Technische Universität Dortmund
Fakultät Statistik

Extending Model-Based Optimization with Resource-Aware Parallelization and for Dynamic Optimization Problems

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

von

MSc.
Jakob Richter

Erstgutachter: Prof. Dr. Jörg Rahnenführer
Zweitgutachter: Prof. Dr. Andreas Groll

Contents

1 Introduction

Sequential model-based optimization has gained a lot of attention since Jones et al. (1998) proposed the *Efficient Global Optimization* (EGO) algorithm. However, the foundations for EGO have been laid much earlier. To improve the estimations of gold values in the earth for mining, Krige (1951) pioneered the principle of distance based estimations. Until then the gold value for a new spot was estimated by averaging the values of surrounding equidistant probes and unusual high or low values were "'adjusted' by arbitrary methods". Krige (1951) assumed that the gold value at each point is distributed following a log-normal distribution with different parameters for the mean and the variance. For any new spots the parameters of this distribution have to be estimated. These parameters are estimated based on known values in combination with their distances to the new spot. Even though the original work by Krige is far from the contemporary understanding of *Gaussian process regression* he still lends his name to the method, also known as *Kriging*. While the theory of Gaussian processes stems from the 1950s, the estimation of the parameters of a Gaussian process based on observations, in other words, fitting a Gaussian process to data, is relatively new and well explained and widely discussed in Rasmussen et al. (2006). The key idea remains the same as in the original work by Krige: The outcome for each observation follows a distribution, i.e. the normal distribution for Gaussian process regression. The parameters of this distribution are dependent on the location of this observation and have to be estimated based on known outcomes. Typically for spatial methods, points in the vicinity are accounted for with a higher weight.

Following the initial motivation of Krige, to find the location where the highest gold value can be estimated, *Kriging* became essential for optimization of all kind

of functions. The gold value in the earth can be seen as a function of spatial coordinates. The same principle can be applied to many scenarios in real life to formulate optimization problems. The productivity of a factory can be seen as a function that depends on the number of machines, number of workers and other variables that have influence on the productivity. In engineering, the stability of a structure can be seen as a function of the mix of materials, design choices and production parameters. The lift of an airplane wing can be modeled as a function depending on different curvature parameters. These examples show that optimization can be applied in many scenarios. Furthermore, they have in common that it is unfeasible and in some cases even impossible to try out all possible settings in order to find an optimal setting. Also there is no evident mathematical formula that describes the connection between the variables and the objective value. As Golovin et al. (2017) puts it:

> Any sufficiently complex system acts as a black-box when it becomes easier to experiment with than to understand.

If an evaluation of a single setting causes non-negligible costs, such as time, money, energy etc., these problems are commonly referred to as *Expensive Black-Box Optimization* problems.

If the number of trials is limited, *design of experiments* is a popular field in statistics to find a set of points that cover the search space optimally according to a certain criterion. Statistical methods are used to model the influence of explanatory variables on the response variable. Such a regression model can estimate the response variable for new values of the explanatory variables. In this regard, such a model behaves like a mathematical function itself. Given a vector of explanatory variables it returns the predicted numerical outcome.

This motivates the usage of such a model as a proxy for optimization, as the model can be evaluated much faster then the original expensive black-box, making optimization easier. The idea is, that the optimization of the function derived from the regression model yields values of the explanatory variables that lead to a value closer to the true optimum of the response variable than the already tried values.

If we assume that the regression model reflects the true interaction between the explanatory variables and the response, the optimum of the surrogate would give us the optimal choice of the explanatory variables for the black-box. This idea was the first approach to *model-based optimization*.

It is justified to assume that the true interaction between the explanatory variables and the response cannot be modeled entirely but that the interaction is captured to a certain degree. We especially care about the accuracy of the regression model in the area of the optimum. Coming back to the example of Krige, we would probe for gold especially in areas where the regression model predicts high gold values. Afterwards, we would use the result of these newly acquired probes to further improve our regression model. Finally, we would drill again where our updated regression model predicts the highest gold values. This basically describes the principle of *sequential model-based optimization* (SMBO).

Eventually, it is not optimal to only drill where the regression model predicts the highest gold value because this would prevent the exploration of new areas. Therefore, instead of using the location that maximizes the surrogate prediction, we choose the location that maximize an *acquisition function*. These *acquisition functions* combine the surrogate's mean and uncertainty point estimates into one value that reflects the "promisingness" of a new location. Thus, maximizing the *acquisition function* leads to proposals that are balanced between exploration of new regions and exploitation of regions where the surrogate predicts an optimum.

SMBO has been applied to optimize the hyperparameter settings for machine learning methods (Snoek et al., 2012), to optimize machine learning pipelines for survival analysis (Richter, Madjar, et al., 2019), to optimize production processes, to guide material studies, in general algorithm configuration (Hutter et al., 2011) and further more. SMBO is applied to all kinds of expensive black-box optimization problems where other optimizers that require many evaluations, such as evolutionary optimizers, fail due to runtime limitations.

In fact, such problems can be highly complex in various terms and each challenge introduces a new theoretical field of black-box optimization. The cost to obtain an

output can be very high, unknown in advance and vary based on the input. The output can be noisy and the noise can be heteroscedastic. The relation between input and output can change over time. The input space can contain categorical or ordinal values. In this work we will only discuss the optimization of deterministic objective functions with numeric input.

In this thesis two new MBO methods will be presented. Each tackles a different problem of black-box optimization with MBO.

The first contribution is RAMBO (Richter, Kotthaus, Bischl, et al., 2016; Richter, Kotthaus, A. Lang, et al., 2017), which is a framework for *parallel resource-aware model-based optimization*. It offers a solution to the problem of evaluating multiple black-box configurations with different runtimes simultaneously within MBO. Strategies that tackle this problem can be divided into synchronous and asynchronous methods. Instead of proposing one configuration in an iterative fashion, as done by ordinary SMBO, synchronous methods usually propose as many configurations as there are parallel black-box instances available. Previously proposed synchronous methods neglect the problem of heterogeneous runtimes which causes idling, when evaluations end at different times. Asynchronous methods can be seen as multiple SMBO processes that run independently on each parallel instance but that share their knowledge about evaluated points. RAMBO is a synchronous method that uses runtime predictions to propose configurations that can be evaluated with little idling. As a consequence, RAMBO can evaluate more configurations than other synchronous parallel MBO methods within the same time. We show how RAMBO compares against the most common synchronous and asynchronous approaches on two different sets of synthetic benchmark functions. One set has easily predictable runtimes and we expect RAMBO to be able to reduce idling times. The other set has hardly predictable runtimes and serves as an exemplary worst case. The results show that synchronous and asynchronous methods each have their advantages and disadvantages and that RAMBO can outperform common synchronous MBO methods if the runtime is predictable but still obtains comparable results in the worst case.

The second contribution is an extension of MBO towards dynamic optimization problems. We present two approaches that enable MBO to handle black-box functions where the relation between input and output changes over time. The *window approach* trains the surrogate only on the most recent observations. The *time-as-covariate approach* includes the time as an additional input variable in the surrogate, giving it the ability to learn the effect of the time. We show how the proposed approaches handle the scenarios of no drift, sudden drift and incremental drift on a set of synthetic benchmark functions, where the change happens in a controlled fashion. The results show that the new methods improve the performance if a drift is present and that the *time-as-covariate approach* works better on lower-dimensional problems where it is easier for the surrogate to capture the influence of the time.

The research leading to this thesis has received funding from the Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 – Providing Information by Resource-Constrained Data Analysis – Project A3. The research on resource-aware model-based parallelization was done in collaboration with Helena Kotthaus who also wrote her dissertation (Kotthaus, 2018) partly on this topic. The work on the concept drift adaptation was supported by discussions with Prof. Dr. Jian-Jia Chen and Junjie Shi. The maintenance and further development of `mlrMBO` (Bischl, Richter, et al., 2017) is an essential part of this work. The R package `mlrMBO` offers a versatile MBO toolbox and all developed methods are available as derivatives of this package.

2 Symbols and Notation

All symbols that are used consistently throughout this work are listed below. Index values i and j are context sensitive. Superscripts are written in brackets to not be confused with exponents. Model-based optimization is an iterative algorithm and as a consequence the value of symbols such as θ^+, $\hat{\mu}(\theta)$, $\hat{s}(\theta)$, $\mathcal{D}$ etc. change iteratively. For readability an iteration index is suppressed.

Symbol	Meaning
Model-based Optimization (MBO)	
Θ	Domain, search space
d	Dimensionality of the search space
θ	Input, configuration, optimization parameters, point in Θ
θ_l	Single optimization parameter, $l \in 1, \ldots, d$
$f(\theta) \colon \Theta \to \mathbb{R}$	Objective function that w.l.o.g. is to be minimized
ν	Function value, outcome
$f(\theta^*) = \nu^*$	Global optimum and optimal value of f
$\hat{\theta}^*$	Estimated point for global optimum
θ^+	Proposed point to be evaluated
$\theta^{(i)}$	Already evaluated point, $i \in 1, \ldots, n$
$\hat{\mu}(\theta)$	Surrogate model mean prediction
$\hat{\mathrm{s}}(\theta)$	Surrogate model uncertainty as standard deviation
$\mathcal{D}$	Design that the surrogate model is trained on
$\mathrm{acq}(\theta)$	Arbitrary acquisition function that, w.l.o.g. has to be maximized
σ_n^2	Noise variance, nugget effect
σ_f^2	Signal variance

Symbol	Meaning
Parallel MBO	
k	Number of workers
q	Number of proposals per iteration
b	Number of busy workers
$\theta^{+(i)}$	Proposal i out of q total proposals, $i \in 1, \ldots, q$
$\theta^{(j)}_{\text{busy}}$	Proposal that is currently under evaluation, $j \in 1, \ldots, b$
$\tilde{\nu}_{j,\text{busy}}$	Temporary fake outcome for the unknown outcome of $f(\theta^{(j)}_{\text{busy}})$
$\tilde{\mathcal{D}}$	Design that contains temporary fake outcomes
$t(\theta)$	Runtime: Time needed to evaluate $f(\theta)$
$\hat{t}(\theta)$	Estimated runtime for $f(\theta)$
Scheduling	
J	Set of jobs
p_j	Priority of job $j \in J$
MBO-CD	
t	Time since optimization start
t_Δ	Window size
f_t	Function at time t
$\hat{\mu}_t$	Surrogate model mean prediction for time t
$\hat{s}_t$	Surrogate model uncertainty for time t
θ^*_t	True optimum at time t
θ^+_t	Proposal for time t

The following table lists the most often used abbreviations and the location of their definition in alphabetical order.

Abbreviation	**Meaning**	**Definition**
AEI	Augmented Expected Improvement	(3.4.8) p. 30
CB	Confidence Bound	(3.4.5) p. 29
EEI	Expected Expected Improvement	(4.3.1) p. 46
EI	Expected Improvement	(3.4.4) p. 28
FE	Fitness Error	(5.6.1) p. 64
qCB	q-Confidence Bound	(4.2.1) p. 40
TEI	Temporal Expected Improvement	(5.4.3) p. 62

3 Model-Based Optimization (MBO)

In this chapter, the individual components of the MBO framework will be introduced. In Section 3.1 the fundamental MBO framework will be presented independently from any assumptions and choices of the surrogate or acquisition function. The succeeding sections are dedicated to the single components of the MBO framework. In Section 3.2 the role of the initial design and the definition of the search space is briefly discussed. Section 3.3 is dedicated to the first building block of the MBO framework: the surrogate. The Gaussian process regression will be introduced here since it is used as surrogate throughout this work. Furthermore, the effect of the choice and configuration of the surrogate on the optimization will be discussed. The second essential MBO building block, the acquisition function, is the topic of Section 3.4. The concrete acquisition functions that are of importance in this work will be discussed here. Finally, the particularities of the termination criteria for MBO will be discussed in Section 3.5 and related work that was not mentioned until that point is mentioned in Section 3.6.

The terms (sequential) model-based optimization, or shortly MBO (Hutter et al., 2011), efficient global optimization (EGO, Jones et al., 1998) and Bayesian (global) optimization (Močkus, 1975) are often used interchangeable. However, there are slight differences: MBO is the most general term that is used for the idea of using surrogates to guide the optimization, efficient global optimization is the term coined in Jones et al. (1998). MBO is also referred to as Bayesian optimization if Kriging is used as a surrogate. Kriging is also known as Gaussian process regression.

3.1 Fundamental Framework

In the following, the fundamental model-based optimization framework will be introduced. Given an expensive black-box function $f(\theta)\colon \Theta \to \mathbb{R}$ the optimization goal is to find (without loss of generality)

$$\theta^* = \arg\min_{\theta \in \Theta} f(\theta). \tag{3.1.1}$$

We assume that no further information about the structure of f is known. This especially includes the absence of derivative information of f. Furthermore and importantly, f is expensive and thus effectively limiting the number of evaluations we have to obtain an optimization result. We call Θ the search space. The dimensionality of Θ is denoted with $d \in \mathbb{N}$. Θ has to be bounded. For most parts in this work Θ is considered a subset of $\mathbb{R}^d$ which is not necessarily required by all methods.

The key idea behind the model-based optimization framework is to use a so-called surrogate model that predicts the outcomes of f for unknown values of θ. The general approach is illustrated as a flow chart in Figure 3.1.

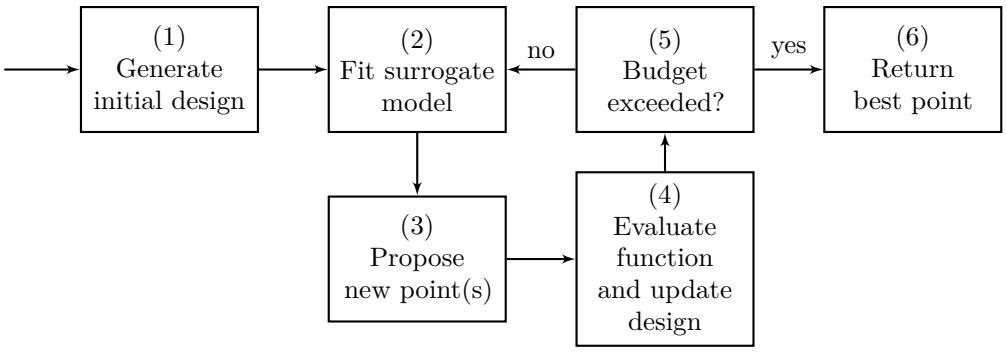

Figure 3.1: General SMBO approach.

The steps are explained in the following:

(1) An initial design of n_{init} points $\theta^{(i)}$ $(i = 1, \ldots, n_{\text{init}})$ is sampled from Θ and f is evaluated at these points to obtain outcomes $\nu^{(i)} = f(\theta^{(i)})$. The design $\mathcal{D} = \{\left(\theta^{(i)}, \nu^{(i)}\right) \mid i = 1, \ldots, n_{\text{init}}\}$ forms the training data for the surrogate model.

(2) Fit the surrogate model on $\mathcal{D}$, whereas ν serves as the dependent variable.

(3) Obtain a proposal point θ^{+} that will be evaluated on f. Therefore, an *acquisition function* is optimized and the optimum serves as the proposal point θ^{+}. These points should either have a good expected objective value or a high potential to find new minima in unexplored regions.

(4) Evaluate the proposed points on the objective f to obtain outcome $\nu^{(i+1)}$. Append new evaluations to design: $D = D \cup \{\left(\theta^{(i+1)}, \nu^{(i+1)}\right)\}$.

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

(6) Otherwise, return the proposed solution for the optimization problem.

A detailed discussion for each step can be found in the following sections 3.2 to 3.5. In Listing 1 the pseudo code is given for sequential MBO.

Algorithm 1 Sequential model-based optimization algorithm with single point proposal for deterministic functions.

Require: expensive black-box function $f(\theta)\colon \Theta \to \mathbb{R}$, acquisition function $\text{acq}(\theta)\colon \Theta \to \mathbb{R}$

1: sample n_{init} points from Θ: $\{\theta^{(i)} | i = 1, \ldots, n_{\text{init}}\}$
2: evaluate $\nu^{(i)} = f(\theta^{(i)})$ for $i = 1, \ldots, n_{\text{init}}$
3: $D \leftarrow \{\left(\theta^{(i)}, \nu^{(i)}\right) | i = 1, \ldots, n_{\text{init}}\}$
4: **while** budget is not exhausted **do**
5: fit surrogate model on $\mathcal{D}$ to obtain estimators $\hat{\mu}, \hat{s}$
6: propose new point $\theta^{+} \leftarrow \arg\max_{\theta} \text{acq}(\theta)$
7: evaluate $\nu \leftarrow f(\theta^{+})$
8: $D \leftarrow D \cup \{(\theta^{+}, \nu)\}$
9: **return** optimal setting $\hat{\theta}^{*} = \arg\min_{\theta \in D} f(\theta)$

3.2 Initial Design

For the initial design two aspects have to be considered: First, the number of points and secondly, how to sample the points from the search space Θ.

If the search space is purely numeric ($\Theta \subset \mathbb{R}^d$), a rule of thumb is $n_{\text{init}} = 4 \cdot d$. An established method to sample the points is to use a space-filling Latin hypercube sample (LHS, McKay et al., 1979). However, a comparison of the effects of different sample sizes and sampling strategies, such as LHS or random sample, by Morar et al. (2017) stated that the optimal initial design varies from problem to problem and no clear recommendation could be given.

Alternatively, MBO can be initialized with an already evaluated design, which can be seen as a warm start. If available, this is always preferable, because it brings all available knowledge about f into the optimization. However, if this design is only densely populated around a certain area it should be augmented with additional samples to allow for a reasonable good fit of the initial surrogate model.

In some situations a distribution is given for the location of the optimum. Then it is beneficial to sample the n_{init} points according to this a-priori distribution, because it increases the chance that the surrogate is accurate in the desired region. However, this should not be confused with rescaling. Rescaling a certain variable is recommended, if the effect on the outcome of changing this variable by a small value depends on its value. For example, this is often the case for optimization parameters θ_l that are theoretically valid in $(0, \infty)$. A change of $\theta_l = 0.1$ to $\theta_l = 0.2$ is expected to have the same effect on the outcome as a change of $\theta_l = 100$ to $\theta_l = 200$. In those cases we should optimize $\tilde{\theta}_l \in (-\infty, \infty)$ and obtain the input θ_l by applying the transformation $\theta_l = 2^{\tilde{\theta}_l}$ as illustrated in Figure 3.2 before evaluating $f(\theta)$. Also other transformations are possible.

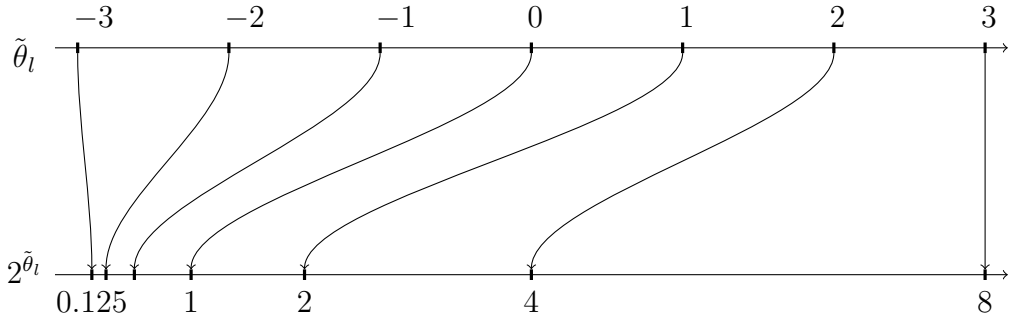

Figure 3.2: A transformation of the search space is a practical consideration to allow the surrogate to fit more accurately in regions where the objective function is sensitive for changes of a certain parameter.

3.3 Surrogate Model

The most common surrogate used for MBO is *Gaussian process regression* (also known as *Kriging*). This combination dates back to the EGO Algorithm presented in Jones et al. (1998). Its usage is widespread with various practical results. It has two desirable characteristics for MBO which are: First, the predicted uncertainty for known points is 0 for deterministic optimization problems. Second, the further away a point is from known points, the higher the uncertainty. More details on the Gaussian process regression are given in Section 3.3.1. One drawback of Gaussian process regression is that it only poorly handles categorical variables. As in this work all optimization problems are defined on a purely numerical domain only a short review of methods for mixed-valued search spaces will be given in Section 3.3.2.

In this section surrogate models are introduced as machine learning models. This is motivated by the fact that within the `mlrMBO` toolbox and within the general formulation of the SMBO framework (compare Algorithm 1) the surrogate is interchangeably by any regression method.

For readability and compatibility across the chapters, the variables that contain the same values will maintain the same notation, although their meaning changes in the context of the surrogate. The search space Θ becomes the feature space for the

model and accordingly the design $\mathcal{D}$ becomes the training data with feature vectors (covariates) $\theta_1, \ldots, \theta_d$ and the label vector (outcome) ν. The mean prediction of the surrogate for input θ is denoted with $\hat{\mu}(\theta)$ and its uncertainty prediction is denoted with $\hat{s}(\theta)$.

The following sections are motivated by the fact that specific knowledge about certain aspects of the methods used to build the surrogate is helpful to avoid mistakes that lead to bad optimization performance. It has to be clear how the surrogate handles different characteristics of the search space Θ and how the surrogate obtains uncertainty estimations. In contrast to a machine learning task, where the best method can be simply selected by its predictive performance, for model-based optimization it is difficult to define a performance measure that reflects the purposefulness of the surrogate model. Any measure based on averaging the residuals might put too much emphasis on regions in Θ where accuracy is not important. After all, the accuracy only has to be high in regions of potential optima, i.e. where the surrogate predicts a minimal outcome. Additionally, the uncertainty prediction has to be meaningful in such a way that it converges towards the noise variance (i.e. 0 for deterministic problems) the more we know about a specific point and it has to increase the further we are away from known points.

3.3.1 Gaussian Process Regression (Kriging)

The Gaussian process regression is the standard surrogate for model-based optimization, especially if referred to by *Bayesian optimization*. The relationship between the label ν and the covariates θ_l is modeled by a Gaussian process with an a-priori mean function and an a-priori covariance function. The following description corresponds to the derivations in Rasmussen et al. (2006).

To understand how the regression works, we first acknowledge that a Gaussian process is completely specified by its mean function $m \colon \Theta \to \mathbb{R}$ and its covariance function $k \colon \Theta \times \Theta \to \mathbb{R}$. Accordingly, if we model the objective function f using

Gaussian process regression we assume that:

$$f \sim GP\,(m, k)\,. \tag{3.3.1}$$

Following the definition of a Gaussian process, we can now assume that for any finite set of points, given as matrix $X = (\theta^{(1)}, \ldots, \theta^{(n)})'$, the outcome

$$\mathbf{f} = (f(\theta^{(1)}), \ldots, f(\theta^{(n)}))' \tag{3.3.2}$$

follows a multivariate normal distribution:

$$\begin{aligned} \mathbf{f}|X &\sim \boldsymbol{\mathcal{N}}(\mathbf{m}(X), K(X,X) + \sigma_n^2 I), \text{ with} \\ \mathbf{m}(X) &= (m(\theta_X^{(1)}), \ldots, m(\theta_X^{(n)}))' \end{aligned} \tag{3.3.3}$$

and $K(X, Z) \in \mathbb{R}^{n_X \times n_Z}$ as the matrix of all pairwise covariances given by the function k where the entry at position (i, j) contains the covariance between the i-th row $\theta_X^{(i)}$ in X and the j-th row $\theta_Z^{(j)}$ in Z given by $k(\theta_X^{(i)}, \theta_Z^{(j)})$. Furthermore, σ_n^2 is the noise variance of the outcomes, also called nugget effect, if we assume that $\nu = f(\theta) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ iid., i.e. the noise is assumed to be homoscedastic. If the function is assumed to be deterministic, we set $\sigma_n^2 = 0$. As a consequence, a single outcome follows a normal distribution:

$$f(\theta) \sim \mathcal{N}(m(\theta), k(\theta, \theta) + \sigma_n^2). \tag{3.3.4}$$

Accordingly, $m(\theta)$ and $k(\theta, \theta)$ directly give us the desired mean prediction $\hat{\mu}(\theta)$ and uncertainty prediction $\hat{s}(\theta)$ for any point θ.

So far we specified the Gaussian process without prior knowledge. For Gaussian regression we look at the Gaussian process that is conditioned on our training observations of function outcomes ν (as realizations of $\mathbf{f}$) with the corresponding input values in matrix X from the training data $\mathcal{D}$. For any finite set of random variables $\mathbf{f}_*$, with the associated values X_* coming from the conditional Gaussian

process, the conditional Gaussian distribution can then be derived to:

$$\begin{aligned}\mathbf{f}_*|X_*, X, \mathbf{f} \sim \boldsymbol{\mathcal{N}}(&\mathbf{m}(X_*) + K\left(X_*, X\right)\left(K(X,X) + \sigma_n^2 I\right)^{-1}\left(\mathbf{f} - \mathbf{m}(X)\right), \\ &K(X_*, X_*) - K(X_*, X)\left(K(X,X) + \sigma_n^2 I\right)^{-1} K(X, X_*)).\end{aligned} \tag{3.3.5}$$

Similar to Equation (3.3.4) the mean prediction $\hat{\mu}(\theta)$ can directly be obtained from the mean parameter of the distribution and by replacing $\mathbf{f}$ with the observed realizations ν and setting $X_* = \theta'$. The same holds for the uncertainty estimation $\hat{s}(\theta)$ which is then equivalent to the second parameter of the Gaussian distribution.

