The Journal of Open Source Software

ParMOO: A Python library for parallel multiobjective simulation optimization

Tyler H. Chang [1¶] **and Stefan M. Wild** [2,1,3]

1 Mathematics and Computer Science Division, Argonne National Laboratory, USA **2** Applied Mathematics and Computational Research Division, Lawrence Berkeley National Laboratory, USA **3** NAISE, Northwestern University, USA ¶ Corresponding author

DOI: 10.21105/joss.04468

Software

- Review
- Repository
- Archive

Editor: Kelly Rowland

Reviewers:

- @Viech
- @JianqiaoMao

Submitted: 13 May 2022
Published: 03 February 2023

Summary

A multiobjective optimization problem (MOOP) is an optimization problem in which multiple objectives are optimized simultaneously. The goal of a MOOP is to find solutions that describe the tradeoff between these (potentially conflicting) objectives. Such a tradeoff surface is called the Pareto front. Real-world MOOPs may also involve constraints – additional hard rules that every solution must adhere to. In a multiobjective *simulation* optimization problem, the objectives are derived from the outputs of one or more computationally expensive simulations. Such problems are ubiquitous in science and engineering.

ParMOO is a Python framework and library of solver components for building and deploying highly customized multiobjective simulation optimization solvers. ParMOO is designed to help engineers, practitioners, and optimization experts exploit available structures in how simulation outputs are used to formulate the objectives for a MOOP. We elaborate on these structures and provide two examples in future sections.

Additionally, ParMOO is:

- an open-source project on GitHub,
- `pip`-installable via PyPI or `conda`-installable via conda-forge, and
- fully documented.

Multiobjective Simulation Optimization Software

Existing open source, actively maintained Python packages for solving multiobjective simulation optimization problems include `pymoo` (Blank & Deb, 2020), `pymoso` (Cooper & Hunter, 2020), `Dragonfly` (Kandasamy et al., 2020), `Playtpus` (Hadka, 2015), `jMetalPy` (Benítez-Hidalgo et al., 2019), and `pygmo` (Biscani & Izzo, 2020). Non-multiobjective-optimization-specific Python packages that are often used for implementing multiobjective optimization solvers include `BoTorch` (Balandat et al., 2020) and `DEAP` (Fortin et al., 2012). Other non-Python packages include the Fortran solvers `MODIR` (Campana et al., 2018) and `VTMOP` (Chang et al., 2022), and the Matlab toolboxes `PlatEMO` (Tian et al., 2017) and `BoostDFO` (Tavares et al., 2022).

The above-listed software packages:

a) are not restricted to a particular MOOP application,
b) have source code publicly available for download,
c) are suitable for or contain sub-modules for solving a general form of the multiobjective simulation optimization problem, and
d) provide sufficient documentation for a new user to get started without requiring counsel from the authors.

Statement of Need

All of the previously mentioned software packages are high-quality and/or feature-complete in some sense. However, ParMOO is distinct for the following reasons:

1) ParMOO is designed to be flexible enough to support diverse scientific workflows and solve a wide variety of real-world problem types.
2) ParMOO provides interfaces and solver techniques that are suitable for both introductory and expert users.
3) ParMOO provides bells and whistles that are required in production-quality solvers, such as checkpointing and logging.
4) By layering on top of libEnsemble (Hudson et al., 2022), ParMOO provides an easy-to-use interface for distributing expensive simulation calculations over high performance computing (HPC) resources.
5) ParMOO provides complete documentation, including instructions for potential contributors.
6) ParMOO is designed around extensibility and continuous integration, with the intention of adding support for new features, solvers, techniques, and problem types, some of which may be beyond what we originally envisioned.
7) In situations where there is an exploitable structure in how the simulation outputs are used to define the objectives and/or constraints, ParMOO can exploit this structure by modeling simulation outputs independently.

While many existing solvers provide one or more of properties 1-5, at this time, no other solver has *all* of these properties at once. Additionally, to our knowledge, properties 6 and 7 are unique to ParMOO.

