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

Qiqi Duan[*,1], Guochen Zhou[*,1], Chang Shao[*,1,2],
Zhuowei Wang[2], Mingyang Feng[1], Yijun Yang[1,2], Qi Zhao[1], Yuhui Shi[1]
[1]Southern University of Science and Technology (SUSTech), Shenzhen
[2]University of Technology Sydney (UTS), Sydney
[*]Equal Contributions.

Abstract: In this paper, we present a pure-Python open-source library, called *PyPop7*, for black-box optimization (BBO). It provides a unified and modular interface for more than 60 versions and variants of different black-box optimization algorithms, particularly population-based optimizers, which can be classified into 12 popular families: Evolution Strategies (ES), Natural Evolution Strategies (NES), Estimation of Distribution Algorithms (EDA), Cross-Entropy Method (CEM), Differential Evolution (DE), Particle Swarm Optimizer (PSO), Cooperative Coevolution (CC), Simulated Annealing (SA), Genetic Algorithms (GA), Evolutionary Programming (EP), Pattern Search (PS), and Random Search (RS). It also provides many examples, interesting tutorials, and full-fledged API documentations. Through this new library, we expect to provide a well-designed platform for benchmarking of optimizers and promote their real-world applications, especially for *large-scale* BBO. Its source code and documentations are available at github.com/Evolutionary-Intelligence/pypop and pypop.readthedocs.io/, respectively.

1. Introduction

Black-box optimization (BBO) problems widely exist in scientific and engineering fields [1]. Recently, direct policy search [2] and black-box attacks [3] of deep neural networks are two representative examples, to name a few. To tackle these challenging black-box problems, a variety of powerful optimization algorithms (particularly their large-scale variants) have been proposed from different research communities (e.g., artificial intelligence/machine learning, mathematical programming, statistics, control, electronic engineering). In this paper, we incorporate many of these recent advances on BBO into an open-source pure-Python library called *PyPop7*. The main objective of this library is to provide a well-designed platform for benchmarking of black-box optimizers and promote their real-world applications, especially for *large-scale* BBO.

Recently, Hansen *et al.* [4] released a well-documented platform called *COCO* for comparing continuous optimizers in a black-box setting, after experiencing ten-years development. However, *COCO* focused on only the design of benchmarking functions and did not provide any black-box optimizers. A similar work is the popular platform named *NeverGrad*, developed recently by Facebook's scientists [5]. Although it provided basic versions of several black-box optimizers, *NeverGrad* did not widely cover their variants[1] for large-scale BBO (LSBBO). By providing many of their newest LSBBO variants, our *algorithms-design-centric* library can be regarded as an important complementary to the above two *benchmarking-functions-centric* libraries.

2. A Modular Framework for Black-Box Optimizers (BBO)

For readability and maintainability, PyPop7 provides a *unified* programming interface with a *modular* code structure for >60 versions and variants of BBO from different research communities, particularly **population-based** optimizers [6,7]. They can be roughly classified into 12 popular algorithm families: Evolution Strategies (ES) [8], Natural Evolution Strategies (NES) [9], Estimation of Distribution Algorithms (EDA) [10], Cross-Entropy Method (CEM) [11], Differential Evolution (DE) [12], Particle Swarm Optimizer (PSO) [13], Cooperative Coevolution (CC) [14], Simulated Annealing (SA) [15], Genetic Algorithms (GA) [16,17], Evolutionary Programming (EP) [18], Pattern Search (PS) [19], and Random Search (RS) [20].

For almost all black-box optimizers, their *sampling*-based nature (i.e., considering only the *zeroth-order* information of the objective function) typically results in the well-established "*curse of dimensionality*" issue for large-scale optimization. See this landmark theoretical paper [21] for the analysis of convergence rate (under convex functions). To alleviate this issue, a variety of new sophisticated techniques have been proposed over the past ten years, which can be roughly classified into 5 families: decomposition/embedding of search space [14,22], low-memory approximation [23], low-rank learning [24], variance-reduction sampling [25], and efficient (self-)adaptive sampling [26].

2.1 Computational Efficiency

For computational efficiency, this library depends heavily on two core scientific computing Python libraries (i.e., *NumPy* [27] and *SciPy* [28]). More specifically, the *numpy.array* data structure and its functions are chosen as the basic way to store and operate the population (e.g., sampling, updating, indexing, and sorting), which can lead to significant speedups than the built-in data structure *list*.

