The Journal of Open Source Software

Minimalist And Customisable Optimisation Package

Jérôme Buisine[1], **Samuel Delepoulle**[1], **and Christophe Renaud**[1]

1 Univ. Littoral Côte d'Opale, LISIC Calais, France, F-62100

DOI: 10.21105/joss.02812

Software

- Review
- Repository
- Archive

Editor: Melissa Weber Mendonça

Reviewers:

- @stsievert
- @torressa

Submitted: 09 September 2020
Published: 12 March 2021

Summary

Optimisation problems are frequently encountered in science and industry. Given a real-valued function f defined on a set called the search space X, optimising the function f consists of finding a point $x \in X$ that has the optimal value $f(x)$, or at least constructing a sequence $(x_t)_{t\in\mathbf{N}} \in X^{\mathbb{N}}$ that is close to the optimum. Depending on the search space X, optimisation problems can be globally classified as discrete problems (e.g. $X = \{0,1\}^n$) or as continuous problems (e.g. $X = \mathbb{R}^n$). Tools for modelling and solving discrete (Soni, 2017) and continuous (Agarwal et al., 2020) problems are proposed in the literature.

In this paper, `Macop` for `Minimalist And Customisable Optimisation Package`, is proposed as a discrete optimisation Python package which doesn't implement every algorithm in the literature, but provides the ability to quickly develop and test your own algorithm and strategies. The main objective of this package is to provide maximum flexibility, which allows easy implementation when experimenting new algorithms.

Based on a common interaction loop (see Figure 1) of all the algorithms, `Macop` wants to allow users to quickly focus on one of the main parts of this loop.

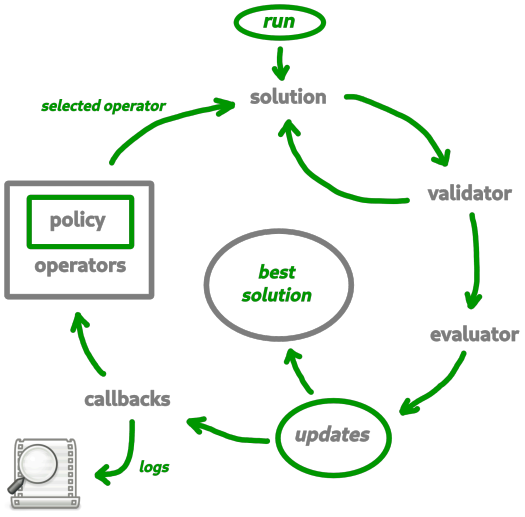

Figure 1: Macop common interation loop.

Statement of Need

Most of the operational research libraries developed in Python offer users either problems and algorithms where it is possible to choose parameters to obtain optimal (or near optimal)

results such as proposed in (Maher et al., 2016), or, libraries targeted to a specific problem or algorithm such as (Perry, 2019). Another package is proposed in (Soni, 2017) which is a comprehensive gradient-free optimization framework written in Python. It seems very similar to `Macop`. However, hiearchic dependencies between algorithms, the notion of callbacks and adaptive operator selection are proposed within `Macop`.

On the other hand, available libraries (Hart et al., 2017; Perez et al., 2012) in the literature did not allow to attach custom evaluation function to each algorithm used in this hierarchy of algorithms. Indeed, it is sometimes possible that the main algorithm manages local searches. Each local search may evaluate the solution differently using a different evaluation function of the parent algorithm (the main algorithm). Such as example, using a surrogate mathematical model (Leprêtre et al., 2019) with a quick-to-evaluate function if the real evaluation function is very expensive in time. This is why in `Macop`, each algorithm can have its own mechanism (or partially), i.e. its evaluation function, its operators for obtaining new solution, as well as its solution update policy. This is independent of the parent algorithm to which it is linked. This means that only the results (solutions found) are exchanged.