The target audience for ParMOO includes scientists, engineers, optimizers, and other practitioners, who are looking to build or use custom solvers for simulation- or experimentation-based MOOPs.

Our Methodology

In our *statement of need*, we outlined the properties that make ParMOO unique. In this section, we outline our strategy for achieving these goals. In particular, properties 1, 6, and 7 are nontrivial.

First, in order to achieve flexibility and customizability without sacrificing ease of use, we have focused on implementing a multiobjective response surface methodology (RSM) framework, which encompasses a wide range of existing techniques. Using the RSM framework, we decompose multiobjective simulation optimization problems into four central components: i) an initial search/design of experiments, used to generate the initial data set; ii) multiple surrogates, used to model the simulation outputs based on existing data; iii) one or more families of acquisition functions, used to scalarize the problem and guide the optimization solver to multiple distinct solutions; and iv) a single-objective optimization solver, used to solve the scalarized surrogate problems, in order to produce batches of candidate solution points.

In order to achieve property 1, we provide a customizable embedding layer, which can be used to embed categorical, integer, mixed-variable, and other input types into a continuous latent space, where the above components can be easily applied. We also support nonlinear relaxable constraints, by using a multiobjective progressive barrier method.

In order to achieve property 6, we use an object-oriented design, where our `MOOP` class references abstract base classes (ABCs) for each of the above components i–iv, in order to solve a MOOP via RSM. This allows us to quickly customize solver components in a modular fashion, by extending their existing interface. In unforeseen circumstances, we can even extend the

MOOP class itself in order to achieve a completely new behavior or customize our method for distributing simulation evaluations based on a novel scientific workflow.

Finally, for property 7, we are the first RSM solver to model simulation outputs separately from objective and constraint functions. This is useful in situations where the objectives are structured algebraic functions of the simulation outputs (e.g., a sum-of-squared outputs), or where one or more objectives does not depend on the simulations at all. In these situations, the additional structure that is available in exactly how the simulation outputs are being used to formulate the problem is made available to ParMOO's solvers, and can be exploited to improve approximation bounds and convergence rates, and to reduce the need for expensive simulation evaluations.

Example Problems

To demonstrate the utility of ParMOO and the importance of property 7, we describe two current applications.

First, ParMOO is currently being used to calibrate energy density functional (EDF) models, by minimizing the error between expensive simulation outputs and experimental data. Let R_1, ..., R_m denote the m deviations between m-dimensional experimental data D and m-dimensional outputs of an EDF model S. Then, we want to calibrate S by solving the multiobjective problem

$$\min_{x\in[0,1]^n} \left(\sum_{i\in C_1} R_i^2, \sum_{j\in C_2} R_j^2, \sum_{k\in C_3} R_k^2 \right)$$

where C_1, C_2, and C_3 are a partitioning of the indices 1, ..., m into three observable classes, each with different observation and measurement errors; and where x is a set of n unknown modeling parameters for S, normalized to lie in the unit hypercube. In this context, the simulation-based structure comes from the known sum-of-squares equation of the empirical loss function. By modeling, the m simulation outputs in S separately from the three objectives, ParMOO is able to exploit this sum-of-squares structure, similarly as in the single-objective software POUNDERS (Wild, 2017). This example also illustrates ParMOO's ability to utilize parallel resources (property 4), since the expensive EDF simulations are being distributed over HPC resources using libEnsemble.

Second, ParMOO is being used to automate material design and manufacturing in a wet lab-based environment, where each "simulation evaluation" corresponds to the experimental synthesis and characterization of a particular material. In this example, the goal is to maximize the yield and minimize the byproduct of an experimental chemical synthesis, which is carried out in a continuous-flow reactor and characterized using nuclear magnetic resonance spectroscopy, while also maximizing the reaction temperature, which is a directly controllable variable. The simulation-based structure in this problem comes from the known dependence between the directly controllable objective (the reaction temperature), while still accounting for the two experimental "blackbox" objectives (the total material yield and byproduct). This example also demonstrates how ParMOO is able to easily integrate with the material scientists' tools and workflow (property 1), which had to be facilitated using third-party libraries since the interface to the physical experiment could not be wrapped in a simple callable Python function.