Still we need to determine m and k before we can calculate the predictions. For MBO m is usually set to be a constant ($m(\theta) = \beta_0$) and k is usually calculated with some distance-based kernel estimator C using the Euclidean distance between two points. Snoek et al. (2012) propose to use the Matérn 5/2 Kernel which is also used in the Bayesian optimization framework *Spearmint* and will also be used for this work. So we calculate

$$\begin{aligned}k(\theta_X, \theta_Z) &= \sigma_f^2 \cdot C(|\theta_X - \theta_Z|), \text{ with} \\ C(r) &= C_{5/2}(r) = \left(1 + \frac{\sqrt{5}r}{\rho} + \frac{5r^2}{3\rho^2}\right) \cdot \exp\left(-\frac{\sqrt{5}r}{\rho}\right) ,\end{aligned} \tag{3.3.6}$$

σ_f^2 as the *signal* variance parameter or also called covariance amplitude (Snoek et al., 2012) and ρ as the scaling parameter.

If we assume the choice of the kernel as fixed this leaves us with the positive parameters β_0, σ_f^2, σ_n^2 and ρ. These parameters can be obtained by optimizing the marginal likelihood which is explained in detail in Rasmussen et al. (2006). We have to note that the maximum likelihood optimization itself forms an optimization problem which cannot be reliably solved for certain priors by many implementations. Especially for small values of r the likelihood becomes numerically instable. Therefore, in `mlrMBO` points that are too close to each other are avoided. Furthermore, it can happen that the likelihood maximization leads to values of ρ close to 0, so that the kernel (see Equation (3.3.6)) estimates a covariance of 0 for nearly all distances. This has the effect that the prediction is a flat surface with $\hat{\mu}(\theta) = \beta_0$ and $\hat{s}^2(\theta) = \sigma_f^2 + \sqrt{\sigma_n^2}$.

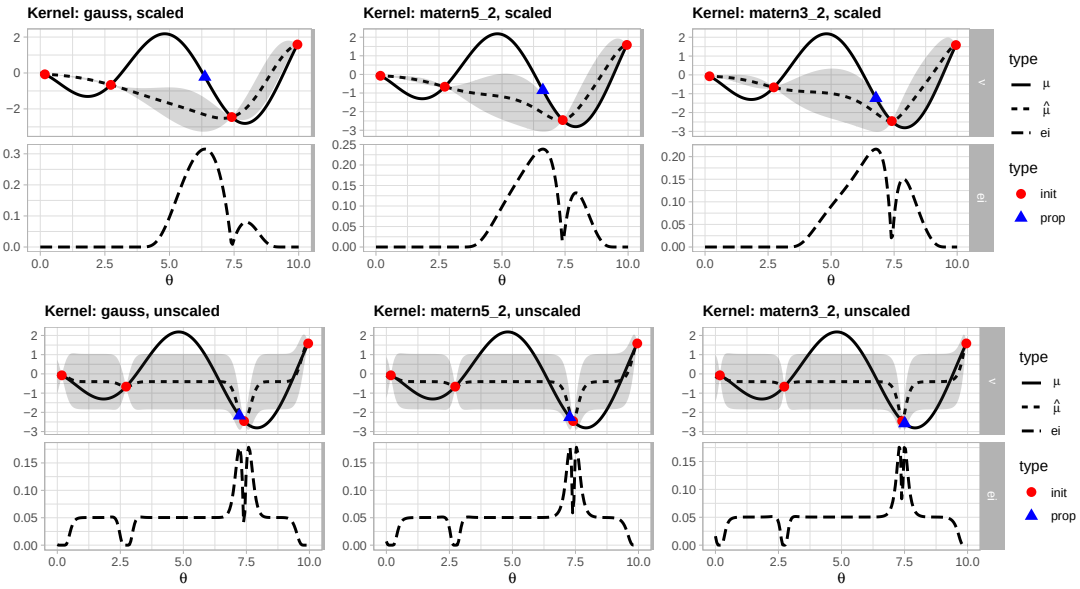

Figure 3.3: Exemplary influence of the kernel on the estimation $\hat{\mu}$ (---) and $\hat{s}$ (gray area). All plots show the surrogate model fitted on the initial design (●). In each column the Kriging is fitted using one of the three kernels *Gaussian*, *Matérn* 5/2 and *Matérn* 3/2. In the top row the input θ is scaled to $[0, 1]$, whereas for the bottom row the input remains untransformed. The true objective function is drawn in a solid line (—). The lower part of each plot shows the *acquisition function* that is maximized to obtain the next proposal (▲).

Furthermore, it becomes clear that with the choice of the Euclidean distance, Gaussian process regression is only well defined for a purely real-valued search space Θ and the length scale of each covariate matters. As a consequence it is often preferable to scale all covariates θ_l to $[0, 1]$, if no further knowledge is available. In Figure 3.3 a Gaussian process regression is shown for different choices of the Kernel and with and without scaling θ_l to $[0, 1]$. This example demonstrates, that the scaling of the covariates to $[0, 1]$ strongly influences the predictions. In contrast, the choice of the kernel does not have such a dramatic impact in this example, as the predictions appear similar. However, small differences lead to different optima in the acquisition function and over multiple iterations the generated proposals within the MBO procedure might differ from optimization runs with different kernels.

Summing up, we have defined Gaussian process regression for real-valued covariates. In this work only covariates of such domain will be used. Gaussian regression for categorical covariates and for mixed domains is a dedicated research topic. However, in some particular cases common techniques to decode categorical covariates into numerical covariates, such as one-hot encoding, might work.

3.3.2 Other Methods

As mentioned, Kriging has desirable properties of extrapolation and interpolation. For purely categorical domains specific distance measures or kernels exist but their usage for MBO is not widespread and still subject to research (Swiler et al., 2014; Horn et al., 2019; Ru et al., 2019). A more established way to deal with categorical variables in Θ is to use an alternative regression method within the MBO framework. Any regression method that supports categorical covariates and offers an uncertainty estimation can potentially be used. A popular choice is the random forest, which was successfully applied by Hutter et al. (2011) and by Bischl, Richter, et al. (2017) to optimize over mixed-valued search spaces. However, the random forest has certain drawbacks, which are illustrated in Figure 3.4. The advantage of not making any assumptions on the covariates and their relationship to the label comes at a cost of two major drawbacks: First, the random forest yields no extrapolation and can only predict values within the range of the training values, as the prediction in unseen areas is only the mean of those observations that are in the same terminal nodes of the individual trees. Second, uncertainty estimation of the random forest is the highest where the individual trees have disagreeing predictions, which is usually the case in areas where the target function is very steep. Both drawbacks have to be taken into account to avoid proposals in areas that are not of interest.

In other regression methods the tree approach is used to divide the search space into purely numerical subspaces. As soon as a node of the tree consists of a purely numerical space a normal Gaussian regression can be fitted. Such a method is proposed as *Bayesian Treed Gaussian Process* in Gramacy et al. (2008). Originally,

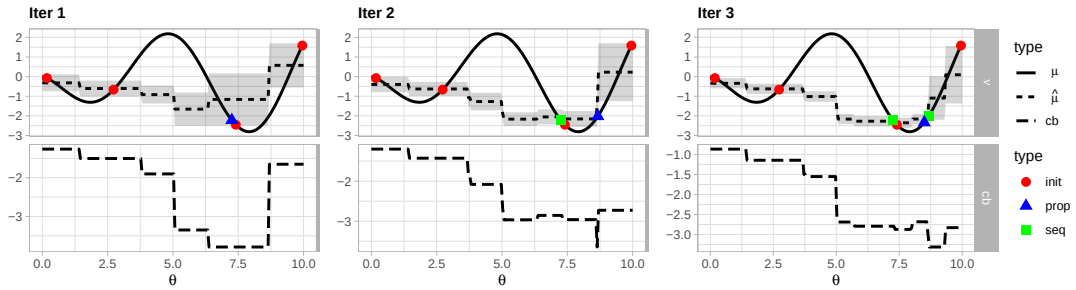

Figure 3.4: Same initial situation as in Figure 3.3. Here, we see three exemplary MBO iterations with a random forest with 100 trees and a minimal node size of one as surrogate model. In this example, the chosen acquisition function has to be minimized. In iteration 1 the wide plateau of the acquisition function leads to a proposal that is close to an already existing one. In iteration 2 the acquisition function has its local minima where the mean prediction has the highest jump.

it is developed to allow fitting non-stationary processes. The non-stationary process is split in the definition space Θ using a specialized tree. However, as Snoek (2013) pointed out, a problem lies at the borders of each segmentation. The Gaussian process that is fitted within each tree node is not aware of the points in neighboring segments. As a result the uncertainty is high at the borders seen from both sides. Therefore, a point in such regions will be often selected by the acquisition function and evaluating this point in one segment will not decrease the uncertainty in the neighboring segment. However, in each MBO iteration the segmentation of Θ can be different, potentially lowering the chance of over-evaluating certain areas. Assael et al. (2015) propose to only split at the observations instead of between them and to include the observations in both tree nodes.

Forrester and Keane (2009) consider the use of an SVM as a surrogate but note that the uncertainty estimation will converge to the noise of the real function. This is a problem with many surrogates (including tree based surrogates). Their uncertainty does not reflect the uncertainty of the model but only the noise in the underlaying training data. This makes the use of many acquisition functions problematic as they rely on higher uncertainty in regions where no observations are present.

3.4 Acquisition Functions

As mentioned, the idea of model-based optimization is that it is cheaper to optimize the surrogate than the true objective. Optimizing the surrogate directly would mean that instead of $f(\theta)$ we minimize $\hat{\mu}(\theta)$. If we assume that the surrogate is accurate, this would be a good idea. But SMBO is an iterative process for the reason to improve the accuracy in relevant areas of Θ. The strategy to evaluate the point θ^+ that minimizes $\hat{\mu}(\theta)$ in each iteration would likely not lead to an increased accuracy. For most situations such θ^+ would be identical or very close to the best observed point in $\mathcal{D}$, as illustrated in Figure 3.5.

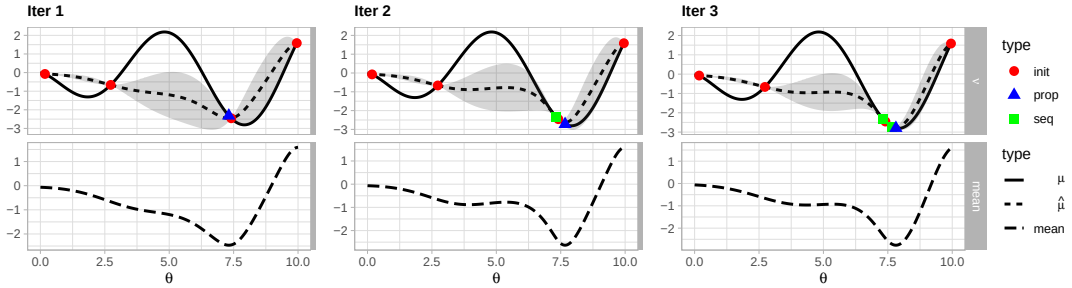

Figure 3.5: Visualization of MBO iterations, with the same starting conditions as in Figure 3.3. The *mean response* is drawn as a dashed line in the upper pannel. The lower panel shows the acquisition function which is identical to the mean response in this example. The point that minimizes the mean response is drawn as a triangle (▲). Once it is evaluated and part of the design $\mathcal{D}$ it is drawn as a rectangle (■).

To avoid this pure *exploitation* of the global minimum in $\hat{\mu}$, the optimization of the *acquisition function* replaces the direct optimization on $\hat{\mu}$. The aim of the acquisition function (also called *infill criterion*) is to deliver a good heuristic that drives the selection of the next proposal θ^+ within the MBO framework. To obtain the next proposal the acquisition function is maximized (or minimized, depending on the formulation):

$$\theta^+ := \arg\max_{\theta} \text{acq}(\theta). \tag{3.4.1}$$

In other words, the acquisition function should be high where there is a high chance of a global minimum of f. To achieve that, most acquisition functions combine the

surrogates estimation of the outcome $\hat{\mu}$ and the point-wise uncertainty $\hat{s}$, following the idea that the acquisition function should be high for low values of $\hat{\mu}$ and/or high values of $\hat{s}$. Various acquisition functions are discussed in the literature, whilst the *expected improvement* (EI) arguably is the most popular choice. However, the EI is built on the assumption that the posterior of ν is normally distributed. This cannot always be justified and motivates the choice of other acquisition functions such as the *confidence bound*.

Optimization of the acquisition function It is assumed that the surrogate yields cheap predictions $\hat{\mu}$ and $\hat{s}$ and as a consequence the evaluation of the acquisition function is also cheap. However, the optimization of the acquisition function forms its own optimization problem. For small and simple search spaces the acquisition function can be optimized reliably. Unfortunately, for high-dimensional search spaces it still remains a challenge because of the multimodality of the acquisition function and the *curse of dimensionality*. Additionally, as certain surrogates such as the random forest yield a discontinuous acquisition function, the use of gradient-based optimizers is restricted. Therefore derivative-free optimizers such as the evolutionary search algorithms *CMA-ES* (Hansen et al., 2001) are a popular choice. Also extensive iterative random search strategies are applied. The *focus search* that is used in `mlrMBO` (Bischl, Richter, et al., 2017) and also in this work falls into this category. In the default setting it evaluates a random design of 1000 points on the search space. The found optimal point then serves as the center of the new search space that is $\frac{1}{4}$th of the size in each dimension. These two steps are repeated five times.

As a side remark, in Section 3.3.1 it was stated that in order to obtain the Kriging hyperparameters a likelihood is maximized. It was also mentioned that this optimization can lead to constant predictions for $\hat{\mu}$ and $\hat{s}$. All presented acquisition functions are only dependent on these two values and as a consequence the acquisition function will be constant if these values are constant across Θ. In this case the optimization of a constant acquisition function will return a random point as θ^+. This problem was also highlighted in Benassi et al. (2011). They propose a Bayesian approach to incorporate the uncertainty of these parameters

into the calculation of the expected improvement, which results in a more stable optimization. In many practical implementations (also in the R-package `mlrMBO`) this problem is ignored with the hope that the design is big enough to reliably maximize the likelihood and obtain Kriging parameters that do not yield constant predictions.

3.4.1 Expected Improvement

The *expected improvement* is defined as the expected value of the potential improvement $I(\theta)$:

$$\text{EI}(\theta) := \text{E}(I(\theta)), \tag{3.4.2}$$

whereas $I(\theta)$ is the random variable that models the improvement over the currently best observed function value $\nu_{\min}$:

$$I(\theta) := \max\left\{\nu_{\min} - f(\theta), 0\right\}. \tag{3.4.3}$$

Here, $f(\theta)$ is interpreted as a random variable as in equation (3.3.4). Under the same assumptions as for the Gaussian process regression, $f(\theta)$ is normally distributed with $f(\theta) \sim \mathcal{N}(\hat{\mu}(\theta), \hat{s}^2(\theta))$. Under this assumption, $\text{EI}(\theta)$ can be expressed analytically (Jones et al., 1998) in closed form as

$$\text{EI}(\theta) = (\nu_{\min} - \hat{\mu}(\theta))\,\Phi\left(\frac{\nu_{\min} - \hat{\mu}(\theta))}{\hat{s}(\theta)}\right) + \hat{s}(\theta)\phi\left(\frac{\nu_{\min} - \hat{\mu}(\theta)}{\hat{s}(\theta)}\right), \tag{3.4.4}$$

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

In Figure 3.6 an exemplary optimization process with three iterations is given, where the EI is used as *acquisition function*. This example demonstrates, that the EI leads to exploratory (Iteration 1) and also exploitative proposals (Iteration 2 and 3) which in the consequence can lead to a fast optimization progress.

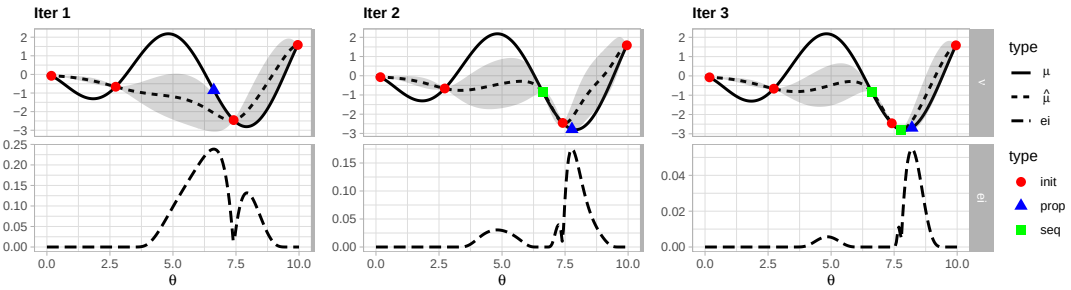

Figure 3.6: Visualization of MBO iterations, with the same starting conditions as in Figure 3.3. The EI acquisition function is drawn below as a dashed line (--). The optimal point of the acquisition function is drawn as a triangle (▲) at the respective position on the true outcome of the objective function. Once it is evaluated it is drawn as a rectangle (■).

3.4.2 Confidence Bound

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

$$\text{CB}(\theta, \lambda) = \hat{\mu}(\theta) - \lambda \hat{s}(\theta), \tag{3.4.5}$$

where $\lambda > 0$ is a hyperparameter that controls the "exploration vs. exploitation" trade-off. A higher value of λ implies more exploration. In Figure 3.7 the CB is illustrated for $\lambda = 2$. Note that for readability, in this formulation CB is actually

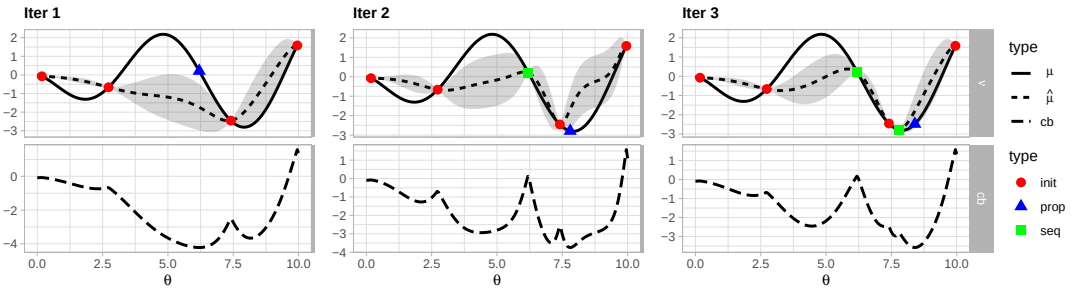

Figure 3.7: The same objective function and settings as in Figure 3.6 but with confidence bound with $\lambda = 2$ as acquisition function. Here, the CB is drawn as a minimization problem for an easier intuition.

minimized, contrary to our definition of the acquisition function. To comply with the definition the CB is multiplied with -1.

3.4.3 Augmented Expected Improvement

If the optimization problem is stochastic, i.e.

$$\theta^* = \arg\min_{\theta \in \Theta} f(\theta) + \varepsilon, \tag{3.4.6}$$

with $\varepsilon \sim N(0, \sigma_n^2)$ iid., the previous acquisition functions are not an optimal choice because they do not account for the noise in the observed function outcomes. For stochastic optimization special acquisition functions are proposed. The *augmented expected improvement* (Huang et al., 2006) is a popular choice and extends the *expected improvement* (Section 3.4.1).

Instead of using the best observed function value $\nu_{\min}$ to calculate the improvement it uses the *effective best solution*:

$$\theta^{**} := \arg\min_{\theta \in \mathcal{D}} \hat{\mu}(\theta) + c \cdot \hat{s}(\theta), \tag{3.4.7}$$

with c as a tuning parameter that is usually set to 1. It adds an uncertainty term to each observed function value to obtain a pessimistic estimate. An optimistic reference value would prevent re-evaluation of outcomes that perform well just due to noise.

The augmented expected improvement is calculated similarly to the EI as follows:

$$\begin{aligned} \text{AEI}(\theta) = (\hat{\mu}(\theta^{**}) - \hat{\mu}(\theta)) \cdot \Phi\left(\frac{\hat{\mu}(\theta^{**}) - \hat{\mu}(\theta))}{\hat{s}(\theta)}\right) \\ + \hat{s}(\theta) \cdot \phi\left(\frac{\hat{\mu}(\theta^{**}) - \hat{\mu}(\theta)}{\hat{s}(\theta)}\right) \cdot \left(1 - \underbrace{\frac{\sigma_n}{\sqrt{\sigma_n^2 + \hat{s}^2(\theta)}}}_{\text{correction}}\right), \end{aligned} \tag{3.4.8}$$

with σ_n^2 denoting the variance of the random error (nugget effect) in the Kriging model. The higher the model uncertainty $\hat{s}^2(\theta)$, the smaller the correction term. The closer $\hat{s}^2(\theta)$ gets to σ_n^2, the bigger the correction. This means that the AEI is decreased in areas where the surrogates uncertainty estimation is close to noise of the function. This avoids additional evaluations in areas where the surrogate has sufficient knowledge. The example in Figure 3.8 shows that the AEI favors more exploitative evaluations to increase the model certainty in a specific area.

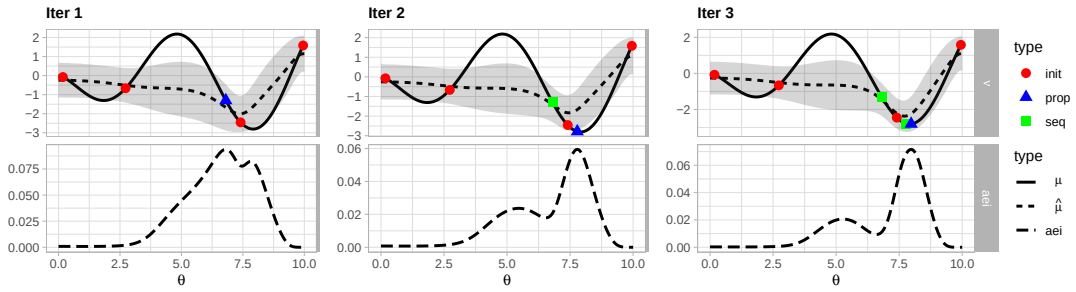

Figure 3.8: The same objective function as in Figure 3.6 but with the *augmented expected improvement* as acquisition function. The Kriging parameter σ_n^2 is fixed to 0.5 which results in an overall higher uncertainty estimation (gray area).

In this example we set $\sigma_n^2 = 0.5$ even though the function is deterministic. In practice the noise variance is often not known and will be estimated as one parameter of the Kriging regression. Other surrogates might not be able to give an estimate for σ_n^2. In such a case one could argue that the residual variance can be taken as a replacement.

3.4.4 Other Acquisition Functions

The presented acquisition functions just reflect a selection of the most commonly used ones. There is also active research on this field. In some early texts the *probability of improvement* is often suggested as an acquisition function but as it only maximizes the probability and neglects the amount of improvement it is known to not work well (Jones, 2001). In Srinivas et al. (2012) an optimal choice for the λ parameter of CB is inferred under certain conditions. In the overview paper

by Shahriari et al. (2016) other popular strategies such as *Thompson sampling* (TS) and *entropy search* (ES) are mentioned. For TS the acquisition function is a sample of the conditional Gaussian process and it is assumed that this strategy has an exploratory characteristic. ES measures the expected information gain from evaluating a certain point but can only be calculated for discrete search spaces. The *predictive entropy search* removes this problem and can be calculated via MC sampling. It is also highlighted that no single acquisition function works best over all problems and that in fact the optimal criterion can change during the course of optimization. Therefore, strategies that combine multiple acquisition functions such as the *entropy search portfolio* have been proposed.

3.5 Termination

Usually the termination of the MBO procedure is defined by a budget. This can either be the number of optimization iterations, function evaluations, a maximum time span or a combination of those. It is not suggested to terminate based on the progress that is made in the last number of iterations, because it can happen that the progress appears to be zero before a jump happens. If MBO does exploratory evaluations the best found value does not necessarily improve over a number of iterations but the overall knowledge increases and a jump into a new valley becomes more likely. Another argument against progress-based termination criteria is that popular assumptions like convexity or Lipschitz continuity on the objective function cannot be made, although Pintér (2013) states that the Lipschitz constant can be estimated during optimization and it can be assumed that it holds for most objectives. Anyhow, it is impossible to make accurate conclusions about the future progress of an ongoing optimization based on the past optimization path and therefore we only use budget-based termination in this work.

Calculation of the Final Result When the optimization has terminated, the final result $\hat{\theta}^*$ has to be determined. If the function is deterministic (i.e. $\sigma_n^2 = 0$) the

most common option is to choose the point $\theta^{(i)}$ from the design $\mathcal{D}$ that yielded the minimum value for ν.

Another option is to determine the point θ that minimizes the surrogate mean prediction $\hat{\mu}(\theta)$, with the drawback that the function value is uncertain unless we do obtain it in an additional evaluation. In case this evaluation reveals that the last value is worse than the best point from the design, we would return the best point from the design.

In case the function is stochastic, each observation in the design does not reflect the true mean of the objective function. Following the first presented strategy could lead to a result that was just chosen because the one evaluation yielded the best outcome "by chance". If we obtain $\hat{\theta}^*$ by minimizing $\hat{\mu}(\theta)$ we risk to obtain a result in an area where the surrogate is uncertain. To avoid that we may restrict the minimization to the points we already know:

$$\hat{\theta}^* := \arg\min_{\theta \in \mathcal{D}} \hat{\mu}(\theta). \tag{3.5.1}$$

In other words, we assume that for already evaluated points the surrogate mean prediction is close to the true mean of the objective function.

3.6 Related Work

Model-based optimization and Bayesian optimization is subject of recent and ongoing research.

Besides the R-package `mlrMBO` (Bischl, Richter, et al., 2017), a further major software contribution is the SMAC (Hutter et al., 2011) framework that uses random forests as a surrogate and promotes itself to efficiently optimize over wide categorical search spaces. The Python package `hyperopt` (Bergstra et al., 2013) does not use a surrogate that models the function outcome but uses a tree-structured Parzen-estimator that yields the desired probabilities for the EI directly.

