SoftwareX 12 (2020) 100520

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

MOBOpt — multi-objective Bayesian optimization

Paulo Paneque Galuzio [a,b,*], Emerson Hochsteiner de Vasconcelos Segundo [b], Leandro dos Santos Coelho [c,d], Viviana Cocco Mariani [b,c]

[a] Positivo University, Prof. Pedro Viriato Parigot de Souza, 5300, Zip code 81280-330, Curitiba, Brazil
[b] Department of Mechanical Engineering, Pontifical Catholic University of Parana, Imaculada Conceicao, 1155, Zip code 80215-901, Curitiba, Brazil
[c] Department of Electrical Engineering, Federal University of Parana, Cel. Francisco Heraclito dos Santos, 100, Zip code 81531-980, Curitiba, Brazil
[d] Industrial and Systems Engineering Graduate Program, Pontifical Catholic University of Parana, Imaculada Conceicao, 1155, Zip code 80215-901, Curitiba, Brazil

ARTICLE INFO

Article history:
Received 10 March 2020
Received in revised form 19 May 2020
Accepted 19 May 2020

MSC:
65K10
62C10

Keywords:
Optimization problems
Multi-objective optimization
Bayesian optimization algorithm

ABSTRACT

This work presents a new software, programmed as a Python class, that implements a multi-objective Bayesian optimization algorithm. The proposed method is able to calculate the Pareto front approximation of optimization problems with fewer objective functions evaluations than other methods, which makes it appropriate for costly objectives. The software was extensively tested on benchmark functions for optimization, and it was able to obtain Pareto Function approximations for the benchmarks with as many as 20 objective function evaluations, those results were obtained for problems with different dimensionalities and constraints.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2020_91
Code Ocean compute capsule	
Legal Code License	GPL-3.0
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	
If available Link to developer documentation/manual	https://github.com/ppgaluzio/MOBOpt/wiki
Support email for questions	galuzio.paulo@protonmail.com

1. Motivation and significance

Optimization of designs and processes constitutes an ubiquitous open problem in science and engineering. Even though there are efficient optimization methods available that work reasonably well for complex problems (for example, see [1]), if the function to be optimized is very costly, most of these methods become undesirable as they rely on a very large number of evaluations of the objective functions [2–4]. Mathematically, the problem of optimization can be simply stated as finding the argument values that return the minimal (or maximal) value of a given function, which is referred to as objective function or objective. In most real-world problems there are more than one, usually conflicting, objective functions to be simultaneously optimized, which leads to the following categorization for the problem: single-objective optimization when there is only one objective function; multi-objective optimization for up to four objectives; and many-objective optimization for more than four.

To optimize many simultaneous objectives is challenging since, in most situations, they are conflicting, i.e., the optimal values are in different regions of the search space, and optimizing one objective function means that other objectives will be non-optimal.

* Corresponding author at: Positivo University, Prof. Pedro Viriato Parigot de Souza, 5300, Zip code 81280-330, Curitiba, Brazil.
E-mail address: galuzio.paulo@protonmail.com (P.P. Galuzio).

https://doi.org/10.1016/j.softx.2020.100520

In this scenario, the solution to the optimization problem is a set that represents the compromise among the many objectives, called the Pareto front of the problem. To address this problem there are a number of different optimization methods proposed in the literature, of which a significant fraction are heuristics or metaheuristics [5,6], many inspired by biological problems [7–11]. Most of these approaches rely on a large number of evaluations of the objectives, which are then interpreted as a fitness or cost function assigned to an individual, later subjected to some biologically inspired selection and evolution criteria. Although efficient, these methods need to probe the values of the objectives several times, which becomes prohibitive if the objective functions are very costly, as the result of some time consuming experiment or of some numerically expensive simulation [3].

In order to optimize this class of continuous real-valued costly functions, Bayesian optimization methods were proposed as an alternative for the existing ones [12–16]. Substantiated by the Bayes theorem, they present two major advantages over other conventional methods [2,17]: *(i)* they are highly efficient regarding the number of objective function evaluations; *(ii)* they do not require any analytical knowledge of the objectives, allowing the methods to perform well with black-box functions. Also, Bayesian methods work well even when the objective functions are multimodal or non-convex [17,18].

For expensive objective functions, the simulation bottleneck lies on the number of evaluations of the objectives required by optimization algorithms, in which case one looks for a method that needs fewer iterations [19]. A popular approach consists of building a model of the objectives, which are evaluated as few as possible, and through successively minimizing a loss function built on top of such model, obtain an approximation to the Pareto front. The choice of the loss function is where most existing Bayesian algorithms differ [20–27]. The reduced number of evaluations of the objectives lead to undersampled solutions of the optimization problem, which is observed in current implementations of Bayesian algorithms [28,29]. The method proposed in this paper addresses this problem by taking advantage of well established multi-objective optimization methods that are known to obtain qualitatively good Pareto fronts [30].