Acknowledgements

We would like to thank Jeffrey Larson, Stephen Hudson, and John-Luke Navarro for their advice on documentation, automated testing, and package setup.

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute under Contract No. DE-AC02-06CH11357.

This work was supported by the National Science Foundation CSSI program under award number OAC-2004601 (BAND Collaboration).

References

Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A. G., & Bakshy, E. (2020). BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (Vol. 33, pp. 21524–21538). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/f5b1b89d98b7286673128a5fb112cb9a-Paper.pdf

Benítez-Hidalgo, A., Nebro, A. J., García-Nieto, J., Oregi, I., & Del Ser, J. (2019). jMetalPy: A Python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation*, *51*, 100598. https://doi.org/10.1016/j.swevo.2019.100598

Biscani, F., & Izzo, D. (2020). A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software*, *5*(53), 2338. https://doi.org/10.21105/joss.02338

Blank, J., & Deb, K. (2020). pymoo: Multi-objective optimization in Python. *IEEE Access*, *8*, 89497–89509. https://doi.org/10.1109/ACCESS.2020.2990567

Campana, E. F., Diez, M., Liuzzi, G., Lucidi, S., Pellegrini, R., Piccialli, V., Rinaldi, F., & Serani, A. (2018). A multi-objective DIRECT algorithm for ship hull optimization. *Computational Optimization and Applications*, *71*(1), 53–72. https://doi.org/10.1007/s10589-017-9955-0

Chang, T. H., Watson, L. T., Larson, J., Neveu, N., Thacker, W. I., Deshpande, S., & Lux, T. C. H. (2022). Algorithm 1028: VTMOP: Solver for blackbox multiobjective optimization problems. *ACM Transactions on Mathematical Software*, *48*(3), 36:1--34. https://doi.org/10.1145/3529258

Cooper, K., & Hunter, S. R. (2020). PyMOSO: Software for multi-objective simulation optimization with R-PERLE and R-MinRLE. *INFORMS Journal on Computing*, *32*(4), 1101–1108. https://doi.org/10.1287/ijoc.2019.0902

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, *13*(1), 2171–2175. https://www.jmlr.org/papers/v13/fortin12a.html

Hadka, D. (2015). *Platypus - multiobjective optimization in Python* (Version 1.0.4). GitHub. https://platypus.readthedocs.io/en/latest

Hudson, S., Larson, J., Navarro, J.-L., & Wild, S. (2022). libEnsemble: A library to coordinate the concurrent evaluation of dynamic ensembles of calculations. *IEEE Transactions on Parallel and Distributed Systems*, *33*(4), 977–988. https://doi.org/10.1109/tpds.2021.3082815

Kandasamy, K., Vysyaraju, K. R., Neiswanger, W., Paria, B., Collins, C. R., Schneider, J., Poczos, B., & Xing, E. P. (2020). Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with Dragonfly. *Journal of Machine Learning Research*, *21*(81), 1–27. http://jmlr.org/papers/v21/18-223.html

Tavares, S., Brás, C. P., Custódio, A. L., Duarte, V., & Medeiros, P. (2022). Parallel strategies for direct multisearch. In *Numerical Algorithms: Vols. Online first*. https://doi.org/10.1007/s11075-022-01364-1

Tian, Y., Cheng, R., Zhang, X., & Jin, Y. (2017). PlatEMO: A MATLAB platform for evolutionary multi-objective optimization [educational forum]. *IEEE Computational Intelligence Magazine*, *12*(4), 73–87. https://doi.org/10.1109/MCI.2017.2742868

Wild, S. M. (2017). Solving derivative-free nonlinear least squares problems with POUNDERS. In T. Terlaky, M. F. Anjos, & S. Ahmed (Eds.), *Advances and Trends in Optimization with Engineering Applications* (pp. 529–540). SIAM. https://doi.org/10.1137/1.9781611974683.ch40