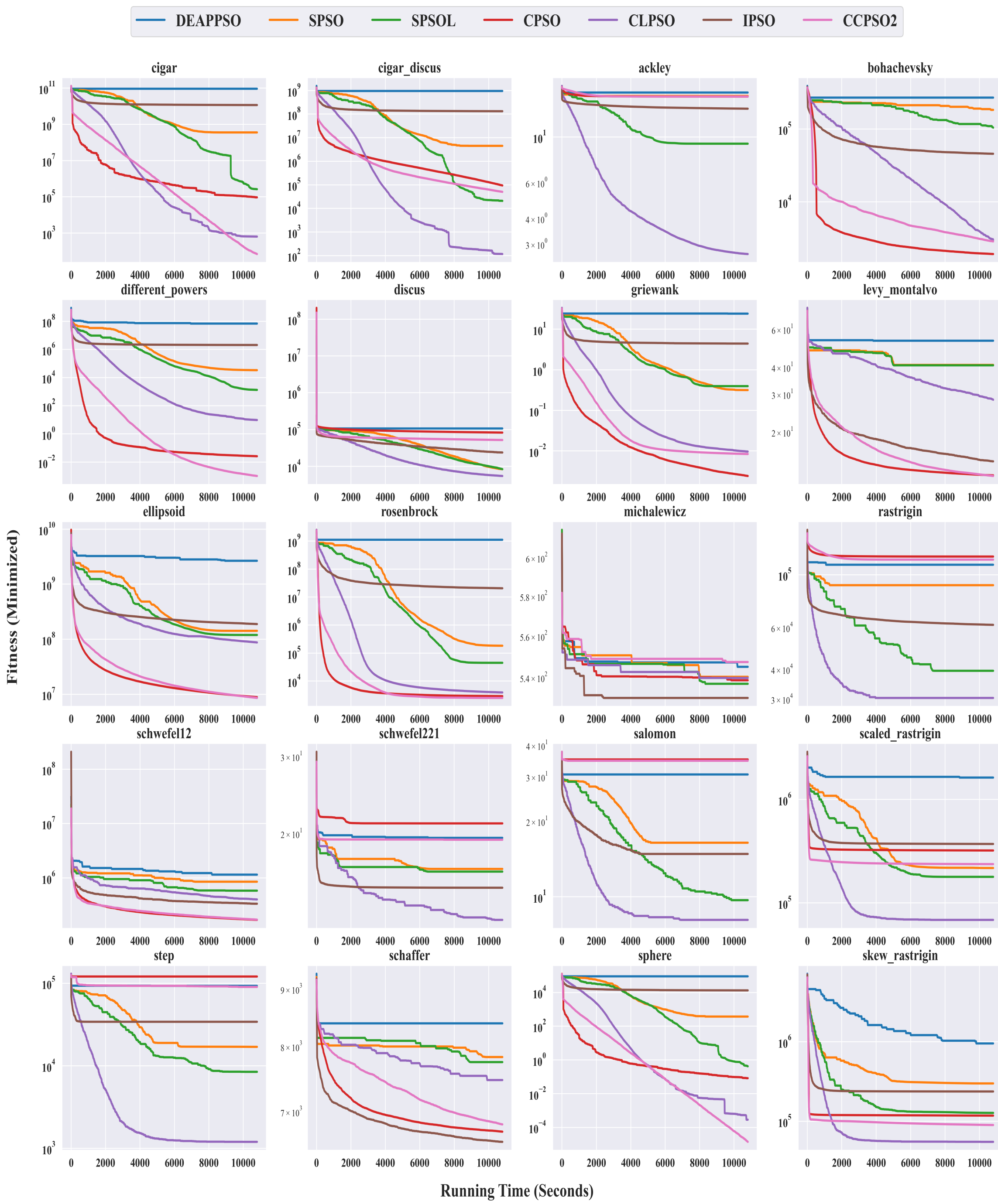

Figure 5: Median convergence rate comparisons of 7 PSO versions on 20 high-dimensional computationally-expensive test functions (with the standard rotation-and-shift operations of quadratic complexity for benchmarking).

