mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions

Bernd Bischl[a], Jakob Richter[b], Jakob Bossek[c], Daniel Horn[b], Janek Thomas[a], Michel Lang[b]

[a] *Ludwig-Maximilians-Universität München, Germany*
[b] *TU Dortmund University, Germany*
[c] *Westfälische-Wilhelms Universität Münster, Germany*

Abstract

We present `mlrMBO`, a flexible and comprehensive `R` toolbox for model-based optimization (MBO), also known as Bayesian optimization, which addresses the problem of expensive black-box optimization by approximating the given objective function through a surrogate regression model. It is designed for both single- and multi-objective optimization with mixed continuous, categorical and conditional parameters. Additional features include multi-point batch proposal, parallelization, visualization, logging and error-handling. `mlrMBO` is implemented in a modular fashion, such that single components can be easily replaced or adapted by the user for specific use cases, e.g., any regression learner from the `mlr` toolbox for machine learning can be used, and infill criteria and infill optimizers are easily exchangeable. We empirically demonstrate that `mlrMBO` provides state-of-the-art performance by comparing it on different benchmark scenarios against a wide range of other optimizers, including DiceOptim, rBayesianOptimization, SPOT, SMAC, Spearmint, and Hyperopt.

Keywords: Model-Based Optimization, Bayesian Optimization, Black-Box Optimization, Hyperparameter Tuning, Parameter Configuration, R

1. Introduction

Black-box functions are systems that require a number of input parameters to produce one or multiple (numeric) outputs. In most cases these are (a) expensive to evaluate in terms of time and/or monetary cost, and (b) knowledge of their internal working is not available, which often manifests through the absence of derivatives. Such problems occur in production engineering [e.g. 1], where the inputs are possible settings of industrial machines or used materials

Email addresses: `bernd_bischl@gmx.net` (Bernd Bischl), `jakob.richter@tu-dortmund.de` (Jakob Richter), `bossek@wi.uni-muenster.de` (Jakob Bossek), `daniel.horn@tu-dortmund.de` (Daniel Horn), `janek.thomas@stat.uni-muenchen.de` (Janek Thomas), `lang@statistik.tu-dortmund.de` (Michel Lang)

and the output is one or multiple measurements regarding the quality of fabricated parts. Since this makes a single evaluation expensive, one tries to find the optimal settings of production steps in a minimal number of tries. Design of Computer Experiments (DACE) [2] is a discipline focused on solving such problems and sequential model-based optimization (SMBO) [3] has become the state-of-the-art optimization strategy in recent years.

The generic SMBO procedure starts with an initial design of evaluation points, and then iterates the following steps:

1. Fit a regression model to the outcomes and design points obtained so far,
2. query the model to propose a new, promising point, often by optimizing a so-called infill criterion or acquisition function,
3. evaluate the new point with the black-box function and add it to the design.

Several adaptations and extensions, e.g., multi-objective optimization [4], multi-point proposal [5, 6], more flexible regression models [7] or alternative ways to calculate the infill criterion [8] have been investigated recently.

A different field of application for SMBO is the hyperparameter optimization for machine learning methods [e.g. 9, 10, 11]. Here, the black-box is a machine learning method and the objective(s) is one or multiple performance measure(s), validated via resampling on a data set of interest. The black-box function can be more complex, for example a machine learning pipeline which includes preprocessing, feature selection and model selection.

After a brief comparison with related software in Subsection 1.1 and clarification of our main contributions in Subsection 1.2, we introduce the general SMBO procedure in more detail in Section 2. Section 3 highlights the capabilities of our software `mlrMBO`, showcased by some code examples. In Section 4 we empirically demonstrate that `mlrMBO` achieves state-of-the-art performance on a wide range of synthetic and real-world single- and multi-objective scenarios. Section 5 gives an outlook on future work.

1.1. Related Software

We will briefly present an overview of available software for model-based optimization, starting with implementations based on the Efficient Global Optimization algorithm (EGO), i.e., the SMBO algorithm proposed by Jones et al. [3] using Gaussian processes (GPs), and continue with extensions and alternative approaches.

Both `DiceOptim` [12] and `rBayesianOptimization` [13] are `R` packages that offer EGO implementations. A sophisticated EGO implementation can be found in the Python package `Spearmint` [14]. It focuses on hyperparameter optimization of machine learning algorithms with enhancements regarding variable costs of experiments and parallelization. All three packages offer different GP kernels and infill criteria, but only support numerical (non-conditional) parameters

and, except for Spearmint, no multi-criteria optimization or parallelization is available. A multi-criteria version of Spearmint is introduced in [15].

The C++ library `BayesOpt` [16] contains an extended version of EGO, including Student-t processes, support of mixed and conditional parameters as well as meta-criteria algorithms to automatically find reasonable infill criteria during optimization. It offers interfaces for Python, Matlab and Octave.

`SMAC` [7] is one of the most established frameworks and allows to optimize mixed parameter spaces as it uses a random forest instead of a GP for regression. Besides general black-box optimization, it is focused on algorithm configuration. However, `SMAC` is limited to single-criteria optimization and parallelization is not supported.

`Hyperopt` [8] is an optimization package in Python that supports numerical, categorical and conditional parameters. Instead of a regression it uses a tree of Parzen estimators (TPE) to compute point suggestions. It supports distributed parallel and asynchronous execution. `Hyperopt` can be used for general black-box optimization, but is mainly focused on machine learning tasks.

Another `R` implementation for sequential black-box optimization is `SPOT` [17]. It is a toolbox with different modeling techniques and offers a wide variety of statistical methods. `SPOT` contains sophisticated algorithms to handle functions with noisy evaluations, is able to handle constrains in functions and supports multi-objective optimization.

1.2. Main Contributions and Prior Applications

The main contribution of this paper is the presentation of the `R` package `mlrMBO`, which implements a generic SMBO framework and provides a large variety of different SMBO methods due to its modular structure. `mlrMBO` is even more flexible than SPOT in its choice of surrogate models as it is connected to the `R` package `mlr` [18] which interfaces more than 60 machine learning regression algorithms. Besides the default SMBO procedure, `mlrMBO` focuses on three domains: Mixed parameter space optimization, multi-point proposals and multi-objective optimization. Even combinations of the three domains are possible, which to our knowledge no other software is currently capable of. `mlrMBO` is easy to use as many default implementations for the individual steps of the SMBO procedure are directly supported in a plug-and-play style. Simple interfaces are available to extend the package with user specific variants.

Benchmarks show that `mlrMBO` achieves state-of-the-art performance in each domain. Additionally, `mlrMBO` has been successfully applied in some practical settings. In [19, 20] it was used to optimize the hyperparameters of machine learning pipelines (joint pre-processing and model hyperparameters) for support vector machines and general machine learning models, respectively, in a single objective setting. Hess et al. [21] proposed an `mlrMBO` ensemble-based approach to identify the best surrogate model during optimization through reinforcement learning. Horn et al. [22] considered a multi-objective benchmark and optimized the runtime-accuracy trade-offs of several approximate support vector machine solvers. Horn and Bischl [11] introduced the general capability of `mlrMBO` to

solve multi-objective machine learning tasks. Steponavičė et al. [23] investigated the impact of different initial design sampling techniques on the performance of multi-objective model-based optimization methods by using `mlrMBO`.

2. Sequential Model-Based Optimization

This section describes the general SMBO setup and presents the individual building blocks in Subsection 2.1. While SMBO is modular and can thus be customized for a variety of different tasks, we highlight the most prominent combinations of components described in the literature like EGO [3] (Subsection 2.2) or SMAC-like [7] optimizers (Subsection 2.3). Subsections 2.4 and 2.6 introduce parallelization through multi-point proposal, and multi-objective optimization.