1.1. Mathematical description

An optimization problem can be represented by a pair $(\Omega, \vec{f})$ [31], where $\Omega \subseteq \mathbb{R}^n$ is the search space, and n is its dimension. $\vec{f}$ is a m- dimensional real valued vector function of Ω, such as:

$$\vec{f} : \Omega \mapsto \mathbb{Y}, \tag{1}$$

where $\mathbb{Y} \subseteq \mathbb{R}^m$ is the objective space. The n components of $\vec{x} \in \Omega$ are also called *design variables*. To solve the multi-objective optimization problem means finding the set of values $\{\vec{x}^\star\} \subset \Omega$, called the Pareto Set, which elements satisfy the condition:

$$\vec{x}^\star = \underset{\text{s.t. } \vec{x}\in\Omega}{\operatorname{argmax}} \left(f_1(\vec{x}), f_2(\vec{x}), \ldots, f_m(\vec{x})\right) \tag{2}$$

However, for most typical problems, there is no $\vec{x} \in \Omega$ that simultaneously maximize (or minimize) all the m objective functions, as these are *conflicting*.

Two objective functions $f_i(\vec{x})$ and $f_j(\vec{x})$ are said to be conflicting if the values of $\vec{x}$ that lead to the maximum value of f_i are different from the values that maximize f_j, in a way that these two objectives cannot be maximized with the same value of $\vec{x}$. In this context, the Pareto set $\{\vec{x}^\star\}$ represents a compromise between the conflicting objectives, which is represented through the concept of dominance, defined as follows:

$\vec{a} \succ \vec{b}$ ($\vec{a}$ dominates $\vec{b}$) if $\vec{f}(\vec{a}) \geq \vec{f}(\vec{b})$, where $f_i(\vec{a}) > f_i(\vec{b})$ for at least one component i of $\vec{f}$.

The comparisons between vectors are made elementwise, i.e., for $\vec{a} \in \mathbb{R}^m$ and $\vec{b} \in \mathbb{R}^m$, we say that $\vec{a} > \vec{b}$ if and only if $a_\mu > b_\mu$ for all $\mu \in \{1, 2, 3, \ldots, m\}$. Therefore, the Pareto Set is given by:

$$\{\vec{x}^\star\} = \{\vec{x}^* \in \Omega \mid \vec{x}^* \succ \vec{x},\ \forall \vec{x} \in \Omega\}, \tag{3}$$

which represents the subset of search space that contains only non-dominated points. The image of $\{\vec{x}^\star\}$ in objective space is called the Pareto front of the problem and it is represented by the set $\mathcal{F}$ [3]:

$$\mathcal{F} = \{\vec{f}(\vec{x}) \mid \vec{x} \in \{\vec{x}^\star\}\} \tag{4}$$

To generalize the optimization problem, one can impose further restrictions on the search space on the form of N_g inequality constraints [32], which can be arbitrarily represented by:

$$g_k(\vec{x}) \leq 0 \qquad k = 1, \ldots, N_g \tag{5}$$

where $g_k(\vec{x})$ are functions on the search space. Regions of Ω in which the constraints are satisfied are called *valid*. When there are constraints in the problem, the Pareto set definition must be updated:

$$\{\vec{x}^\star\} = \{\vec{x}^* \in \Omega \mid \vec{x}^* \succ \vec{x},\ \forall \vec{x} \in \Omega \ \wedge\ g_k(\vec{x}) \leq 0,\ k = 1, \ldots, N_g\}. \tag{6}$$

2. Software description

A general description of the algorithm behind the Bayesian method is given in Section 2.1. Following, details of the current implementation are provided in Section 2.2.

2.1. Software architecture

The Bayesian method is built upon the idea of constructing models for the objective functions, so that the models can be optimized instead of the objectives themselves [2]. In this way, the total number of objective functions evaluations is expected to be reduced to the minimal necessary to build the model accurately in the vicinity of the Pareto front of the problem. A schematic description of the algorithm proposed is depicted in Fig. 1, where each major logic step of it is labeled by a number from 1 to 8.