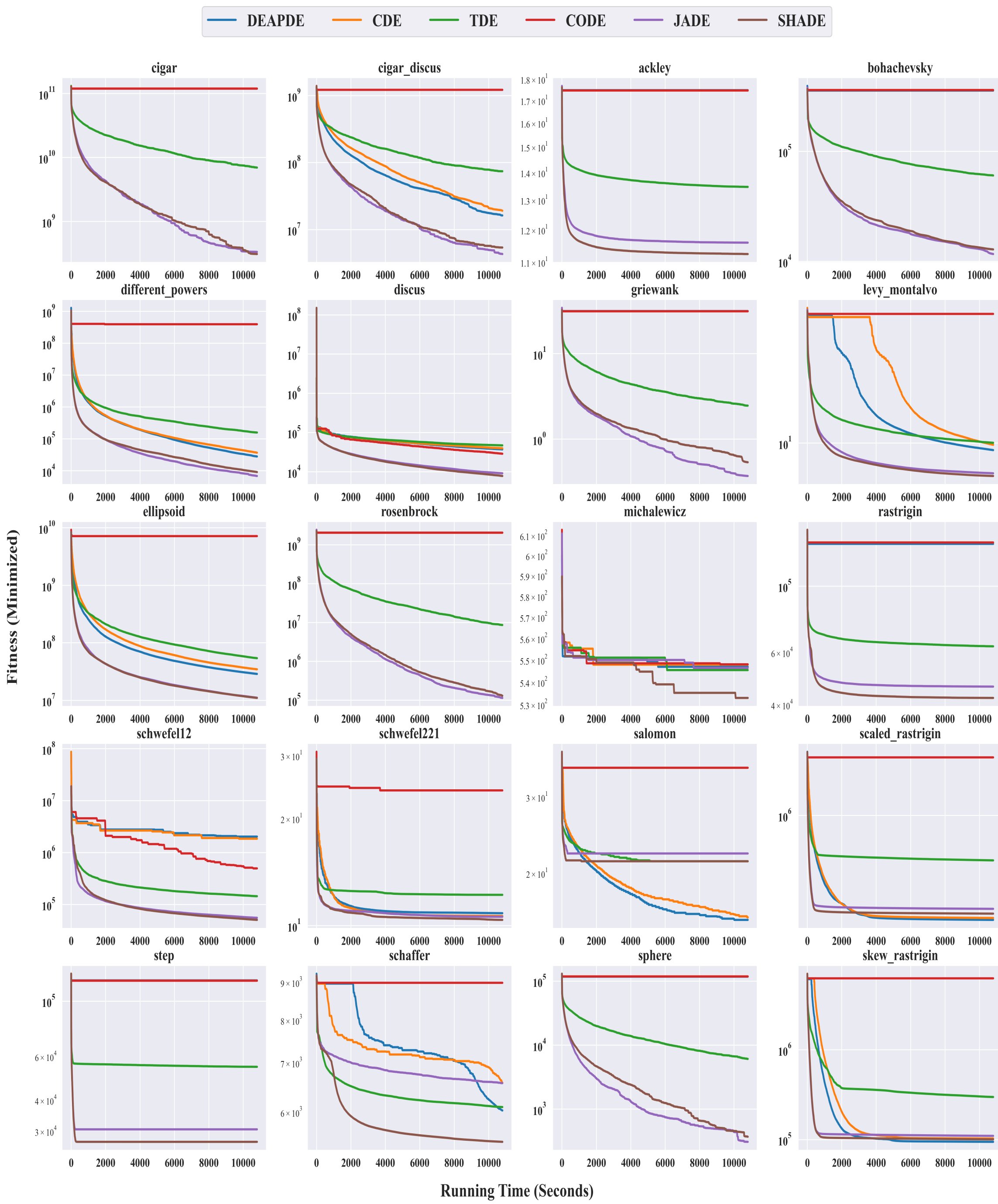

Figure 6: Median convergence rate comparisons of 6 DE versions on 20 high-dimensional computationally-expensive test functions (with the standard rotation-and-shift operations of quadratic complexity for benchmarking).

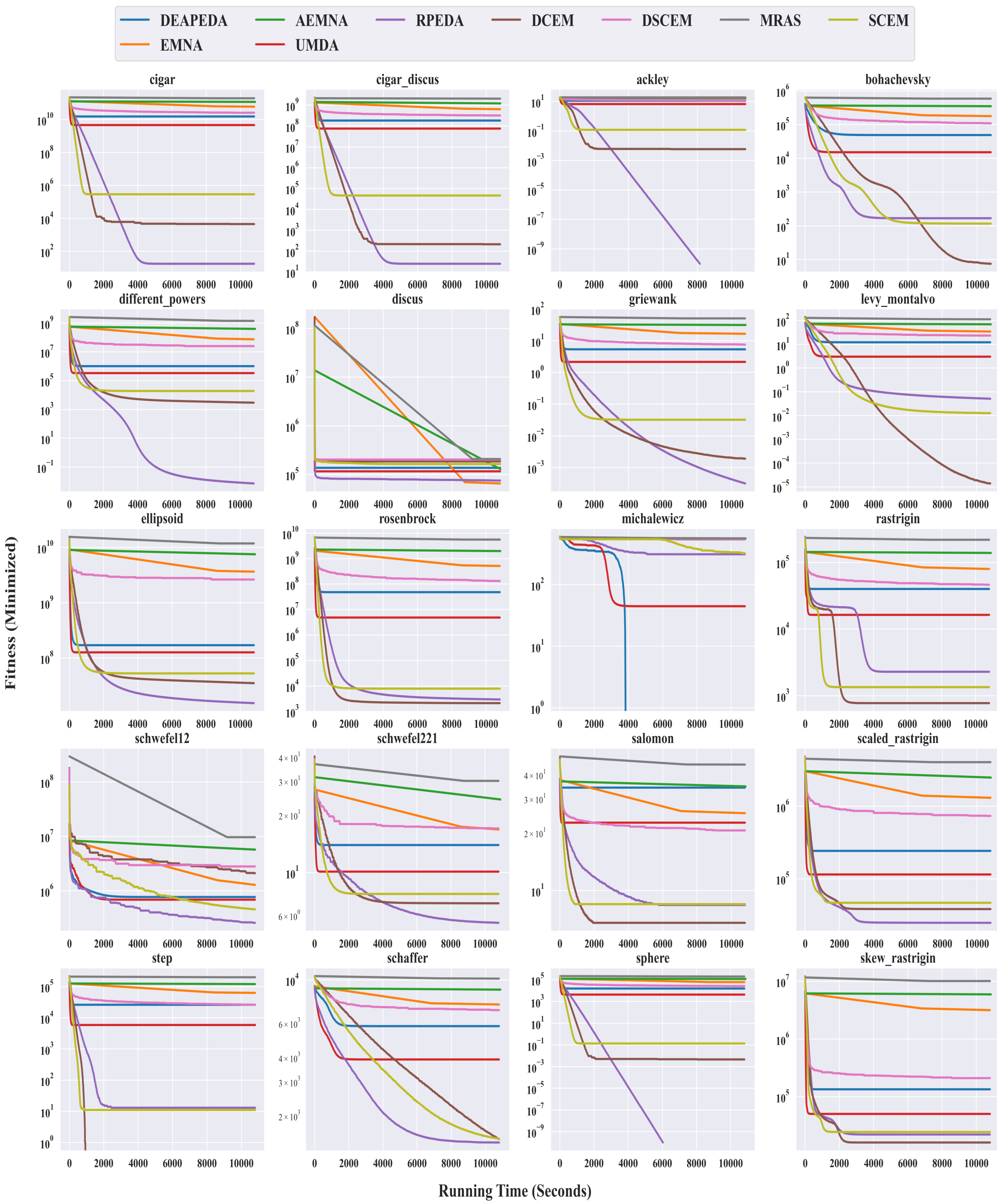

Figure 7: Median convergence rate comparisons of 9 EDA versions on 20 high-dimensional computationally-expensive test functions (with the standard rotation-and-shift operations of quadratic complexity for benchmarking).