2.1. Sequential model-based optimization

Let $f(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}$ be an arbitrary black-box function with a d-dimensional input domain $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_d$ and a deterministic output y. Each $\mathcal{X}_i$ $(i = 1, \cdots, d)$ can be either numeric and bounded (i.e. $\mathcal{X}_i = [l_i, u_i] \subset \mathbb{R}$) or a finite set of s categorical values ($\mathcal{X}_i = \{v_{i1}, \ldots, v_{is}\}$). Without loss of generality, we want to find the input $\boldsymbol{x}^*$ with

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in \mathcal{X}}{\arg\min}\, f(\boldsymbol{x}).$$

In the context of model-based optimization, we usually assume that f is expensive to evaluate, hence the total number of function evaluations is limited by a budget. At the heart of SMBO are so-called surrogate models $\hat{f}$ which cheaply estimate the expensive black-box function f and which are iteratively updated and refined. The general approach is illustrated in Figure 1. The figure outlines the following steps, whereas each step is explained in more detail in the following subsections:

(1) An initial design of n_{init} points $\boldsymbol{x}^{(j)}$ $(j = 1, \ldots, n_{\text{init}})$ is sampled from $\mathcal{X}$ and f is evaluated at these points to yield outcomes $y^{(j)} = f(\boldsymbol{x}^{(j)})$. The tuples $\left(y^{(j)}, \boldsymbol{x}^{(j)}\right)$ constitute the data to build the initial surrogate model $\hat{f}$ in the next step.

(2) Fit a surrogate model to all evaluated points $\boldsymbol{x}^{(j)} \in \mathcal{X}$ and corresponding values $y^{(j)}$.

(3) An *infill criterion* proposes m points $\boldsymbol{x}^{(j+i)}$ $(i = 1, \ldots, m)$. The criterion is defined on $\mathcal{X}$ and operates on the surrogate $\hat{f}$ to determine points which are promising for the optimization. These points should either have a good expected objective value or high potential to improve the quality of the surrogate model.

(4) The proposed points are evaluated using f and the new tuples $\left(y^{(j+i)}, \boldsymbol{x}^{(j+i)}\right)$ are added to the design.

(5) If the budget is not exhausted (and no other termination criteria is met), go to step (2).

(6) If the budget is exhausted or another termination criteria is met, return the proposed solution for the optimization problem.

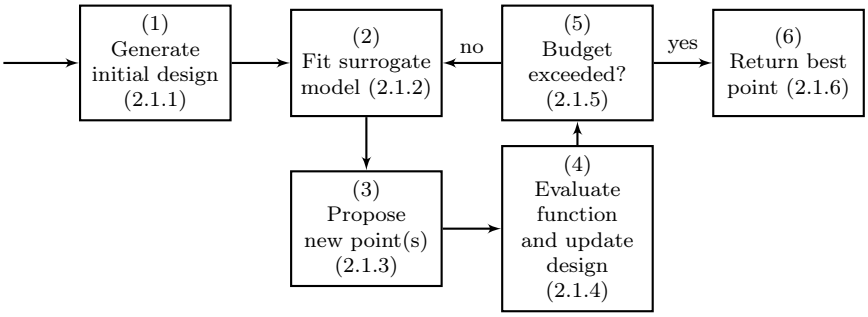

Figure 1: General SMBO approach.

2.1.1. Initial Design

The initial design specifies the points of the input domain at which the black-box function is evaluated to build the initial surrogate model $\hat{f}$. If too few points are chosen or if the points do not cover $\mathcal{X}$ well, the fit of $\hat{f}$ may be poor and thus points proposed based on $\hat{f}$ may be suboptimal for the progress of the optimization. Fitting a surrogate model may even be impossible. On the other hand, a large initial design may reduce the available budget too much. `mlrMBO` provides various options for the initial design: The user can specify it manually or generate designs either completely at random, coarse grid designs or by using space-filling Latin Hypercube Designs [24].

2.1.2. Surrogate Models

One of the main factors that determines the choice of surrogate model $\hat{f}$ is the structure of the input space $\mathcal{X}$. If $\mathcal{X} \subset \mathbb{R}^d$, *Kriging* [3] is the recommended choice and provides state-of-the-art performance. In Section 2.2, the Kriging-based EGO approach is discussed in more detail. If the search space $\mathcal{X}$ also includes categorical parameters on the other hand, *random forests* are a viable alternative [7] as they can handle such parameters directly, without the need to encode the categorical parameters as numeric. `mlrMBO` allows the use of any of the many regression models available in the `R` package `mlr`, which itself can also be easily extended to support custom regression learners [25].

While Kriging models and random forests already provide uncertainty estimation natively, generic bagging can be applied to arbitrary regression models to retrieve standard error estimators in `mlr`.

2.1.3. Infill Criteria

The infill criterion, or sometimes called acquisition function, guides the optimization and tries to trade-off exploitation and exploration. This is usually achieved by combining $\hat{\mu}(\boldsymbol{x})$ and $\hat{s}(\boldsymbol{x})$ (or $\hat{s}^2(\boldsymbol{x})$) in a single formula in a well-balanced fashion, where the posterior mean $\hat{\mu}(\boldsymbol{x})$ and posterior standard deviation $\hat{s}(\boldsymbol{x})$ (or posterior variance $\hat{s}^2(\boldsymbol{x})$) are estimated by the surrogate model $\hat{f}$. $\hat{s}(\boldsymbol{x})$ and $\hat{s}^2(\boldsymbol{x})$ are sometimes also called "local uncertainty estimators". Assuming that our model $\hat{f}$ is somewhat "spatial" in the sense that higher values of $\hat{s}(\boldsymbol{x})$ indicate regions of the search space that few of our design points lie close to and / or we have not learned the structure of f very well at $\boldsymbol{x}$, we are therefore looking for points with low $\hat{\mu}(\boldsymbol{x})$ and high $\hat{s}(\boldsymbol{x})$.

Arguably the most popular choice is the *expected improvement*

$$\mathrm{EI}(\boldsymbol{x}) := \mathrm{E}(I(\boldsymbol{x}))$$

where the random variable $I(\boldsymbol{x})$ defines the potential improvement at $\boldsymbol{x}$ over the currently best observed function value $y_{\min}$:

$$I(\boldsymbol{x}) := \max\left\{y_{\min} - Y(\boldsymbol{x}), 0\right\}.$$

Here, $Y(\boldsymbol{x})$ is a random variable that should express the posterior distribution at $\boldsymbol{x}$, estimated with $\hat{f}$. For a Gaussian process, $Y(\boldsymbol{x})$ is normally distributed with $Y(\boldsymbol{x}) \sim N(\hat{\mu}(\boldsymbol{x}), \hat{s}^2(\boldsymbol{x}))$. Under this assumption, $\mathrm{EI}(\boldsymbol{x})$ can be expressed analytically in closed form as

$$\mathrm{EI}(\boldsymbol{x}) = (y_{\min} - \hat{\mu}(\boldsymbol{x}))\,\Phi\left(\frac{y_{\min} - \hat{\mu}(\boldsymbol{x}))}{\hat{s}(\boldsymbol{x})}\right) + \hat{s}(\boldsymbol{x})\phi\left(\frac{y_{\min} - \hat{\mu}(\boldsymbol{x})}{\hat{s}(\boldsymbol{x})}\right), \quad (1)$$

where Φ and ϕ are the distribution and density function of the standard normal distribution, respectively.

A simpler approach to balance $\hat{\mu}(\boldsymbol{x})$ and $\hat{s}(\boldsymbol{x})$ for a point $\boldsymbol{x}$ is given by the *lower confidence bound*

$$\mathrm{LCB}(\boldsymbol{x}, \lambda) = \hat{\mu}(\boldsymbol{x}) - \lambda\hat{s}(\boldsymbol{x}), \quad (2)$$

where $\lambda > 0$ is a constant that controls the "mean vs. uncertainty" trade-off.

Furthermore, `mlrMBO` currently support pure mean $\hat{\mu}(\boldsymbol{x})$ minimization (pure exploitation) and pure uncertainty $\hat{s}(\boldsymbol{x})$ maximization (pure exploration) and further criteria for multiple point proposals (see Section 2.4), noisy optimization (see Section 2.5), and multi-objective optimization (See section 2.6).