Adoptions and special methods for various scenarios have been proposed to the model-based optimization framework. The following section gives a brief overview of interesting related topics that are not dealt with in this thesis.

Sometimes a cheaper objective function exists that correlates with the expensive objective function. In Huang et al. (2006) and Forrester, Sóbester, et al. (2007) Co-Kriging is used to incorporate the knowledge of outcomes of the cheaper function to estimate and optimize the expensive function and as a consequence, reduce the number of necessary expensive function evaluations. In Richter (2015) the method is generalized to be used for arbitrary surrogates and for an array of cheaper objective functions that increase in cost and fidelity representing the expensive objective. This is particularly practical for optimizing the hyperparameters of machine learning methods as cheaper objective functions of lower fidelity can be acquired by evaluating the performance of the machine learning method on a subsample of the training data.

The performance of MBO methods on high-dimensional optimization problems can still be seen as doubtful as there are little publications on real applications on that topic. Following the *curse of dimensionality*, the number of design points needed by the surrogate to model the objective increases dramatically. As pointed out in Z. Wang et al. (2013) and Wang (2016) usually not all dimensions have a significant influence on the outcome. Therefore they propose to transform the design into a lower dimension using a random projection in each iteration. On a set of synthetic functions Wang (2016) and Munteanu et al. (2019) were able to show that for selected problems the optimization on the lower dimensional projections can still reflect the optimization problem and can be handled better by the surrogate.

4 Parallel MBO

In this chapter the adaptations that are necessary to enable the MBO framework to evaluate multiple black-boxes in parallel are presented. Section 4.1 gives the motivation for parallel MBO and introduces some specific terminology and key concepts. The concepts can be divided in synchronous (Section 4.2) and asynchronous (Section 4.3) methods. Each section presents the specific methods used in this thesis, whereas the newly proposed synchronous method *RAMBO* is presented in its own Section 4.5. The RAMBO framework was first presented in Richter, Kotthaus, Bischl, et al. (2016) and extended with the priority refinement in Richter, Kotthaus, A. Lang, et al. (2017).

4.1 Prerequisites

As already stated, MBO is a popular technique for global optimization of expensive black-box functions. Deviating from the sequential formulation that was given in Chapter 3 and that is depicted in Figure 4.1, it is often indispensable to apply parallelization to speed up the optimization. This is usually achieved by evaluating

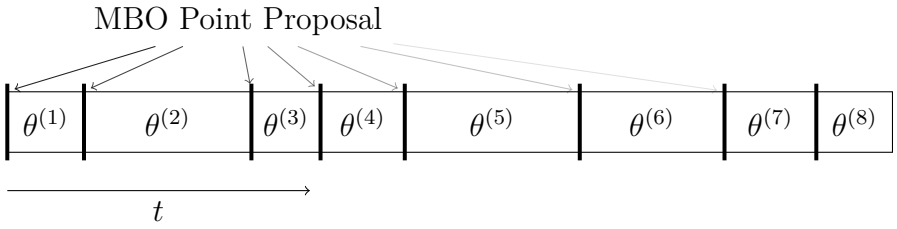

Figure 4.1: Sequential MBO on one worker.

as many different input configurations of the objective function per iteration as there are available workers that can evaluate the black-box.

In the context of algorithm configuration (i.e. optimization of parameters of a computer algorithm) such worker usually is a single core of a CPU that evaluates the algorithm exclusively. It has to be noted that many algorithms can make use of parallel hardware systems themselves. For instance, in machine learning, the performance of a specific method and its hyperparameter setting is usually evaluated by conducting a k-fold cross-validation. These k folds form independent problems that can be evaluated in parallel. Even on a smaller scale a single evaluation can often be parallelized, e.g. for a random forest, each single tree can be constructed independently and can form a separate process. These independent processes can be parallelized naturally. Similar examples can be found for many objective functions. Nevertheless, functions remain that cannot be parallelized or that cannot be parallelized to to an extend that makes use of all resources. For instance, if we have an implementation of an objective function (e.g. a machine learning algorithm) that can be evaluated on one multi-core system but not on multiple computers, we would want to evaluate this algorithm in parallel on all available computers in a cluster. Therefore we need an additional layer of parallelization. In the following, we ignore any possible internal parallelization of the objective and assume that parallel evaluation of the objective during optimization is necessary in order to utilize all available resources.

> **Definition 4.1.1.** The instance that evaluates the objective function will be called **worker**, no matter if it is a single CPU or a group of CPUs within a bigger system. The total number of available workers will be denoted with k.

The focus of this chapter is to introduce methods that optimize an objective function in a situation where multiple workers are available.

The proposal of multiple optimization parameters in one MBO iteration is topic of various publications which are summarized in Section 4.2. If the number of such multi-point proposals within one iteration equals the number of available

workers, we call this synchronous parallelization. The proposal of multiple points per iteration is also called batch proposal. The parallel evaluation of a batch leads to problems if runtimes of the objective function are heterogeneous. As depicted in Figure 4.1, heterogeneous runtimes do not create a problem for sequential MBO, regarding worker utilization. Unfortunately, as shown in Figure 4.2, if the same configurations were evaluated in parallel, idling times would occur, because some workers finish faster than others.

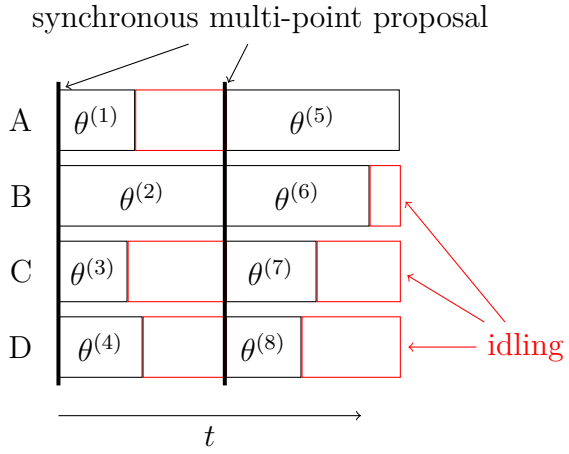

Figure 4.2: Parallel MBO with synchronous multi-point proposal on four workers. Black boxes represent time intervals with evaluations. Red boxes represent time intervals with idling.

Definition 4.1.2. A single evaluation of a proposal $\theta^{+(i)}$ on the objective function f forms a **job**.

Workers that finish earlier than others of the same batch will idle until all workers have finished their jobs and the MBO procedure generates a new batch of proposals. In consequence, resources are wasted by waiting, because the time could instead be used to evaluate more configurations of the objective function.

Care has to be taken to propose and evaluate configurations in such a way that idle times are avoided. This will ultimately lead to a faster optimization as more evaluations can be carried out in the same amount of time. The strategy to avoid

idling presented in this thesis is based on applying common scheduling approaches on a set of proposed points. It is described in detail in Section 4.5.

Another approach to avoid idling times is to establish an asynchronous parallelization. Here, all workers share the knowledge about previously evaluated points and points that are currently under evaluation. Based on this information each worker generates its own proposal. Therefore, this approach efficiently eliminates idling times as illustrated in Figure 4.3. This comes with the challenge to take the ongoing

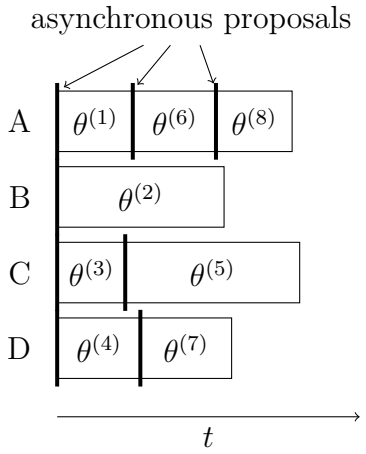

Figure 4.3: Asynchronous MBO on four workers.

evaluations into consideration for the point proposal. If ongoing evaluations are neglected, it can happen that a new proposal is very similar to a point that is under evaluation. The proposal is generated because the uncertainty in the proximity of the point under evaluation is still estimated to be high. Different strategies to incorporate points that are under evaluation into the proposal are explained in Section 4.3.

As a final remark to the introduction: If the evaluation of the objective function can be efficiently parallelized itself, it should be considered to do so, instead of parallelizing the optimization. The profit that each evaluation brings to the optimization is the highest for sequential MBO, because in each iteration the one single proposal is generated based on maximal knowledge, i.e. all previous evaluations. However, for multiple proposals within a single iteration each additional

proposal is "just an elaborate guess", which lowers the expected profit of each additional evaluation. Therefore, reducing the number of parallel proposals makes the optimization more efficient per evaluation.

4.2 Synchronous Parallelization

We have defined the sequential MBO procedure in Listing 1. It becomes clear that each iteration depends on the results of the previous iteration. Therefore, it is sequential by design and there is no apparent way to make use of multiple workers, e.g. parallel hardware systems, within this algorithm.

In a scenario where k workers are available, a natural idea is to propose k different configurations in each MBO iteration. This is called *synchronous parallelization* because at each iteration the results of the evaluations of the objective function are fed back to the surrogate synchronously. There are various strategies to obtain multiple proposals in each iteration. For importance in this work are the *qCB* method (Hutter et al., 2012) and the *Kriging believer* approach (Ginsbourger, Le Riche, et al., 2010). Others are listed as related work in Section 4.4.

Multi-point proposals derive not only one single proposal θ^+ from a surrogate model, but q proposals $\theta^{+(1)}, \ldots, \theta^{+(q)}$ simultaneously. The q proposed points should be sufficiently different from each other to avoid multiple evaluations of nearly identical input configurations. The determination of what is "sufficiently different" is a major concern. Since f is a black-box, a slight change of θ_l can result in a nearly identical outcome, or in a completely different outcome of f. Although we treat f as a black-box, we assume the first case, i.e. that the function is "sufficiently smooth".

4.2.1 Multiple Proposals by Multiple Confidence Bounds

The *qCB* criterion is one of the simplest methods to acquire multiple proposals. It was introduced by Hutter et al. (2012) as follows:

$$\text{qCB}(\theta, \lambda_j) = \hat{\mu}(\theta) - \lambda_j \hat{s}(\theta) \ , \text{ with } \lambda_j \sim \text{Exp}(\Lambda^{-1}) \tag{4.2.1}$$

and is an intuitive extension of the *CB* criterion (see Section 3.4.2).

To obtain q proposals we first sample q values $\lambda_1, \ldots, \lambda_q$ from the exponential distribution with an expected value of Λ (e.g. $\Lambda = 2$). This results in q different acquisition functions. Each can be optimized independently and will lead to q different proposals $\theta^{+(1)}, \ldots, \theta^{+(q)}$.

Since the λ_j controls the impact of the standard deviation, sampling multiple λ_j varies the trade-off between exploration and exploitation of each proposal. It has to be noted that the choice of Λ^{-1} as the parameter of the exponential distribution can have a high effect on which points are proposed, similar to the choice of λ for the normal *CB* criterion. A low value of Λ results in low values λ_i which result in exploitative proposals that are close to minima of the surrogate. A high values of Λ results in high values λ_i which generates exploratory proposals. There is no guarantee that proposals obtained by different values of λ are sufficiently different from each other, as illustrated by Figure 4.4.

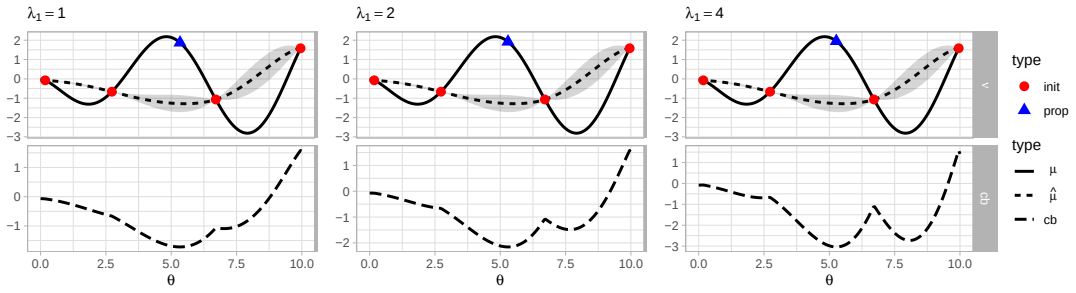

Figure 4.4: The same objective function and settings as in Figure 3.6 but with the *qCB* acquisition function and three different values for λ_j. We see that in this example, even though the λ values vary, nearly the same points are proposed.

Algorithm 2 Surrogate Believer Multi-point MBO Algorithm

Require: expensive black-box function $f(\theta)\colon \Theta \to \mathbb{R}$, acquisition function $\text{acq}(\theta)\colon \Theta \to \mathbb{R}$

1: sample n_{init} points from Θ: $\{\theta^{(j)} \mid j = 1, \ldots, n_{\text{init}}\}$
2: evaluate $\nu^{(j)} \leftarrow f(\theta^{(j)})$ for $j = 1, \ldots, n_{\text{init}}$
3: $D \leftarrow \{\left(\theta^{(j)}, \nu^{(j)}\right) \mid j = 1, \ldots, n_{\text{init}}\}$
4: **while** budget is not exhausted **do**
5: $\quad \tilde{\mathcal{D}} \leftarrow D$
6: $\quad$ **for** $i \in \{1, \ldots, q\}$ **do**
7: $\quad\quad$ fit surrogate model on $\tilde{\mathcal{D}}$
8: $\quad\quad$ propose new point $\theta^{+(i)} \leftarrow \arg\max_\theta \text{acq}(\theta)$
9: $\quad\quad$ $\hat{\nu}^{(i)} \leftarrow f(\theta^{+(i)})$
10: $\quad\quad$ $\tilde{\mathcal{D}} \leftarrow \tilde{\mathcal{D}} \cup \{\left(\theta^{+(i)}, \hat{\nu}^{(i)}\right)\}$
11: $\quad$ evaluate $\nu^{(i)} \leftarrow f(\theta^{+(i)})$ in parallel for $i = 1, \ldots, q$
12: $\quad$ $D \leftarrow D \cup \{\left(\theta^{+(i)}, \nu^{(i)}\right) \mid i = 1, \ldots q\}$
13: **return** optimal setting $\hat{\theta}^* = \arg\min_{\theta \in D} f(\theta)$

4.2.2 Surrogate Believer

The name *Kriging believer* which was coined by Ginsbourger, Le Riche, et al. (2010) is a bit unfortunate because the method can be applied for any surrogate method. Therefore, it will be called *surrogate believer* in this work. In theory it can also be used with any acquisition function, although it is most commonly used in connection with the *expected improvement*.

It extends the sequential MBO method with additional steps outlined in Algorithm 2. The first point $\theta^{+(1)}$ is proposed using the acquisition function (e.g. EI), exactly as in the sequential MBO method. Now, to obtain a second proposal, the design $\mathcal{D}$ of evaluated points is extended with $\theta^{+(1)}$ and a fake value for ν. This fake value is obtained from the surrogate mean prediction for this proposal: $\hat{\nu}^{(1)} = \hat{\mu}(\theta^{+(1)})$. The design containing the fake value(s) is denoted as $\tilde{\mathcal{D}}$. This decreases the model uncertainty around $\theta^{+(1)}$. In effect, the EI in this area decreases so that the following proposals will be not in the direct vicinity. Afterwards, the surrogate is trained on $\tilde{\mathcal{D}}$ and the acquisition function is used to obtain the second proposal $\theta^{+(2)}$. To obtain further proposals we continue the same way and add $\theta^{+(2)}$ and

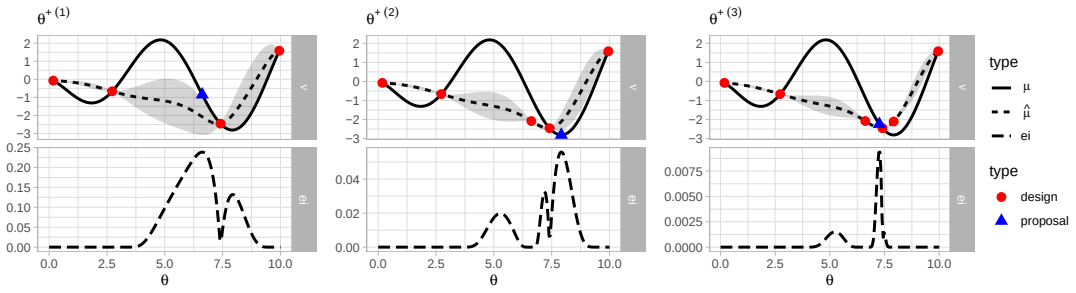

Figure 4.5: The same objective function and settings as in Figure 3.6. Here, we create $q = 3$ proposals after the initial design with the surrogate believer approach. The EI is used as acquisition function and drawn on the bottom of each panel. The first panel shows the initial design (●) and the obtained proposal (▲). The proposal is not evaluated but included as a temporary fake value in the design in the second panel. Since the fake value is taken from the mean prediction (---) the new point in the temporary design does not change the mean prediction but only the estimated uncertainty (gray area). The second proposal is obtained from maximizing the EI that is based on the temporary design. The third proposal is generated likewise by including both previous fake values in the design.

the fake value $\hat{\nu}^{(2)} = \hat{\mu}(\theta^{+(2)})$ to the design $\tilde{\mathcal{D}}$. These steps are continued until all desired q proposals are generated. In Figure 4.5 these steps are visualized for $q = 3$ proposals after the initial design.

4.3 Asynchronous Parallelization

Asynchronous execution approaches the problem of parallelizing MBO from a different angle. In the synchronous case, the problem can occur that an evaluation of the objective function takes much longer than the others. This results in idle workers, as a new batch of evaluations is only started after all evaluations have finished.

Instead of calculating the set of proposals in a single batch, in the asynchronous setting each proposal is generated on each worker independently. Instead of waiting for a new set of proposals, each worker calculates its own proposal $\theta^{+(i)}$ and evaluates it directly. This means that whenever one worker has finished its evaluation it can

directly start with the next one, therefore effectively reducing idle times to zero. Figure 4.3 on page 38 illustrates this method for four workers.

Asynchronous MBO introduces new challenges in comparison to its synchronous counterpart. On a technical side, each worker has to be able to access the design $\mathcal{D}$ of all evaluated points. Some asynchronous methods require additional communication between workers, as we will see. The main concern is, similar to multi-point proposal for synchronous MBO, to not evaluate similar points on multiple workers at the same time. However, this challenge occurs in different forms in asynchronous MBO. These challenges can be differentiated in three cases:

Case A: A worker finishes while all others are evaluating jobs. This is the most considered case in literature on asynchronous optimization. Here, one worker has just finished evaluating a job at t_{now} and is about to generate a new proposal to evaluate. In this case the worker has the knowledge of all evaluations completed until time point t_{now}. Also the additional knowledge about the input configurations $\theta^{+(i)}$ that are currently under evaluation is completely available.

Case B: A worker finishes while at least one worker is generating a proposal. This case seems to be neglected in literature. It can be argued that it does not need to be taken into account because the runtime of a single job is much longer than the generation of a proposal. Therefore, the chance that this case occurs is so small that it can be neglected. However, the chance that the termination of a job falls into the time window of another worker generating a proposal increases with the amount of used workers. The chance additionally increases because, depending on the method, the generation of a proposal takes longer the more pending evaluations have to be taken into account.

Case C: Multiple workers start the proposal at the same time. At first glance this appears to be a very improbable case. However, this case occurs if we start all workers on the optimization on a given design $\mathcal{D}$ at the same time, e.g. at the beginning of the optimization with a given initial design.

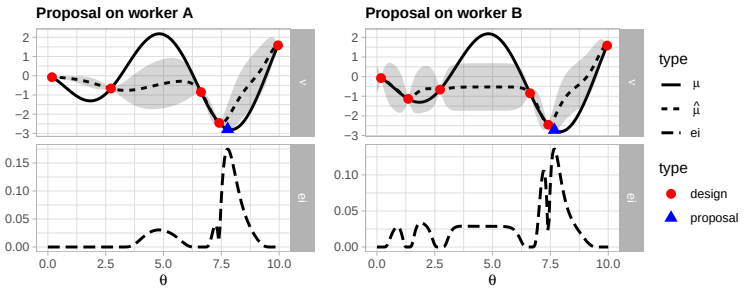

Figure 4.6: In this example worker A generates a proposal (▲) based on an initial design (●) of five points using the EI acquisition function (drawn at the bottom). Worker B generates a proposal in the same way with an additional point in the design. We can see that the additional point changes the EI but not the proposal that is generated.

The easiest solution is to use a stochastic proposal generation mechanism. This can be a surrogate that is stochastic (e.g. the random forest) or a stochastic acquisition function, e.g. the CB criterion with a randomly sampled value for λ similar to the multiple proposals by multiple confidence bounds in Section 4.2.1 or both options combined as proposed in Hutter et al. (2012). If the proposals are diverse for the same given design $\mathcal{D}$, it will prevent the evaluation of similar points for all presented cases and there is no direct need to include the knowledge of pending evaluations. However, it has the drawback that not all available knowledge is incorporated in the proposal generation.

If the proposal mechanism is deterministic, as it would be if we combined the Gaussian process regression as surrogate with the expected improvement, two proposals generated on the same design $\mathcal{D}$ will be identical. Even if the designs are different, the obtained proposal still can be very similar as the example in Figure 4.6 demonstrates for an exemplary *Case A*. Nearly identical proposals can occur if a point that does not affect the optimum of the acquisition function is added to the design. Such a point might be far away from the optimum of the acquisition function and the according outcome ν might not indicate a global minimum. If the optimum of the acquisition function does not change with a new point, the subsequent proposal also does not change. We recall that for Gaussian process regression the influence of a training point (e.g. the evaluation of worker B that is

added to the design) on the prediction of another point depends on the distance between both, the scaling, the chosen kernel and the kernel parameters that are data dependent themselves. In another case the response surface might only change insignificantly if the result is in accordance with the outcome that the surrogate already predicted.

Having this problem in mind, we introduce different strategies for each case. For **Case A** we need to take the pending evaluation of worker A into consideration to avoid proposing a point that is similar to $\theta^{+(A)}$. The *Expected Expected Improvement* (Section 4.3.1) and the *Surrogate Believer* (Section 4.3.2) are two strategies presented in this work to incorporate pending evaluation into the proposal.

If we neglect **Case B** it can happen that worker B is proposing the same point as worker A, for the same reason as in *Case A*. As worker B does not now which proposal worker A will generate it can happen that it will propose a similar point as worker A. As there is no way of obtaining knowledge about the point that worker A will propose, the only option is to wait until worker A has generated the proposal. Afterwards, worker B can continue as in *Case A*. Another option is to neglect this case, as it might happen rarely as mentioned before.

If *Case B* occurs often, it might be worth to also consider **Case C** for the following reason: If multiple workers are waiting for worker A to finish generating a proposal, they can either be put into a *queue* or be *grouped*. In the *queue* each worker has to wait for the previous worker to finish its proposal generation until the worker generates its own proposal. The time spent waiting can sum up to a significant amount of time. For the *group* we can generate multi-point proposals as soon as the worker A finished its proposal. Therefore, we could combine methods from the synchronous parallelization (Section 4.2) and the asynchronous parallelization to generate multiple proposals that incorporate the pending evaluations.

In the following, we will mainly focus on *Case A* in combination with deterministic point proposals. Therefore, the following two methods incorporate the ongoing evaluation in their point-proposal.

4.3.1 Expected Expected Improvement

Ginsbourger, Janusevskis, et al. (2011) motivate the expected EI (EEI) by stating that for the proposals which are under evaluation we know the distribution of the outcome ν. The distribution of the outcome is obtained through the surrogate, i.e. the Gaussian process regression. This allows us to incorporate the uncertainty about the outcome ν in the EI, which results in the EEI. Given b workers that are currently evaluating jobs, the unknown values of $f(\theta_{\text{busy}}^{(j)})$ with $j \in \{1, \ldots, b\}$ and $1 \leqslant b \leqslant k$ are integrated out via Monte Carlo sampling, which is computationally demanding.

For each Monte Carlo iteration values $\tilde{\nu}_{1,\text{busy}}, \ldots, \tilde{\nu}_{b,\text{busy}}$ are drawn from the posterior distribution of the surrogate regression model at $\theta_{\text{busy}}^{(1)}, \ldots, \theta_{\text{busy}}^{(b)}$. In each Monte Carlo iteration, these values are combined with the set of already known evaluations to form a temporary design with fake values. This temporary design is used to fit the surrogate model and to obtain an EI function. The EEI can then simply be calculated by averaging the individual expected improvement functions that are deduced from each Monte Carlo sample:

$$\widehat{\text{EEI}}\,(\theta) = \frac{1}{n_{\text{sim}}} \sum_{i=1}^{n_{\text{sim}}} \text{EI}_i(\theta) \ , \tag{4.3.1}$$

whereas n_{sim} denotes the number of Monte Carlo iterations.

4.3.2 Surrogate Believer

The Kriging believer approach from Section 4.2.2 can be adapted to incorporate the pending evaluations of $\theta_{\text{busy}}^{(j)}$ with $j \in \{1, \ldots, b\}$, whereas b denotes the number of pending evaluations. It can be seen as a simplification of the EEI from the previous Section 4.3.1. We ignore the fact that the outcome of $f(\theta_{\text{busy}}^{(j)})$ is not known exactly and treat each pending evaluation as if its outcome can be directly predicted by the surrogate. Therefore, we derive temporary fake outcomes from the surrogate: $\tilde{\nu}_{j,\text{busy}} = \hat{\mu}(\theta_{\text{busy}}^{(j)})$. The temporary design $\tilde{\mathcal{D}}$ is created by combining the design of

completed evaluations $\mathcal{D}$ with the fake values. Finally, we simply calculate the EI over the surrogate that is fitted on design $\tilde{\mathcal{D}}$. This has the advantage of being computationally cheaper than the EEI.

4.4 Related Work

There are further methods to generate multi-point proposals for synchronous MBO.