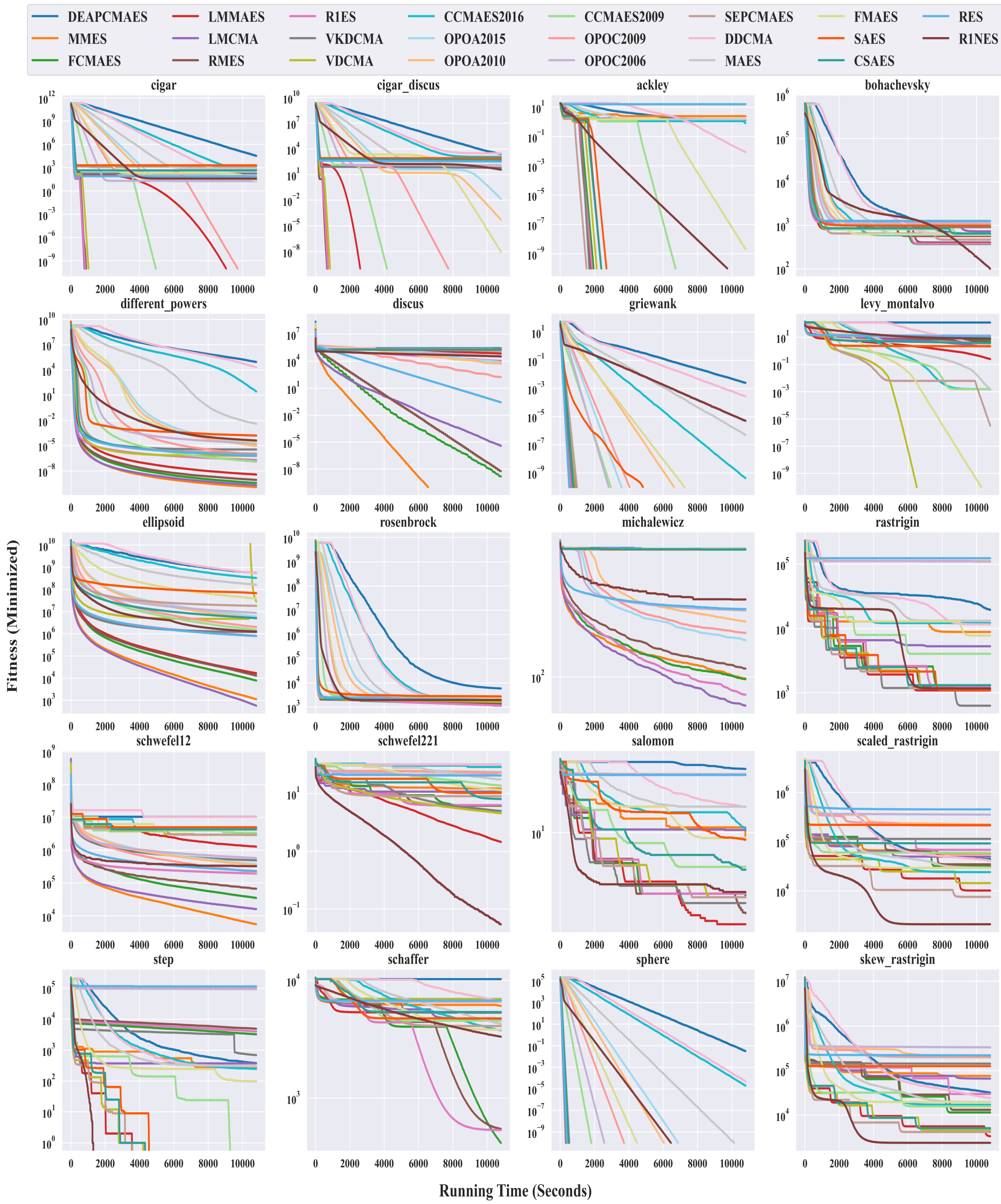

Figure 8: Median convergence rate comparisons of 23 ES versions on 20 high-dimensional computationally-expensive test functions (with the standard rotation-and-shift operations of quadratic complexity for benchmarking).