Hence, motivation behind **Macop** is a flexible discrete optimisation package allowing a quick implementation of problems. In particular it meets the following needs:

- **Common basis:** the interaction loop during the solution finding process proposed within the package is common to all heuristics. This allows the user to modify only a part of this interaction loop if necessary without rendering the process non-functional;
- **Hierarchy:** a hierarchical algorithm management system is available, especially when an algorithm needs to manage local searches. This hierarchy remains transparent to the user. The main algorithm will be able to manage and control the process of searching for solutions;
- **Flexibility:** although the algorithms are dependent on each other, it is possible that their internal management (search mechanism) is different. This means that the ways in which solutions are evaluated and updated, for example, may be different;
- **Abstraction:** thanks to the modular separability of the package, it is quickly possible to implement new problems, solutions representation, way to evaluate, update solutions within the package;
- **Extensible:** the package is open to extension, i.e. it does not partition the user in these developer choices. It can just as well implement continuous optimization problems if needed while making use of the main interaction loop proposed by the package;
- **Easy Setup:** as a pure Python package distributed is `pip` installable and easy to use.

Target Audience

This package would meet the expectations of people wishing to:

- Solve a problem using an evolutionary algorithm but without developing their own frawmework. They can rely on what the package already proposes but also on its generic and flexible contribution in order to adapt their own content;
- Conduct research work leading to the rapid modification of meta-heuristics and the interaction of different algorithms. More precisely:
  - test new combinations of algorithms. Changing algorithms during evaluations, e.g. different local searches;
  - provide reinforcement learning during searches (e.g. adaptive operator choice strategy).

  - test new multi-objective methods quickly thanks to the proposed algorithmic hierarchy allowing to easily decompose the multi-objective problem into single-objective sub-problems.

- Take advantage of a system for launching calculations from a backup in order to avoid any loss in case of unwanted program interruption;
- Quickly model a problem that is still unknown, i.e. the type of solution and the evaluation function, while taking advantage of the interaction loop proposed by the package.

Description

At the beginning of the development of this library, the idea of making it as modular as possible was topical. The library divide into sub-module forms considered to be the most important to build and solve an optimisation problem.

The package consists of main several modules:

- **solutions:** representation of the solution;
- **validator:** such as constraint programming, a `validator` is a function which is used to validate or not a solution data state;
- **evaluator:** stores problem instance data and implements a `compute` method in order to evaluate a solution;
- **operators:** mutators, crossovers operators to update and obtain new solution;
- **policies:** the way you choose the available operators (might be using reinforcement learning);
- **algorithms:** generic and implemented optimisation research algorithms;
- **callbacks:** callbacks to automatically keep track of the search space advancement and restart from previous state if nedded.

The primary advantage of using Python is that it allows you to dynamically add new members within the new implemented solution or algorithm classes. This of course does not close the possibilities of extension and storage of information within solutions and algorithms. It all depends on the current need.

In `macop.algorithms` module:

Both single and multi-objective algorithms have been implemented for demonstration purposes.

A hierarchy between dependent algorithms is also available, based on a parent/child link, allowing quick access to global information when looking for solutions, such as the best solution found, the number of global evaluations.

The mono-objective Iterated Local Search (Lourenço et al., 2003) algorithm has been implemented. This algorithm aims to perform local searches (child algorithms linked to the main algorithm) and then to explore again (explorations vs. exploitation trade-off). On the multi-objective side, the MOEA/D algorithm (Zhang & Li, 2007) has been implemented by using the weighted-sum of objectives to change multi-objectives problem into a set of mono-objective problems (Tchebycheff approach can also be used (Alves & Almeida, 2007)). Hence, this algorithm aims at decomposing the multi-objective problem into μ single-objective problems in order to obtain the Pareto front (Kim & De Weck, 2005) where single-objective problems are so-called child algorithms linked to the multi-objective algorithm.

The main purpose of these developed algorithms is to show the possibilities of operational search algorithm implementations based on the minimalist structure of the library.

In `macop.solutions` module:

Currently, only combinatorial solutions (discrete problem modelisation) are offered, with the well-known problem of the knapsack as an example. Of course, it's easy to add your own representations of solutions. Solutions modeling continuous problems can also be created by anyone who wants to model his own problem.