2.2 Repeatability

For the randomized optimizer, properly controlling its random process is very key to well repeat its numerical experiments. For this library, the random seed for each optimizer needs to be *explicitly* given for repeatability [29], according to the newest *NumPy*'s suggestion for *Random Sampling*.

For each black-box optimizer considered in this library, we try our best to provide a repeatability report (involving comparisons with the original reference), if possible.

3. Usage Case

In this section, we provide an optimization example to show PyPop7's easy-to-use programming interface unified for all black-box optimizers:

---

[1] Although it incorporated one specific test suite for large-scale BBO (LSBBO), this LSBBO test suite is built mainly on the "*Partially Additive Separability*" assumption. Since such an assumption is hard to satisfy on most real-world applications, we will not consider the research line based on this "*Partially Additive Separability*" assumption in this library.

```
>>> import numpy as np
>>> def rosenbrock(x): # the notorious test function to be minimized in the optimization community
...     return 100 * np.sum(np.power(x[1:] - np.power(x[:-1], 2), 2)) + np.sum(np.power(x[:-1] - 1, 2))
>>> ndim_problem = 1000 # dimension of fitness (cost) function to be minimized
>>> problem = {'fitness_function': rosenbrock, # fitness function to be minimized
...            'ndim_problem': ndim_problem, # dimension
...            'lower_boundary': -5 * np.ones((ndim_problem,)), # search boundary
...            'upper_boundary': 5 * np.ones((ndim_problem,))}
>>> from pypop7.optimizers.es.lmmaes import LMMAES # to choose any optimizer you prefer in this library
>>> options = {'fitness_threshold': 1e-10, # terminate when the best-so-far fitness is lower than 1e-10
...           'max_runtime': 3600, # terminate when the actual runtime exceeds 1 hour (i.e., 3600 seconds)
...           'seed_rng': 0, # seed of random number generation (which must be set for repeatability)
...           'x': 4 * np.ones((ndim_problem,)), # initial mean of search (mutation/sampling) distribution
...           'sigma': 0.3, # initial global step-size of search distribution
...           'verbose': 500}
>>> lmmaes = LMMAES(problem, options) # initialize the optimizer
>>> results = lmmaes.optimize() # run its (time-consuming) search process
>>> # print the best-so-far fitness and used function evaluations returned by the black-box optimizer
>>> print(results['best_so_far_y'], results['n_function_evaluations'])
```

For more examples, refer to its documentation homepage: pypop.readthedocs.io/.

4. Conclusions

In this paper, we provide a well-designed open-source Python library for black-box optimization with a modular code structure and full-fledged API documentations. We expect it to be used as a benchmarking platform of *large-scale* BBO. In the future, we will enhance its optimization capability via the following two new functionalities:

- Parallel and distributed computing (see [30] based on this new library),
- Automated algorithm selection and configuration [31].

Reference

1. Conn, A.R., Scheinberg, K. and Vicente, L.N., 2009. Introduction to derivative-free optimization. Society for Industrial and Applied Mathematics.
2. Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M.I. and Stoica, I., 2018. Ray: A distributed framework for emerging AI applications. In USENIX Symposium on Operating Systems Design and Implementation (pp. 561-577).
3. Ilyas, A., Engstrom, L., Athalye, A. and Lin, J., 2018, July. Black-box adversarial attacks with limited queries and information. In International Conference on Machine Learning (pp. 2137-2146). PMLR.
4. 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), pp.114-144.
5. Meunier, L., Rakotoarison, H., Wong, P.K., Roziere, B., Rapin, J., Teytaud, O.,