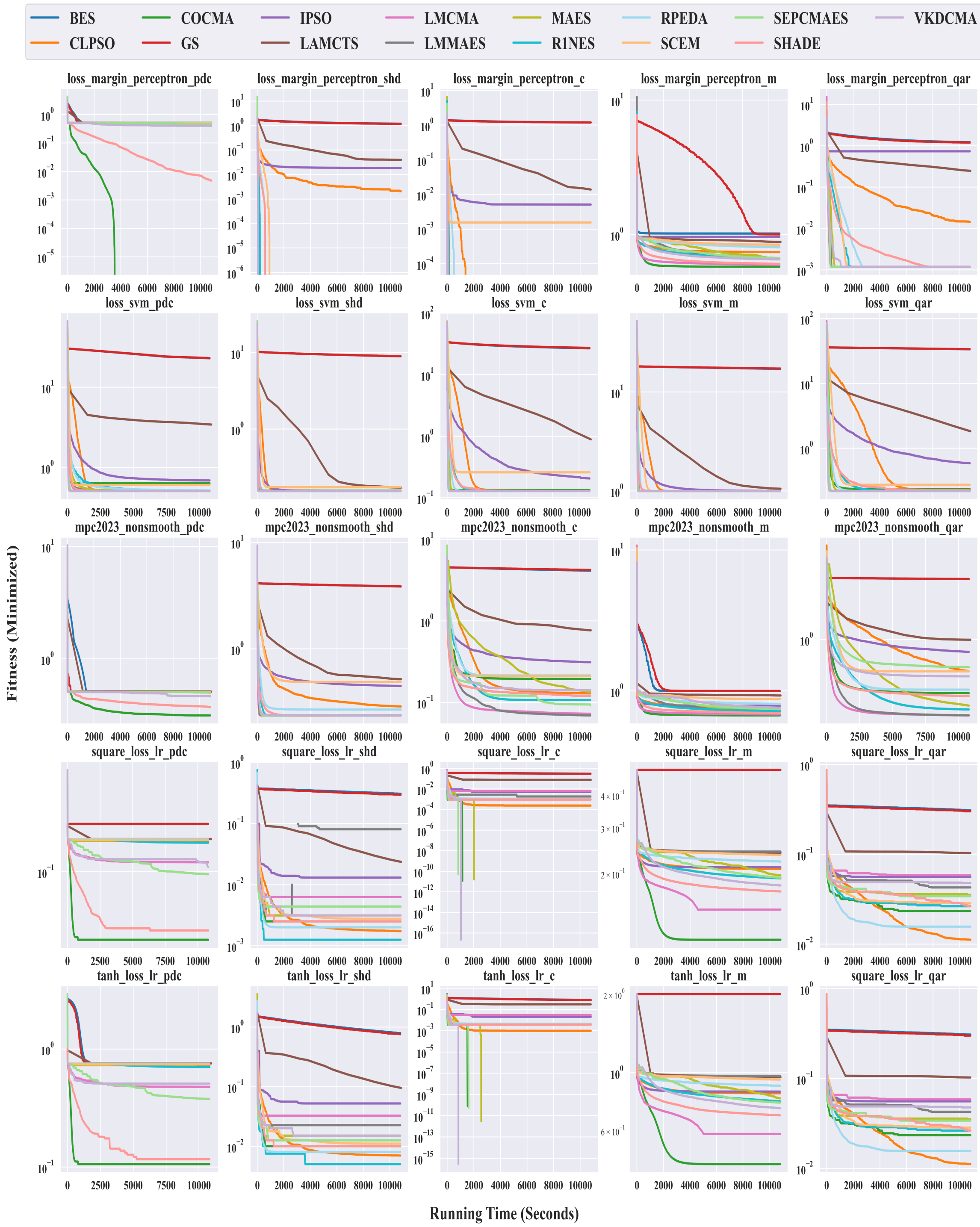

Figure 9: Comparisons of convergence curves of 15 large-scale optimizers on 25 black-box classification tasks given the maximal runtime limit (3 hours) and the fitness threshold (1e-10).

different versions/variants of black-box optimizers in our library, seemingly in accordance with the well-established No Free Lunch Theorems (NFLT) (Wolpert and Macready, 1997).

4. Two Use Cases for Large-Scale BBO

To empirically demonstrate how to properly use PyPop7, in this section we will provide two optimization examples. The first is to show its easy-to-use programming interface unified for all black-box optimizers. The following Python script shows how one large-scale ES variant called LMMAES (Loshchilov et al., 2019) minimizes the popular Rosenbrock test function (Kok and Sandrock, 2009).

```
>>> import numpy as np
>>> from pypop7.benchmarks.base_functions import rosenbrock # notorious test function
>>> ndim_problem = 1000 # dimension of fitness (cost) function to be minimized
>>> problem = {"fitness_function": rosenbrock, # fitness function to be minimized
...            "ndim_problem": ndim_problem, # function dimension
...            "lower_boundary": -5.0*np.ones((ndim_problem,)), # lower search boundary
...            "upper_boundary": 5.0*np.ones((ndim_problem,))} # upper search boundary
>>> from pypop7.optimizers.es.lmmaes import LMMAES # or using any other optimizers
>>> options = {"fitness_threshold": 1e-10, # fitness threshold to terminate evolution
...            "max_runtime": 3600, # to terminate evolution when runtime exceeds 1 hour
...            "seed_rng": 0, # seed of random number generation for repeatability
...            "x": 4.0*np.ones((ndim_problem,)), # initial mean of search distribution
...            "sigma": 3.0, # initial global step-size of search distribution
...            "verbose": 500} # to print verbose information every 500 generations
>>> lmmaes = LMMAES(problem, options) # to initialize this black-box optimizer
>>> results = lmmaes.optimize() # to run its time-consuming search process on high dimensions
>>> # to print the best-so-far fitness found and the number of function evaluations used
>>> print(results["best_so_far_y"], results["n_function_evaluations"])
```

The second is to present the benchmarking process of one black-box optimizer on the well-documented COCO/BBOB platform (Varelas et al., 2020), which is shown below.

```
>>> import os
>>> import webbrowser # for post-processing in the browser
>>> import numpy as np
>>> import cocoex # experimentation module of 'COCO'
>>> import cocopp # post-processing module of 'COCO'
>>> from pypop7.optimizers.es.maes import MAES
>>> suite, output = "bbob", "COCO-PyPop7-MAES"
>>> budget_multiplier = 1e3 # or 1e4, 1e5, ...
>>> observer = cocoex.Observer(suite, "result_folder:" + output)
>>> minimal_print = cocoex.utilities.MiniPrint()
>>> for function in cocoex.Suite(suite, "", ""):
...     function.observe_with(observer) # to generate data for 'cocopp' post-processing
...     sigma = np.min(function.upper_bounds - function.lower_bounds) / 3.0
...     problem = {"fitness_function": function,
...                "ndim_problem": function.dimension,
...                "lower_boundary": function.lower_bounds,
...                "upper_boundary": function.upper_bounds}
...     options = {"max_function_evaluations": function.dimension * budget_multiplier,
...                "seed_rng": 2022,
```

```
...              "x": function.initial_solution,
...              "sigma": sigma}
...     solver = MAES(problem, options)
...     print(solver.optimize())
>>> cocopp.main(observer.result_folder)
>>> webbrowser.open("file://" + os.getcwd() + "/ppdata/index.html")
```

For more examples, please refer to its online documentations: pypop.rtfd.io. Note that we have provided at least one example for each black-box optimizer in its corresponding API online document.

5. Conclusion

In this paper, we have provided an open-source pure-Python library (called PyPop7) for BBO with modular coding structures and full-fledged online documentations. Up to now, this light-weighted library has been used not only by our own work, e.g., (Duan et al., 2022) and (Duan et al., 2023), but also by other work, such as, prompt tuning of vision-language models (Yu et al., 2023), nonlinear optimization for radiotherapy[3], and robotics planning/-control (Zhang et al., 2024; Lee et al., 2023). Please refer to its online documentations for an up-to-date summary of its applications.