The *qEI* criterion was proposed by Ginsbourger, Le Riche, et al. (2010). It directly optimizes the expected improvement over q points. As the *qEI* criterion is computationally expensive, Chevalier et al. (2013) proposed approximations for $q \leqslant 10$. But these also require expensive Monte Carlo Sampling.

The *constant liar* proposed by Ginsbourger, Le Riche, et al. (2008) is a simplified alternative to the surrogate believer method (Section 4.2.2). Instead of using $\hat{\mu}(\theta^{+(i)})$ as a fake value, a constant value is taken for all q proposals. Ginsbourger, Le Riche, et al. (2008) look at three ways to choose the constant value, namely $\min(\nu)$, $\max(\nu)$ and $\text{mean}(\nu)$, with ν being the vector of all outcomes in the design $\mathcal{D}$. According to their work, $\min(\nu)$ gave the best optimization result, $\max(\nu)$ led to well spread points across the search space and the *surrogate believer* did not lead to any improvement. However, this statement in Ginsbourger, Le Riche, et al. (2008) is solely based on a single two-dimensional test function (Branin), so it should not be generalized. Also the fake value $\max(\nu)$ can lead to numerical problems while fitting a Gaussian process regression, because of the big differences of the function outcomes within close vicinity.

In Bischl, Wessing, et al. (2014) multi-objective optimization is used to find a set of proposals that has a high diversity as well as a high expected improvement.

Another asynchronous approach is presented in Kandasamy et al. (2018). They propose to use *Thompson Sampling*, where a single realization from the Gaussian process is drawn. This realization is a function which can be optimized analogously

to an acquisition function to generate a proposal. They claim that this ensures enough diversity to not have to implement any heuristics or expensive calculations to consider proposals that are under evaluation. Those evaluations are simply ignored.

4.5 Resource-Aware Model-Based Optimization

The main goal of resource-aware model-based optimization is to use the resources as efficiently as possible to reduce the time needed to obtain an optimization result.

As described in the previous chapter, asynchronous MBO directly leads to a complete worker utilization, if we neglect waiting problems as described for *Case B* and *C* in Section 4.3. However, besides the advantage of an increased worker utilization, asynchronous execution can also potentially cause additional runtime overhead due to the higher number of surrogate fits, especially when the number of workers increases. Therefore, our experiments include a comparison with the above described asynchronous and synchronous approaches to investigate the advantages and disadvantages.

Instead of using asynchronous execution to efficiently utilize parallel computer architectures, our new approach uses the synchronous execution combined with resource-aware scheduling. The key idea is to use a second regression model that estimates the runtime of each proposal. Similar to the surrogate, which is used to estimate the outcome of the objective function, the runtime regression model is used to predict the elapsed time to evaluate an input configuration. The estimations are based on the runtime measured on previous runs. We use the estimated runtimes to calculate how these jobs can be efficiently distributed on the available workers. This distribution is calculated following a knapsack scheduling strategy. The aim of the scheduling is to select and distribute the jobs in such a way that the ideling times are reduced.

The theoretical Resource-Aware Model-Based Optimization (RAMBO) framework is outlined in Figure 4.7. It shows different aspects of the resource-aware parallel

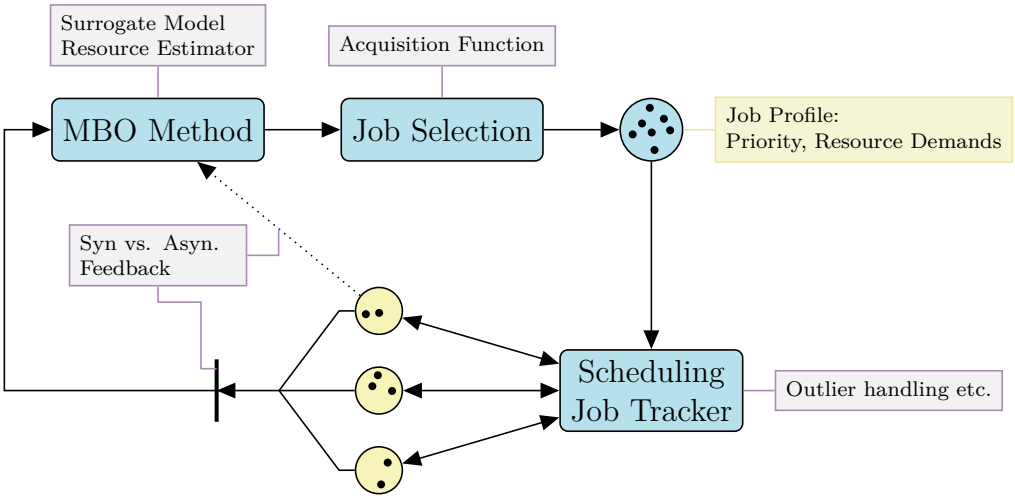

Figure 4.7: Schematic representation of the RAMBO framework. The *MBO Method* defines which regression method is used to build the surrogate and the resource (runtime) estimation. The *Job Selection* uses an acquisition function to select a set of possible jobs. Each job has a *Job Profile* which contains the information needed by the *Scheduling* to efficiently distribute the jobs to the workers. A theoretical *Job Tracker* could terminate jobs if the expected outcome does not promise further optimization progress, e.g. because of an updated surrogate.

evaluation within the MBO framework. The underlying MBO method remains and is extended with a resource estimator (Section 4.5.3) that works similar to the surrogate but predicts the needed resources for each configuration θ. The job selection derives a set of proposals from the surrogate and the resource estimator. The set of jobs goes to the job scheduler (Section 4.5.1) which will distribute the jobs on the workers based on their profiles. Some aspects like the job tracker and the outlier handling, which would allow cancellation of running jobs, are not implemented, but form the basis for further research on the RAMBO framework.

4.5.1 Job Scheduler

In the literature schedulers are defined in many ways and for various purposes. For the purposes in this thesis we define a schedule for a list of jobs $J = \{1, \ldots, q\}$ and k available workers as follows:

> **Definition 4.5.1.** A **schedule** $X \in (0,1)^{k \times q}$ is a matrix with k rows and q columns. Element x_{ij} is 1 iff $i \in \{1, \ldots, k\}$ and $j \in J$, job j is calculated on worker i. Otherwise $x_{ij} = 0$.

The purpose of a scheduler is to calculate an optimal schedule and therefore it is defined as follows:

> **Definition 4.5.2.** The job **scheduler** is given a set of jobs J. Each job has a given priority p_j and a given runtime t_j, $j \in J$. The maximal allowed time for evaluation of jobs is given as $t_{\max}$. For k available workers the scheduler obtains the optimal *schedule* X^* that maximizes the sum of priorities
>
> $$X^* = \arg\max_{x_{ij}} \sum_{i=1}^{k} \sum_{j \in J} p_j x_{ij} \tag{4.5.1}$$

under the restriction that the maximal allowed time t_{max} is not exceeded

$$t_{max} \geqslant \sum_{j \in J} t_j x_{ij} \ \forall i \in 1, \ldots, k \tag{4.5.2}$$

and that no job is executed more than once

$$1 \geqslant \sum_{i=1}^{k} x_{ij} \ \forall j \in J. \tag{4.5.3}$$

This definition implies that not all jobs in J have to be scheduled. For example, jobs with a low priority and a high runtime might not get evaluated.

4.5.2 Scheduling Priority

One prerequisite for scheduling is a priority value for each job. The priorities of the proposed points should reflect their usefulness for optimization. We propose to derive the priority directly from the acquisition function. Therefore, we use the qCB acquisition function (see Section 4.2.1) to obtain a set of job proposals. The qCB is suitable because the individual proposals are independent of each other in contrast to multiple proposals obtained by *Surrogate Believer*, *Constant Liar* or *qEI*. In consequence, there is no inherent order within the set of obtained proposals $\theta^{+(j)}$ from the qCB acquisition function. This means that there is no general rule to determine how promising or important individual proposals are in comparison to each other. In theory proposals obtained by confidence bounds with a high value of λ have a more exploratory characteristic as they focus on uncertainty regions. In contrast, proposals obtained by confidence bounds with a low value λ are more exploitative as they are in regions where the surrogate predicts an optimal outcome. We decided that the highest priority should go to proposals with an exploitative characteristic. In other words, only additional resources should be used for exploration. Therefore, we define the priority for each proposal as $p_j := -\lambda_j$, i.e. we give the highest priority to the proposal $\theta^{+(j)}$ that was proposed using the smallest value of λ_j.

Afterwards we refine the job priorities to prevent jobs in close proximity from each having high priority values. The idea is to lower the priority of a job if a job with higher priority is in the direct vicinity. The distance between the jobs is calculated using the Euclidean distance in the search space Θ, i.e. $d_{j_1,j_2} = |\theta^{+(j_1)} - \theta^{+(j_2)}|$. The goal is to avoid parallel evaluations of very similar proposals $\theta^{+(j)}$ and encourage the scheduler to select sets of jobs with proposals more scattered in the search space. This is necessary because the qCB does not include a penalty for the proximity of selected points.

The **priority refinement** procedure is defined as follows: Given a set of q jobs, we apply hierarchical clustering using the complete linkage method. In the first step ($i = 1$) of the procedure, the job with the highest priority is given the first position in $\tilde{J}$. For each following step $i \geqslant 2$ all jobs are split into i clusters. Of these i clusters the $i - 1$ clusters that contain jobs which are already in $\tilde{J}$ are discarded, leaving one cluster. The job with the highest priority within this cluster is put into position i in $\tilde{J}$ and the counter i is increased by one. The procedure is continued until q jobs have assigned positions in $\tilde{J}$. It generates an ordering in $\tilde{J}$ where each job has the highest priority of the most distant cluster to its predecessor following the hierarchy induced by the clustering. The job with position 1 in $\tilde{J}$ gets assigned the highest priority value q, position 2 gets $q - 1$ and so on until the last job gets the lowest priority 1.

The complete priority refinement algorithm is also given as pseudo code in Listing 3. Note that the hierarchical clustering is just generated once and that the different cluster sizes are obtained by cutting the dendrogram at an appropriate height. Also no jobs are removed from the hierarchical clustering, instead clusters with jobs in $\tilde{J}$ are just ignored when choosing j^*. Naturally, from the j clusters in each iteration of the refinement algorithm, there are always $j - 1$ clusters that are ignored and one “new” cluster that has a candidate for j^*.

Algorithm 3 Priority refinement algorithm for hierarchically clustered jobs.

Require: jobs $J = \{1, \ldots, q\}$, job priorities $\{p_1, \ldots, p_q\}$, hierarchical clustering h_c of jobs.

1: $\tilde{J} \leftarrow \varnothing$
2: **for** $i = 1 \rightarrow q$ **do**
3: $\quad \tilde{h_c} \leftarrow$ divide h_c into i clusters
4: $\quad$ remove clusters from $\tilde{h_c}$ that contain jobs in $\tilde{J}$
5: $\quad j^* \leftarrow \arg\max_{j \in \tilde{h_c}} p_j$
6: $\quad \tilde{J} \leftarrow \tilde{J} \cup j^*$
7: $\quad \tilde{p}_{j^*} \leftarrow q - i + 1$
8: **return** refined job priorities $\tilde{p}_j$, with $j \in 1, \ldots, q$

4.5.3 Resource Estimation

A separate regression is used to model the resource demands, i.e. the runtimes t_j, for the generated proposals $\theta^{+(j)}$. A sensible choice is to choose the same regression method that is used for the surrogate, since both models are defined on the same domain. In the same fashion as for the MBO algorithm, in each MBO iteration the resource estimation model is fitted on the runtimes of previous evaluations in dependency of their optimization parameter settings θ.

As a side remark, for an application with actually measured runtimes a log-transformation is recommended, as runtimes are often distributed with a positive skew, i.e. right-tailed. Log-transformed runtimes can be closer to a normal distribution, which will likely lead to a better regression fit if a Gaussian process regression is chosen.

4.5.4 Resource-Aware Knapsack Scheduling

The scheduling strategy follows two goals. First, the aim is to reduce the idle time. Second, the time until feedback is obtained should be as short as possible, so we can profit from new knowledge, i.e. obtained function values, as fast as possible. If the second goal was not given, the direct consequence would be to evaluate many jobs in one iteration, because it is easier to reduce idle time for a bigger set of

jobs. This, however, would be a bad idea since we do not benefit from the newly obtained knowledge that is generated by a finished job until the complete schedule has been processed. To fulfill both goals we develop a heuristic that balances both aspects.

Our algorithm is embedded in the MBO framework (cmp. Figure 3.1). It adapts the point proposal in Step (3) and needs a slight modification of the point evaluation in Step (4). First, we propose more points than we actually execute within one iteration. Afterwards, we use the scheduler to select and schedule a subset of promising proposals. The evaluation is adapted, so that it evaluates the proposals on the k workers according to the schedule.

We use the qCB acquisition function (see Section 4.2.1) to obtain q proposals, with q randomly drawn values of $\lambda_j \sim \text{Exp}(\frac{1}{2})$, as in Richter, Kotthaus, Bischl, et al. (2016). We set $q = 8 \cdot k$ to obtain a set of proposals larger than the number of available workers. These q proposals form the set of jobs $J = \{1, \ldots, q\}$. Having more jobs than workers allows the scheduler to schedule multiple jobs on one worker and to discard jobs with a high runtime and a low priority. The priority of each job is set as explained in Section 4.5.2. For each job in J we obtain the estimated runtime $\hat{t}_j$ from the resource estimator. To ensure a fast feedback of model updates we set $t_{\max}$ to the estimated runtime of the job with the highest priority. At the same time we want to maximize the profit, given by the priorities of each job, within each MBO iteration. This maximization problem can be solved by a job scheduler as defined in Section 4.5.1. As a solver we utilize the $0-1$ multiple knapsack algorithm, implemented in the R-package `adagio` for global optimization routines (Borchers, 2018), which solves the maximization problem in Equation (4.5.1) approximately. The knapsacks are the available workers and their individual capacity is $t_{\max}$. The items are the jobs J, their weights are the estimated runtimes $\hat{t}_j$ and their values are the priorities p_j.

Before we apply the solver, we manually schedule the job with the highest priority to the first CPU exclusively. This job defines the upper time bound $t_{\max}$. Accordingly, all jobs with higher runtimes are directly discarded. Then the scheduling solver is applied to assign the remaining candidates in J to the remaining $k-1$ CPUs.

Given that the runtime estimates are accurate, this leads to a schedule X^* that can be run in parallel within the given time and maximizes the sum of priorities. If a worker is left without a job, we query the resource estimator for a job with an estimated runtime smaller or equal to $\hat{t}_{\max}$ to fill the gap, regardless of their predicted outcome. This procedure ensures that no idling occurs while we could instead evaluate another configuration.

As mentioned beforehand, the evaluation step within the MBO framework has to be adapted. This is a mere technical matter as we have to make sure that the jobs assigned to each worker are actually evaluated on said worker. From the schedule we can derive the starting times relative to the first one, i.e. the first k jobs start at $t = 0$. The job $k + 1$ starts after the shortest job from the first k jobs finished, because the corresponding worker is the first worker to get free, and so on (see Figure 4.3). We pass the list of jobs sorted by their relative starting time to the system and they will be evaluated on the desired workers. This is because the system evaluates them in the given order and as soon as one worker becomes available, the next job in the list will be assigned to it. One important aspect has been neglected so far: The foundation of the scheduling are the estimated runtimes $\hat{t}$. If the actual runtimes differ too much, the schedule will not be correct anymore. As just mentioned the system will always assign the next job in the list to the next free worker. This will balance out wrong schedules to a certain degree.

5 MBO with Concept Drift

In this chapter, two adaptations are presented that extend the MBO framework to handle dynamic optimization problems (DOPs) which are functions that change over the course of time. Section 5.1 motivates the use of the MBO idea for DOPs. In Section 5.2 we will specify concept drifts for optimization problems. In the following we present the two MBO adaptations: The *window approach* in Section 5.3 and the *time-as-covariate approach* in Section 5.4. Since the error measurement for DOPs is different from static optimization problems and heavily influences the choice of the optimization strategy, we dedicate Section 5.6 to define the scenario and a suitable online error measure.

5.1 Prerequisites

Usually, within the MBO framework the objective function is assumed to be constant over time. However, in real life the relationship between the input θ and the output ν often changes over time. Such a problem can be a process that is influenced by external factors which cannot be controlled. These can be factors that can be measured but not controlled (e.g. outside temperature) or factors that just can be measured latently (e.g. because they change over time). Another practical example is hyperparameter tuning for online machine learning algorithms. Here, the algorithm has to predict labels for new data that sequentially become available. The true labels also become available after the prediction is made and as a consequence the available training data grows. For *offline* learning it is well established to tune the hyperparameters (Thornton et al., 2013) of a

learning algorithm for a specific problem to achieve better predictive performance. For *online* learning hyperparameter tuning should be applied constantly as the nature of the problem can change over time. First, because more training data become available, second because the relationship between features and label can change and third because the structure of the data can change. All three points make continuous tuning of hyperparameters necessary to achieve best predictive performance at each point in time.

Taking the time into consideration changes the optimization problem in Equation (3.1.1) to a *dynamic optimization problem* (DOP), as we are interested in the optimum for each point in time t:

$$\theta_t^* = \arg\min_{\theta \in \Theta} f_t(\theta). \tag{5.1.1}$$

To solve DOPs with MBO we propose two adaptations that can be easily integrated into the MBO framework (Section 3.1). First, we introduce a simple *window approach* in Section 5.3. It limits the design that is used to train the surrogate to the most recent observations. Second, we introduce the *time-as-covariate approach* in Section 5.4S. It includes the time as an additional covariate in the design, giving the surrogate the ability to learn the effect of the time on the outcome.

5.2 Concept Drifts for Dynamic Optimization Problems

The term *Concept Drift* is often used in the context of Online Machine Learning (Gama et al., 2014). We differentiate between three different kinds of concept drifts as illustrated in Figure 5.1. If the change of the objective functions happens instantaneously it is labeled as *sudden drift*. An *incremental drift* describes changes that happen continuously but not necessarily endlessly. It could be a smooth transition from one state to another but also a never ending continuous change. If different states are active repetitively its called a *recurring drift*. In

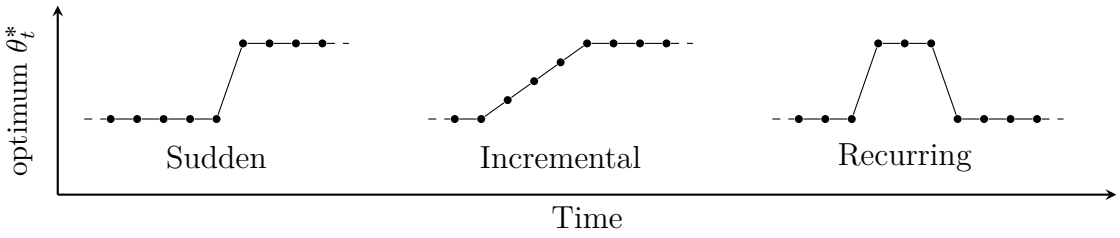

Figure 5.1: Patterns of different concept drifts, figure adapted from Gama et al. (2014).

the literature these definitions are usually used in connection with drifts between different concepts. A concept in the context of online learning is the relationship between the features $\mathbf{x}$ and the label $\mathbf{y}$ of a dataset.

In the dynamic optimization literature, the default is to assume a sudden change of the objective function. Sudden changes happen at discrete points in time but can have different severities. The severity roughly describes how far the optimum moves away in the search space Θ due to the sudden change. A probable reason for the absence of incremental changes in DOP literature might be that the proposed algorithms (e.g. genetic algorithms) are mostly tailored for changes that only occur after many evaluations of the objective function were made (Cruz et al., 2011). In such a scenario it is not assumed that the function changes continuously for each evaluation. However, in our scenarios we want to include this case. Especially as for expensive black-box optimization it is more likely that changes happen at a rate where we are not capable of running hundreds of evaluations on one steady state of the objective function. So we borrow the drift vocabulary of the online machine learning literature for our DOPs. Accordingly, we consider objective functions with incremental, sudden and reoccurring drifts. In this work we only focus on the first two drift types.

5.3 Window Approach

The idea of the *window approach* is to drop older evaluations from the design $\mathcal{D}$, since they likely are not valid anymore. Evaluations are invalid at the current time t if they do not reflect the relationship between the input θ and the output ν that is currently expressed through $f_t(\theta)$. This change of f_t over time can be seen as a concept drift.

To limit the knowledge of the surrogate to evaluations that are likely to be still valid, we subset the design $\mathcal{D}$ to evaluations that were made within the last t_Δ time units:

$$\mathcal{D}_{t_\Delta} = \{(\theta^{(j)}, \nu^{(j)}) \mid (\theta^{(j)}, \nu^{(j)}) \in \mathcal{D} \wedge j \in [t_{\text{now}} - t_\Delta, t_{\text{now}}]\} \ . \tag{5.3.1}$$

Observations outside the window are ignored and thus the uncertainty predicted by the surrogate increases in the corresponding areas. As a consequence, the chance increases that the acquisition function will suggest to re-evaluate the objective function in areas of discarded points. This way the surrogate will always maintain an estimate of the true objective which is close to the state of the function at the current time. Similar to window-based approaches in time series analysis, the choice of the window width symbolizes a trade-off. On the one hand, a large window provides more information for the surrogate and a more precise estimation of the true objective function if no change of the objective function happens, i.e. no concept drift occurs. However, if a change happens the surrogate will only be able to adapt slowly to that change. On the other hand, if the window is too small, it can happen that the surrogate is not able to get a correct notion of the true objective function as it relies on less information. The advantage of a small window size can be a faster adaption to changes in the objective function.

Note that evaluations cannot be considered deterministic anymore as the same θ leads to different outcomes ν depending on the time that they are evaluated at. Therefore, we have to configure the MBO framework to deal with stochastic objective functions. This affects the acquisition function, where the EI is not a valid choice anymore and can be replaced with the AEI (see Section 3.4.3). The

CB acquisition function (see Section 3.4.2) is unaffected as it has no reference value. It also affects the surrogate which has to be aware that each observation in the design has some uncertainty. If Kriging is chosen as a surrogate, the *nugget effect* σ_n^2 should be estimated to account for the additional uncertainty.

5.4 Time as Covariate

For this approach the idea is to let the surrogate completely model the effect of the time. Depending on the regression method, this allows to model complex interactions between the time t and the features θ. Accordingly, the time is included in the surrogate model as an additional covariate t. Introducing an additional dimension renders the surrogate fit more difficult, e.g. because more parameters have to be estimated if Kriging is used as a surrogate. As mentioned, the choice of the surrogate model heavily determines the ability to cope with different drifts. For example, a discontinuity in the input space, like it is introduced by a sudden drift, is a challenge for a Kriging surrogate. Additionally, this approach relies on a certain extrapolation capability of the surrogate, as we are interested for predictions at the current time t_{now} but only have observations from the past.

To propose an input setting for the next iteration, i.e. t_{now}, we optimize the acquisition function on the hyperplane with fixed time $t = t_{\text{now}}$. Therefore, the acquisition function has to be adapted. For the *confidence bound* (see Section 3.4.2) the change is merely technical:

$$\text{CB}_t(\theta, \lambda) = \hat{\mu}_t(\theta) - \lambda \hat{s}_t(\theta) \ , \tag{5.4.1}$$

whereas $\hat{\mu}_t$ denotes the mean prediction of the surrogate for a fixed time t and $\hat{s}_t$ the corresponding uncertainty estimation at time t.

In contrast to the CB, the EI (Equation (3.4.4) on page 28) relies on the best observed outcome y_{min}. This is problematic, as for DOPs it is likely that y_{min} is not valid anymore, because the relation between the corresponding input and the previously evaluated y_{min} has changed for the current state of the objective

function. Therefore, we propose an adaption of the EI that uses an *effective best point* as a reference value instead of the possibly invalid $y_{\min}$. The *effective best point* is a pessimistic estimate of which already evaluated point could be the best point at the current time t:

$$\theta^{**} = \underset{\theta \in \mathcal{D}}{\arg\min}\, \hat{\mu}_t(\theta) + c \cdot \hat{s}_t(\theta) \ , \tag{5.4.2}$$

with c as a tuning parameter which is set to $c = 1$ as default. Using this *effective best point* to calculate the *expected improvement* yields, what we propose as the *Temporal Expected Improvement*:

$$\text{TEI}_t(\theta) = \text{E}\left(\max\left\{\hat{\mu}_t(\theta^{**}) - f_t(\theta), 0\right\}\right), \tag{5.4.3}$$

This approach is inspired by the *Augmented Expected Improvement* (see Section 3.4.3). Accordingly, we assume $f_t(\theta)$ to be a normally distributed random variable with $f_t(\theta) \sim \mathcal{N}(\hat{\mu}_t(\theta), \hat{s_t}^2(\theta))$. In contrast to the AEI, we set the noise variance σ_n to zero, because in dependence of the time the objective function is deterministic, i.e. for the same t and θ the function always returns the same result. Since we set $\sigma_n = 0$, the TEI does not include the additional correction term that is needed for the AEI in Equation (3.4.8). Therefore, the TEI can be calculated analytically as follows:

$$\text{TEI}_t(\theta) = \left(\hat{\mu}_t(\theta^{**}) - \hat{\mu}_t(\theta)\right) \Phi\left(\frac{\hat{\mu}_t(\theta^{**}) - \hat{\mu}_t(\theta)}{\hat{s}_t(\theta)}\right) + \phi\left(\frac{\hat{\mu}_t(\theta^{**}) - \hat{\mu}_t(\theta)}{\hat{s}_t(\theta)}\right) \tag{5.4.4}$$

with ϕ and Φ as the standard normal density and distribution function.