2.1.4. Infill Optimization

The infill optimizer searches for the point $\boldsymbol{x}$ which yields the best infill value. Unlike the original optimization problem on f, the optimization on the infill criterion can be considered inexpensive. While this is still a black-box optimization problem, points can be evaluated more lavishly, and Jones et al. [3] propose a branch and bound algorithm for this task. `mlrMBO` defaults to a more generic approach, which we call *focus search*, outlined in Algorithm 1. It is able to handle

numeric parameter spaces, categorical parameter spaces, as well as mixed and hierarchical spaces. The algorithm starts with a large random design from which all points are evaluated by the surrogate regression model to determine the most promising point. Next, focus search shrinks the search space around the best point and samples new random points for the now focused search space. The shrinkage of search space is iterated n_{iters} times. The complete procedure can be restarted n_{restart} times to avoid local optima. Finally the best point over all restarts and iterations is returned. Evolutionary algorithms like CMA-ES [26] or custom user-defined optimizers can be selected alternatively.

Algorithm 1 Infill Optimization: Focus Search.

Require: infill criterion $c : \mathcal{X} \to \mathbb{R}$, control parameters n_{restart}, n_{iters}, n_{points}

for $u \in \{1, ..., n_{\text{restart}}\}$ **do**
  Set $\tilde{\mathcal{X}} = \mathcal{X}$
  for $v \in \{1, ..., n_{\text{iters}}\}$ **do**
    generate random design $\mathcal{D} \subset \tilde{\mathcal{X}}$ of size n_{points}
    compute $\boldsymbol{x}^*_{u,v} = (x^*_1, ..., x^*_d) = \arg\min_{\boldsymbol{x} \in \mathcal{D}} c(\boldsymbol{x})$
    shrink $\tilde{\mathcal{X}}$ by focusing on $\mathbf{x}^*$:
    for each search space dimension $\tilde{\mathcal{X}}_i$ in $\tilde{\mathcal{X}}$ **do**
      if $\tilde{\mathcal{X}}_i$ numeric: $\tilde{\mathcal{X}}_i = [l_i, u_i]$ **then**
        $l_i = \max\{l_i, x^*_i - \frac{1}{4}(u_i - l_i)\}$
        $u_i = \min\{u_i, x^*_i + \frac{1}{4}(u_i - l_i)\}$
      end if
      if $\tilde{\mathcal{X}}_i$ categorical: $\tilde{\mathcal{X}}_i = \{v_{i1}, \ldots, v_{is}\}$, $s > 2$ **then**
        $\bar{x}_i$ = sample one category uniformly from $\tilde{\mathcal{X}}_i \backslash x^*_i$
        $\tilde{\mathcal{X}}_i = \tilde{\mathcal{X}}_i \backslash \bar{x}_i$
      end if
    end for
  end for
end for
Return $\boldsymbol{x}^* = \underset{u \in \{1,...,n_{\text{restart}}\}, v \in \{1,...,n_{\text{iters}}\}}{\arg\min} c(\boldsymbol{x}^*_{u,v})$

2.1.5. Termination

Multiple termination criteria can be used in `mlrMBO`. Commonly a limit is set for the total number of evaluations of f or for the number of SMBO iterations. Alternatively, the optimization can be terminated after a given time or after a time budget for function evaluations is exhausted. The optimization can also be stopped as soon as a predefined objective value is reached. Furthermore, the user can create custom termination rules.

2.1.6. Final Point

Finally, the final solution $\boldsymbol{x}^*$ has to be determined. Usually the best point observed during the optimization is picked. Fitting a last surrogate model to find the best point predicted is a viable option, especially if f is noisy.

2.2. Efficient Global Optimization (EGO)

Kriging models [27] are arguable the most popular choice for a surrogate model because they are very flexible and provide a local uncertainty estimator [3].

In general, we consider a numeric-only input domain $\mathcal{X} \subset \mathbb{R}^d$. Jones et al. [3] were the first who introduced surrogate models for the sequential optimization of box-constrained functions with real-valued arguments. Their Efficient Global Optimization (EGO) algorithm employs Kriging models together with the expected improvement infill criterion (see Equation 1). Maximizing the EI results in an infill criterion that balances exploitation of the model structure and exploration of regions with high uncertainty and has proven to be highly effective [3]. It can ensure global convergence [28, 29] (which is somewhat unrealistic to expect under the usually tight budget constraints that exist for many expensive black-box optimization problems).

Figure 2 illustrates the point proposal at the 3rd (left) and the 4th iteration (right) of an EGO run on a $1d$ cosine mixture function. It illustrates how high uncertainty ($\hat{s}$) and a low value of $\hat{\mu}$ contribute to the EI and thus to the selection of the next point and the ability of model-based optimization to find the optimum even for multi-modal functions.

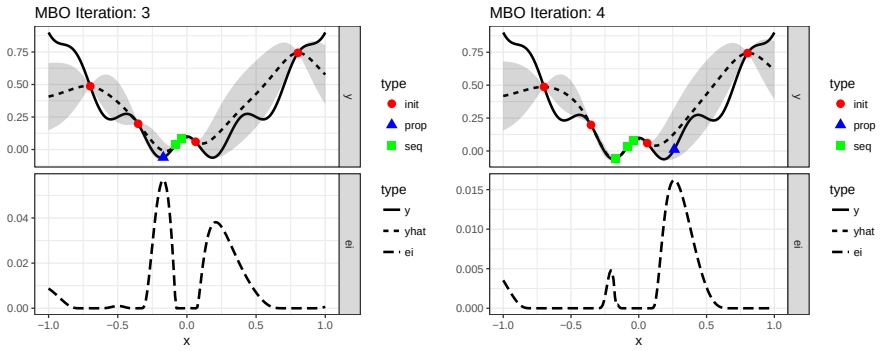

Figure 2: State at the 3rd (left) and 4th iteration (right) of an exemplary EGO run on a $1d$ cosine mixture function. The upper part shows the real function f as a solid line and its estimation $\hat{\mu}$ dotted. The uncertainty is indicated by the shaded area. Initial design points are displayed as red circles, sequential points as green squares. The lower part shows the respective value for the EI. The optimum of the EI defines the point that proposed to be evaluated next (blue triangle).

2.3. Mixed Space Optimization

Real life scenarios often include mixed-valued as well as hierarchical parameter spaces with conditional parameters. An example is the tuning of a support vector machine, for which the parameter space is illustrated in Figure 3. Depending on the choice of the *kernel*, the hyperparameter γ has to be optimized for the *radial* kernel (so it is conditional on the setting of *kernel*), but it is not present (or we could say: active) for the *linear* kernel. In contrast to γ, the

hyperparameter C is unconditionally always active. Kriging is not really suited for such problem domains, since covariance kernels natively supporting those types of data are still subject to research [30].

For the *initial design* all options support categorical parameters as well as hierarchical dependencies (feasible values of a parameter depend on the values of other parameters).

For the *surrogate* we need a regression model that is more flexible and can handle categorical features as well as missing values to support dependent parameters. A slightly modified *random forest* can be used for this purpose. If a hyperparameter is not active in a design point in the training set (due to unfulfilled conditions), we will mark its value as missing. Although the random forest could potentially directly handle missing values, many implementations do not. Hence, we impute these values in the following way: For categorical parameters we code missing values as a new level, and for numerical parameters we code the imputed value out of the range of the box-constraints of the parameter under consideration. This is known as the *separate-class method* and was shown to perform best for decision trees in a prediction-oriented study, when missingness is related to the outcome [31].

In order to still use *infill criteria* as LCB and EI, we also have to compute an uncertainty estimate $\hat{s}(\boldsymbol{x})$ for the random forest. For bagging-like predictors this can be computed or approximated in various ways from the bootstrap. We refer the reader to [32, 33] for further details. In `mlr` the uncertainty estimator can be deviated from an expensive extra bootstrap around the random forest, the jackknife, the infinitesimal jackknife, or a simple estimator which extracts the standard error simply from the internal bootstrap of the random forest. In our experience, the jackknife estimator works most reliably, so it is the current default for `mlrMBO` with random forests as surrogate. However, it should be noted that the random forest is not really a spatial model as a Gaussian process and therefore the properties of the uncertainty estimator are less intuitive in comparison to the ones from Kriging models. Our following results still indicate that we obtained state-of-the-art results with this default, and we deem this aspect a matter for further research.