As next steps, we plan to further enhance its capability of BBO from five aspects, as shown in the following:

- Massive parallelism (Chalumeau et al., 2024; Lange, 2023),
- Constrains handling (Hellwig and Beyer, 2024),
- Noisy optimization (Häse et al., 2021; Hansen et al., 2009; Beyer, 2000),
- Meta-learning/optimization (Lange et al., 2023; Vicol, 2023; Li et al., 2023; Vicol et al., 2021), and
- Automatic algorithm design, in particular automated algorithm selection/configuration (Schede et al., 2022; Kerschke et al., 2019).

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grant 72401122, in part by Guangdong Basic and Applied Basic Research Foundation under Grants No. 2024A1515012241 and 2021A1515110024, in part by the Shenzhen Fundamental Research Program under Grant No. JCYJ20200109141235597, and in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant No. 2017ZT07X386.

3. https://github.com/pyanno4rt/pyanno4rt

References

Y. Akimoto and N. Hansen. Online model selection for restricted covariance matrix adaptation. In *PPSN*, pages 3–13, 2016.

Y. Akimoto and N. Hansen. Diagonal acceleration for covariance matrix adaptation evolution strategies. *Evol. Comput.*, 28(3):405–435, 2020.

Y. Akimoto, A. Auger, et al. Global linear convergence of evolution strategies on more than smooth strongly convex functions. *SIAM J. Optim.*, 32(2):1402–1429, 2022.

S.-i. Amari. Natural gradient works efficiently in learning. *Neural Comput.*, 10(2):251–276, 1998.

B. Amos and D. Yarats. The differentiable cross-entropy method. In *ICML*, pages 291–302, 2020.

C. Aranha, C. L. Camacho Villalón, et al. Metaphor-based metaheuristics, a call for action: The elephant in the room. *Swarm Intell.*, 16(1):1–6, 2022.

D. V. Arnold and H.-G. Beyer. A comparison of evolution strategies with other direct search methods in the presence of noise. *Comput. Optim. Appl.*, 24(1):135–159, 2003.

T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Handbook of evolutionary computation*. CRC Press, 1997.

T. Bäck, C. Foussette, et al. *Contemporary evolution strategies*. Springer, 2013.

S. Baluja. Genetic algorithms and explicit search statistics. In *NeurIPS*, 1996.

S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *ICML*, pages 38–46, 1995.

R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

R. Bellman. *Adaptive control processes: A guided tour*. Princeton University Press, 1961.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Stat. Sci.*, 8(1):10–15, 1993.

H.-G. Beyer. Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice. *Comput. Meth. Appl. Mech. Eng.*, 186(2-4):239–267, 2000.

H.-G. Beyer and H.-P. Schwefel. Evolution strategies – a comprehensive introduction. *Nat. Comput.*, 1:3–52, 2002.

H.-G. Beyer and B. Sendhoff. Simplify your covariance matrix adaptation evolution strategy. *IEEE Trans. Evol. Comput.*, 21(5):746–759, 2017.

F. Biscani and D. Izzo. A parallel global multiobjective framework for optimization: Pagmo. *J. Open Source Softw.*, 5(53):2338, 2020.

R. Bollapragada and S. M. Wild. Adaptive sampling quasi-newton methods for zeroth-order stochastic optimization. *Math. Program. Comput.*, 15(2):327–364, 2023.

R. Bollapragada, R. Byrd, et al. Adaptive sampling strategies for stochastic optimization. *SIAM J. Optim.*, 28(4):3312–3343, 2018.

J. Bolte, L. Miclo, et al. Swarm gradient dynamics for global optimization: The mean-field limit case. *Math. Program.*, 205(1-2):661–701, 2024.

E. Bonabeau, M. Dorigo, et al. *Swarm intelligence: From natural to artificial systems.* Oxford University Press, 1999.

C. Bouttier and I. Gavra. Convergence rate of a simulated annealing algorithm with noisy observations. *J. Mach. Learn. Res.*, 20(4):1–45, 2019.

D. Brockhoff, A. Auger, et al. Mirrored sampling and sequential selection for evolution strategies. In *PPSN*, pages 11–21, 2010.

D. Brookes, A. Busia, et al. A view of estimation of distribution algorithms through the lens of expectation-maximization. In *GECCOC*, pages 189–190, 2020.

S. H. Brooks. A discussion of random methods for seeking maxima. *Oper. Res.*, 6(2): 244–251, 1958.

L. Bungert, T. Roith, et al. Polarized consensus-based dynamics for optimization and sampling. *Math. Program.*, 2024.

F. Campelo and C. Aranha. Lessons from the evolutionary computation bestiary. *Artif. Life*, 29(4):421–432, 2023.

F. Chalumeau, B. Lim, et al. QDax: A library for quality-diversity and population-based algorithms with hardware acceleration. *J. Mach. Learn. Res.*, 25(108):1–16, 2024.

T. Chen, J. van Gelder, et al. Classification with a disordered dopant-atom network in silicon. *Nature*, 577(7790):341–345, 2020.

S. Choudhury, B. Narayanan, et al. Generative machine learning produces kinetic models that accurately characterize intracellular metabolic states. bioRxiv, 2023.

C. Cipriani, H. Huang, et al. Zero-inertia limit: From particle swarm optimization to consensus-based optimization. *SIAM J. Math. Anal.*, 54(3):3091–3121, 2022.

A. Corana, M. Marchesi, et al. Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm—corrigenda for this article is available here. *ACM Trans. Math. Softw.*, 13(3):262–280, 1987.

G. Cui, M. L. Wong, et al. Machine learning for direct marketing response models: Bayesian networks with evolutionary programming. *Manag. Sci.*, 52(4):597–612, 2006.