In order to build the model of the objectives it is necessary to sample them at different regions of the search space, preferably we would like the samples to be as close as possible to the Pareto set of the problem, so as to increase the accuracy of the model in this region and consequently reduce the total number of functions evaluations. However, in the first steps of the algorithm, as nothing is yet known about the Pareto set, it is necessary to evaluate the objectives N_{init} times, at randomly selected points. The information about our observations of each objective $f_i(\vec{x})$ is stored in the set $\mathcal{D}_i(t)$, defined as:

$$\mathcal{D}_i(t) = \{\vec{x}^{(1)}, f_i(\vec{x}^{(1)});\ \vec{x}^{(2)}, f_i(\vec{x}^{(2)});\ \ldots;\ \vec{x}^{(t)}, f_i(\vec{x}^{(t)})\}, \tag{7}$$

which is just a collection of pairs of points in the search space $\vec{x}^{(t)}$ and its image of the ith objective function $f_i(\vec{x}^{(t)})$. In Eq. (7), t stands for the number of points sampled so far in the algorithm, such that, at the start of the method $t = N_{\text{init}}$. This is the first step represented in Fig. 1.

Given $\mathcal{D}_i(t)$, it is possible to construct a model $\eta_i(\vec{x})$ for the ith objective function $f_i(\vec{x})$:

$$\eta_i(\vec{x};\, t) = \mathcal{G}\left[\mathcal{D}_i(t);\ C_{3/2}\right](\vec{x}) - \zeta \sum_{k=1}^{N_g} \Theta(g_k(\vec{x})), \tag{8}$$

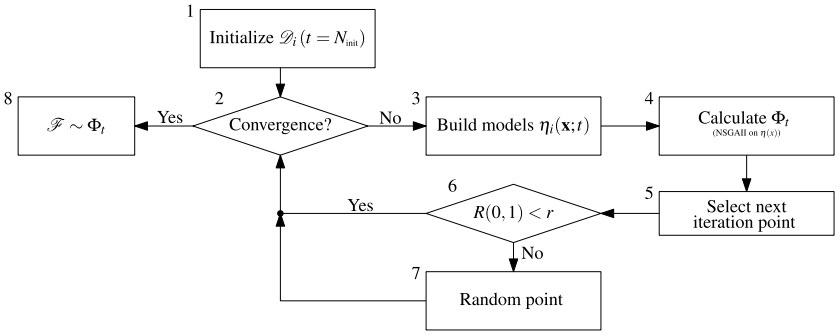

Fig. 1. Schematic representation of optimization algorithm; each step is labeled by a number from 1 to 8 for reference.

where ζ is a penalty constant factor, $\Theta(\cdot)$ is the Heaviside step function and $g_k(\vec{x})$ are the constraints defined in Eq. (5); $\mathcal{G}[\mathcal{D}; K]$ is a *Gaussian Process* (GP), where the first argument was defined in Eq. (7), and the second argument (K) is the *kernel* or covariance function used in the GP. Observe that in the valid region, the penalty term vanishes and $\eta_i \approx f_i$. Also, as no model is built for $g_k(\vec{x})$, expensive constraints are prohibitive, which is a limitation of the method. The construction of the model corresponds to the third step of the algorithm, as depicted in Fig. 1.

Gaussian Processes are distributions over functions [33], completely specified by their mean and covariance function [2]. They can be understood as a random function that, for each given value $\vec{x}$, returns the mean and variance of a Gaussian distribution that better describes $f_i(\vec{x})$, given our knowledge of f_i contained in $\mathcal{D}_i$, and our estimate of how these observations correlate to each other represented by the kernel.

In this work we used the Matérn covariance function $C_\nu(\vec{x}, \vec{x}')$ [33,34], given by:

$$C_\nu(\vec{x}, \vec{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{\ell} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu} r}{\ell} \right), \tag{9}$$

where ℓ is a typical length scale, $\Gamma(\cdot)$ is the Gamma function, $K_\nu(\cdot)$ is the modified Bessel function of the second kind, $r = \|\vec{x} - \vec{x}'\|$ is the distance between the two arguments of the kernel and ν is a positive parameter, such that the process $\eta(\vec{x})$ is k-times mean-square differentiable if and only if $\nu > k$ [33]. In this work, we adopted $\nu = 3/2$, so we require only once differentiability in our models for the objective functions.

Pareto front approximation. After t observations of the objective functions, a model $\eta_i(\vec{x}; t)$ is obtained by applying Eq. (8), which can then be used to calculate an approximation to the Pareto front of the objectives ($\mathcal{F}$). The Pareto front approximation (PFA) at time t is denoted Φ_t, and the Pareto set approximation (PSA) that generates Φ_t is denoted χ_t, i.e., $\vec{\xi}_\iota^{\,(t)} \in \chi_t \subset \Omega \iff \vec{f}(\vec{\xi}_\iota^{\,(t)}) \in \Phi_t$. The PFA can be obtained by employing a well established multi-objective optimization algorithm to the models η_i, which are much faster to evaluate than the actual objectives. If the models are accurate enough in the vicinity of the Pareto front of the problem, then the PFA is a good approximation of the Pareto front, i.e., $\Phi_t \approx \mathcal{F}$. In this work, we selected the non-dominated sorting genetic algorithm II (NSGAII) [30] to find Φ_t. This is step 4 in Fig. 1. It is important to note that the NSGA-II is implemented in the method without any particular adaptation, from its perspective the approximate objectives $\eta_i(\vec{x}; t)$ are optimized as if they were the actual objectives of the optimization problem being solved.