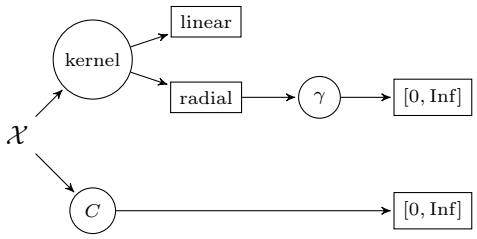

Figure 3: Dependent search space for the tuning of a support vector machine. Circles denote parameters, rectangles denote parameter ranges, arrows denote the hierarchical structure.

2.4. Multi-Point Proposal

The expensive nature of the optimization problem makes parallelization, i.e. the evaluation of different configurations on multiple CPUs, an important extension to speed up the SMBO process. Recently many methods have been proposed to simultaneously propose m points in each iteration. We showcase three methods implemented in `mlrMBO`, which are also discussed in [6]. A straightforward approach is *qLCB* [34], an extension of the LCB criterion. Instead of one fixed λ, multiple λ_k ($k = 1, \ldots, m$) are drawn from an exponential distribution to obtain m points $\boldsymbol{x}^{(j+1)}, \ldots, \boldsymbol{x}^{(j+m)}$:

$$\text{qLCB}(\boldsymbol{x}, \lambda_k) = \hat{\boldsymbol{y}}(\boldsymbol{x}) - \lambda_k \hat{s}(\boldsymbol{x}), \quad \lambda_k \sim \text{Exp}\left(\frac{1}{\lambda}\right).$$

The criterion is than optimized separately for every λ_k, so that overall m points are proposed. Proposals that were obtained by optimizing the qLCB for a low value of λ_k exploit the model and are in proximity of the best found y so far. For high values of λ_k the proposals will be of exploratory nature. This ensures that in one SMBO iteration all proposals balance exploitation and exploration.

Another approach to propose multiple points using the expected improvement is known as *constant liar* [5]. Here we obtain $\boldsymbol{x}^{(j+1)}$ in the same way as for the ordinary EI. To obtain $\boldsymbol{x}^{(j+2)}$ we assume that the evaluation at point $\boldsymbol{x}^{(j+1)}$ is done and update the surrogate model with a made up target value y. Exemplary choices for the made up value are $\min(\boldsymbol{y})$, $\max(\boldsymbol{y})$, the mean $\bar{\boldsymbol{y}}$, or the predicted posterior mean $\hat{\mu}(\boldsymbol{x}^{(j+1)})$ of the surrogate model. The latter approach is also often referred to as *kriging believer*.

Bischl et al. [6] propose the multi-objective infill model-based optimization (MOIMBO) approach. The posterior mean $\hat{\mu}(\boldsymbol{x})$ and variance $\hat{s}(\boldsymbol{x})$ are not scalarized in a single function (as done by EI or (q)LCB), instead a multi-objective optimization strategy (see Section 2.6) is used to optimize them jointly and propose a whole set of optimal points. To ensure that the points are diverse, a distance measure, e.g. the nearest neighbor distance, can be used as a third objective.

2.5. Noisy Optimization

Noisy optimization assumes that the objective function f is stochastic. Usually, one now faces the problem to optimize $\text{E}[f(x)]$ instead of $f(x)$ and common strategies are intelligent repetition strategies [35] or adapted infill criteria. `mlrMBO` currently only offers the latter (but of course the user can always opt to perform averaging in the objective function, e.g. by naively averaging over a constant number of repetitions himself).

A popular infill criterion for noisy functions is the *expected quantile improvement* [36] which is an extension of EI. Instead of looking for an improvement over best value observed so far (the $y_{\min}$ in the EI formula), we exchange this with a so called "plug-in" value $q_{\min}$:

$$\text{EQI}(\boldsymbol{x}) = (q_{\min} - q(\boldsymbol{x}))\, \Phi\left(\frac{q_{\min} - q(\boldsymbol{x})}{s_{q(\boldsymbol{x})}}\right) + s_{q(\boldsymbol{x})} \phi\left(q_{\min} - q(\boldsymbol{x}) s_{q(\boldsymbol{x})}\right), \quad (3)$$

where $q_{\min}$ is the lowest β-quantile $q(\boldsymbol{x}_i)$ for all previously evaluated points $\boldsymbol{x} \in \{\boldsymbol{x}^1, \dots \boldsymbol{x}^n\}$, and β is a user control parameter for the EQI. The estimated β-quantile at point $\boldsymbol{x}$ is given by $q(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \Phi^{-1}(\beta)\hat{s}(\boldsymbol{x})$. This implies that the criterion will be non-zero at already evaluated points allowing re-evaluations or evaluations very close to already evaluated design points to increase knowledge of promising points.

`mlrMBO` offers also the so called "augmented expected improvement" and its modular design makes extensions towards further criteria functions straightforward. For a further in-depth discussion of this topic we refer the reader to [37] and their benchmark for noisy MBO approaches.

2.6. Model-Based Multi-Objective (MBMO) Optimization

Multi-objective optimization problems are characterized by a set of target functions $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \dots, f_k(\boldsymbol{x}))$ which have to be optimized simultaneously. Since there is no total order in $\mathbb{R}^k$, for $k \geq 2$, the concept of *Pareto dominance* is used. A point $\boldsymbol{x}$ pareto-dominates another point $\tilde{\boldsymbol{x}}$, $\boldsymbol{x} \preceq \tilde{\boldsymbol{x}}$, if $f_i(\boldsymbol{x}) \leq f_i(\tilde{\boldsymbol{x}})$ for $i = 1, \dots, k$ and $\exists\, j\, f_j(\boldsymbol{x}) < f_j(\tilde{\boldsymbol{x}})$, i.e., $\boldsymbol{x}$ needs to be as good as $\tilde{\boldsymbol{x}}$ in each component and strictly better in at least one. A point $\boldsymbol{x}$ is said to be *non-dominated* if it is not dominated by any other point. The set $P = \{\boldsymbol{x} \mid \nexists\, \tilde{\boldsymbol{x}}\ \tilde{\boldsymbol{x}} \preceq \boldsymbol{x}\}$ of all non-dominated points is called the *Pareto set*. It contains all incomparable trade-off solutions. In multi-objective optimization the goal is to approximate the Pareto set or the *Pareto front* $f(P)$, i.e., the image of P under f.

In recent years some approaches were published that generalize single-objective SMBO algorithms like EGO for the multi-objective case. We distinguish between 3 different MBMO algorithm classes: First, *scalarization based algorithms* that use EGO to optimize a scalarized version of the black-box functions with random weights for the scalarization in each iteration. Second, *Pareto based algorithms* that fit individual models for each objective and perform multi-objective optimization of infill criteria on these models. Third, *direct indicator based algorithms* that also fit individual models, but perform a single objective optimization of an infill criterion aggregating all models. `mlrMBO` supports 4 different MBMO algorithms, covering all 3 classes: ParEGO [38] as scalarization based, MSPOT [39] as Pareto based, and both SMS-EGO [40] and ε-EGO [41] as direct indicator based algorithms.

A much more detailed discussion of these methods, their multi-point variants, and what is currently implemented in `mlrMBO` is given in [4].

3. mlrMBO R Package

We implemented the software package `mlrMBO` for the statistical programming language `R`. It is designed as a modular framework. The individual components of model-based optimization such as the infill criterion or the stopping conditions (cf. Section 2) can easily be combined in a plug-and-play fashion to respect the specific characteristics of the optimization problem at hand. In the following we give a short introduction of this process which is split into multiple steps.