K. Deb, A. Anand, et al. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evol. Comput.*, 10(4):371–395, 2002.

N. Demo, M. Tezzele, et al. A supervised learning approach involving active subspaces for an efficient genetic algorithm in high-dimensional optimization problems. *SIAM J. Sci. Comput.*, 43(3):B831–B853, 2021.

Y. Diouane, S. Gratton, et al. Globally convergent evolution strategies. *Math. Program.*, 152(1):467–490, 2015.

Q. Duan, G. Zhou, et al. Collective learning of low-memory matrix adaptation for large-scale black-box optimization. In *PPSN*, pages 281–294, 2022.

Q. Duan, C. Shao, et al. Cooperative coevolution for non-separable large-scale black-box optimization: Convergence analyses and distributed accelerations. arXiv preprint, 2023.

A. E. Eiben and J. Smith. From evolutionary computation to the evolution of things. *Nature*, 521(7553):476–482, 2015.

J. Fan, R. Lau, et al. Utilizing domain knowledge in neuroevolution. In *ICML*, pages 170–177, 2003.

E. Fermi. Numerical solution of a minimum problem. Technical report, Los Alamos Scientific Lab., Los Alamos, NM USA, 1952.

D. B. Fogel. Evolutionary programming: An introduction and some current directions. *Stat. Comput.*, 4(2):113–129, 1994.

D. B. Fogel. An overview of evolutionary programming. In *Evolutionary Algorithms*, pages 89–109. Springer, 1999.

D. B. Fogel and L. J. Fogel. An introduction to evolutionary programming. In *ECAE*, pages 21–33, 1995.

L. J. Fogel, A. J. Owens, et al. Intelligent decision-making through a simulation of evolution. *Trans. Hum. Factors Electron.*, HFE-6(1):13–23, 1965.

M. Fornasier, L. Pareschi, et al. Consensus-based optimization on the sphere: Convergence to global minimizers and machine learning. *J. Mach. Learn. Res.*, 22(237):1–55, 2021.

S. Forrest. Genetic algorithms: Principles of natural selection applied to computation. *Science*, 261(5123):872–878, 1993.

F.-A. Fortin, F.-M. D. Rainville, et al. DEAP: Evolutionary algorithms made easy. *J. Mach. Learn. Res.*, 13(70):2171–2175, 2012.

K. Gao and O. Sener. Generalizing gaussian smoothing for random search. In *ICML*, pages 7077–7101, 2022.

C. García-Martínez, M. Lozano, et al. Global and local real-coded genetic algorithms based on parent-centric crossover operators. *Eur. J. Oper. Res.*, 185(3):1088–1113, 2008.

D. E. Goldberg. Genetic and evolutionary algorithms come of age. *Commun. ACM*, 37(3): 113–119, 1994.

D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Mach. Learn.*, 3(2):95–99, 1988.

F. J. Gomez and J. Schmidhuber. Co-evolving recurrent neurons learn deep memory POMDPs. In *GECCO*, pages 491–498, 2005.

F. J. Gomez, R. Miikkulainen, et al. Solving non-markovian control tasks with neuroevolution. In *IJCAI*, pages 1356–1361, 1999.

F. J. Gomez, J. Schmidhuber, et al. Accelerated neural evolution through cooperatively coevolved synapses. *J. Mach. Learn. Res.*, 9:937–965, 2008.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, 2001.

N. Hansen, A. S. P. Niederberger, et al. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Trans. Evol. Comput.*, 13(1):180–197, 2009.

N. Hansen, A. Auger, et al. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optim. Methods Softw.*, 36(1):114–144, 2021.

C. R. Harris, K. J. Millman, et al. Array programming with NumPy. *Nature*, 585(7825): 357–362, 2020.

F. Häse, M. Aldeghi, et al. Olympus: A benchmarking framework for noisy optimization and experiment planning. *Mach. Learn.: Sci. Technol.*, 2(3):035021, 2021.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

X. He, Z. Zheng, et al. MMES: Mixture model-based evolution strategy for large-scale optimization. *IEEE Trans. Evol. Comput.*, 25(2):320–333, 2021.

M. Hellwig and H.-G. Beyer. Analyzing design principles for competitive evolution strategies in constrained search spaces. arXiv preprint, 2024.

S. I. Higgins, T. Conradi, et al. Limited climatic space for alternative ecosystem states in africa. *Science*, 380(6649):1038–1042, 2023.

J. H. Holland. Outline for a logical theory of adaptive systems. *J. ACM*, 9(3):297–314, 1962.

R. Hooke and T. A. Jeeves. "Direct search" solution of numerical and statistical problems. *J. ACM*, 8(2):212–229, 1961.

J. Hu, M. C. Fu, et al. A model reference adaptive search method for global optimization. *Oper. Res.*, 55(3):549–568, 2007.

H. Huang, J. Qiu, et al. Consensus-based optimization for saddle point problems. *SIAM J. Control Optim.*, 62(2):1093–1121, 2024.

M. Hüttenrauch and G. Neumann. Robust black-box optimization for stochastic search and episodic reinforcement learning. *J. Mach. Learn. Res.*, 25(153):1–44, 2024.

A. Ilyas, L. Engstrom, et al. Black-box adversarial attacks with limited queries and information. In *ICML*, pages 2137–2146, 2018.

D. R. Jones, M. Schonlau, et al. Efficient global optimization of expensive black-box functions. *J. Glob. Optim.*, 13(4):455–492, 1998.

A. Kabán, J. Bootkrajang, et al. Toward large-scale continuous EDA: A random matrix theory perspective. *Evol. Comput.*, 24(2):255–291, 2016.

A. F. Kaupe. Algorithm 178: Direct search. *Commun. ACM*, 6(6):313–314, 1963.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *ICNN*, pages 1942–1948 vol.4, 1995.

J. F. Kennedy, R. C. Eberhart, et al. *Swarm intelligence*. Morgan Kaufmann, 2001.

P. Kerschke, H. H. Hoos, et al. Automated algorithm selection: Survey and perspectives. *Evol. Comput.*, 27(1):3–45, 2019.