The method works iteratively, meaning that it needs a rule for selecting the next point in search space. To improve the quality of the PFA, i.e., to reduce the difference between Φ_t and $\mathcal{F}$, it is of interest to select points close to $\{\vec{x}^\star\}$ in the search space, in such a way as to make the models $\eta_i(\vec{x})$ more representative of their respective objective functions $f_i(\vec{x})$ in the vicinity of $\mathcal{F}$. However, at the same time it is important to evaluate undersampled regions of the search space Ω so as to guarantee that no region of $\mathcal{F}$ is overlooked and the algorithm gets trapped with an under-representation of the Pareto front. These two contradicting approaches – sampling closer to $\{\vec{x}^\star\}$ or sampling Ω uniformly – represent the trade-off between *exploitation* and *exploration*.

In order to pursue exploitation, we assume that our current PFA is a good approximation to the true Pareto front at current time, then the corresponding PSA approximates the actual Pareto set with the information available at time t, i.e., $\chi_t \approx \{\vec{x}^\star\}$, therefore points taken from χ_t are good candidates to be the next iteration point in the algorithm, i.e., $\vec{x}^{\,(t+1)} = \vec{\xi}_{\iota\text{next}}^{\,(t)} \in \chi_t$, where:

$$\iota\text{next} = \underset{\iota \in 1,\ldots,N}{\text{argmax}} \left[q \left(\frac{d_{\iota,f} - \mu_f}{\sigma_f} \right) + (1-q) \left(\frac{d_{\iota,x} - \mu_x}{\sigma_x} \right) \right]. \tag{10}$$

In Eq. (10), $d_{\iota,x}$ corresponds to the least distance from the ιth point in χ_t to all other t points $\{\vec{x}^{\,(1)}, \ldots \vec{x}^{\,(t)}\}$ probed by the algorithm; μ_x is the average of all $d_{\iota,x}$ and σ_x their standard deviation. The same quantities are calculated in objective space and are represented by the index f. The method selects as the next iteration point the one that belongs to χ_t and is the farthest away from all previously observed points. This analysis can be made at the search or the objective space, the parameter q in Eq. (10) quantifies that choice, if $q = 1$ or $q = 0$ only distances in objective space or search space are considered, respectively. Intermediate values of q are also accepted and weight both spaces accordingly. This is step 9 in Fig. 1.

At every step, with probability given by the parameter $r \in [0, 1]$, a randomly selected component of $\vec{x}^{\,(t+1)}$ is replaced by $\mathcal{R}(x_{\mu,\min}, x_{\mu,\max})$, where $\mathcal{R}(a, b)$ is a uniform random number in the interval $[a, b)$ and $x_{\mu,\min}$ and $x_{\mu,\max}$ are the limits to the μ-th component of $\vec{x}$. This is step 6 in Fig. 1, and is implemented to increase the exploration of the search space by the method. The iteration stops when the set number of iterations was achieved.

2.2. Software functionalities

The Multi-Objective Bayesian optimization algorithm is implemented as a Python class in the MOBOpt package. Its usage is centered around the `MOBayesianOpt` class, which can be instantiated as:

```
import mobopt as mo

Optimizer = mo.MOBayesianopt(target=objective, NObj=n_obj,
                             pbounds=bounds)
```

Where `target` is the function to be optimized, `NObj` is the number of objective functions, `pbounds` is a numpy array [35]