Moreau, A. and Doerr, C., 2022. Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking. IEEE Transactions on Evolutionary Computation, 26(3), pp.490-500.
6. Eiben, A.E. and Smith, J., 2015. From evolutionary computation to the evolution of things. Nature, 521(7553), pp.476-482.
7. Miikkulainen, R. and Forrest, S., 2021. A biological perspective on evolutionary computation. Nature Machine Intelligence, 3(1), pp.9-15.
8. Ollivier, Y., Arnold, L., Auger, A. and Hansen, N., 2017. Information-geometric optimization algorithms: A unifying picture via invariance principles. Journal of Machine Learning Research, 18(18), pp.1-65.
9. Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J. and Schmidhuber, J., 2014. Natural evolution strategies. Journal of Machine Learning Research, 15(27), pp.949-980.
10. Brookes, D., Busia, A., Fannjiang, C., Murphy, K. and Listgarten, J., 2020, July. A view of estimation of distribution algorithms through the lens of expectation-maximization. In Proceedings of Genetic and Evolutionary Computation Conference Companion (pp. 189-190). ACM.
11. Hu, J., Fu, M.C. and Marcus, S.I., 2007. A model reference adaptive search method for global optimization. Operations Research, 55(3), pp.549-568.
12. Laganowsky, A., Reading, E., Allison, T.M., Ulmschneider, M.B., Degiacomi, M.T., Baldwin, A.J. and Robinson, C.V., 2014. Membrane proteins bind lipids selectively to modulate their structure and function. Nature, 510(7503), pp.172-175.
13. Tang, D., Ye, Q., Yuan, S., Taylor, J., Kohli, P., Keskin, C., Kim, T.K. and Shotton, J., 2019. Opening the black box: Hierarchical sampling optimization for hand pose estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 41(9), pp.2161-2175.
14. Gomez, F., Schmidhuber, J. and Miikkulainen, R., 2008. Accelerated neural evolution through cooperatively coevolved synapses. Journal of Machine Learning Research, 9(31), pp.937-965.
15. Bouttier, C. and Gavra, I., 2019. Convergence rate of a simulated annealing algorithm with noisy observations. Journal of Machine Learning Research, 20(1), pp.127-171.
16. Chen, T., van Gelder, J., van de Ven, B., Amitonov, S.V., de Wilde, B., Euler, H.C.R., Broersma, H., Bobbert, P.A., Zwanenburg, F.A. and van der Wiel, W.G., 2020. Classification with a disordered dopant-atom network in silicon. Nature, 577(7790), pp.341-345.
17. Holland, J.H., 1962. Outline for a logical theory of adaptive systems. Journal of the ACM, 9(3), pp.297-314.
18. Fogel, D.B., 1994. Evolutionary programming: An introduction and some current directions. Statistics and Computing, 4(2), pp.113-129.
19. Hooke, R. and Jeeves, T.A., 1961. "Direct search" solution of numerical and statistical problems. Journal of the ACM, 8(2), pp.212-229.
20. Bergstra, J. and Bengio, Y., 2012. Random search for hyper-parameter

optimization. Journal of Machine Learning Research, 13(2).

21. Nesterov, Y. and Spokoiny, V., 2017. Random gradient-free minimization of convex functions. Foundations of Computational Mathematics, 17(2), pp.527-566.
22. Kabán, A., Bootkrajang, J. and Durrant, R.J., 2016. Toward large-scale continuous EDA: A random matrix theory perspective. Evolutionary Computation, 24(2), pp.255-291.
23. Loshchilov, I., 2017. LM-CMA: An alternative to L-BFGS for large-scale black box optimization. Evolutionary Computation, 25(1), pp.143-171.
24. Li, Z. and Zhang, Q., 2018. A simple yet efficient evolution strategy for large-scale black-box optimization. IEEE Transactions on Evolutionary Computation, 22(5), pp.637-646.
25. Gao, K. and Sener, O., 2022, June. Generalizing Gaussian Smoothing for Random Search. In International Conference on Machine Learning (pp. 7077-7101). PMLR.
26. He, X., Zheng, Z. and Zhou, Y., 2021. MMES: Mixture model-based evolution strategy for large-scale optimization. IEEE Transactions on Evolutionary Computation, 25(2), pp.320-333.
27. Harris, C.R., Millman, K.J., Van Der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J. and Kern, R., 2020. Array programming with NumPy. Nature, 585(7825), pp.357-362.
28. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J. and Van Der Walt, S.J., 2020. SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature methods, 17(3), pp.261-272.
29. Sonnenburg, S., Braun, M.L., Ong, C.S., Bengio, S., Bottou, L., Holmes, G., LeCun, Y., Pereira, F. and Rasmussen, C.E., 2007. The need for open source software in machine learning. Journal of Machine Learning Research, 8, pp.2443-2466.
30. Duan, Q., Zhou, G., Shao, C., Yang, Y. and Shi, Y., 2022. Collective learning of low-memory matrix adaptation for large-scale black-box optimization. In International Conference on Parallel Problem Solving from Nature (pp. 281-294). Springer, Cham.
31. Kerschke, P., Hoos, H.H., Neumann, F. and Trautmann, H., 2019. Automated algorithm selection: Survey and perspectives. Evolutionary Computation, 27(1), pp.3-45.