Definition of the black-box function. For the first step `mlrMBO` relies on the `smoof` package [42] which provides a unified interface to work with black-box functions. Many test functions that are frequently used to benchmark optimizers are already included. Additionally, the package provides the functions `make{Single,Multi}ObjectiveFunction()` as constructors for custom test functions. Mandatory arguments are the function itself, a name and a parameter set. In the simplest case, the latter is defined by names and box constraints, which can be specified concisely using the `ParamHelpers` package. For more complex settings, it is also possible to connect parameters with arbitrary transformation functions (e.g., to vary a parameter on the log-scale) or declare dependencies between parameters. The following listing gives an example for the definition of the black-box $f(\mathbf{x}) = (x_2 - 0.1x_1^2 + x_1 - 6)^2 + \cos(x_1)$ with $x_1 \in [-5, 10], x_2 \in [0, 15]$:

```
fn = makeSingleObjectiveFunction(
  name = "my_blackbox",
  fn = function(x) (x[2] - 0.1 * x[1]^2 + x[1] - 6)^2 + cos(x[1]),
  par.set = makeParamSet(
    makeNumericParam("x1", lower = -5, upper = 10),
    makeNumericParam("x2", lower = 0, upper = 15)
  )
)
```

Definition of the Initial Design. To specify the points to be evaluated to initialize the surrogate an initial design has to be specified. It is recommended to use a Latin Hypercube Design by calling `generateDesign()` and passing the number of desired points. If no design is given by the user, `mlrMBO` will generate a *maximin* Latin Hypercube Design of size 4 times the number of the black-box function's parameters.

Definition of the surrogate regression model. `mlrMBO` builds up on the `mlr` package [18], which offers a unified interface for a plethora of machine learning methods in `R`. For surrogate regression, Kriging (`makeLearner("regr.km")`) and random forests (`makeLearner("regr.randomForest")`) are popular choices, but other regression methods can be selected as well. Keep in mind that if expected improvement or LCB is chosen as the infill criterion, the surrogate either has to provide an uncertainty estimator, or has to be combined with a bagging approach using the `makeBaggingWrapper()` in `mlr`. If no regression method is supplied by the user, the fallback is a Kriging model with a Matern-$3/2$ kernel and the "GENetic Optimization Using Derivatives" (genoud) fitting algorithm in a fully numeric setting, and a random forest with jackknife variance estimation otherwise.

Definition of the control flow. Basic settings like the number of proposed points in each SMBO iteration or the error handling are set via `makeMBOControl()`

which returns a base control object. This object can be further extended to adjust the different component of the SMBO methodology. `setMBOControlInfill()` adjusts the infill criterion and the infill criterion optimizer. If the infill optimization is unspecified, `mlrMBO` uses LCB as infill criterion with $\lambda = 1$ in a fully numeric setting and $\lambda = 2$ if at least one one discrete parameter is present. To optimize the criterion, focus search with $n_{\text{restarts}} = 3$, $n_{\text{iters}} = 5$ and $n_{\text{points}} = 1000$ is used by default. For multi-point proposals or multi-objective optimization, `setMBOControlMultiPoint()` and `setMBOControlMultiObj()` are used, respectively. If multiple points are proposed, they can be evaluated simultaneously using different parallelization (i.e. multicore, sockets, and MPI) and high-performance computation systems (e.g., Slurm, LSF, OpenLava, TORQUE, or Docker Swarms) with the `R` packages `parallelMap` and `batchtools` [43]. Finally, `setMBOControlTermination()` controls the termination criteria.

Putting it all together. The actual optimization is finally started by calling the `mbo()` function with the (optional) initial design, the black-box function, the (optional) surrogate regression method, and the control object as arguments. The following listing demonstrates an application of `mlrMBO` to optimize our example black-box.

```
library(mlrMBO)

# Create initial random Latin Hypercube Design of 10 points
library(lhs) # for randomLHS
des = generateDesign(n = 5L * 2L, getParamSet(fn), fun = randomLHS)

# Specify kriging model with standard error estimation
surrogate = makeLearner("regr.km", predict.type = "se",
  covtype = "matern3_2")

# Set general controls
ctrl = makeMBOControl()
ctrl = setMBOControlTermination(ctrl, iters = 30L)
ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritEI())

# Start optimization
mbo(fn, des, surrogate, ctrl)
```

The resulting object contains the full optimization path, with all x and y values, runtime of function evaluations, final state, potential error messages as well as optionally all fitted surrogate models. Diagnostic visualizations of the optimization are available by calling `plot()` and for one and two dimensional input domains with single- or multi-objective targets, each step of the optimization process can be visualized by calling `exampleRun()` or `exampleRunMultiObj()`. For instance, Figure 2 has been created with `exampleRun()` and `plotExampleRun()`.

4. Benchmarks

In this section, the performance of `mlrMBO` is evaluated on three extensive benchmarks. First, we compare `mlrMBO` against other black-box optimizers connected to `R` (Section 4.1), then against state-of-the-art optimizers that are not available in `R` through the optimization benchmark framework HPOlib [44] (Section 4.2). Furthermore, we summarize a previously published simulation study of multi-objective optimization using `mlrMBO` [4] (Section 4.3). All benchmarks were conducted using the `batchtools` [43] package for `R`.

4.1. Model-Based Single-Objective Optimization

We run our implementation on various single-objective optimization tasks and compare it with the three EGO implementations available in `R`: `DiceOptim` [12], `rBayesianOptimization` [13] and `SPOT` [17]. Additionally, to ensure that an EGO approach is suitable, we also consider a basic random search as well as the popular covariance matrix adaptation evolution strategy (CMA-ES) based on the `R` package `cmaesr` [26].

Benchmarks. The methods are evaluated on a set of six 5-dimensional, continuous, and single-objective test functions: *Alpine01*, *Deflected Corrugated Spring*, *Schwefel*, *Ackley*, *Griewank* and *Rosenbrock*. All are defined in the `R`-package `smoof` and have been subject to optimization benchmarks previously.

Setup. For the initial design, the same pre-generated maximin Latin Hypercube design containing 25 points is used for `mlrMBO`, `DiceOptim`, `SPOT` and the random search. It was not possible to pass a user-defined initial design in `rBayesianOptimization` without provoking an error. Instead, a random design of the same size is generated internally. We allow each algorithm 200 sequential iterations. Since CMA-ES as an evolutionary algorithm does not initialize with a design, it gets an additional budget of 25 iterations (in total 225). All algorithms are run in their default settings carefully chosen by the respective package authors.

Evaluation. The objective values of the proposed solutions are summarized in Figure 4. All methods performed clearly better than the baseline random search approach on all six test functions. In comparison with the other EGO-based algorithms, `mlrMBO` yields a substantial better objective on four test functions and similar objective on the other two. `SPOT` is slightly better than `mlrMBO` on *Griewank*, but worse on three others. The evolutionary CMA-ES is comparable to `mlrMBO` on *Alpine01* and slightly better on *Rosenbrock*, but considerably worse on the four other problems. If we consider the averaged rank of the methods over all test functions as shown in Table 1, `mlrMBO` proves to be the best method overall, with `SPOT` in second place.