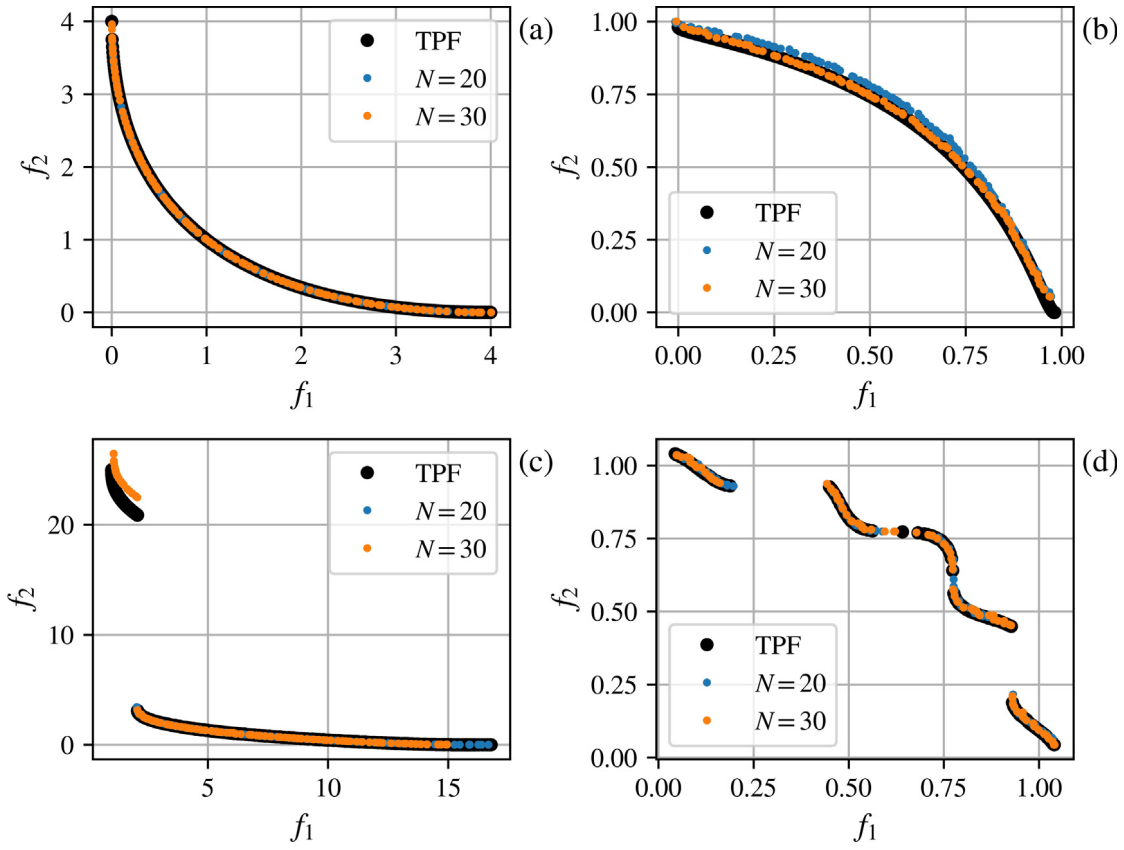

Fig. 2. Pareto fronts found by Bayesian method for selected benchmark functions (a) Schaffer, (b) Fonseca, (c) Poloni, and (d) Tanaka. TPF stands for True Pareto front.

specifying the bounds of the variables in search space. The function `objective` to be passed to the optimizer must be declared as:

```
def objective(x):
    ...
    return numpy.array(f_{1}(x), f_{2}(x), ..., f_{n_obj}(x))
```

The initialization of the algorithm is called by the method `initialize`:

```
Optimizer.initialize(init_points=N_init)
```

where the parameter `init_points` corresponds to the parameter N_{init}, explained in Section 2.1. Finally, the main method of the class is called by:

```
front, pop = Optimizer.maximize(n_iter=N_iter)
```

where `n_iter` corresponds to the number of iteration of the method; the outputs `front` and `pop` correspond to the Pareto front and Pareto Set of the problem, respectively. Optional arguments for the methods, as well as other methods of the class are explained in details in the repository wiki page. The implementation of the algorithm was based on available open source packages [34–37].

3. Illustrative examples

To assess the efficiency of the proposed algorithm, it was tested on a set of benchmark functions [30]. These were chosen because they are well established in the field of optimization, and they have distinct characteristics from each other: their Pareto fronts have different topologies, which allows us to determine the best convergence scenarios; their search space have different dimensionalities; and their constraint conditions are different. For all these benchmark functions the True Pareto front is known, allowing to access the convergence of the method.

Four different benchmarks were used as an example of Pareto front convergence, namely: Schaffer's, Fonseca and Flemming's, Poloni's and Tanaka's. Also, the 2-D ZDT1, from Zitzler, Deb and Thiele's [38], function was used to infer parameter dependency of the method. All of the benchmarks were taken from reference [30] and are minimization problems, they are detailed in Table 1. Tanaka's benchmark was chosen as an example of constrained problem, and it is subjected to the following constraint:

$$g_1(x) = -x_1^2 - x_2^2 + 1 + 0.1\cos\left(16\arctan\left(\frac{x_1}{x_2}\right)\right) \leq 0 \tag{11}$$

$$g_2(x) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5 \tag{12}$$

In Fig. 2 the Pareto front approximations obtained for each of the benchmark functions are represented for different number of function evaluations (denoted N). In all figures the black dots correspond to the True Pareto front of the problem. It can be seen that even for as few as 20 objective function evaluation, for the Schaffer benchmark (Fig. 2(a)), the algorithm is able to find accurate approximations for the Pareto fronts. Even for the most complicated functions, such as the Poloni benchmark with a discontinuous Pareto front, with few evaluations the method converges to a satisfactory solution. Also, the imposition of constraint does not limit the efficiency of the method, as can be seen in Fig. 2(d).