5.5 Related Work

DOPs have been in the focus of evolutionary optimization strategies, where a function evaluation is considered comparably inexpensive. Excellent surveys are given in Branke (2005) and Cruz et al. (2011). Independent from this work, Nyikosa et al. (2018) proposed to apply Bayesian optimization on dynamic optimization

problems. They exclusively use the CB as acquisition function and introduce the usage of specific temporal Kernels for the Gaussian process surrogate. This has the advantage that the temporal Kernel k_T can be specifically chosen to handle periodicity or other temporal peculiarities. However, they assume that the process has to have a separable covariance:

$$\text{Cov}(f_t(\theta), f_{t'}(\theta')) = k_S(\theta, \theta') \cdot k_T(t, t'), \tag{5.5.1}$$

which is a strong assumption since it implies that the time t and the input θ have independent influence on the outcome ν. In this chapter, we do not follow this approach which gives us the advantage to freely choose the regression method for the surrogate.

5.6 Error Measurement

To evaluate the performance of our optimizer we need an evaluation measure that considers the online characteristic of our optimization. In comparison to regular MBO we are not only interested in the distance to the optimum of the final result of the optimization process, but also in the distance to the optimum at any given time point t. In particular for DOPs the choice of the error measure can heavily influence the final ranking of the optimization algorithms. A plethora of evaluation measures exists in the DOP literature (Cruz et al., 2011). Many are tailored for the characteristics of evolutionary optimization algorithms, e.g. *Current best-of-generation evolution*, *Mean best-of-generation* etc. The mentioned measures and most others take only the best solution of a set of evaluations, e.g. a single generation, into account. Each generation is evaluated on the same state of the objective function, i.e. at the same point in time. Usually multiple generations are used to optimize the objective until a state change happens. This means that many evaluations of the objective can perform badly without affecting the performance measure.

In our scenario the setup is quite different. As we deal with function evaluations that take a significant amount of time, we have to assume that the state of the objective function already changed after each evaluation. This continuous state change does not allow us to run multiple evaluations on the same state of the objective function successively.

In our benchmark scenario we know the true optimal outcome ν_t^* for any given point in time. Therefore we can simply calculate the *fitness error*

$$\text{err}_{\text{FE}} = f_t(\theta_t^+) - \nu_t^* \tag{5.6.1}$$

for the minimization of f at any given time point t. For the evaluation of the whole optimization run we average the errors over all time points which gives us the *mean fitness error* (MFE).

A motivational example can be given by a machine that produces a product. The machine can be controlled by the input parameters θ. In the best case (θ_t^*) all products are perfect and no loss occurs. Unfortunately, θ_t^* is unknown and changes over the time due to external factors. The goal is to set θ_t^+ in such a way that over the course of time the average loss is as low as possible. As we only have one machine, every evaluation counts into the error measurement. This is a substantial difference from the error measurement in the evolutionary setting where only the error of the best evaluation within an evolution or a batch is taken into account. Note that in Equation (5.6.1) we use θ_t^+ as input setting instead of $\hat{\theta}_t^*$ as in the original formulation of the dynamic minimization problem in Equation (5.1.1). The latter would be the estimation of the best input setting. For ordinary sequential MBO the proposal of the best point is usually obtained by taking the configuration θ that led to the best outcome (see Section 3.5). Such a point would likely be an optimum for an outdated state of our objective function. Alternatively, we could propose the input setting that minimizes $\hat{f}_t$ as the best input setting $\hat{\theta}_t^*$. This proposal could minimize the error in a single iteration of our optimization process. However, using such a proposal in each iteration will likely lead to stagnation or even decrease of the performance as we will not be able to explore new areas in our search space and probably not be able to adapt to a drift. Summing up, we use the

input setting suggested by the acquisition function θ_t^+ to measure the error because it is the only input configuration that is evaluated within the chosen set-up. This implies that exploration can negatively affect the overall error with the benefit of exploring the search space.

6 Parallel MBO Benchmark

The purpose of the benchmark in this chapter is to analyze which MBO parallelization strategy (see Chapter 4) works best for black-box functions with heterogeneous runtimes within a limited time budget. The benchmark considers the scenario of reliably and unreliably predictable runtimes as well as two different degrees of parallelization on 4 and 16 workers. The benchmark was first published in Richter, Kotthaus, A. Lang, et al. (2017).

6.1 Objective Functions with Heterogeneous Runtimes

To cover a variety of problems, we consider two categories of synthetic base functions that will serve as objective functions:

1. Functions with a smooth surface: $\texttt{rosenbrock}_d$ and $\texttt{bohachevsky}_d$ with dimension $d \in \{2, 5\}$. They are likely to be fitted well by the Kriging regression that is used as a surrogate and to predict the runtime.
2. Highly multimodal functions: $\texttt{ackley}_d$ and $\texttt{rastrigin}_d$ ($d \in \{2, 5\}$). We expect that Kriging models have problems to achieve a good fit here.

All base functions are implemented in the `R` package `smoof` (Bossek, 2017). Their 2-dimensional versions are illustrated in Figure 6.1

In our optimization scenario, which is mainly motivated by machine learning problems, we want to consider heterogeneous runtimes. The goal is to build a

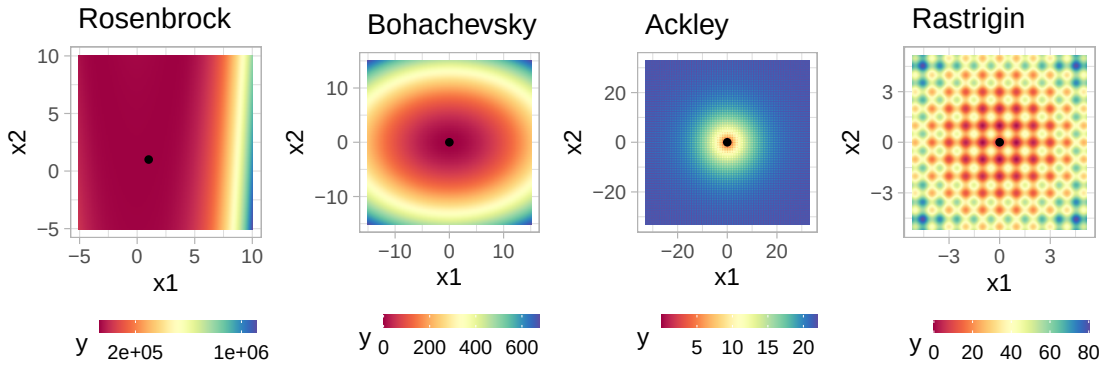

Figure 6.1: All base functions in their 2-dimensional versions that are used in this benchmark. The global optimum is marked with a dot.

benchmark with synthetic functions that reflects our optimization scenario. As synthetic functions have no significant runtime, it is necessary to simulate artificial runtimes. The artificial runtime in dependence of the input θ is given by the same set of functions as mentioned above. The functions with artificial runtime are derived by combining a base function with a time function. The time function determines the number of seconds it takes to calculate the objective value of the base function. E.g., for the combination $\texttt{rastrigin}_2.\texttt{rosenbrock}_2$ it would require $\texttt{rosenbrock}_2(\theta)$ seconds to retrieve the objective value $\texttt{rastrigin}_2(\theta)$ for an arbitrary point θ. Technically, we simulate the runtime by sleeping $\texttt{rosenbrock}_2(\theta)$ seconds before returning the objective. We simulate the runtime with either $\texttt{rosenbrock}_d$ or $\texttt{rastrigin}_d$ and analyze all combinations of our four objective functions, except where the objective and the time function are identical.

A prerequisite for this approach is the unification of the input space. Therefore, we scale values from the input space of the time function to the input space of the base function. The output of the time functions is scaled to return values between 5 min to 60 min.

6.2 Setup

We consider the following optimizers and optimization set-ups:

`rs`: Random search, serving as base-line.

`qCB`: Synchronous approach using qCB where in each iteration $q = k$ points are proposed as explained in Section 4.2.1.

`ei.bel`: Synchronous approach using the surrogate believer approach with k proposals in each iteration as explained in Section 4.2.2).

`asyn.eei`: Asynchronous approach using EEI with $n_{\text{sim}} = 100$ Monte Carlo iterations (Section 4.3.1). If a worker finishes and another worker is busy generating a proposal, the worker will wait until the proposal is generated and can be included in the EEI calculation. However, no queue is generated if multiple workers are waiting. See *Case B* in Section 4.3.

`asyn.ei.bel`: Asynchronous surrogate believer approach as explained in Section 4.3.2. The same waiting mechanism as for `asyn.eei` is applied here.

`rambo`: The new synchronous resource-aware approach that schedules $q = 8 \cdot k$ proposals obtained by the *qCB* acquisition function on the k workers to reduce idling as explained in Section 4.5.

`qCB` and `ei.bel` are implemented in the `R` package `mlrMBO` (Bischl, Richter, et al., 2017), which builds upon the machine learning framework `mlr` (Bischl, M. Lang, et al., 2016). `asyn.eei`, `asyn.ei.bel` and `rambo` are also based on `mlrMBO`. All aforementioned optimizers (except the random search) use a Gaussian process regression as a surrogate. The covariance is estimated with a Matern$\frac{5}{2}$-kernel and σ_n^2 is set to 0 since the optimization is deterministic. `rambo` uses the same settings for the regression model that predicts the runtimes.

Additionally, we compare our implementations to:

smac: Asynchronous approach that turns k independent optimization runs of the SMAC Algorithm into a single asynchronous optimization with k workers by sharing the surrogate model data (also called shared-model-mode [1]).

SMAC cannot be initialized with a predefined design. Therefore, it was allowed the same initial budget as the other optimizers. It was started with the default parameters and the shared-model-mode activated. SMAC uses a random forest as surrogate and the EI criterion. The shared-model-mode does not include knowledge about pending evaluations.

The optimizations are repeated 10 times and conducted on $k = 4$ and $k = 16$ CPUs (workers). For each repetition a random initial design is generated by Latin hypercube sampling with $n = 4 \cdot d$ points. The initial design is the same for all optimizers within a repetition. We allow each optimization to run for 4 h on 4 workers and for 2 h on 16 workers in total which includes all computational overhead and idling.

All computations were performed on a Docker Swarm cluster using the `R` package `batchtools` (M. Lang et al., 2017).

6.3 Evaluation

To have a comparable measure across different objective functions we standardize the objective value to $[0, 1]$, with 0 being the best. The target value 0 is determined by the average of the best ν reached by any optimization method after the complete time budget across all 10 replications. The reference value 1 is determined by the average of the best ν reached in the initial design across all 10 replications. Note, that the initial design is identical for all algorithms for a given problem but

[1] Hutter, F., Ramage, S.: Manual for SMAC version v2.10.03-master. Department of Computer Science, UBC. (2015), `www.cs.ubc.ca/labs/beta/Projects/SMAC/v2.10.03/manual.pdf`

different for each replication. This excludes the initial runs of `smac` which are not taken into consideration here. We call this standardized objective value *accuracy*. This *accuracy* gives the distance between the best found point at time t and the target value. If an optimization method needs 2 h to reach an accuracy of 0.5, this means that within 2 h half of the way to 0 has been accomplished, after starting at 1. We compare the differences between optimization methods at three accuracy levels 0.5, 0.1 and 0.01.

6.3.1 Quality of Resource Estimation

The quality of resource-aware scheduling naturally relies on the accuracy of the resource estimation. Without reliable runtime predictions, the schedules potentially cannot be applied as planned and an efficient utilization of the workers is not guaranteed. As Figure 6.2 by way of example shows, the runtime prediction for the $\texttt{rosenbrock}_2$ time function works well as the residual values are getting smaller over time, while the runtime prediction for $\texttt{rastrigin}_2$ is comparably imprecise, taking into consideration that the real runtimes range between 5 min and 60 min. Similarly for $\texttt{rosenbrock}_5$ the residuals are getting smaller after 30 min, while the residuals for $\texttt{rastrigin}_5$ do not indicate any improvement of the estimation over time. This encourages to consider scenarios separately where runtime prediction is possible ($\texttt{rosenbrock}_d$, Section 6.3.2) and settings where runtime prediction is error-prone ($\texttt{rastrigin}_d$, Section 6.3.3) for further analysis.

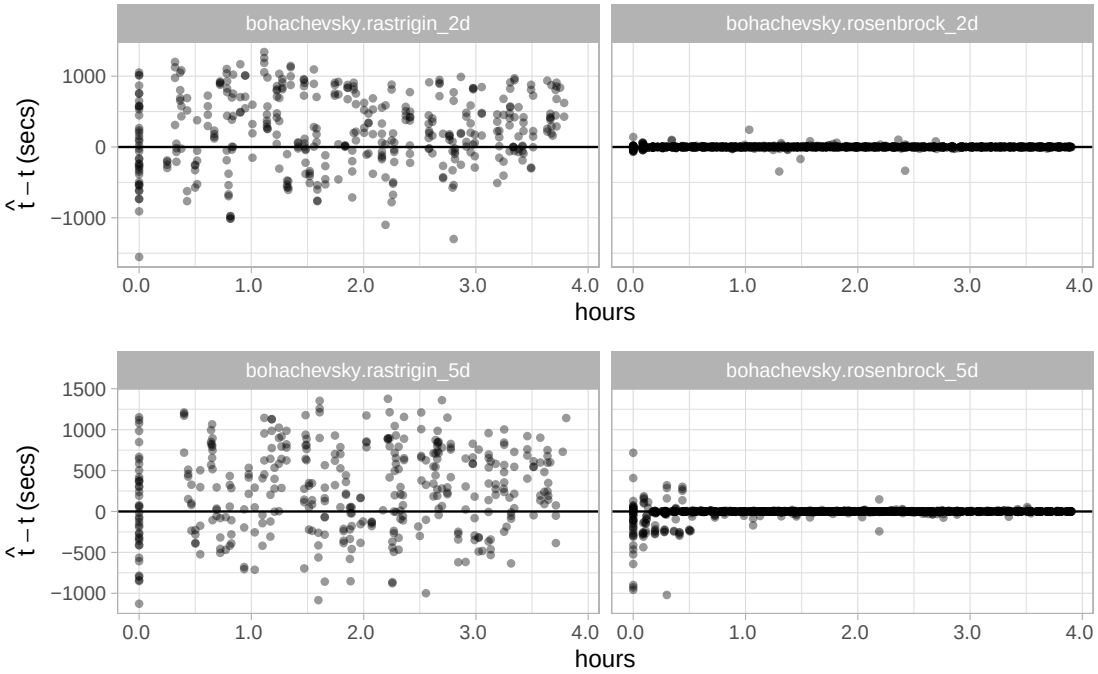

Figure 6.2: Residuals of the runtime prediction in the course of time for the $\texttt{rosenbrock}_d$ and $\texttt{rastrigin}_d$ time functions on 4 workers and $\texttt{bohachevsky}_d$ as objective function. Positive values indicate an overestimated runtime and negative values an underestimation.

6.3.2 High Runtime Estimation Quality: $\texttt{rosenbrock}_d$

In this section we analyze the performance of all optimization methods on all problems with $\texttt{rosenbrock}_d$ as time function. Figure 6.3 shows boxplots for the time required to reach the specified accuracy levels in 10 repetitions within a budget of 4 h real time on 4 workers (upper part) and 2 h on 16 workers (lower part). The faster an optimization method reaches the desired accuracy level, the lower the displayed box and the better the method. If an algorithm did not reach an accuracy level within the time budget, we impute the missing time value with the respective time budget (4 h or 2 h) plus a penalty of 1000 s, noticeable as observations above the respective time budget in the boxplots.

Table 6.1 lists the aggregated ranks over all objective functions, grouped by method, accuracy level, and number of workers (k). To obtain the average ranks, the

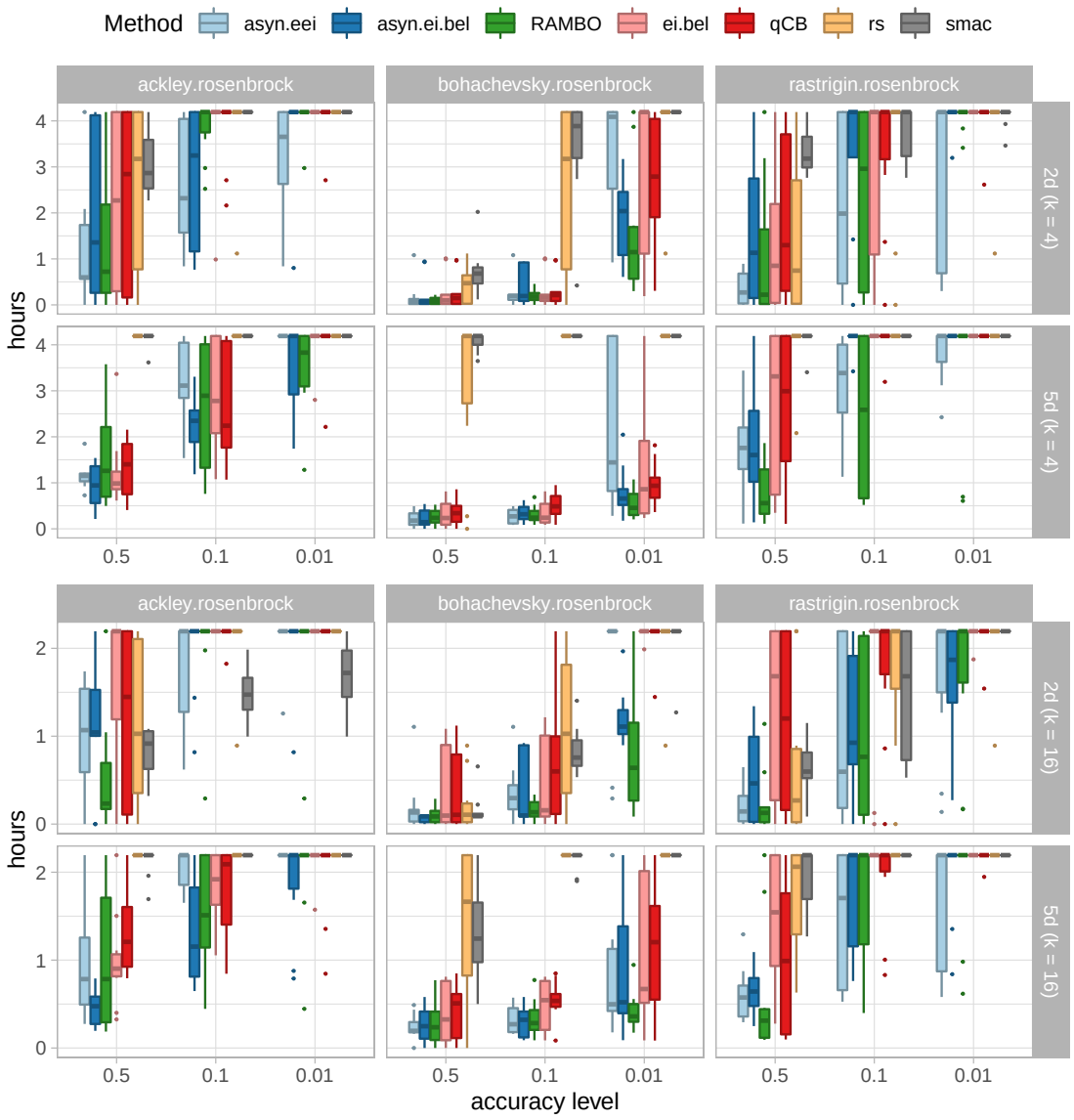

Figure 6.3: Accuracy level vs. execution time for different objective functions using time function $\texttt{rosenbrock}_d$ (lower is better).

Table 6.1: Rankings of the optimization methods on problems with reliable runtime prediction, i.e. $\texttt{rosenbrock}_d$ as time function.

	$k = 4$			$k = 16$		
Method	0.5	0.1	0.01	0.5	0.1	0.01
`asyn.eei`	3.32 (2)	3.52 (1)	4.97 (2)	3.75 (3)	4.30 (3)	5.45 (3)
`asyn.ei.bel`	3.55 (3)	4.10 (3)	4.97 (2)	3.48 (2)	4.08 (2)	4.53 (2)
`RAMBO`	3.17 (1)	3.85 (2)	4.57 (1)	3.13 (1)	3.93 (1)	4.47 (1)
`ei.bel`	4.38 (4)	4.98 (4)	5.90 (5)	5.00 (5)	5.48 (6)	6.28 (6)
`qCB`	4.52 (5)	5.03 (5)	5.63 (4)	4.72 (4)	5.17 (4)	6.10 (4)
`rs`	6.02 (6)	6.67 (6)	6.83 (7)	5.50 (7)	6.48 (7)	6.87 (7)
`smac`	6.22 (7)	6.70 (7)	6.82 (6)	5.32 (6)	5.47 (5)	6.17 (5)

optimization methods are ranked w.r.t. their performance for each replication, problem, accuracy level and k before they are aggregated with the mean. If there are ties because an accuracy level was not reached, all values obtain the worst possible rank.

The results show that the new Resource-Aware MBO (RAMBO) method reaches the defined accuracy levels first in two cases on $k = 4$ workers and always reaches the levels first on 16 workers. `rambo` is closely followed by the asynchronous variant `asyn.eei` on 4 workers but the lead becomes more clear on 16 workers. In comparison to the unscheduled synchronous algorithms (`ei.bel`, `qCB`), `rambo` as well as `asyn.eei` and `asyn.ei.bel` reach the given accuracy levels in shorter time. This is clearly visible for the objective functions that are hard to model ($\texttt{ackley}_d$, $\texttt{rastrigin}_d$). The simpler `asyn.ei.bel` performs better than `asyn.eei` on 16 workers. Except for `smac`, all presented MBO methods outperform base-line `rs` on almost all problems and accuracy levels. The bad average results for `smac` are partly due to its low performance on the $d = 5$ problems and probably because of the disadvantage of using a random forest as a surrogate on purely numerical problems. It can be speculated that the strengths of `smac` are black-box problems of another structure (e.g. search spaces with more discrete values).

For an analysis of the scheduling behavior, Figure 6.4 exemplary visualizes the true mapping of the jobs for all MBO methods on 16 workers for the $d = 5$ versions of the problems from one of the ten stochastic repetitions. The random search `rs` is left out, because it naturally does not lead to idling. Each gray box represents the computation of a job on the respective worker. The red boxes, which are especially visible for `asyn.eei`, show the time that is needed for the proposal generation. The remaining scheduling plots for $k = 4$ workers and on the two-dimensional objectives are given in Figures A.1, A.2 and A.3 in the Appendix A.1 on pages 129ff.

The necessity of a resource estimation for jobs with heterogeneous runtimes becomes obvious, as `qCB` and `ei.bel` can cause long idle times by evaluating jobs together in one iteration with large runtime differences. The scheduling in `rambo` manages to clearly reduce this idle time by two means: Either multiple jobs are proposed to fill gaps created by a highly prioritized job with a relatively long runtime, or only jobs with similar runtime are proposed within a single MBO iteration.

The Monte Carlo approach of `asyn.eei` generates a high computational overhead as indicated by the red boxes. This reduces the effective number of evaluations, which decreases the optimization performance. Idling occurs because the calculation of the EEI is encouraged to wait for ongoing EEI calculations to include their proposals. The time needed for a single EEI proposal increases with the number of already evaluated points as the training of the surrogate takes longer with more training points. `asyn.ei.bel` and `smac` have a comparably low overhead and thus basically no idle time. This seems to be an advantage for `asyn.ei.bel` on 16 CPUs where it performs better on average than its complex counterpart `asyn.eei`.

Summed up, if the resource estimation that is used in `rambo` has a high quality, `rambo` is able to outperform the considered MBO algorithms `qCB`, `ei.bel`, and `smac` on the given set of functions. This indicates that the increased resource utilization obtained by the scheduling in `rambo` leads to the same accuracy within shorter time, especially if the number of available workers increases. This also implies that within a given time-frame `rambo` potentially reaches higher accuracy levels.

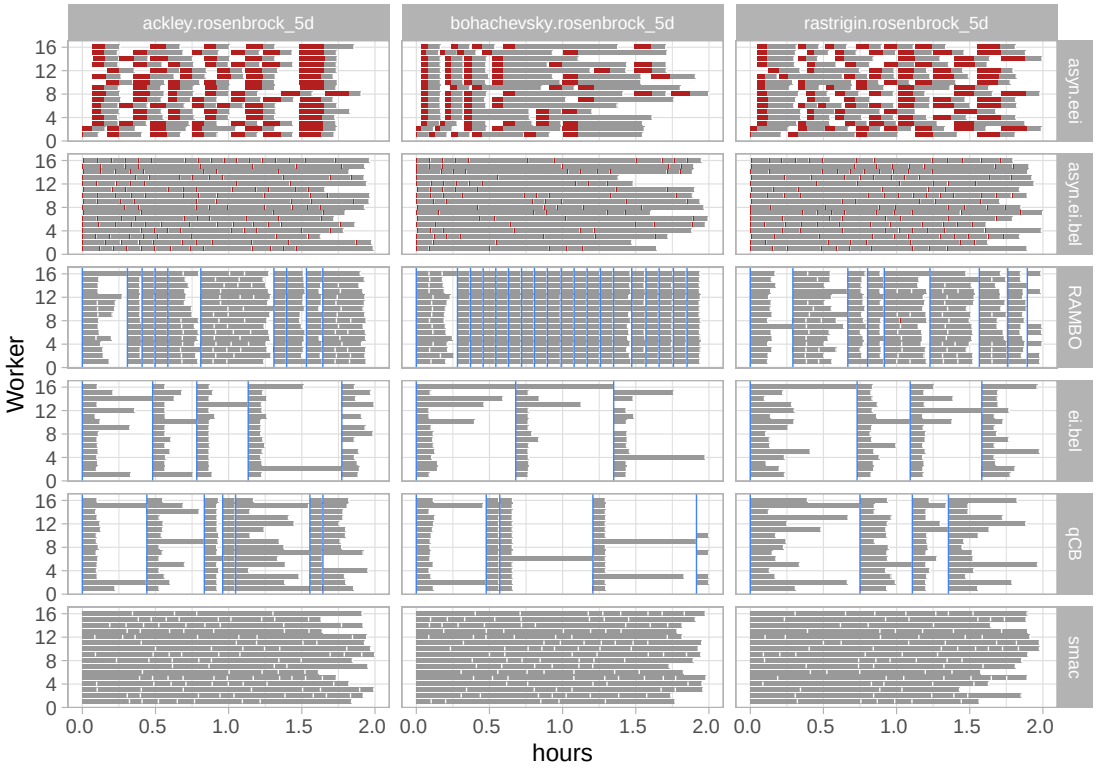

Figure 6.4: Worker usage of the different optimization methods on the functions with the $\texttt{rosenbrock}_5$ time function, which can be modeled reliably by the resource estimator. Time is plotted on the x-axis and the mapping of the jobs on $k = 16$ workers on the y-axis. Each gray box represents the runtime of job on a specific worker. Each red box represents the time needed for the point proposal. If it is not shown, the time is insignificantly short. For `smac` the proposal time is not measured individually but is included in the overall runtime. The gaps represent idle time. For the synchronous MBO algorithms (`rambo`, `qCB`, `ei.bel`) the blue vertical lines indicate the beginning of each MBO iteration.