Besides the quality of the solution, runtime, and computational overhead should also be considered. The timings for a complete optimization run in

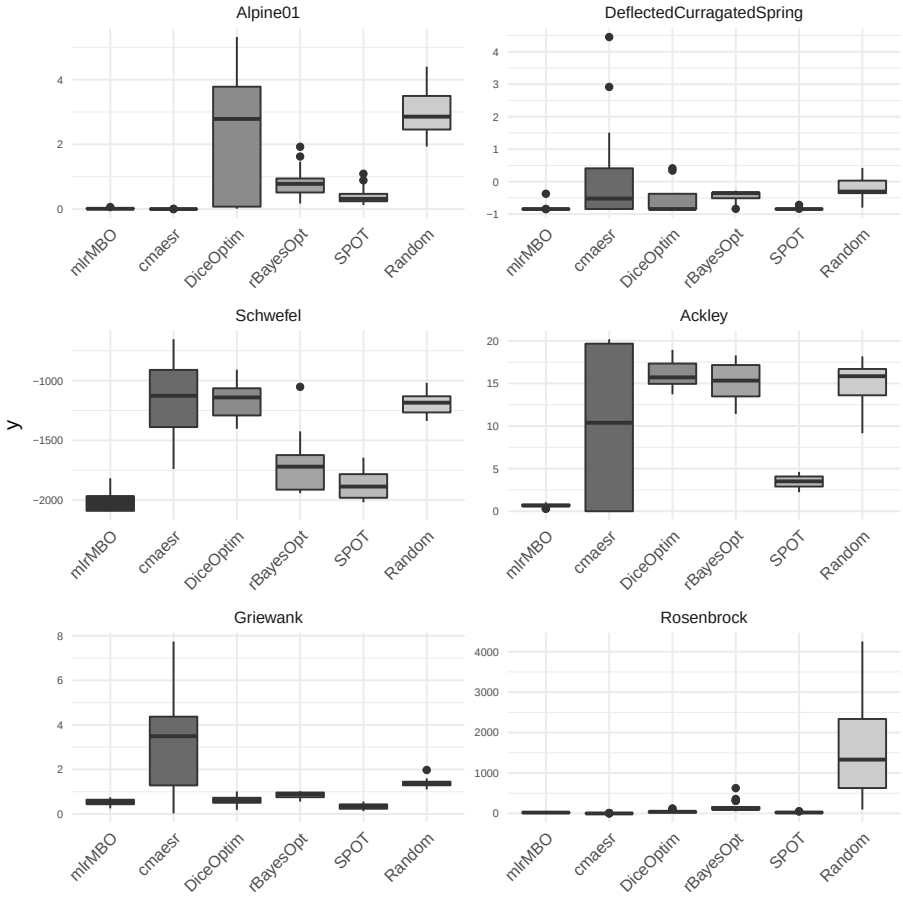

Figure 4: Best objective value (on y axis) found by respective algorithms on respective test function.

minutes are listed in Table 1. Note that we are basically measuring the overhead of the optimization algorithms, as the synthetic test functions are evaluated in microseconds. The random search unsurprisingly comes with the least overhead, followed by CMA-ES as implemented in the package `cmaesr`. The EGO-based approaches consume considerably more time by fitting the surrogate model and optimizing the infill criterion. Here, `mlrMBO` is slower than `DiceOptim` but still more than twice as fast as `SPOT` and orders of magnitudes faster than `rBayesianOptimization`. However, keep in mind that EGO is tailored for expensive problems. If we paid each function evaluation with just one minute of computation time, the differences between 200 min for random search and 212 min for `mlrMBO` seems to be a reasonable price to pay for a much better objective value.

Algorithm	Average Rank	Average runtime in minutes
`mlrMBO`	1.95	8.03
`SPOT`	2.48	27.88
`cmaesr`	3.17	0.01
`DiceOptim`	3.97	3.35
`rBayesOpt`	4.24	695.96
`Random`	5.19	0.00

Table 1: Average ranks and runtime on artificial test functions.

4.2. Model-Based Single-Objective Optimization in Mixed Spaces

The second benchmark compares `mlrMBO` to three[1] other state-of-the-art Bayesian optimizers which are not connected to `R`: Spearmint [14], hyperopt (called TPE in the following) [8] and SMAC [7]. We use the hyperparameter optimization library HPOlib [44], which contains a large number of standardized benchmarks. Besides purely numerical problems, the HPOlib also defines problems with mixed and dependent parameter spaces. We evaluate the methods on four synthetic functions (*branin*, *camelback*, *michalewicz* and *har6*), three parameter optimization problems on grids (linear discriminant analysis (*lda*), logistic regression (*logreg*) and a support vector machine (*svm*)), as well as a deep neural network (*hpnnet*) with 15 parameters, and a deep belief network (*hpdbnet*) with 35 parameters. The latter two problems were originally proposed by Bergstra et al. [8]. `mlrMBO` uses its default settings, i.e., a Gaussian process as surrogate model for all solely numerical problems and a random forest for the problems with mixed and dependent parameter spaces (*hpnnet* and *hpdbnet*). We deviate from the defaults only for the initial design of *hpnnet* and *hpdbnet*. Here, the number of allowed function evaluations compared to the dimension of the parameter set is very small, therefore the initial design has only size $2d$ instead of the default $4d$. Spearmint uses an internal dummy encoding of all categorical parameters for its Gaussian process. The number of iterations on each benchmark as well as the specific settings of all other optimizers are defined in HPOlib.

Evaluation. The results of ten runs on each benchmark are summarized in Figure 5. On each of the four synthetic test functions, `mlrMBO` outperforms both SMAC and TPE and has similar performance compared to Spearmint, except for *michalewicz* where `mlrMBO` outperforms all competitors. For the grid optimizations, `mlrMBO` also performs exceptionally well on every single one, while each other optimizer results in worse performance on at least one of the three problems, which overall places `mlrMBO` on the first place in numeric settings (cf. Table 2). Regarding the neural network and deep belief network, `mlrMBO` achieves similar results as SMAC and slightly better results than Spearmint

[1]Since `BayesOpt` is neither connected to HPOlib nor possesses an `R` interface, we refer the reader to the benchmarks in [16] and do not consider `BayesOpt` in our analysis.

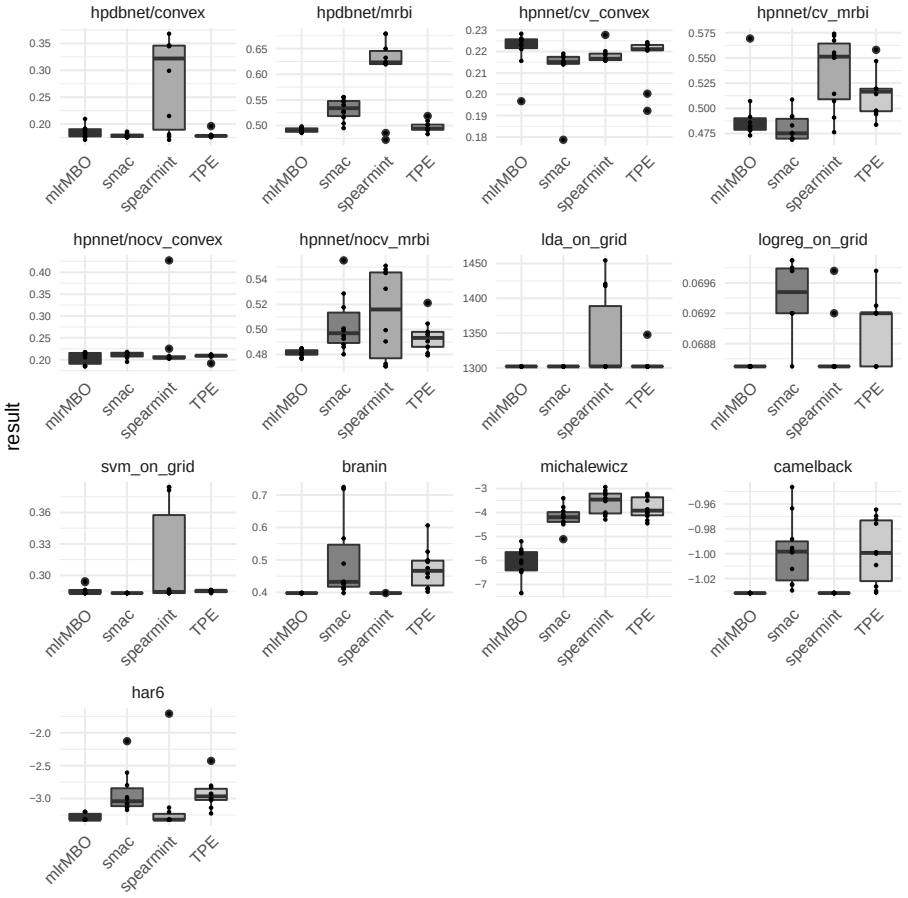

Figure 5: Best objective value (on y axis) found by respective optimizer on 13 HPOlib test functions. For details on the test functions we refer to [8] for *hpnnet* and *hpdbnet* and to [44] for everything else.

and TPE. Especially Spearmint has a clearly worse performance on three of the six problems, while `mlrMBO` is only worse than its competitors on *hpdbnet/cv_convex*. As a result, Table 2 shows that `mlrMBO` also places first w.r.t. aggregated mean ranks for mixed hyperparameter spaces. We can clearly see that `mlrMBO` is on par with other state-of-the-art Bayesian optimization software, even in highly complex settings.