In order to provide further quantification of the convergence of the code, there are several metrics that can be used, one of the

Table 1
Description about benchmark functions.

Name	n	Bounds	Functions
ZDT1	2	$[0,\ 1]$	$f_1(x) = x_1$ $f_2(x) = g(x)\left[1 - \sqrt{\frac{x_1}{g(x)}}\right]$ $g(x) = 1 + 9\frac{\sum_{i=2}^{n} x_i}{n-1}$
Schaffer	1	$[-10^3,\ 10^3]$	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$
Fonseca	3	$[-4,\ 4]$	$f_1(x) = 1 - \exp\left(-\sum_{i=1}^{3}\left(x_i - \frac{1}{\sqrt{3}}\right)^2\right)$ $f_2(x) = 1 - \exp\left(-\sum_{i=1}^{3}\left(x_i + \frac{1}{\sqrt{3}}\right)^2\right)$
Poloni	2	$[-\pi,\ \pi]$	$f_1(x) = \left[1 + (A_1 - B_1)^2 + (A_2 - B_2)^2\right]$ $f_2(x) = \left[(x_1+3)^2 + (x_2+1)^2\right]$ $A_1 = 0.5\sin 1 - 2\cos 2 + \sin 2 - 1.5\cos 2$ $A_2 = 1.5\sin 1 - \cos 1 + 2\sin 2 - 0.5\cos 2$ $B_1 = 0.5\sin x_1 - 2\cos x_1 + \sin x_2 - 1.5\cos x_2$ $B_2 = 1.5\sin x_1 - \cos x_1 + 2\sin x_2 - 0.5\cos x_2$
Tanaka	2	$[0,\ \pi]$	$f_1(x) = x_1$ $f_2(x) = x_2$

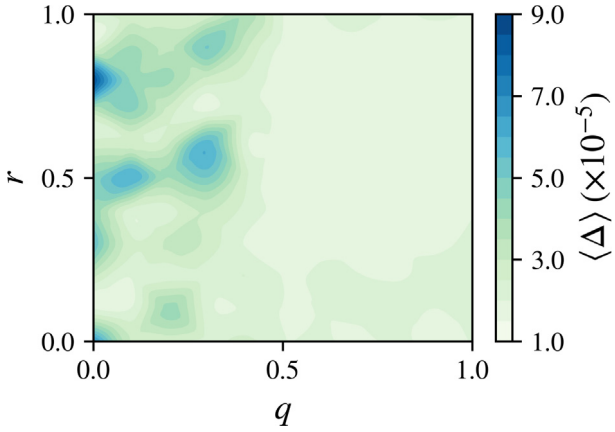

Fig. 3. Average generational distance $\langle\Delta\rangle$ as a function of q and r for the ZDT1 benchmark function. Smaller values are better.

simplest is the generational distance (Δ) defined as:

$$\Delta(\Phi, \mathcal{F}) = \frac{\sqrt{\sum_{i=1}^{|\Phi|} d_i^2}}{|\Phi|}, \tag{13}$$

where $d_i = \min_{\vec{f}\in\mathcal{F}} |F(\vec{\phi}_i) - F(\vec{f})|$ is the smallest distance from $\vec{\phi}_i \in \Phi$ to the closest solution in $\mathcal{F}$ [39], where Φ is the optimal solution set found by the algorithm, and $\mathcal{F}$ is the True Pareto front of the problem.

In Fig. 3, Δ is represented as a function of the method parameters q and r, in order to provide some general guide for the parameter choice, which can be problem dependent. Fig. 3 was calculated with 10 simulations for the 2-D ZDT1 benchmark, for different values of q and r. For each simulation there were $N_{\text{init}} = 5$ initial points and 50 iterations. Generational distance was calculated and averaged for the 20 last iterations in every simulation.

4. Impact

Optimization problems are not restricted to academic questions, but in fact have applications throughout different fields of science and technology. Even though there are many available methods that address this problem, most of them rely on many objective functions evaluations, which can become prohibitive whenever the objectives are numerically costly or are the outcome of experiments. In these situations, it is desirable to obtain an approximation to the Pareto front with as few evaluations of the objectives as possible.

The current software, by using a Bayesian algorithm, is able to calculate the Pareto front of the objectives efficiently in terms of objective functions evaluations. It does so through a simple user interface, written in Python, which allows it to be easily interfaced with other programming languages through the use of numpy arrays [35]. The user interface is straightforward and easy of use, structured around one simple class and few control parameters.