S. Kirkpatrick, C. D. Gelatt, et al. Optimization by simulated annealing. *Science*, 220 (4598):671–680, 1983.

S. Kok and C. Sandrock. Locating and characterizing the stationary points of the extended rosenbrock function. *Evol. Comput.*, 17(3):437–453, 2009.

T. G. Kolda, R. M. Lewis, et al. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Rev.*, 45(3):385–482, 2003.

V. Koob, R. Ulrich, et al. Response activation and activation–transmission in response-based backward crosstalk: Analyses and simulations with an extended diffusion model. *Psyc. Rev.*, 130(1):102–136, 2023.

O. Krause, D. R. Arbonès, et al. CMA-ES with optimal covariance update and storage complexity. In *NeurIPS*, pages 370–378, 2016.

A. Laganowsky, E. Reading, et al. Membrane proteins bind lipids selectively to modulate their structure and function. *Nature*, 510(7503):172–175, 2014.

J. C. Lagarias, J. A. Reeds, et al. Convergence properties of the nelder-mead simplex method in low dimensions. *SIAM J. Optim.*, 9(1):112–147, 1998.

R. T. Lange. Evosax: JAX-based evolution strategies. In *GECCO*, pages 659–662, 2023.

R. T. Lange, T. Schaul, et al. Discovering evolution strategies via meta-black-box optimization. In *ICLR*, 2023.

P. Larrañaga, editor. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer, 2002.

Y. LeCun, Y. Bengio, et al. Deep learning. *Nature*, 521(7553):436–444, 2015.

C.-Y. Lee and X. Yao. Evolutionary programming using mutations based on the levy probability distribution. *IEEE Trans. Evol. Comput.*, 8(1):1–13, 2004.

Y. Lee, K. Lee, et al. The planner optimization problem: Formulations and frameworks. arXiv preprint, 2023.

O. Li, J. Harrison, et al. Variance-reduced gradient estimation via noise-reuse in online evolution strategies. In *NeurIPS*, pages 45489–45501, 2023.

S. Li, T. Driver, et al. Attosecond coherent electron motion in auger-meitner decay. *Science*, 375(6578):285–290, 2022a.

Y. Li, M. Cheng, et al. A review of adversarial attack and defense for classification methods. *Am. Stat.*, 76(4):329–345, 2022b.

Z. Li and Q. Zhang. A simple yet efficient evolution strategy for large-scale black-box optimization. *IEEE Trans. Evol. Comput.*, 22(5):637–646, 2018.

J. Liang, A. Qin, et al. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Trans. Evol. Comput.*, 10(3):281–295, 2006.

S. Liu, B. Kailkhura, et al. Zeroth-order stochastic variance reduction for nonconvex optimization. In *NeurIPS*, pages 3727–3737, 2018.

S. Liu, P.-Y. Chen, et al. SignSGD via zeroth-order oracle. In *ICLR*, 2019.

I. Loshchilov. LM-CMA: An alternative to L-BFGS for large-scale black box optimization. *Evol. Comput.*, 25(1):143–171, 2017.

I. Loshchilov, T. Glasmachers, et al. Large scale black-box optimization by limited-memory matrix adaptation. *IEEE Trans. Evol. Comput.*, 23(2):353–358, 2019.

M. Lutz. *Learning python: Powerful object-oriented programming*. O'Reilly, 2013.

S. Mannor, R. Y. Rubinstein, et al. The cross entropy method for fast policy search. In *ICML*, pages 512–519, 2003.

Y. Mei, M. N. Omidvar, et al. A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. *ACM Trans. Math. Softw.*, 42(2):13:1–24, 2016.

J. M. Melis, I. Siwanowicz, et al. Machine learning reveals the control mechanics of an insect wing hinge. *Nature*, 628(8009):795–803, 2024.

N. Metropolis, A. W. Rosenbluth, et al. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092, 1953.

L. Meunier, H. Rakotoarison, et al. Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking. *IEEE Trans. Evol. Comput.*, 26(3): 490–500, 2022.

R. Miikkulainen and S. Forrest. A biological perspective on evolutionary computation. *Nat. Mach. Intell.*, 3(1):9–15, 2021.

M. Mitchell, J. Holland, et al. When will a genetic algorithm outperform hill climbing. In *NeurIPS*, pages 51–58, 1993.

D. E. Moriarty and R. Miikkulainen. Efficient learning from delayed rewards through symbiotic evolution. In *ICML*, pages 396–404, 1995.

D. E. Moriarty and R. Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Mach. Learn.*, 22:11–32, 1996.

P. Moritz, R. Nishihara, et al. Ray: A distributed framework for emerging AI applications. In *OSDI*, pages 561–577, 2018.

J. A. Nelder and R. Mead. A simplex method for function minimization. *Comput. J.*, 7(4): 308–313, 1965.

Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Found. Comput. Math.*, 17(2):527–566, 2017.

Y. Ollivier, L. Arnold, et al. Information-geometric optimization algorithms: A unifying picture via invariance principles. *J. Mach. Learn. Res.*, 18(18):1–65, 2017.

L. Panait, K. Tuyls, et al. Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *J. Mach. Learn. Res.*, 9:423–457, 2008.

F. Pedregosa, G. Varoquaux, et al. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(85):2825–2830, 2011.

M. A. Potter and K. A. Jong. A cooperative coevolutionary approach to function optimization. In *PPSN*, pages 249–257, 1994.

M. A. Potter and K. A. D. Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evol. Comput.*, 8(1):1–29, 2000.

M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Comput. J.*, 7(2):155–162, 1964.

J. Rapin and O. Teytaud. Nevergrad - a gradient-free optimization platform. GitHub, 2018.

L. Rastrigin. The convergence of the random search method in the external control of many-parameter system. *Autom. Remote Control*, 24:1337–1342, 1963.

I. Rechenberg. The evolution strategy. a mathematical model of darwinian evolution. In *ISS*, pages 122–132, 1984.

R. Ros and N. Hansen. A simple modification in CMA-ES achieving linear time and space complexity. In *PPSN*, pages 296–305, 2008.