4.3. Model-Based Multi-Objective Optimization

In the following section we will briefly summarize a benchmark performed by Horn et al. [4]. We will focus on the results concerning the four MBMO algorithms implemented in `mlrMBO`. For the detailed experimental setup and further details we refer to the original paper.

Optimizer	Avg. rank	Avg. rank (numeric only)	Avg. rank (mixed only)
`mlrMBO`	1.90 **(1)**	1.64 **(1)**	2.20 **(1)**
`smac`	2.65 **(3)**	2.90 **(3)**	2.35 **(2)**
`spearmint`	2.61 **(2)**	2.32 **(2)**	2.95 **(4)**
`TPE`	2.85 **(4)**	3.14 **(4)**	2.50 **(3)**

Table 2: Average ranks on HPOlib problems, Results were ranked in each replication and then averaged over the replications and problems. Numeric only ranks are based on benchmark 7 to 13 and mixed only ranks are based on 1 to 6.

Benchmarks. The benchmark was performed on nine artificial test functions. On the one hand, four well known multi-objective test functions were used, namely *ZDT1*, *ZDT2* and *ZDT3* each with dimension $d = 5$ and number of objectives $k = 2$, as well as *DTLZ2* using $d = 5$ and $k \in \{2, 5\}$. On the other hand, four additional test functions were constructed by combining various single-objective problems, in such a way that each combination of $d \in \{2, 5\}$ and $k \in \{2, 5\}$ is used. These are called *GOMOP* in the following.

Setup. To simulate an expensive setting, all algorithms had a budget of only $40d$ function evaluations. The popular popular evolutionary multi-objective algorithm NSGA2 [45] and a random search serve as baseline for the four implementation in `mlrMBO`: SMS-EGO, ϵ-EGO, SMS-Ego, and MSPOT (cf. Section 2.6).

Evaluation. Various performance measures for comparing different approximations have been introduced. The most popular measure may be the dominated hypervolume (also known as S-Metric). In the bi-objective case the hypervolume simply measures the area between the discrete approximation of the Pareto front and a pessimistic reference point. If an approximation reaches a higher hypervolume value, it is considered superior.

The final Pareto front approximations were normalized to the interval $[1, 2]^m$ with respect to a reference set. In Figure 6, the hypervolume values of all runs with respect to the reference point $(1.1)^k$ are shown for 20 replications per test function. We see that most MBMO algorithms outperform both baseline algorithms on nearly all test functions, only ParEGO fails occasionally. Moreover, its performance is always worse than ϵ-EGO and SMS-EGO. The three remaining MBMO algorithms perform comparably well, whereas SMS-EGO shows top or near-top performance on all test functions.

5. Conclusion

We introduced the `R` package `mlrMBO`, a modular toolbox for model-based optimization in the `R` programming language. We gave a brief introduction to software specific aspects and features. Furthermore, we performed comprehensive benchmarks of `mlrMBO` against other black-box optimizers in different

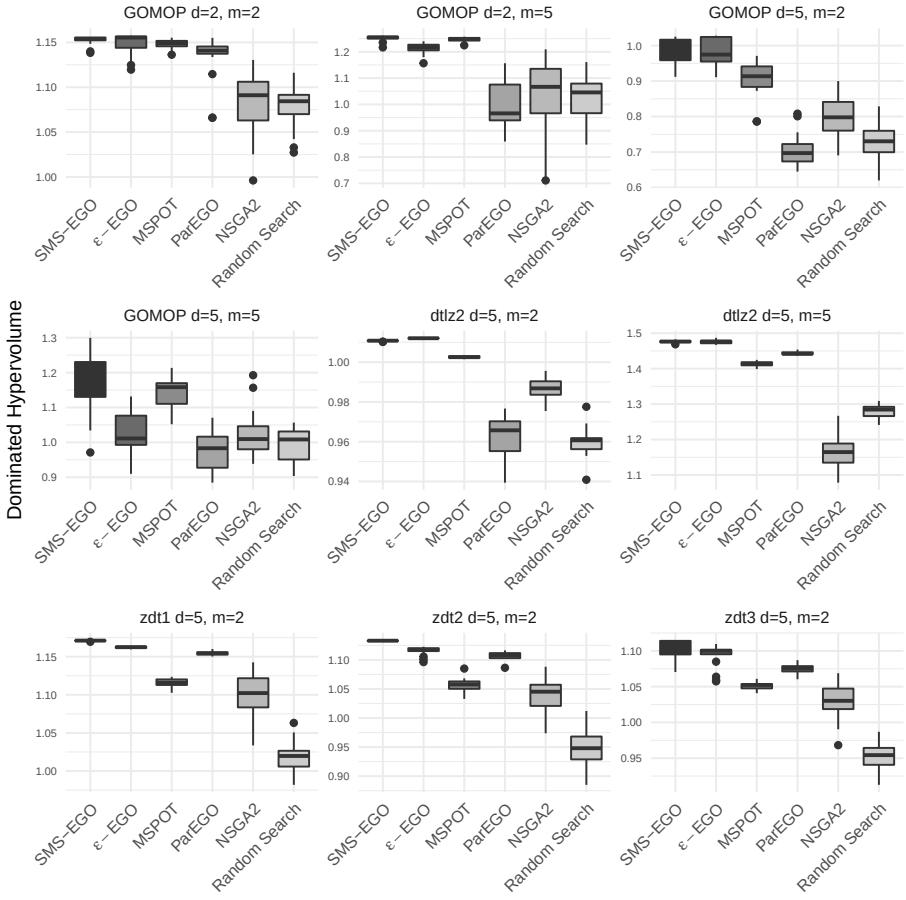

Figure 6: Normalized dominated hypervolume (on y axis) for the respective algorithm on respective test functions.

scenarios. In the single-objective benchmark `mlrMBO` proved state-of-the-art performance regarding solution quality in comparison with the CMA evolutionary strategy, random search, and alternative SMBO implementations, while still being reasonably fast. Furthermore, `mlrMBO` is on par with the well known optimization frameworks SMAC, Spearmint, and TPE as shown by benchmarks using HPOlib. The benchmark study on expensive multi-objective optimization revealed SMBO-based methods, in particular SMS-EGO, to show excellent performance. Both the state-of-the-art NSGA-II evolutionary algorithm as well as the baseline random search algorithm were outperformed on all nine test functions (only ParEGO occasionally failed). All in all the results demonstrate the suitability of the `mlrMBO` toolbox in particular for expensive optimization scenarios in `R` for single- and multi-objective tasks, with continuous or mixed parameter spaces.

Acknowledgments

Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project A3 (http://sfb876.tu-dortmund.de) and by the Competence Network for Technical, Scientific High Performance Computing in Bavaria (KONWIHR) in the project "Implementierung und Evaluation eines Verfahrens zur automatischen, massiv-parallelen Modellselektion im Maschinellen Lernen".

References

1. Sieben, B., Wagner, T., Biermann, D.. Empirical modeling of hard turning of aisi 6150 steel using design and analysis of computer experiments. *Production Engineering* 2010;4(2):115–125.

2. Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P.. Design and analysis of computer experiments. *Statistical science* 1989;4(4):409–423.

3. Jones, D.R., Schonlau, M., Welch, W.J.. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization* 1998;13(4):455–492.

4. Horn, D., Wagner, T., Biermann, D., Weihs, C., Bischl, B.. Model-based multi-objective optimization: Taxonomy, multi-point proposal, toolbox and benchmark. In: Gaspar-Cunha, A., Henggeler Antunes, C., Coello, C.C., eds. *Evolutionary Multi-Criterion Optimization*; vol. 9018 of *Lecture Notes in Computer Science*. Springer; 2015:64–78.