Also, the software provides the user with many metrics so that convergence quantification can be assessed. In cases of longer simulations, there are methods implemented that allow the user to easily save the status of the simulation so that it can be resumed without the need of reinitialization.

This Bayesian method was tested on a more complicated and realistic problem, a thermal-hydraulic plate heat exchanger (HE) [40], where each simulation takes around $\sim$ 15 min on a Intel(R) Core(TM) i5-8265U CPU with 1.60 GHz and 8 GB of RAM. Other applications are, for example, optimizing refrigerator evaporators or hyperparameters of machine learning techniques, problems prohibitive for traditional multi-objective optimization algorithms requiring thousands of objectives evaluations.

5. Conclusions

A multi-objective Bayesian optimization software was outlined. The software is able to calculate the Pareto front of optimization problems with fewer objective functions evaluations than most currently available optimization algorithms. The implementation is made in Python, so that it is easy of use and can be interfaced with many other programming languages. A series of benchmark functions were tested and the software was able to find reliable Pareto front approximations with only a few evaluations of the objectives.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank the National Council of Scientific and Technological Development of Brazil — CNPq (Grants number: 307958/2019-1-PQ, 307966/2019-4-PQ, 404659/2016-0-Univ, 405101/2016-3-Univ), PRONEX "Fundação Araucária, Brazil" 042/2018. Furthermore, the authors wish to thank the Editor and anonymous referees for their constructive comments and recommendations, which have significantly improved the presentation of this paper.

References

[1] Bianchi L, Dorigo M, Gambardella LM, Gutjahr WJ. A survey on metaheuristics for stochastic combinatorial optimization. Nat Comput 2009;8(2):239–87. http://dx.doi.org/10.1007/s11047-008-9098-4.

[2] Brochu E, Cora VM, de Freitas N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010, arXiv:1012.2599.

[3] Hernández-Lobato D, Hernandez-Lobato J, Shah A, Adams R. Predictive entropy search for multi-objective bayesian optimization, In: International conference on machine learning; 2016. p. 1492–1501.

[4] Tsoulos IG, Stavrou V, Mastorakis NE, Tsalikakis D. GenConstraint: A programming tool for constraint optimization problems. SoftwareX 2019;10:100355. http://dx.doi.org/10.1016/j.softx.2019.100355.

[5] Silvestre D. OPTool—An optimization toolbox for iterative algorithms. SoftwareX 2020;11:100371. http://dx.doi.org/10.1016/j.softx.2019.100371.

[6] Ayala HVH, Keller P, de Fátima Morais M, Mariani VC, Coelho LS, Rao RV. Design of heat exchangers using a novel multiobjective free search differential evolution paradigm. Appl Therm Eng 2016;94:170–7. http://dx.doi.org/10.1016/j.applthermaleng.2015.10.066.

[7] Coelho LS, Mariani VC, Guerra FA, da Luz MVF, Leite JV. Multiobjective optimization of transformer design using a chaotic evolutionary approach. IEEE Trans Magn 2014;50(2):669–72. http://dx.doi.org/10.1109/TMAG.2013.2285704.

[8] Coelho LS, Mariani VC, Ferreira da Luz MV, Leite JV. Novel gamma differential evolution approach for multiobjective transformer design optimization. IEEE Trans Magn 2013;49(5):2121–4. http://dx.doi.org/10.1109/TMAG.2013.2243134.

[9] Neto JXV, Junior EJG, Moreno SR, Ayala HVH, Mariani VC, Coelho LS. Wind turbine blade geometry design based on multi-objective optimization using metaheuristics. Energy 2018;162:645–58. http://dx.doi.org/10.1016/j.energy.2018.07.186.

[10] de Vasconcelos Segundo EH, Mariani VC, Coelho LS. Design of heat exchangers using falcon optimization algorithm. Appl Therm Eng 2019;156:119–44.

[11] Gavidia-Calderon C, Beltrán Castañon C. Isula: A java framework for ant colony algorithms. SoftwareX 2020;11:100400. http://dx.doi.org/10.1016/j.softx.2020.100400.

[12] Kushner HJ. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. J Basic Eng 1964;86(1):97–106. http://dx.doi.org/10.1115/1.3653121.

[13] Mockus J. On Bayesian methods of optimization. In: Towards global optimization. North-Holland Amsterdam; 1975, p. 166–81.

[14] Mockus J. Bayesian approach to global optimization: theory and applications, vol. 37. Springer Science & Business Media; 2012.