In `macop.operators` and `macop.policies` modules:

A few mutation and crossover operators have been implemented. However, it remains quite simple. What is interesting here is that it is possible to develop one's own strategy for choosing operators for the next evaluation. The available UCBPolicy class proposes this functionality as an example, since it will seek to propose the best operator to apply based on a method known as the Adaptive Operator Selection (AOS) via the use of the Upper Confidence Bound (UCB) algorithm (Li et al., 2014).

In `macop.callbacks` module:

The use of callback instance allows both to do an action every k evaluations of information, but also to reload them once the run of the algorithm is cut. Simply inherit the abstract Callback class and implement the `apply` method to backup and `load` to restore. It is possible to add as many callbacks as required. As an example, the implemented UCBPolicy has its own callback allowing the instance to reload previously collected statistics and restart using them.

Conclusion

`Macop` aims to allow the modelling of discrete (usually combinatorial) optimisation problems. It is therefore open to expansion and not closed specifically to a kind of problem.

`Macop` proposes a simple structure of interaction of the main elements (algorithms, operators, solutions, policies, callbacks) for the resolution of operational research problems inside an interaction loop. From its generic structure, it is possible, thanks to the flexible programming paradigm of the Python language, to easily allow the extension and development of new algorithms and problems. Based on simple concepts, this package can therefore meet the needs of the rapid problem implementation.

Acknowledgements

This work is supported by *Agence Nationale de la Recherche* : project ANR-17-CE38-0009

References

Agarwal, S., Mierle, K., & Others. (2020). *Ceres solver* (Version 2.0.0). http://ceres-solver.org.

Alves, M. J., & Almeida, M. (2007). MOTGA: A multiobjective Tchebycheff based genetic algorithm for the multidimensional knapsack problem. *Computers & Operations Research*, *34*(11), 3458–3470. https://doi.org/10.1016/j.cor.2006.02.008

Hart, W. E., Laird, C. D., Watson, J.-P., Woodruff, D. L., Hackebeil, G. A., Nicholson, B. L., & Siirola, J. D. (2017). *Pyomo–optimization modeling in python* (Second Edition, Vol. 67). Springer Science & Business Media.

Kim, I. Y., & De Weck, O. L. (2005). Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. *Structural and Multidisciplinary Optimization*, *29*(2), 149–158. https://doi.org/10.1007/s00158-004-0465-1

Leprêtre, F., Verel, S., Fonlupt, C., & Marion, V. (2019). Walsh functions as surrogate model for pseudo-boolean optimization problems. *Proceedings of the Genetic and Evolutionary Computation Conference*, 303–311. https://doi.org/10.1145/3321707.3321800

Li, K., Fialho, Á., Kwong, S., & Zhang, Q. (2014). Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, *18*(1), 114–130. https://doi.org/10.1109/TEVC.2013.2239648

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 320–353). Springer US. https://doi.org/10.1007/0-306-48056-5_11

Maher, S., Miltenberger, M., Pedroso, J. P., Rehfeldt, D., Schwarz, R., & Serrano, F. (2016). PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In G.-M. Greuel, T. Koch, P. Paule, & A. Sommese (Eds.), *Mathematical software – ICMS 2016* (pp. 301–307). Springer International Publishing. https://doi.org/10.1007/978-3-319-42432-3_37

Perez, R. E., Jansen, P. W., & Martins, J. R. R. A. (2012). PyOpt: A Python-based object-oriented framework for nonlinear constrained optimization. *Structures and Multidisciplinary Optimization*, *45*(1), 101–118. https://doi.org/10.1007/s00158-011-0666-3

Perry, M. (2019). Simanneal. In *GitHub repository* (Version 0.5.0). https://github.com/perrygeo/simanneal; GitHub.

Soni, D. (2017). Solid. In *GitHub repository* (Version 0.11). https://github.com/100/Solid; GitHub.

Zhang, Q., & Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, *11*(6), 712–731. https://doi.org/10.1109/TEVC.2007.892759