5. Ginsbourger, D., Le Riche, R., Carraro, L.. Kriging is well-suited to parallelize optimization. In: *Computational Intelligence in Expensive Optimization Problems*. Springer; 2010:131–162.

6. Bischl, B., Wessing, S., Bauer, N., Friedrichs, K., Weihs, C.. MOI-MBO: Multiobjective infill for parallel model-based optimization. In: *Learning and Intelligent Optimization Conference*. 2014:173–186.

7. Hutter, F., Hoos, H.H., Leyton-Brown, K.. Sequential model-based optimization for general algorithm configuration. In: *LION 5*. 2011:507–523.

8. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.. Algorithms for hyper-parameter optimization. In: *Advances in Neural Information Processing Systems*. 2011:2546–2554.

9. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: *Proceedings of ACM SIGKDD*. 2013:847–855.

10. Lang, M., Kotthaus, H., Marwedel, P., Weihs, C., Rahnenführer, J., Bischl, B.. Automatic model selection for high-dimensional survival analysis. *Journal of Statistical Computation and Simulation* 2015;85(1):62–76.

11. Horn, D., Bischl, B.. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. *Symposium Series on Computational Intelligence* 2016;ACCEPTED.

12. Roustant, O., Ginsbourger, D., Deville, Y.. DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software* 2012;51(1):1–55.

13. Yan, Y.. rBayesianOptimization: Bayesian Optimization of Hyperparameters; 2016. URL: `https://CRAN.R-project.org/package=rBayesianOptimization`; R package version 1.0.0.

14. Snoek, J., Larochelle, H., Adams, R.P.. Practical bayesian optimization of machine learning algorithms. In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc.; 2012:2951–2959.

15. Hernández-Lobato, D., Hernández-Lobato, J.M., Shah, A., Adams, R.P.. Predictive entropy search for multi-objective bayesian optimization. In: *Proceedings of the 33nd International Conference on Machine Learning (ICML)*. 2016:1492–1501.

16. Martinez-Cantin, R.. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research* 2014;15:3915–3919.

17. Bartz-Beielstein, T., Zaefferer, M.. A gentle introduction to sequential parameter optimization. Tech. Rep. 2; Bibliothek der Fachhochschule Koeln; 2012. URL: `http://opus.bsz-bw.de/fhk/volltexte/2012/19`.

18. Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M.. mlr: Machine learning in R. *Journal of Machine Learning Research* 2016;17(170):1–5.

19. Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., Weihs, C., Konen, W.. Tuning and evolution of support vector kernels. *Evolutionary Intelligence* 2012;5(3):153–170.

20. Bischl, B., Schiffner, J., Weihs, C.. Benchmarking classification algorithms on high-performance computing clusters. In: Spiliopoulou, M., Schmidt-Thieme, L., Janning, R., eds. *Data Analysis, Machine Learning and Knowledge Discovery*. Studies in Classification, Data Analysis, and Knowledge Organization; Springer; 2014:23–31.

21. Hess, S., Wagner, T., Bischl, B.. PROGRESS: Progressive reinforcement-learning-based surrogate selection. In: Nicosia, G., Pardalos, P., eds. *Learning and Intelligent Optimization*. Lecture Notes in Computer Science; Springer; 2013:110–124.

22. Horn, D., Demircioğlu, A., Bischl, B., Glasmachers, T., Weihs, C.. A comparative study on large scale kernelized support vector machines. *Advances in Data Analysis and Classification* 2016;:1–17.

23. Steponavičė, I., Shirazi-Manesh, M., Hyndman, R.J., Smith-Miles, K., Villanova, L.. On Sampling Methods for Costly Multi-Objective Black-Box Optimization. In: Pardalos, P.M., Zhigljavsky, A., Žilinskas, J., eds. *Advances in Stochastic and Deterministic Global Optimization*. No. 107 in Springer Optimization and Its Applications; Springer International Publishing. ISBN 978-3-319-29973-0 978-3-319-29975-4; 2016:273–296.

24. McKay, M.D., Beckman, R.J., Conover, W.J.. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* 2000;42(1):55–61.

25. Schiffner, J., Bischl, B., Lang, M., Richter, J., Jones, Z.M., Probst, P., Pfisterer, F., Gallo, M., Kirchhoff, D., Kühn, T., Thomas, J., Kotthoff, L.. mlr tutorial. 2016. `arXiv:arXiv:1609.06146`.

26. Bossek, J.. cmaesr: Covariance Matrix Adaptation Evolution Strategy; 2016. URL: `https://CRAN.R-project.org/package=cmaesr`; R package version 1.0.1.

27. Rasmussen, C.E., Williams, C.K.I.. Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning; MIT Press; 2006.

28. Vazquez, E., Bect, J.. Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *Journal of Statistical Planning and Inference* 2010;140(11):3088–3095.

29. Jones, D.R.. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization* 2001;21(4):345–383.

30. Zhou, Q., Qian, P.Z., Zhou, S.. A simple approach to emulation for computer models with qualitative and quantitative factors. *Technometrics* 2012;.

31. Ding, Y., Simonoff, J.S.. An investigation of missing data methods for classification trees applied to binary response data. *Journal of Machine Learning Research* 2010;11:131–170. URL: `http://www.jmlr.org/papers/v11/ding10a.html`.

32. Sexton, J., Laake, P.. Standard errors for bagged and random forest estimators. *Computational Statistics & Data Analysis* 2009;53(3):801–811.

33. Wager, S., Hastie, T., Efron, B.. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *Journal of Machine Learning Research* 2014;15:1625–1651. URL: `http://jmlr.org/papers/v15/wager14a.html`.

34. Hutter, F., Hoos, H.H., Leyton-Brown, K.. Parallel algorithm configuration. In: *Learning and Intelligent Optimization*. Springer; 2012:55–70.

35. Preuss, M.. Considerations of budget allocation for sequential parameter optimization (spo. In: *Workshop on Empirical Methods for the Analysis of Algorithms, Proceedings*. 2006:35–40.

36. Picheny, V., Ginsbourger, D., Richet, Y., Caplin, G.. Quantile-based optimization of noisy computer experiments with tunable precision. *Technometrics* 2013;55(1):2–13.

37. Picheny, V., Wagner, T., Ginsbourger, D.. A benchmark of kriging-based infill criteria for noisy optimization. *Struct Multidiscip Optim* 2013;48(3):607–626.

38. Knowles, J.. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation* 2006;10(1):50–66.

39. Zaefferer, M., Bartz-Beielstein, T., Naujoks, B., Wagner, T., Emmerich, M.. A case study on multi-criteria optimization of an event detection software under limited budgets. In: Purshouse, R., et al., eds. *Proc. 7th Int'l. Conf. Evolutionary Multi-Criterion Optimization (EMO)*. Springer; 2013:756–770.

40. Ponweiser, W., Wagner, T., Biermann, D., Vincze, M.. Multiobjective optimization on a limited amount of evaluations using model-assisted $\mathcal{S}$-metric selection. In: *Proc. 10th Int'l Conf. Parallel Problem Solving from Nature (PPSN)*. 2008:784–794.

41. Wagner, T.. Planning and Multi-Objective Optimization of Manufacturing Processes by Means of Empirical Surrogate Models. Vulkan Verlag, Essen; 2013.

42. Bossek, J.. smoof: Single and Multi-Objective Optimization Test Functions; 2016. URL: `https://github.com/jakobbossek/smoof`; R package version 1.4.

43. Bischl, B., Lang, M., Mersmann, O., Rahnenführer, J., Weihs, C.. BatchJobs and BatchExperiments: Abstraction mechanisms for using R in batch environments. *Journal of Statistical Software* 2015;64(11):1–25.

44. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In: *NIPS workshop on Bayesian Optimization in Theory and Practice*. 2013:1–5.

45. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 2002;6(2):182–197.