[15] Mockus J, Tiesis V, Zilinskas A. The application of bayesian methods for seeking the extremum. In: Towards global optimization, vol. 2, no. 117–129. 1978, p. 2.

[16] Jones DR, Schonlau M, Welch WJ. Efficient global optimization of expensive black-box functions. J Global Optim 1998;13(4):455–92.

[17] Snoek J, Larochelle H, Adams RP. Practical Bayesian optimization of machine learning algorithms. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. Advances in neural information processing systems, vol. 25. Curran Associates, Inc.; 2012, p. 2951–9.

[18] Tajbakhsh SD. A fully Bayesian approach to the efficient global optimization algorithm [Ph.D. thesis], Pennsylvania, USA: The Pennsylvania State University; 2012.

[19] Ponweiser W, Wagner T, Biermann D, Vincze M. Multiobjective optimization on a limited budget of evaluations using model-assisted s-metric selection. In: Rudolph G, Jansen T, Beume N, Lucas S, Poloni C, editors. Parallel problem solving from nature – PPSN X. Berlin, Heidelberg: Springer; 2008, p. 784–94.

[20] Picheny V. Multiobjective optimization using Gaussian process emulators via stepwise uncertainty reduction. Stat Comput 2015;25(6):1265–80. http://dx.doi.org/10.1007/s11222-014-9477-x.

[21] Emmerich MTM, Giannakoglou KC, Naujoks B. Single- and multiobjective evolutionary optimization assisted by Gaussian random field metamodels. IEEE Trans Evol Comput 2006;10(4):421–39.

[22] Emmerich M, Klinkenberg J-w. The computation of the expected improvement in dominated hypervolume of Pareto front approximations, Rapport technique, vol. 34. Leiden University; 2008.

[23] Wagner T, Emmerich M, Deutz A, Ponweiser W. On expected-improvement criteria for model-based multi-objective optimization. In: International conference on parallel problem solving from nature. Springer; 2010, p. 718–27.

[24] Oyama A, Shimoyama K, Fujii K. New constraint-handling method for multi-objective and multi-constraint evolutionary optimization. Trans Japan Soc Aeronaut Space Sci 2007;50(167):56–62.

[25] Keane AJ. Statistical improvement criteria for use in multiobjective design optimization. AIAA J 2006;44(4):879–91. http://dx.doi.org/10.2514/1.16875.

[26] Knowles J. ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. IEEE Trans Evol Comput 2006;10(1):50–66.

[27] Knowles J, Hughes EJ. Multiobjective optimization on a budget of 250 evaluations. In: Coello Coello CA, Hernández Aguirre A, Zitzler E, editors. Evolutionary multi-criterion optimization. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005, p. 176–90.

[28] Knudde N, van der Herten J, Dhaene T, Couckuyt I. GPflowOpt: A Bayesian optimization library using tensorflow. 2017, arXiv preprint – URL arXiv:1711.03845.

[29] Pandita P, Bilionis I, Panchal J, Gautham BP, Joshi A, Zagade P. Stochastic multiobjective optimization on a budget: Application to multipass wire drawing with quantified uncertainties. Int J Uncertain Quantif 2018;8(3):233–49.

[30] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. In: IEEE transactions on evolutionary computation, vol. 6, no. 2. 2002.

[31] Larrañaga P, Lozano J. Estimation of distribution algorithms: A new tool for evolutionary computation. Genetic algorithms and evolutionary computation, Springer US; 2001.

[32] Hestenes MR. Multiplier and gradient methods. J Optim Theory Appl 1969;4(5):303–20.

[33] Rasmussen CE, Williams CKI. Gaussian processes for machine learning. Adaptive computation and machine learning, The MIT Press; 2005.

[34] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.

[35] Virtanen P, Gommers R, Oliphant TE, et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods 2020;17(3):261–72.

[36] Nogueira F. Bayesian optimization: Open source constrained global optimization tool for Python. 2014, URL https://github.com/fmfn/BayesianOptimization.

[37] Fortin F-A, De Rainville F-M, Gardner M-A, Parizeau M, Gagné C. DEAP: Evolutionary algorithms made easy. J Mach Learn Res 2012;13:2171–5.

[38] Zitzler E, Deb K, Thiele L. Comparison of multiobjective evolutionary algorithms: Empirical results. Evol Comput 2000;8(2):173–95.

[39] Jiang S, Ong Y, Zhang J, Feng L. Consistencies and contradictions of performance metrics in multiobjective optimization. IEEE Trans Cybern 2014;44(12):2391–404.

[40] Galuzio PP, Segundo EHV, Coelho LS, Freire RZ, Mariani V. Thermal-hydraulic optimization of plate heat exchanger applying a Bayesian multi-objective, In: 25th International congress of mechanical engineering; 2019. p. 1–10.