6.3.3 Low Runtime Estimation Quality: $\texttt{rastrigin}_d$

The time function $\texttt{rastrigin}_d$ used in the following scenario is difficult to fit by the regression method, as indicated by the residual plot in Figure 6.2. For this reason, the benefit of the knapsack scheduling in `rambo` is expected to be minimal. For example, in a possible worst case multiple supposedly short jobs are assigned to one CPU but their real runtime is considerably longer and causes unwanted idle times.

Similar to the Figure 6.3 from the previous section, Figure 6.5 shows boxplots for the benchmark results, but with $\texttt{rastrigin}_d$ as the time function. Table 6.2 provides the mean ranks of each optimization method, calculated in the same way as in the previous Section 6.3.2.

Despite possible wrong scheduling decisions, `rambo` still manages to outperform `qCB` and performs comparably to `ei.bel`. `asyn.eei` reaches all accuracy levels fastest on 4 workers. Similar to the benchmark results in the previous Section 6.3.2, the simplified `asyn.ei.bel` seems to benefit from its reduced overhead and places first on 16 workers.

`smac` cannot compete with the MBO methods that use a Gaussian process as a surrogate. Overall, `rambo` appears not to be able to outperform the asynchronous MBO methods as unreliable runtime estimates likely lead to suboptimal scheduling decisions. The real worker utilization for the functions with $\texttt{rastrigin}_5$ as time function and $k = 16$ workers is given in Figure 6.6. The remaining scheduling plots for $k = 4$ workers and on the two-dimensional objectives are given in Figures A.4, A.5 and A.6 in the Appendix A.2 on pages 132ff. In comparison to the scheduling plot of the previous section in Figure 6.4, the $\texttt{rastrigin}_5$ apparently leads to longer runtimes on average. As a result, the overhead for point proposals of `asyn.eei` becomes relatively smaller. Additionally, the EEI proposal itself has to consider fewer evaluated points, making it effectively faster. In contrast, `rambo` nearly behaves identical to the other asynchronous methods `ei.bel` and `qCB`. `rambo` is not able to schedule multiple jobs on a single worker, even though the simulated runtimes are heterogeneous. However, it could be that in some cases jobs with a

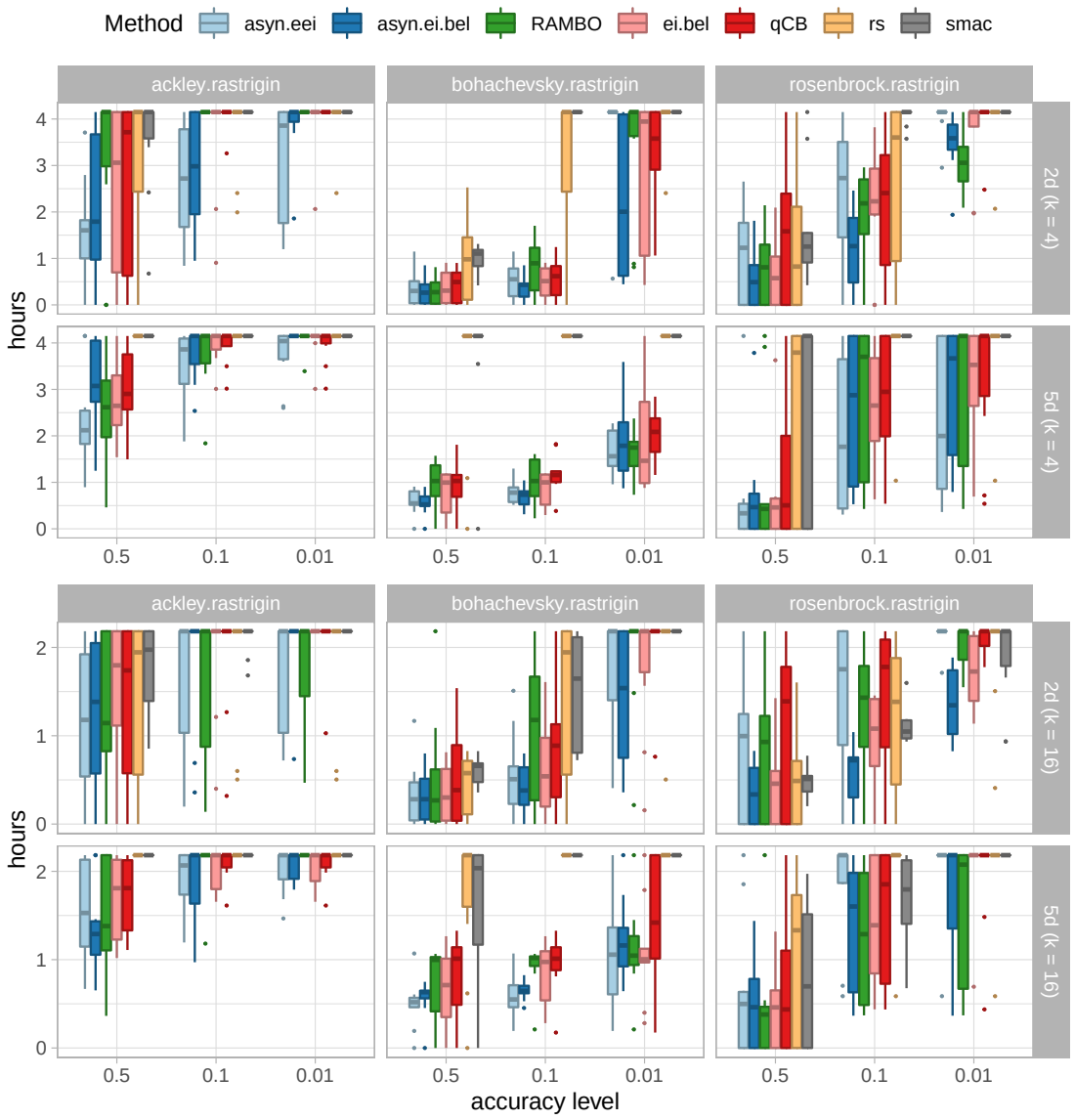

Figure 6.5: Accuracy level vs. execution time for different objective functions using hard-to-fit time function $\texttt{rastrigin}_d$ (lower is better).

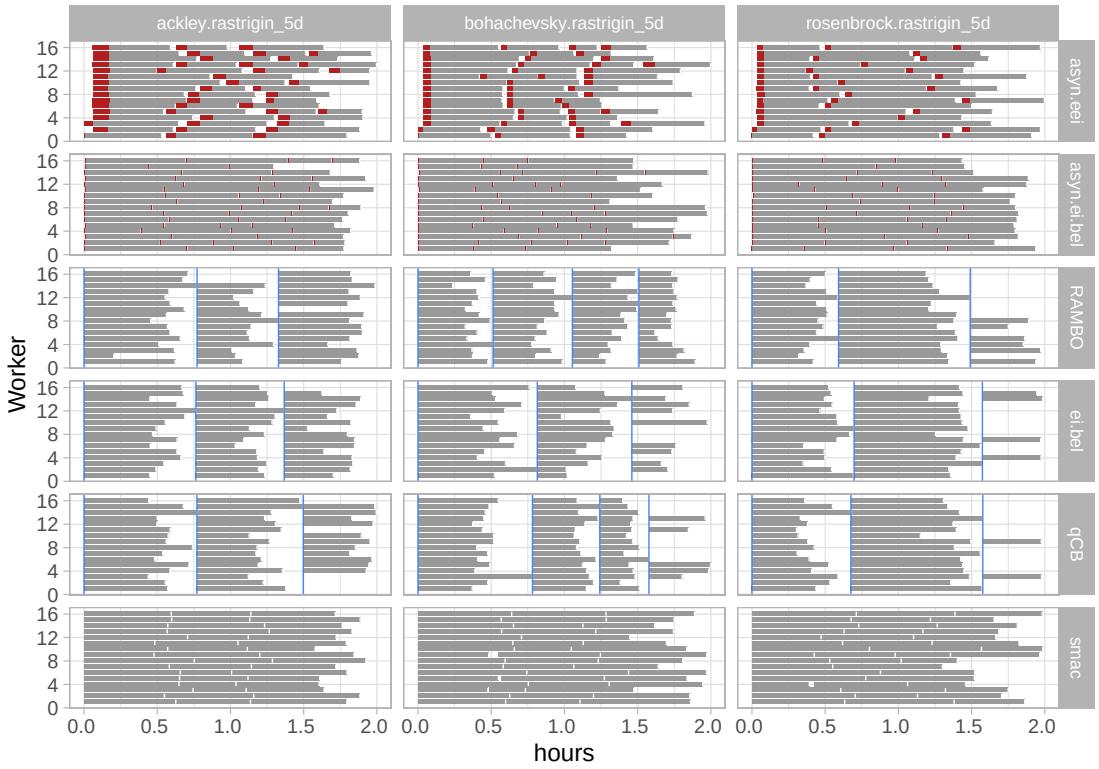

Figure 6.6: Similarly to Figure 6.4 this plot shows the worker utilization for problems with the $\texttt{rastrigin}_5$ time function which can be hardly modeled by the resource estimator. Note that this just visualizes one of the ten stochastic repetitions.

Table 6.2: Rankings of the optimization methods on problems with unreliable runtime predictions (`rastrigin`$_d$ as time function).

	$k = 4$			$k = 16$		
Method	0.5	0.1	0.01	0.5	0.1	0.01
`asyn.eei`	3.65 (1)	3.25 (1)	4.47 (1)	4.42 (3)	4.38 (2)	5.20 (3)
`asyn.ei.bel`	3.88 (2)	3.50 (2)	4.52 (2)	3.90 (1)	3.77 (1)	4.33 (1)
`RAMBO`	4.50 (4)	4.70 (4)	4.72 (3)	4.43 (4)	4.63 (4)	5.17 (2)
`ei.bel`	4.22 (3)	4.42 (3)	4.87 (4)	4.33 (2)	4.55 (3)	5.27 (4)
`qCB`	4.95 (5)	4.80 (5)	5.38 (5)	5.10 (5)	5.00 (5)	5.82 (5)
`rs`	6.30 (7)	6.42 (6)	6.63 (6)	5.80 (7)	6.23 (7)	6.43 (6)
`smac`	5.90 (6)	6.98 (7)	7.00 (7)	5.27 (6)	5.63 (6)	6.72 (7)

long predicted runtime and a low priority are discarded, leading to slightly more evaluations than `qCB` as it can be observed for the `bohachevsky`$_d$ objective on $k = 4$ workers.

Despite the inability to schedule multiple jobs onto a single worker, `rambo` reaches comparable results to `ei.bel` and performs slightly better than `qCB`. Altogether, the asynchronous parallelization methods perform better on the chosen problems with a hard-to-predict runtime.

6.4 Conclusion

This benchmark compared the new knapsack based resource-aware parallel MBO algorithm `rambo` against popular synchronous and asynchronous MBO approaches on a set of synthetic, continuous test functions with artificial runtimes. The artificial runtimes were taken from the function value of two of the functions included in the benchmark. One function is hard to model by the Gaussian process regression, while the other can be easily modeled. For the latter, runtimes could be predicted reliably, leading to an advantage for `rambo`. When the runtimes could not be predicted accurately by the resource estimator, `rambo` was not able to outperform the other methods. Most likely, unreliable runtime estimates lead to suboptimal

scheduling decisions. The results on 16 parallel workers also indicate that `rambo` can handle high grades of parallelization, in contrast to the expensive asynchronous approach that relies on Monte Carlo simulations. On problems with hard-to-predict runtimes the asynchronous approaches performed best on 4 workers and only the simplified asynchronous surrogate believer kept its lead on 16 workers.

If the runtime of jobs is predictable, the `rambo` approach for parallel MBO with high numbers of available workers appears to be a viable optimization method. The runtime estimation quality often is hard to determine in advance. For some real applications like hyperparameter optimization for machine learning methods predictable runtimes can be assumed. Interestingly the choice of the acquisition function and the multiple point proposal strategy do not influence the optimization performance as heavily as the choice of the parallelization strategy, e.g. both synchronous approaches perform similarly.

The good performance of `rambo` is not entirely expected, since the asynchronous approaches easily achieve higher utilization of the workers. Therefore, a higher amount of total jobs evaluated could be expected. More evaluated optimization parameter settings θ should lead to a higher chance of finding optimal solutions, unless the parameters are chosen poorly. The proposal for asynchronous methods always has to handle $k - 1$ missing evaluations, whereas for the synchronous proposal at least one of k proposals can be created with the complete knowledge. The high number of missing evaluations for the asynchronous method can become unfortunate when a pending job finishes shortly after a proposal was generated. If this proposal was generated some seconds later it could have been generated based on more knowledge. Therefore, it seems promising to use a hybrid method that combines the resource estimation and multi-point proposal of the synchronous `rambo` approach but also has the capability to start asynchronous evaluations. The hybrid method should include a validator for the resource estimation that checks whether the resource estimations are reliable by a simple cross-validation on the measured runtimes. If the runtimes are heterogeneous and unreliably estimated the hybrid method will use an asynchronous behavior. If the runtimes are homogeneous the hybrid will use a synchronous behavior. If the runtimes are heterogeneous and

reliably estimated, the method could divide jobs into groups of jobs. Within the groups the runtimes are homogeneous and the jobs are evaluated in a synchronous manner, but the groups are evaluated asynchronously.

Further improvements are possible by formulating a criterion that directly rewards the selection of multiple points as well as it punishes highly expected idle times. Such a group acquisition function could have the following form:

$$\begin{aligned}\mathrm{acq}_{\mathrm{group}}(\{\theta^{+(1)},\ldots,\theta^{+(q)}\}) = \\ \sum_{i=1}^{q}\left[\mathrm{acq}(\theta^{+(i)})\right]\cdot d(\{\theta^{+(1)},\ldots,\theta^{+(q)}\}) - \gamma\cdot s(\{\theta^{+(1)},\ldots,\theta^{+(q)}\}),\end{aligned} \quad (6.4.1)$$

whereas $d(\ldots)$ is a reward that encourages the selection of points that are spread across the search space, such as the average distance between the given points, and $s(\ldots)$ is the expected idle time to encourage the selection of points that can be scheduled without idle time. The tunable parameter γ balances the influence of expected idle time. The acquisition function could be the qCB as it has the desirable property that it is independent of the selected points. Maximizing $\mathrm{acq}_{\mathrm{group}}$ would lead to a set of proposals that are well distributed, have low idle time and are likely good solution candidates.

In the presented benchmarks, only the runtime was considered by the resource estimation. Also, the memory and energy consumption could be taken into consideration. This is of interest for machine learning problems where not only the number of available workers is a limiting factor but also the available random access memory.

7 MBO CD Benchmark

The goal of the benchmark is to compare the optimization performance of different MBO CD approaches on dynamic optimization problems with different types of drifts.

7.1 Synthetic Dynamic Objective Functions

To obtain a set of dynamic test functions we augment static functions that are well-known in the optimization community in such a way that we can control what kind of drift occurs. Therefore, we construct the dynamic optimization problems (DOPs) from three parts: An *objective function*, a *drift function* and a *transformation function*.

The constructed DOP will be a function of the form $f_{\delta,g}(\theta, t)$. To obtain different time-dependent versions of the same objective function a transformation in the domain space will be applied. We obtain the DOP through a composition of the static objective function f and a state-dependent transformation function g: $f(g(\theta, w))$. The transformation function $g(\theta, w)\colon [0,1] \to [0,1]$ transforms the input θ according to the drift state w. The drift function $\delta(t)\colon [0,1] \to [0,1]$ is the function that maps the time t to the drift state w, e.g. $\delta(t) = 0.5$ implies that the time does not change the state of the function. The procedure is explained in more detail hereafter.

Objective Function: First, we take one of the black-box functions $f(\theta)$ from Table 7.1. All of them are implemented in the R-package `smoof` (Bossek, 2017) and have already been used to generate DOPs (Cruz et al., 2011; Nyikosa et al., 2018), but usually with stochastic changes. Functions that are defined on a 1, 2 and

Table 7.1: Objective functions that serve as base functions divided in two sets. d is the dimensionality of the input space Θ, θ^* is the location of the global optimum.

Function	d	θ^*
Set 1		
Ackley	1	$(0.5)'$
	2	$(0.5, 0.5)'$
	5	$(0.5, 0.5, 0.5, 0.5, 0.5)'$
Griewank	1	$(0.5)'$
	2	$(0.5, 0.5)'$
	5	$(0.5, 0.5, 0.5, 0.5, 0.5)'$
Rastrigin	1	$(0.5)'$
	2	$(0.5, 0.5)'$
	5	$(0.5, 0.5, 0.5, 0.5, 0.5)'$
Set 2		
Branin	2	$(0.12, 0.82)'$, $(0.54, 0.15)'$, $(0.96, 0.17)'$
Camelback	2	$(0.49, 0.68)'$, $(0.51, 0.32)'$
Goldstein-Price	2	$(0.5, 0.25)'$

5-dimensional input space are grouped in set 1 and the others in set 2. All functions of set 1 have a parabolic surface combined with cosine waves of different amplitude and frequency as illustrated in Figure 7.1. Consequently they are multimodal. Functions of set 2 are also multimodal but with fever local minima and a smoother surface. Functions that were originally constructed as maximization problems were flipped to become minimization problems. Furthermore, each function is standardized in two ways. The input space is scaled to $[0, 1]^d$ and function values are scaled, so that $f(\theta^*) = 0$ and the median performance is 1. The median performance is obtained by evaluating a grid of $\min(100^d, 10^6)$ values of $\theta \in \Theta$ on the objective function and calculating the median of the outcomes.

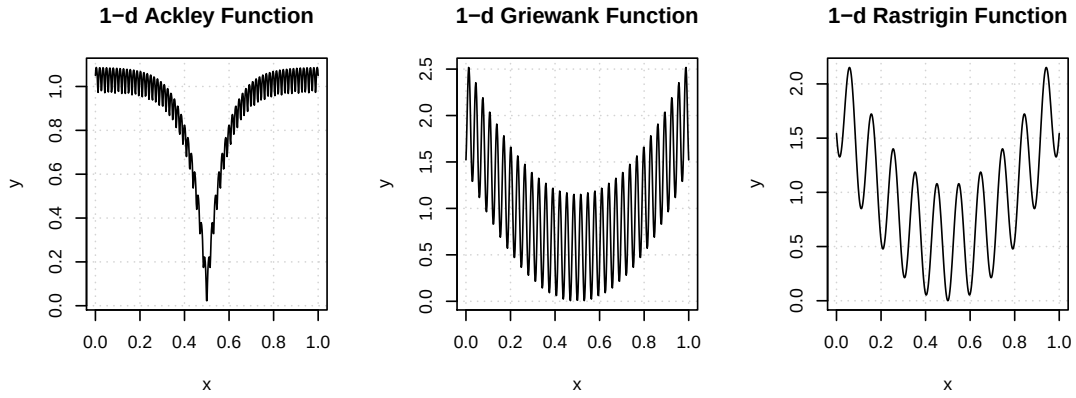

Figure 7.1: All objective functions from set 1 in their $1d$ versions.

Drift Function: The drift function $\delta(t)$ is utilized to introduce different types of drifts into the function, depending on the elapsed time t: We consider the following drift functions $\delta(t)\colon [0,1] \to [0,1]$:

No Drift	$\delta_n(t) = 0.5$,
Sudden Drift at $t = 0.5$	$\delta_s(t) = \mathbf{1}_{[0.5,1]}(t)$ and
Incremental Sinus Drift	$\delta_i(t) = -0.5 \cdot \left(\sin\left(\frac{\pi}{2} - \pi \cdot t\right) - 1\right)$,

with $\delta_s(0) = \delta_i(0) = 0$ and $\delta_s(1) = \delta_i(1) = 1$. The outcome of $\delta(t)$ will define the weight w of the drift, whereas $w = 0.5$ is the middle between two states indicating no derivation from the original objective function. All drift functions are illustrated in Figure 7.2.

Transformation Function: Finally, we define a function $g(\theta, w)$ which transforms the input depending on the state of the drift w. The function will be constructed in such a way that for a fixed transformation intensity K and $w = 0$: $g(\theta, 0) = \theta^{\frac{1}{K}}$, so that the optima are dragged to the left. For $w = 0.5$ no transformation should be applied so that $g(\theta, 0.5) = \theta$ for all values of K. For $w = 1$: $g(\theta, 1) = \theta^K$, which has the effect that all optima are dragged to the right.

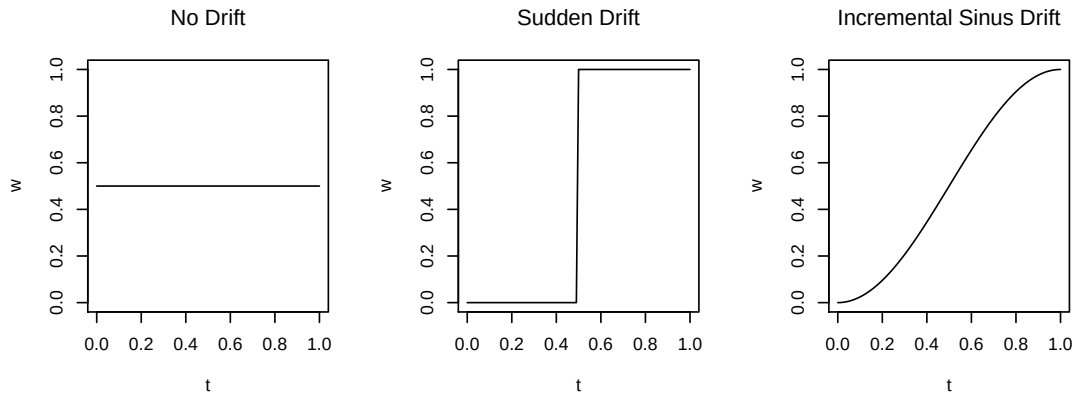

Figure 7.2: The three different drift functions δ_n, δ_i and δ_s.

To achieve this we define the transformation function for each dimension $l = 1, \ldots, d$:

$$g(\theta_l, w) := \theta_l^c \quad \text{with}$$
$$c = \frac{K+1}{1-K} \cdot \left(w - \frac{K}{K-1} \right)^{-1} - 1 \ . \tag{7.1.1}$$

The weight w controls the influence of the transformation and $K \in (1, \infty)$ is the maximal transformation intensity. In this work we chose $K = 3$ for a strong shift if w is 0 or 1. This function is displayed in Figure 7.3 for different values of w and K.

If the global optimum θ^* is known, its position at time t can be derived from the inverse of g. In fact g in Equation (7.1.1) is constructed so that the inverse solves to:

$$g^{-1}(\theta, w) = g(\theta, 1 - w). \tag{7.1.2}$$

For a one dimensional objective function with an optimum at $\theta^* = 0.5$, the optimum for $w = 0$ is at $g(0.5, 1) = 0.125$ and for $w = 1$ the optimum is at $g(0.5, 0) \approx 0.794$. We see that this transformation does not mean that $f(g(\theta, 1))$ is a mirrored version of $f(g(\theta, 0))$. To achieve that, $g(\theta, 1)$ would have to be a point reflection of $g(\theta, 0)$, instead it is reflected at the line of identity.

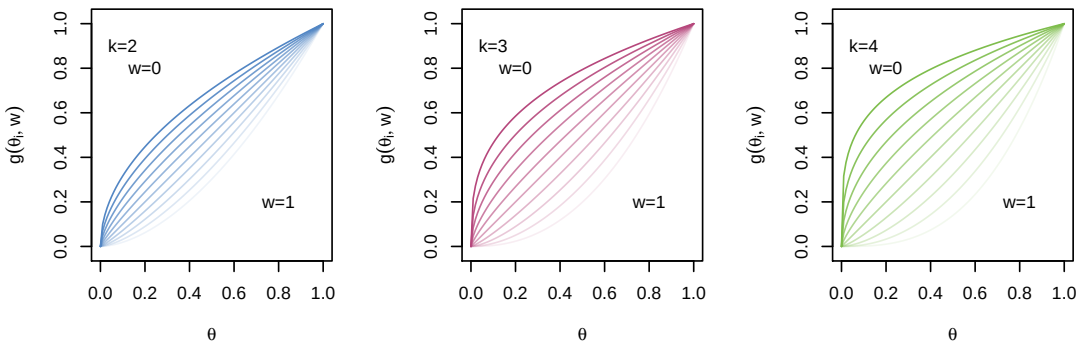

Figure 7.3: The transformation function $g(\theta, w)$ for $K = \{2, 3, 4\}$.