M. T. Rosenstein and A. G. Barto. Robot weightlifting by direct policy search. In *IJCAI*, pages 839–846, 2001.

Y. Ruan, Y. Xiong, et al. Learning to learn by zeroth-order oracle. In *ICLR*, 2020.

R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method: A unified approach to combinatorial optimization, monte-carlo simulation and machine learning.* Springer, 2004.

G. Rudolph. Evolutionary strategies. In *Handbook of Natural Computing*, pages 673–698. Springer, 2012.

T. Salimans, J. Ho, et al. Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint, 2017.

R. Samyak and J. A. Palacios. Statistical summaries of unlabelled evolutionary trees. *Biometrika*, 111(1):171–193, 2024.

T. Schaul, J. Bayer, et al. PyBrain. *J. Mach. Learn. Res.*, 11:743–746, 2010.

T. Schaul, T. Glasmachers, et al. High dimensions and heavy tails for natural evolution strategies. In *GECCO*, pages 845–852, 2011.

E. Schede, J. Brandt, et al. A survey of methods for automated algorithm configuration. *J. Artif. Intell. Res.*, 75:425–487, 2022.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Netw.*, 61:85–117, 2015.

J. Schmidhuber, S. Hochreiter, et al. Evaluating benchmark problems by random guessing. In *A Field Guide to Dynamical Recurrent Networks*, pages 231–235. IEEE, 2001.

J. Schmidhuber, D. Wierstra, et al. Training recurrent networks by evolino. *Neural Comput.*, 19(3):757–779, 2007.

M. Schumer and K. Steiglitz. Adaptive step size random search. *IEEE Trans. Autom. Control*, 13(3):270–276, 1968.

H. P. Schwefel. Evolution strategies: A family of non-linear optimization techniques based on imitating some principles of organic evolution. *Ann. Oper. Res.*, 1(2):165–167, 1984.

B. Shahriari, K. Swersky, et al. Taking the human out of the loop: A review of bayesian optimization. *Proc. IEEE*, 104(1):148–175, 2016.

P. Siarry, G. Berthiau, et al. Enhanced simulated annealing for globally minimizing functions of many-continuous variables. *ACM Trans. Math. Softw.*, 23(2):209–228, 1997.

F. J. Solis and R. J.-B. Wets. Minimization by random search techniques. *Math. Oper. Res.*, 6(1):19–30, 1981.

S. Sonnenburg, M. L. Braun, et al. The need for open source software in machine learning. *J. Mach. Learn. Res.*, 8:2443–2466, 2007.

S. U. Stich. On low complexity acceleration techniques for randomized optimization. In *PPSN*, pages 130–140, 2014.

R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.*, 11(4):341–359, 1997.

T. Sun, Y. Shao, et al. Black-box tuning for language-model-as-a-service. In *ICML*, pages 20841–20855, 2022.

Y. Sun, T. Schaul, et al. A linear time natural evolution strategy for non-separable functions. In *GECCOC*, pages 61–62, 2013.

J. Swan, S. Adriaensen, et al. Metaheuristics "In the large". *Eur. J. Oper. Res.*, 297(2): 393–406, 2022.

D. Tang, Q. Ye, et al. Opening the black box: Hierarchical sampling optimization for hand pose estimation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(9):2161–2175, 2019.

V. Torczon. On the convergence of pattern search algorithms. *SIAM J. Optim.*, 7(1):1–25, 1997.

K. Varelas, A. Auger, et al. A comparative study of large-scale variants of CMA-ES. In *PPSN*, pages 3–15, 2018.

K. Varelas, O. A. El Hara, et al. Benchmarking large-scale continuous optimizers: The bbob-largescale testbed, a COCO software guide and beyond. *Appl. Soft Comput.*, 97: 106737, 2020.

P. Vicol. Low-variance gradient estimation in unrolled computation graphs with es-single. In *ICML*, pages 35084–35119, 2023.

P. Vicol, L. Metz, et al. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In *ICML*, pages 10553–10563, 2021.

P. Virtanen, R. Gommers, et al. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nat. Meth.*, 17(3):261–272, 2020.

L. Wang, R. Fonseca, et al. Learning search space partition for black-box optimization using monte carlo tree search. In *NeurIPS*, pages 19511–19522, 2020.

T. Wang and J. Ba. Exploring model-based planning with policy networks. In *ICLR*, 2020.

X. Wei, H. Yan, et al. Sparse black-box video attack with reinforcement learning. *Int. J. Comput. Vis.*, 130(6):1459–1473, 2022.

D. Whitley. Next generation genetic algorithms: A user's guide and tutorial. In *Handbook of Metaheuristics*, pages 245–274. Springer, 2019.

D. Wierstra, T. Schaul, et al. Natural evolution strategies. In *CEC*, pages 3381–3387, 2008.

D. Wierstra, T. Schaul, et al. Natural evolution strategies. *J. Mach. Learn. Res.*, 15(27): 949–980, 2014.

D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.*, 1(1):67–82, 1997.

M. Wright. Direct search methods: Once scorned, now respectable. In *Numerical Analysis*, pages 191–208. Addison-Wesley, 1996.

S. J. Wright. Coordinate descent algorithms. *Math. Program.*, 151(1):3–34, 2015.

P. Xu, F. Roosta, et al. Second-order optimization for non-convex machine learning: An empirical study. In *SDM*, pages 199–207, 2020.

X. Yao, Y. Liu, et al. Evolutionary programming made faster. *IEEE Trans. Evol. Comput.*, 3(2):82–102, 1999.

S. Yi, D. Wierstra, et al. Stochastic search using the natural gradient. In *ICML*, pages 1161–1168, 2009.

L. Yu, Q. Chen, et al. Black-box prompt tuning for vision-language model as a service. In *IJCAI*, pages 1686–1694, 2023.

Z. Zhang, Y. Wei, et al. An invariant information geometric method for high-dimensional online optimization. arXiv preprint, 2024.

W. Zheng and B. Doerr. From understanding genetic drift to a smart-restart mechanism for estimation-of-distribution algorithms. *J. Mach. Learn. Res.*, 24(292):1–40, 2023.