Combination: To put all pieces together, we first plug the drift function $\delta(t)$ into our transformation function to obtain $g(\theta_l, \delta(t))$. This means that the intensity of the transformation is controlled by the time through the drift function. Applying this scheme, we can combine any static objective function f with any drift function $\delta(t)$ and the transformation function g so that

$$f_{\delta,g}(\theta, t) = f(g(\theta, \delta(t))). \tag{7.1.3}$$

Figures 7.4 and 7.5 visualize how the functions and the optimum change over time for the one and two-dimensional DOPs constructed out of the static objective functions.

For DOPs derived through the presented procedure the optimal value is known for each point in time which allows us to calculate the MFE as defined in Section 5.6. Additionally, the optimal value does not change over time, which is advantageous because it allows to study the effect of the changing location of the optima independently from the effect of a changing optimal value. Although we only analyze changing locations in this work, the presented concept is easily extendable to construct DOPs with changing optimal values by introducing a second transformation function that transforms the outcomes depending on the time.

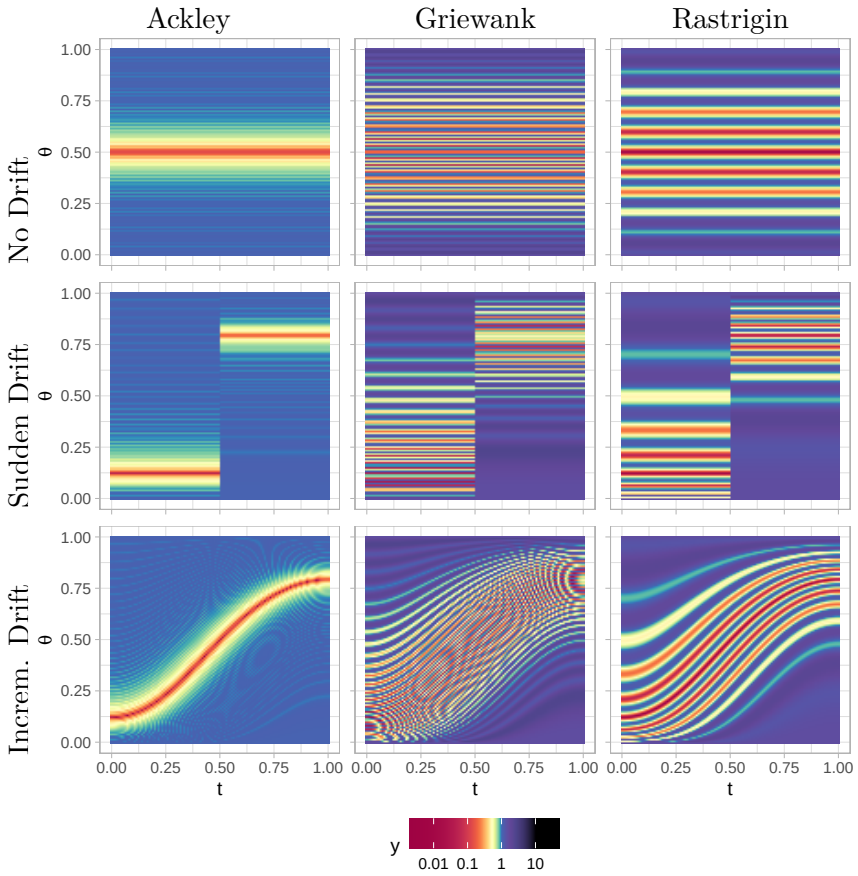

Figure 7.4: All $1d$ functions from set 1 with their different drifts.

The presented procedure has the advantage that the location θ_t^* of the optima can always be calculated analytically if the location of the optima of the original black-box function is known. As a consequence the distance to the optimum in the domain space can be calculated for each point in time which is an advantage because it allows more error measures to be used.

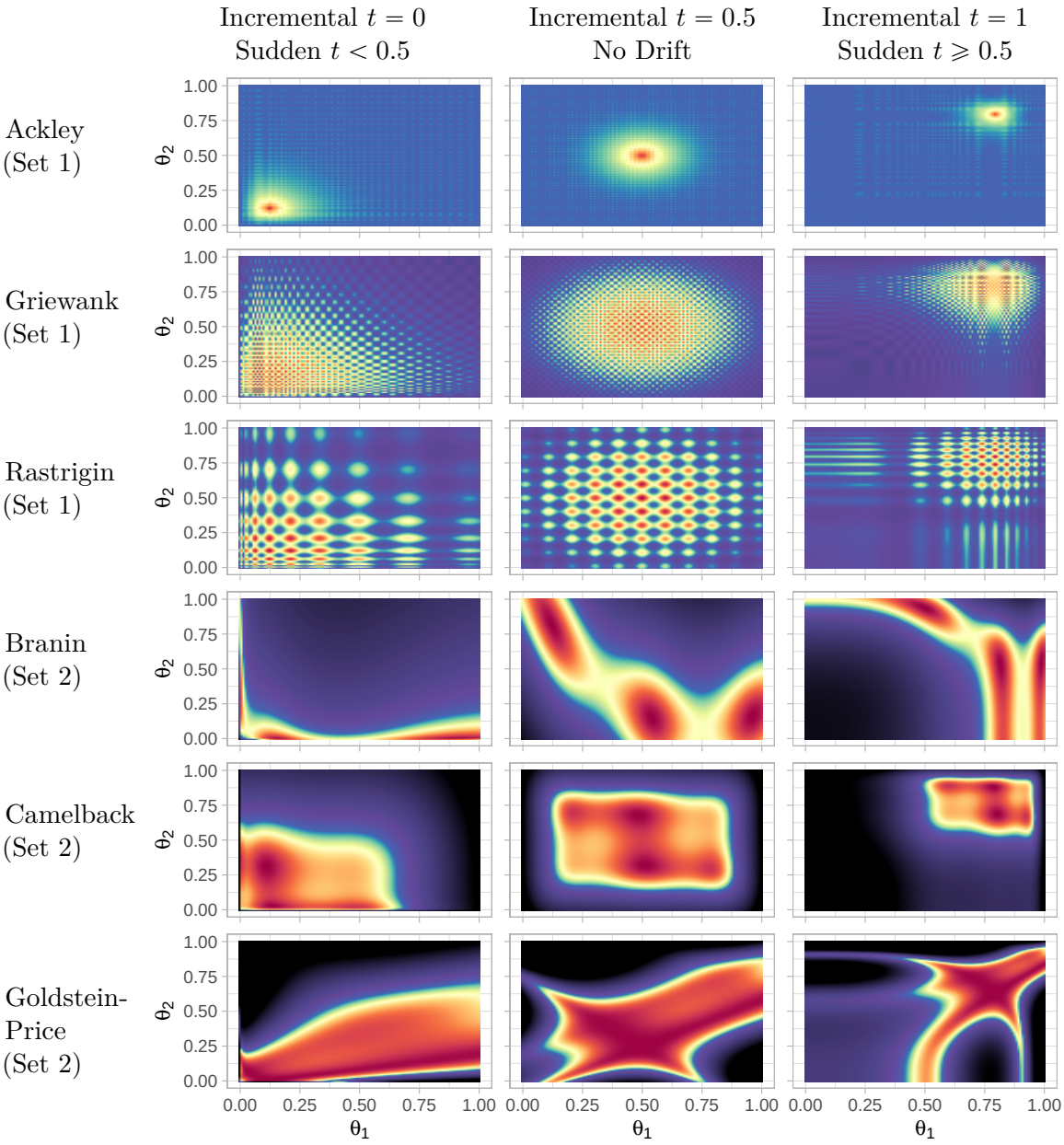

Figure 7.5: All $2d$ functions with incremental drift and their states at $t = 0$, $t = 0.5$ and $t = 1$. For $t = 0.5$ the state equals the constant state of the function with no drift. For $t = 0$ and $t = 1$ the states equal the two different states for the sudden drift. This figure uses the same color scale as Figure 7.4.

7.2 Setup

For the benchmark we consider two trivial baselines, the ordinary drift-unaware Bayesian optimization and the newly developed adaptations as presented in Chapter 5:

rs: Random search that at each time step evaluates a random configuration from the search space serving as a naive base-line.

constant: From the initial design the best configuration is taken and evaluated at each time step.

bo: Sequential MBO executed regularly without notion of concept drifts that might occur. At each time step the suggested point is evaluated.

bo_tw: Concept drift aware MBO with time-window (see Section 5.3) and two different window sizes of 20 and 40 time steps.

bo_tac: Concept drift aware MBO with time as covariate (see Section 5.4).

All MBO-related optimizers use a Gaussian process regression as a surrogate. The covariance is estimated with a Matern$\frac{5}{2}$-kernel. For `bo_tw` the *nugget effect* is estimated during the surrogate fit, to account for the non-deterministic behavior of the objective function over time.

Each of the `bo` optimization algorithms is run with two different acquisition functions.

_cb2 The *confidence bound* with $\lambda = 2$ as defined in Section 3.4.2.

_aei The *augmented expected improvement* as defined in Section 3.4.3 is used in combination with ordinary sequential MBO (`bo`) and `bo_tw`.

_tei The *temporal expected improvement* as defined in Equation (5.4.3) in Section 5.4 is only used in combination with the concept drift aware `bo_tac`.

The number of MBO iterations is chosen equivalent to the number of discrete time steps that are available. We conduct the experiments once by dividing the time frame of $[0, 1]$ into 50 time steps and once into 100 time steps. In other words, we conduct our experiments once with a budget of 50 optimization iterations and once 100 optimization iterations. This allows us to analyze how a different granularity on the time scale influences the optimization.

A single benchmark **problem** consists of a combination of a single DOP (i.e. an objective function with a specific drift type) and the optimization budget. Each optimizer is applied on each problem with 50 stochastic repetitions. For each repetition a different random initial design is generated by Latin hypercube sampling with $4 \cdot d$ points. All optimizers start with the same initial design on one problem within one stochastic repetition.

7.3 Evaluation

The following chapters provide a detailed analysis of the benchmark results for each drift type separately. For each *problem* we will visualize the fitness error averaged over all stochastic repetitions for each time step. This visual tool allows us to analyze how each optimizer behaves over time on a specific problem and for specific drift types.

To asses the performance of each optimizer, instead of looking at each optimization problem individually, the results of multiple problems will be grouped and summarized. A **group** consists of problems with functions that belong to the same function set (see Table 7.1) that have the same dimensionality and which are optimized with the same number of evaluations (50 or 100).

For each *group* two aggregating evaluations are carried out to compare the performance of the optimizers. For both evaluations the performance of each optimizer is calculated by the *mean fitness error* (MFE, see Section 5.6) for each single stochastic repetition of a problem.

The first evaluation method generates a preference graph. This directed graph shows which optimizer was able to beat which optimizer in a pairwise comparison. The pairwise comparison is conducted with a one-sided paired-sample sign test on the MFE values of each repetition and of each problem across two competing optimizers. The MFE values from two optimizers can be seen as paired because each result pair belongs to a specific problem and a specific stochastic repetition (implying the same initial design). Each test is conducted at the 5% significance level without further adjustment for multiple testing, as this analysis is only meant as an exploratory visualization of the stochastic results. Note that this non-parametric test results in the following logical pitfall:

$$A \text{ beats } B \wedge B \text{ beats } C \implies A \text{ beats } C \tag{7.3.1}$$

Corner cases can exist, where the sign test indicates that A is not significantly better than C, but A is significantly better than B and B is significantly better than C.

The results of all pairwise comparisons yield a directed graph. A directed line from Algorithm A to Algorithm B indicates that A yielded significantly better results than B. A transitive reduction on the results removes direct edges if there also exist an indirect path to improve readability. For example, if A beats B and C, and B beats C the edge between A and C is removed. This reduction introduces an ambiguity where we no longer can differentiate between the situation where A beats C significantly and the situation where A only beats C through B, as discussed above. However, if according to the test procedure A beats B and B beats C but A does not beat C in a direct comparision, it still holds that A performed better than C in the majority of cases, with the only restriction that the number of cases is not significant. Accordingly, it is a sensible decision to neglect this corner case for this visual tool and allow the conclusion that A performs better than C.

The second evaluation method analyzes the mean average ranks of each optimizer. Therefore, we first rank the optimizers among each other for each *problem* and each stochastic replication by their MFE. Afterwards, we calculate the mean of all ranks for each optimizer across all problems and replications within one *group*

of problems. For the sake of completeness the averaged MFE values for each problem and optimizer are also included in tabular fashion in the appendix. For both tables we conduct a statistical analysis on the underlying unsummarized data to find out which optimizers do not perform significantly worse than the best performing optimizer. To verify whether there are statistically significant differences between the optimizers, we perform a non-parametric test procedure as recommended in Demšar (2006). First, the Friedman test is employed to test whether there are any statistical differences between all optimizers. If the null hypothesis of no statistical differences between the optimizers cannot be rejected, we will underline all results to indicate that no algorithms are significantly different from each other. Second, if the null is rejected, we test each optimizer against the best performing one on the null hypothesis that the best performing optimizer is not better than the competitor. Therefore, the one-sided paired-sample sign test is applied on the MFE values of each repetition and each problem of the given group and the two competing optimizers. The test is adjusted for multiple comparisons using the Bonferroni-Holm-Correction. All results of optimizers that are not significantly worse than the best performing optimizer will be underlined to indicate that they are potential candidates for the best method. The test procedure is conducted at the global 5% significance level.

In a final comparison we will give a short combined analysis for all drift types that aims to guide a decision in case the drift type and function characteristics are not known beforehand.

7.3.1 No Drift

To understand specific characteristics of the different optimizers it is worthwhile to begin with the case of no drift. It is expected that the ordinary sequential `bo` methods perform best because they do not have to model a temporal influence and operate on the complete evaluation design. Also `bo_tw20` and `bo_tw40` are expected to behave exactly like ordinary `bo` for the first 20 and 40 optimization evaluations. For `bo_tac` we suspect a slightly worse performance than ordinary `bo`,

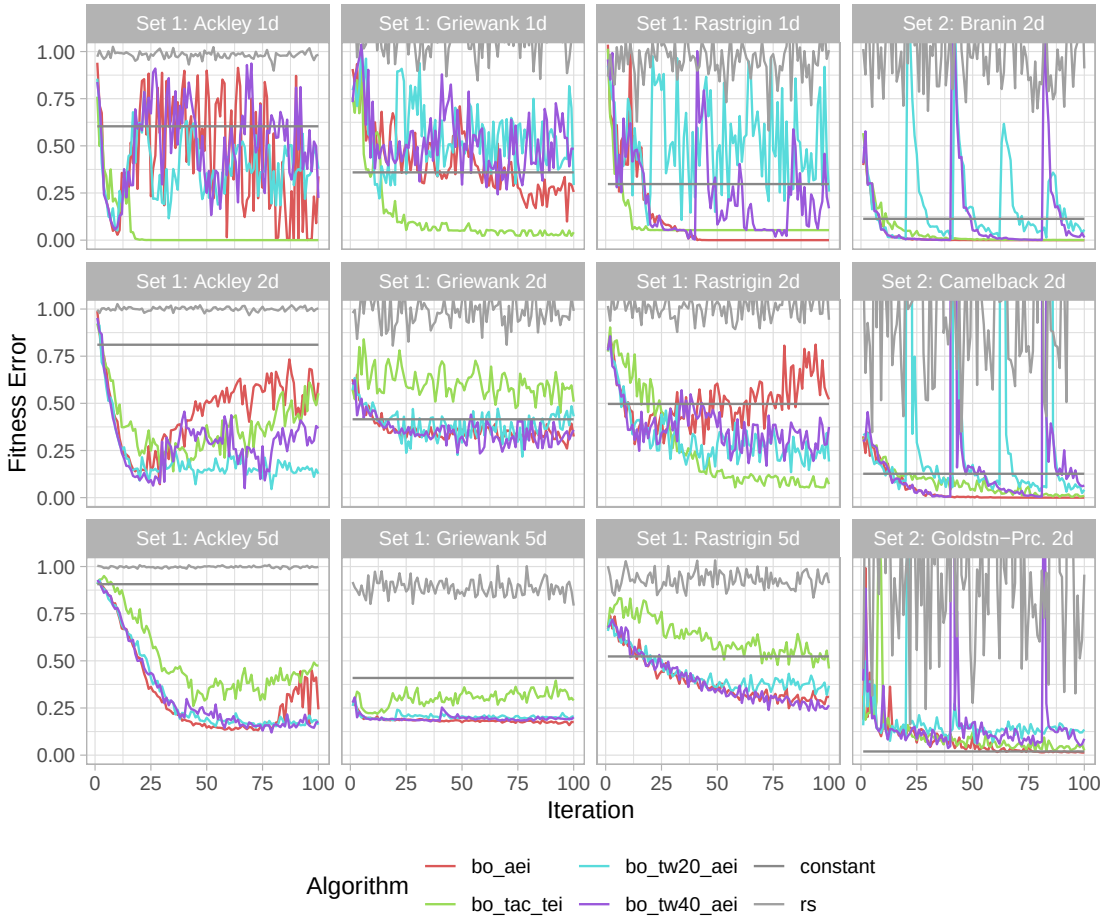

Figure 7.6: Optimization curves for optimizers that use the `aei` or `tei` acquisition function on problems with 100 optimization iterations and no drift.

since `bo_tac` models an additional variable which potentially lowers the prediction quality of the surrogate.

The plot in Figure 7.6 shows the fitness error at each time point averaged over all stochastic repetitions on a subset of problems with 100 optimization iterations and with EI-based acquisition functions. This plot only contains a subset of the results for better readability. The optimization curves for the `cb2` acquisition function and with 50 iterations are found in Appendix B on pages 136ff. These plots help to indicate some problems the optimizers are facing on specific functions.

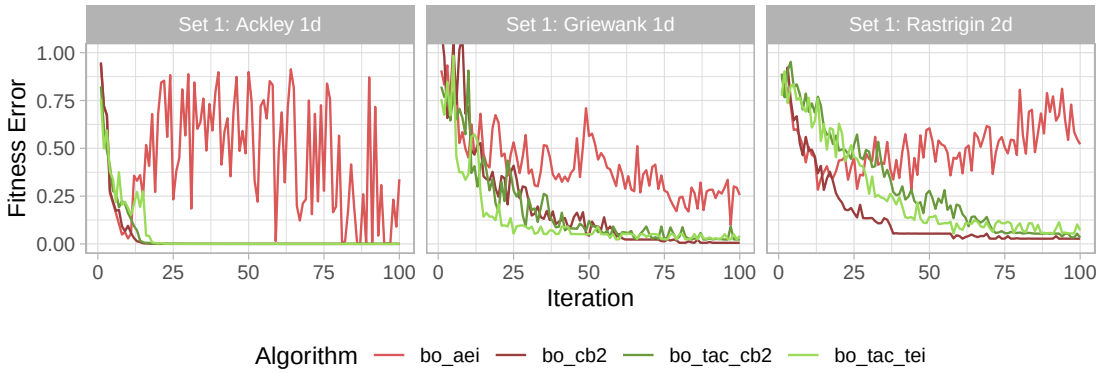

Figure 7.7: Exemplary optimization curves illustrating the effect of the choice of the acquisition function on selected objective functions of set 1 with no drift. The fact that `bo_aei` performs worst and `bo_cb2` performs best illustrates that in some cases it is not the specific MBO strategy that determines the performance but the choice of the acquisition function.

In the following, we will discuss some of the peculiarities before looking at the overall performance of all algorithms.

Contrary to the expectations, the `bo_tac_tei` optimizer shows better performance than `bo_aei` for the `Ackley 1d`, `Griewank 1d` and `Rastrigin 2d` function even though it has to model an additional covariate. These functions are heavily multimodal as they are all overlayed with a cosine wave with a high frequency. Looking at Figure 7.7 we see that using another acquisition function leads to better results for the ordinary Bayesian optimizer `bo_cb2`. This leads to the assumption that the `aei` acquisition function fails in these cases, as it interprets the cosine waves as noise and puts more emphasis on exploration which leads to many evaluations with bad performance. This assumption is backed by Figure 7.8 which shows the evaluated points in the search space and their function outcome for the `Ackley 1d` function for one stochastic repetition. However, the bad performance due to exploration for `bo_aei` is not a general rule, as we can see in the results of `Rastrigin 1d`, `Rastrigin 5d` and `Griewank 5d` in Figure 7.6 for these functions of set 1. It also has to be stressed that exploration is a desired behavior of a global optimizer. However, for specific target functions it can be advantageous to search locally and exploit the knowledge of a local optimum. Adding to that,

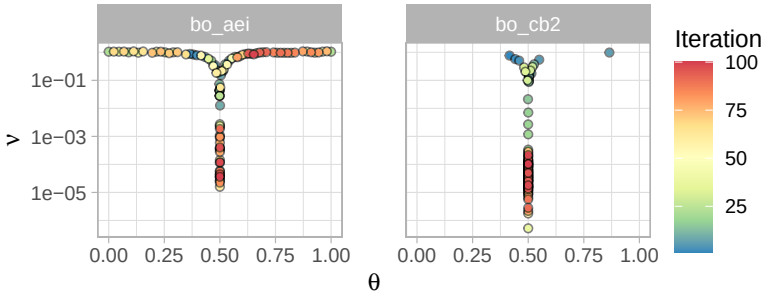

Figure 7.8: For the `Ackely 1d` function the `bo_aei` optimizer does many exploratory evaluations even in the later iterations whereas `bo_cb2` does many evaluations close to the global optimum. Note the *log*-scale.

in our setting, the chosen error measure punishes exploration as each exploratory evaluation with a performance below the average increases the MFE. In a setting without concept drift we would only measure the error of the best found setting and all exploratory evaluations would not decrease the final performance value.

The `constant` approach achieves a low error for some $2d$ problems (`Goldstein-Price 2d` and `Griewank 2d`). For these objective functions the initial design seems to be sufficient to find a good solution and `constant` benefits from not having exploratory evaluations that negatively affect the average error over time.

The bad performance of `bo_aei` can probably be attributed to the exploratory characteristic of the algorithm. In Figure 7.6 we see that the error increases after iteration 25 for `Ackley 2d` and after iteration 75 for `Ackley 5d`. Here `bo_aei` probably has found a good configuration but the acquisition function forbids the algorithm to revisit this region and the exploration only leads to configurations that yield a bad performance. In the following, we will call this the *saturation* effect, where the surrogate has reached a certain security that prevents exploitation in the area of the optimum. Again, if we were interested in the performance on a static optimization problem, only the lowest point of the curve would determine the final performance.

For the window-based optimizers (`bo_tw`) a distinct pattern can be observed. At iteration 20 (for `bo_tw20`) and 40 (for `bo_tw40`) the error suddenly increases which

is especially visible for `Branin 2d` and `Camelback 2d`. At these iterations the information from the initial design falls out of the window and the uncertainty predicted by the surrogate increases in certain regions. Note that all evaluations from the initial design are assigned to $t = 0$. The re-exploration leads to an error peak as bad performing areas are reevaluated. This pattern repeats all 20 (for `bo_tw20`) and 40 (for `bo_tw40`) iterations as information about bad performing configurations gets repeatedly lost.

The additional optimization curves for 50 iterations (Fig. B.1) and for the `cb2` acquisition function with 50 (Fig. B.2) and 100 (Fig. B.3) iterations are given in the Appendix B.1 on pages 136ff. The optimization curves for 50 iterations are given for completeness. Basically, they are equal to the first 50 iterations of the 100 iteration setting. The comparison between the optimization curves from the EI based optimizers in Figure 7.6 and the optimizers that use the `cb2` acquisition function in Figure B.3 demonstrates that the confidence bound does not punish exploitation as much as the `aei`. Therefore, the mentioned saturation effect does not occur that notably, which is especially visible for the `Ackley` functions, where the `cb2` optimizers constantly obtain low error values.

The preference plot in Figure 7.9 shows which optimization algorithms were able to beat each other in a pairwise comparison across all problems of set 1 with the same dimensionality and with the same number of optimization iterations. The plots show that each optimizer with the `cb2` acquisition function can beat its `aei`/`tei` counterpart in $1d$ and $2d$. For $5d$ this only holds for `bo_tac_cb2` that beats `bo_tac_tei`. The plots also indicate that `bo_cb2` performs well across all subsets. The `bo_tac` approaches perform notably worse on the `5d` problems and comparably well to the ordinary `bo_aei`/`bo_cb2` approaches on the $1d$ problems. The results of the time window (`bo_tw`) approaches are inconclusive. Even though no drift occurs in the objective function and the shorter time window approach leads to a less accurate surrogate, `bo_tw40_aei` cannot beat `bo_tw20_aei` on the $1d$ and $2d$ groups of set 1, which can also be accounted to the saturation effect. If we only look at time window approaches with the `cb2` acquisition function, the order is as expected and `bo_tw40_cb2` beats `bo_tw20_cb2` in all groups. This

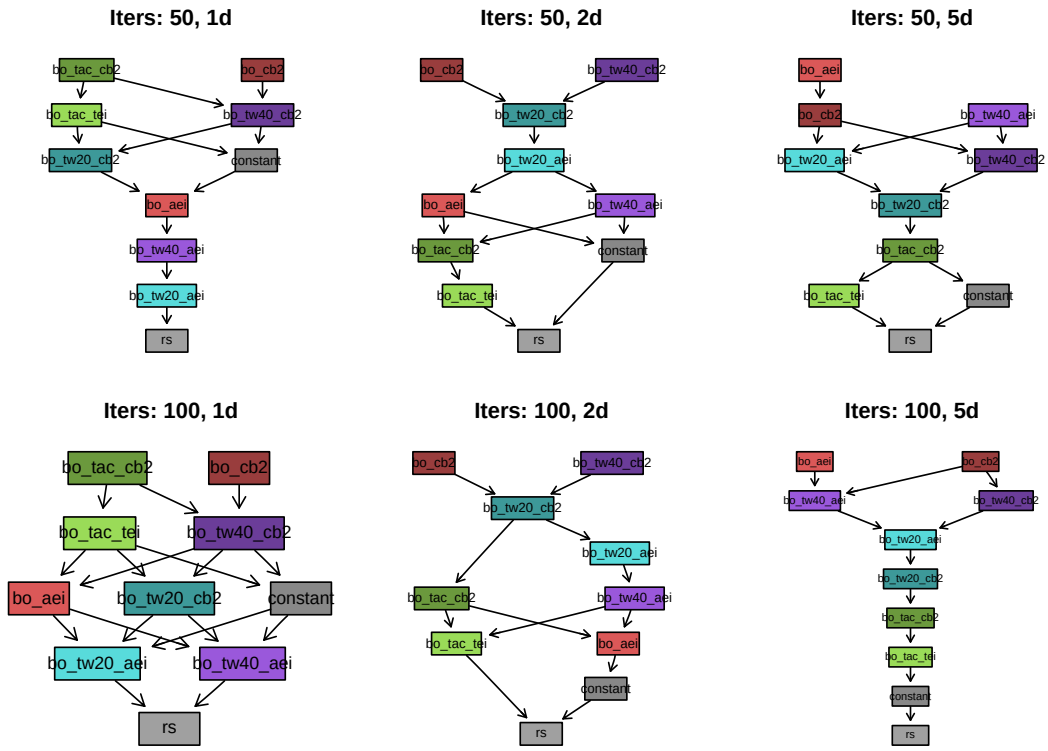

Figure 7.9: Preference graphs for each problem group of set 1 with no drifts.

highlights that the behavior induced by the choice of the acquisition function, can heavily influence the final performance.

The preference plot for problems of set 2 is given in Figure B.4 on page 139. On set 2 optimizers with the `cb2` acquisition function no longer outperform their `aei`/`tei` counterparts. On the contrary, `bo_aei` performs better than `bo_cb2` and in general the optimizers using the `aei`/`tei` acquisition function outperform their `cb2` counterparts.

The average ranked performance of each optimizer is displayed in Table 7.2. The ordinary `bo_cb2` obtains good rankings. Only on the $5d$ problems of set 1 and on set 2 ordinary `bo_aei` performs well. It appears that the optimizers with the `tac` approach perform worse on the $2d$ and $5d$ problems of set 1 than their `tw` counterparts.

Table 7.2: Rankings of the optimization methods on each subset of problems with no drift.

			AEI					CB2					
Set	d	Iterations	bo_aei	bo_tac_tei	bo_tw20_aei	bo_tw40_aei	bo_cb2	bo_tac_cb2	bo_tw20_cb2	bo_tw40_cb2	rs	constant
Set 1	1	50	6.34 (7)	3.39 (3)	7.37 (9)	7.01 (8)	3.32 (2)	**3.24** (1)	5.43 (6)	3.72 (4)	9.87 (10)	5.31 (5)
		100	5.99 (6)	3.23 (3)	7.55 (9)	7.31 (8)	**2.77** (1)	2.94 (2)	6.05 (7)	3.34 (4)	9.89 (10)	5.92 (5)
	2	50	5.17 (6)	7.27 (9)	4.57 (4)	5.09 (5)	**2.98** (1)	6.61 (8)	3.78 (3)	3.45 (2)	9.99 (10)	6.09 (7)
		100	6.28 (7)	7.01 (9)	4.79 (4)	5.15 (5)	2.83 (2)	5.41 (6)	4.13 (3)	**2.74** (1)	9.99 (10)	6.67 (8)
	5	50	**2.93** (1)	7.83 (9)	4.09 (4)	3.37 (2)	3.48 (3)	7.06 (7)	4.94 (6)	4.16 (5)	9.95 (10)	7.19 (8)
		100	3.15 (2)	7.99 (9)	4.48 (5)	3.57 (4)	**2.56** (1)	6.88 (7)	5.45 (6)	3.24 (3)	9.99 (10)	7.68 (8)
Set 2	2	50	2.57 (2)	3.37 (3)	7.45 (8)	5.35 (6)	3.93 (4)	5.17 (5)	8.93 (9)	6.85 (7)	9.61 (10)	**1.78** (1)
		100	2.43 (2)	3.30 (3)	7.62 (8)	5.93 (6)	3.32 (4)	4.71 (5)	8.93 (9)	6.83 (7)	9.86 (10)	**2.06** (1)

For set 1 the optimizers with the `cb2` acquisition function mostly yield better performance than their `aei`/`tei` counterparts and vice versa for set 2. For set 2 surprisingly `constant` is the best performing strategy. As we can see for the other optimizers in Figure 7.6 the outliers with bad performance negatively affect the final performance as well as the bad fitness error in the beginning of the optimization. In Table 7.3 the *mean fitness error* is given for each problem of set 2 individually and in Table B.1 on page 140 the errors for all problems are listed. However, we

Table 7.3: MFE and its standard deviation of each optimization method on each problem of set 2 with no drift averaged over 50 stochastic repetitions.

		AEI				CB2					
Function	Iterations	bo_aei	bo_tac_tei	bo_tw20_aei	bo_tw40_aei	bo_cb2	bo_tac_cb2	bo_tw20_cb2	bo_tw40_cb2	rs	constant
Branin	50	**0.14** ±0.07	0.18 ±0.11	0.52 ±0.11	0.31 ±0.07	0.20 ±0.09	0.30 ±0.14	0.70 ±0.12	0.35 ±0.09	1.54 ±0.25	0.20 ±0.19
	100	**0.07** ±0.03	0.11 ±0.07	0.50 ±0.07	0.27 ±0.04	0.09 ±0.04	0.15 ±0.06	0.64 ±0.09	0.30 ±0.06	1.54 ±0.15	0.20 ±0.19
Camelback	50	0.47 ±0.23	0.74 ±0.71	1.73 ±0.20	1.04 ±0.23	0.75 ±0.30	0.97 ±0.51	2.06 ±0.21	1.28 ±0.22	2.16 ±0.39	**0.13** ±0.08
	100	0.26 ±0.12	0.40 ±0.34	1.58 ±0.13	0.94 ±0.11	0.35 ±0.16	0.54 ±0.35	1.80 ±0.17	1.09 ±0.16	2.23 ±0.28	**0.13** ±0.08
Gldstn-Pr	50	1.77 ±0.95	2.02 ±2.11	3.60 ±1.41	2.60 ±1.15	2.36 ±1.22	2.92 ±2.12	6.17 ±1.91	4.63 ±1.83	8.39 ±2.77	**0.05** ±0.10
	100	0.94 ±0.64	0.96 ±0.82	3.22 ±0.98	2.47 ±0.84	1.32 ±0.63	2.05 ±1.33	5.97 ±1.81	3.71 ±1.08	7.92 ±1.66	**0.05** ±0.10

are not discussing each problem in detail, as we try to draw conclusions that are generally applicable.

We looked at the results with no drift and expected `bo_cb2` or `bo_aei` to perform best. In general these ordinary methods work well and mostly exploratory evaluations decrease their performance for the concept drift specific error measure. On selected functions the choice of the acquisition functions heavily changes the behavior and thus the performance of an optimizer. On functions with higher dimensionality the `tac` approaches perform worse than the drift-unaware optimizers.

7.3.2 Sudden Drift

For all problems with a sudden drift the change happens at $t = 0.5$, so after 25 or 50 optimization iterations. We expect that the fitness error for all optimizers increases dramatically at the sudden change. For the concept drift unaware optimizers `bo_aei` and `bo_cb2` we do not expect any recovery after the sudden change. For the concept drift aware optimizers we expect that the fitness error decreases again after the change and ideally it recovers to values close to the fitness error before the sudden change. The time-as-covariate approaches try to model the influence of the time. Obviously, the surrogate will not be able to anticipate the sudden change. It will need some iterations so that the surrogate realizes that the change should be attributed to the time covariate. Additionally, this discontinuity will challenge the Gaussian process regression, which expects input from a differentiable function. For the mentioned reasons we do not expect the `bo_tac` approaches to perform best. The windowed `bo_tw*` optimizers should reach better performances according to their window size. For `bo_tw20` we expect that at least after 20 iterations the fitness error decreases, since all outdated observations are forgotten until then. For `bo_tw40` we expect the same behavior after 40 iterations.

Until $t = 0.5$ the function is static and therefore we expect the same behavior from the optimizers as in the previous Section 7.3.1. However, since the functions at

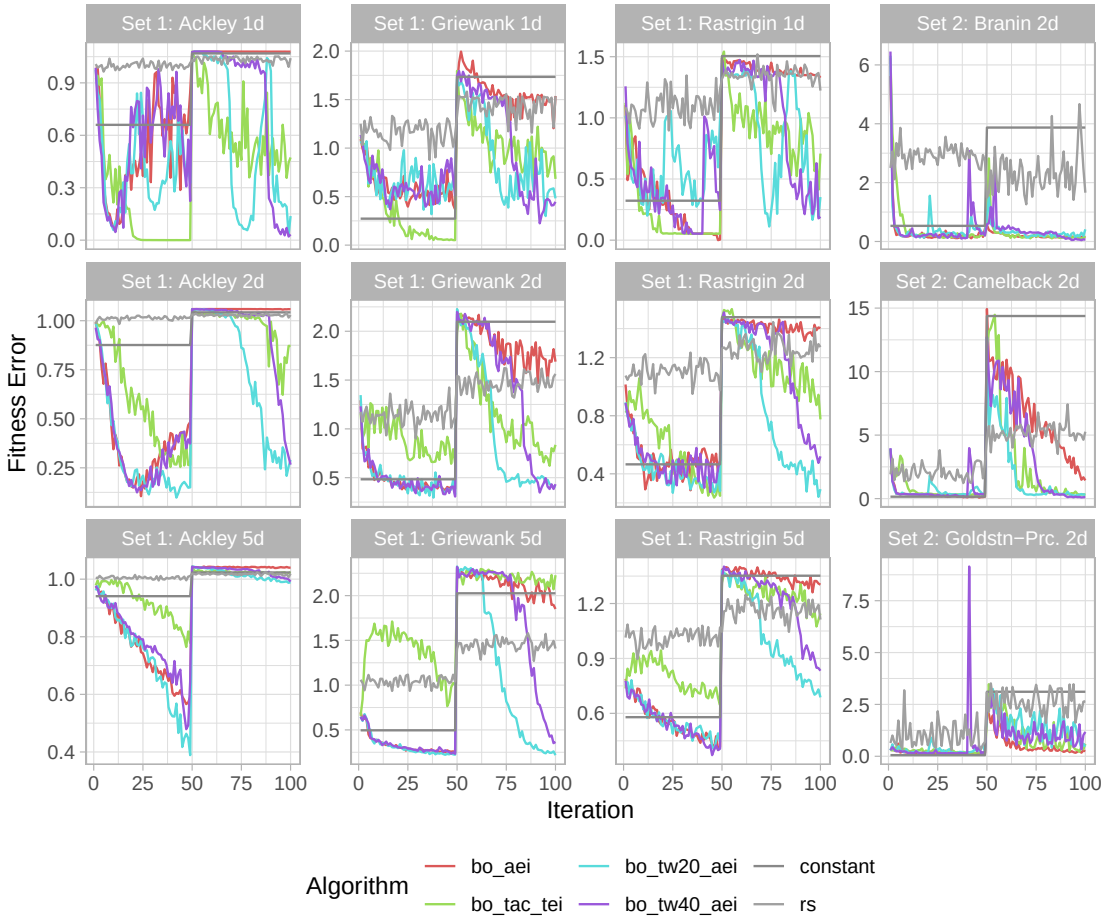

Figure 7.10: Optimization curves for optimizers that use the `aei` or `tei` acquisition function on all problems with 100 optimization iterations and a sudden drift.

$t \in [0, 0.5)$ are different from the functions without drift which are untransformed (see Figures 7.4, 7.5), the performance can vary.

In Figure 7.10 it is visible that `bo_aei` and all other optimizers are directly impacted by the sudden change for all functions. Only for functions of set 2 the impact seems less severe and `bo_aei` can recover better than for functions of set 1. On set 2 `bo_tac_tei` is also able to adapt the changes, whereas for all $5d$ problems of set 1, we hardly see any recovery. It seems that `bo_tac_tei` works better on the lower dimensional problems. The behavior of the windowed optimizers is nearly as expected. It is clearly visible that `bo_tw_20` adapts faster than `bo_tw_40`.

Similar to the *no drift* scenario we see, that after 20 and 40 iterations we have recurring peaks of high error values, which are especially visible for `Ackley 1d` and `Rastrigin 1d` and `Branin 2d`. More exploratory evaluations of the smaller window might negatively affect the performance but the faster adaptation seems to make up for it. Interestingly the surrogate obtained by the smaller window size seems to suffice to reach low error levels, since for most functions both window sizes lead to similarly low error levels. For `Ackley 5d` no optimizer was able to recover after the sudden change.

The additional optimization curves for 50 iterations (Fig. B.5) and for the `cb2` acquisition function with 50 (Fig. B.6) and 100 (Fig. B.7) iterations are given in the Appendix B.2 on pages 141ff. The observable behaviors are similar to the ones from Figure 7.10. In the 50 iteration setting `bo_tac_tei` and `bo_tac_cb2` are less able to recover after the sudden change than within the 100 iteration setting. For `Ackley 1d` and `Ackely 2d` we can observe a performance decrease even before the drift occurs in Figure 7.10 (100 iterations). This saturation effect was already discussed in the previous Section 7.3.1. If the drift occurs earlier (see Figure B.5, 50 iterations) such effects naturally cannot be observed, since the saturation of the surrogate has not been reached. In the direct comparison between the two acquisition functions, we again notice that the saturation effect is less observable for `cb2` based optimizers. The recovery after the sudden change does not seem to be affected by the choice of the acquisition function.

Looking at the preference graphs in Figure 7.11, we clearly see that the smaller window sizes have been beneficial for this benchmark. All optimizers with a window size of 20 iterations beat their counterpart with a windows size of 40. The time-as-covariate approach only works well for the one-dimensional problems of set 1. Similar to our observations on the problems with no drift, the window-based optimizers that use the expected improvement based acquisition functions (`bo_tw*_aei`) perform better on the 5-dimensional problems with 100 iterations than their `bo_tw*_cb2` counterparts. For the other cases the results are somewhat inconclusive giving a slight benefit to the confidence bound.

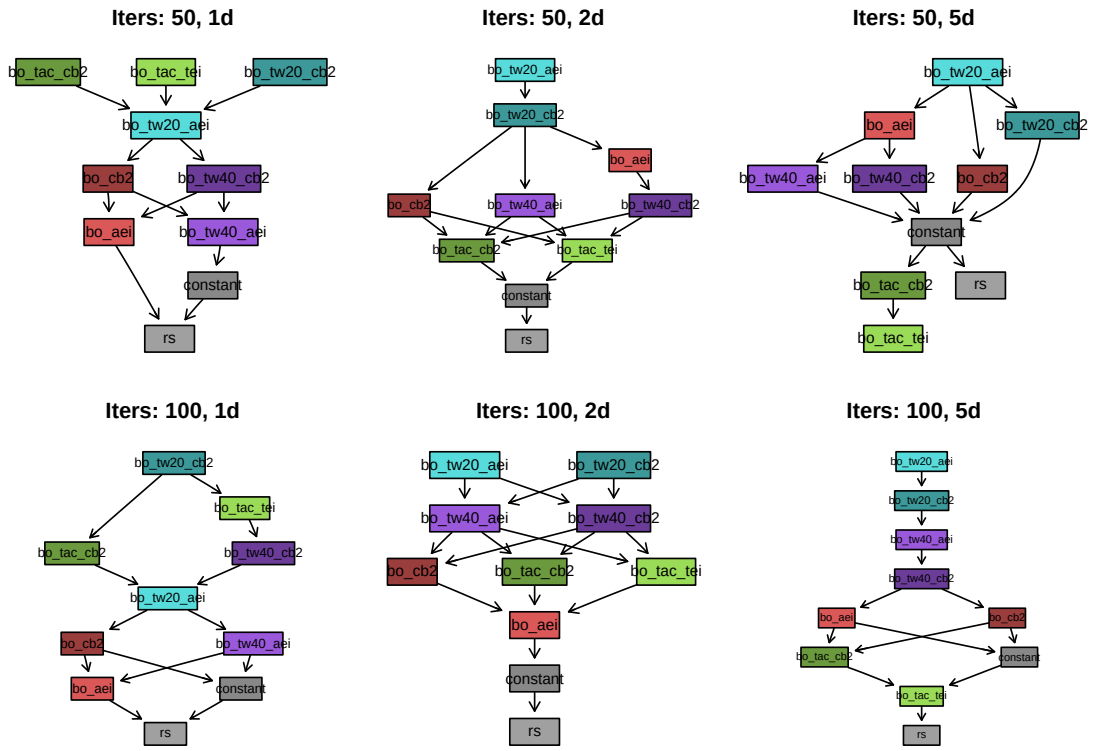

Figure 7.11: Preference graphs for each problem group of set 1 with sudden drifts.

The preference plot for problems of set 2 is given in Figure B.8 on page 144. Similar to our observations on functions with no drift, the optimizers using the `aei`/`tei` acquisition function outperform their `cb2` counterparts on set 2. Also the bigger window size appears to be the better choice for set 2 in contrast to set 1.

In conclusion `bo_tw20_aei` performed well for problems with sudden drift. The averaged ranks in Table 7.4 show that it did not perform best for the one-dimensional problems of set 1 and for set 2. However, if we look at the individual mean fitness errors in Table B.2, we see that `bo_tw20_aei` at least performed comparably well for a few functions out of these subsets, namely `Griewank 1d` and `Camelback 2d`. Also `bo_tw20_cb2` performed best on the one-dimensional problems of set 1, strengthening the claim that the choice of the acquisition function can dominate the performance of an algorithm. For set 2 `bo_tac_tei` performs well, next to the

Table 7.4: Rankings of the optimization methods on each subset of problems with sudden drifts.

			AEI				CB2					
Set	d	Iterations	bo_aei	bo_tac_tei	bo_tw20_aei	bo_tw40_aei	bo_cb2	bo_tac_cb2	bo_tw20_cb2	bo_tw40_cb2	rs	constant
Set 1	1	50	6.86 (8)	3.20 (2)	3.96 (4)	6.65 (7)	5.87 (6)	3.51 (3)	**2.83** (1)	5.70 (5)	9.55 (10)	6.87 (9)
		100	8.07 (9)	2.88 (2)	4.77 (5)	5.94 (6)	6.74 (7)	3.35 (4)	**2.24** (1)	3.30 (3)	9.81 (10)	7.91 (8)
	2	50	4.96 (3)	6.27 (7)	**2.39** (1)	5.15 (4)	5.32 (5)	6.27 (7)	3.37 (2)	5.35 (6)	9.16 (10)	6.77 (9)
		100	6.92 (8)	5.96 (6)	**1.92** (1)	4.43 (4)	6.35 (7)	5.53 (5)	2.03 (2)	4.13 (3)	9.69 (10)	8.04 (9)
	5	50	3.85 (2)	8.23 (10)	**2.92** (1)	4.44 (4)	4.60 (5)	7.90 (8)	4.35 (3)	4.75 (6)	8.19 (9)	5.76 (7)
		100	5.33 (6)	8.37 (9)	**1.99** (1)	3.64 (3)	5.18 (5)	7.35 (8)	2.61 (2)	4.40 (4)	8.89 (10)	7.24 (7)
Set 2	2	50	**4.23** (1)	4.73 (3)	5.07 (5)	4.89 (4)	4.67 (2)	5.75 (7)	6.31 (9)	5.46 (6)	7.91 (10)	5.99 (8)
		100	4.33 (4)	**3.93** (1)	5.58 (6)	4.93 (5)	4.28 (2)	4.28 (2)	7.05 (9)	5.61 (7)	8.75 (10)	6.27 (8)

ordinary drift-unaware optimizers. Interestingly, the number of iterations did not have a notable effect on the rankings of the optimizers.

7.3.3 Incremental Drift

For problems with an incremental drift a constant change of the functions response surface leads to the assumption that drift-unaware optimizers will slowly decrease in performance. We expect window-based optimizers to be able to adapt to the steady change although they will likely lack behind, since the surrogate is always trained on an outdated state of the function. In contrast, the time-as-covariate approach seems to be the most promising in this scenario. In an ideal case the surrogate is able to learn and predict the influence of the time, resulting in an overall low fitness error.

The optimization curves in Figure 7.12 partly confirm our assumptions. The performance of `bo_tei` decreases to the same extent as the `constant` proposal towards the end on functions of set 2. Surprisingly, even the random evaluations of `rs` reach a better performance after iteration 50 for `Griewank 2d`, `Griewank 5d`, `Rastrigin 2d` and `Rastrigin 5d`. In the first 20 and 40 iterations the curves for `bo_tw20_tei` and `bo_tw40_tei` follow the same path as the ones for `bo_tei`. In the first 20 and 40 iterations the curves for `bo_tw20_tei` and `bo_tw40_tei` follow the same path as the ones for `bo_tei`. This is expected, since as long as the initial design is still included in the window, these methods work identical. Still the adaptation to the incremental drift is surprisingly slow for functions of set 1. Similar to the observations made on functions with sudden drift, also here, the smaller window size is beneficial and allows a faster adaptation to the drift. Again, the smaller window size does not increase the fitness error due to a surrogate with less information.

Functions of set 2 have a comparably simple response surface with relatively small regions of very high function outcomes. The `constant` approach suffers from this characteristic, since the previously best setting from the initial design falls into such an area of high function outcomes after half of the time, which can also be seen in Figure 7.5. In contrast all other optimizers visually appear to perform well. Apart from some outliers in the beginning, `bo_tac_tei` reaches the lowest error values throughout the time. For the window-based optimizers we can observe the same behavior on set 2 as for functions without drift in general. The loss of

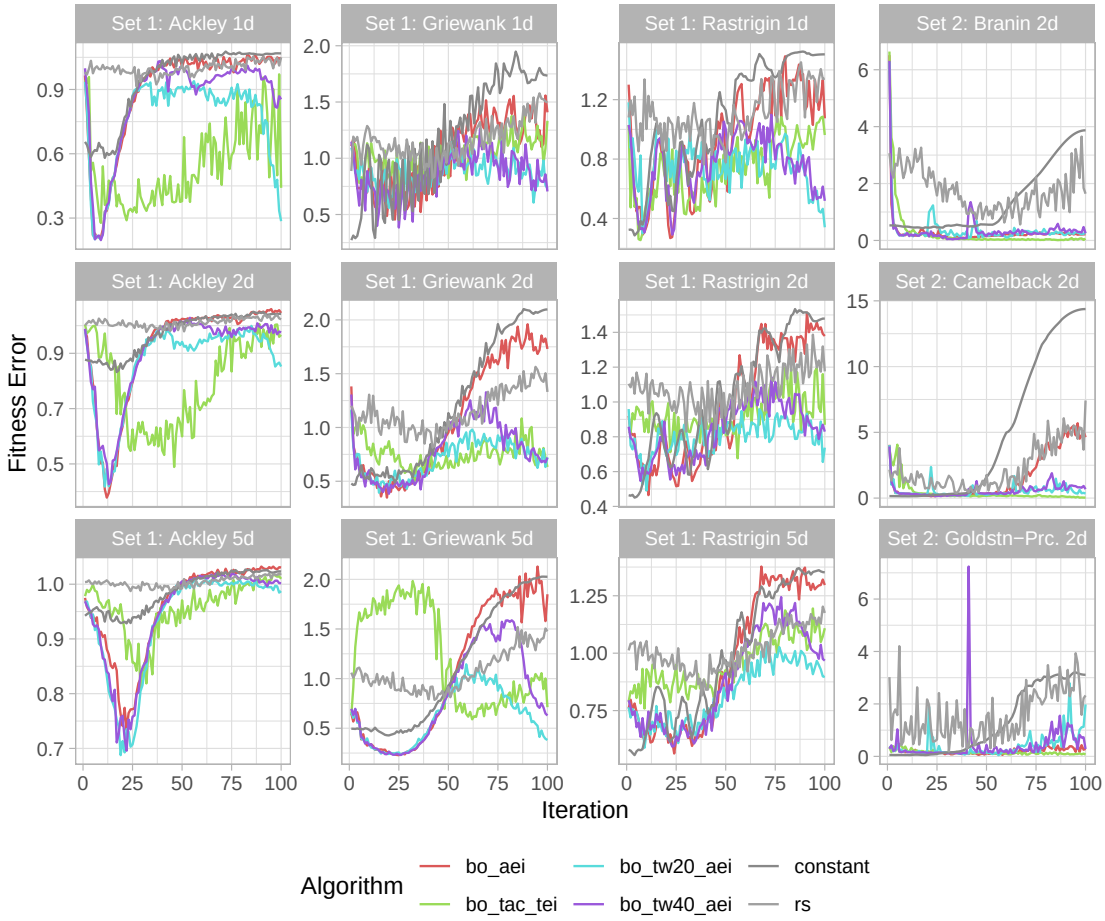

Figure 7.12: Optimization curves for optimizers that use the `aei` or `tei` acquisition function on all problems with 100 optimization iterations and an incremental drift.

informations from the initial design and badly performing points lead to recurring peaks of a high fitness error.

The additional optimization curves for 50 iterations (Fig. B.9) and for the `cb2` acquisition function with 50 (Fig. B.10) and 100 (Fig. B.11) iterations are given in the Appendix B.3 on pages 146ff. In the 50 iteration setting `bo_tac_tei` and `bo_tac_cb2` are less able to follow the incremental change than within the 100 iteration setting, which is especially visible for the `Ackley 2d`, `Ackley 5d` and `Griewank 5d` functions. As expected, the performance of `bo_tw40` is similarly bad

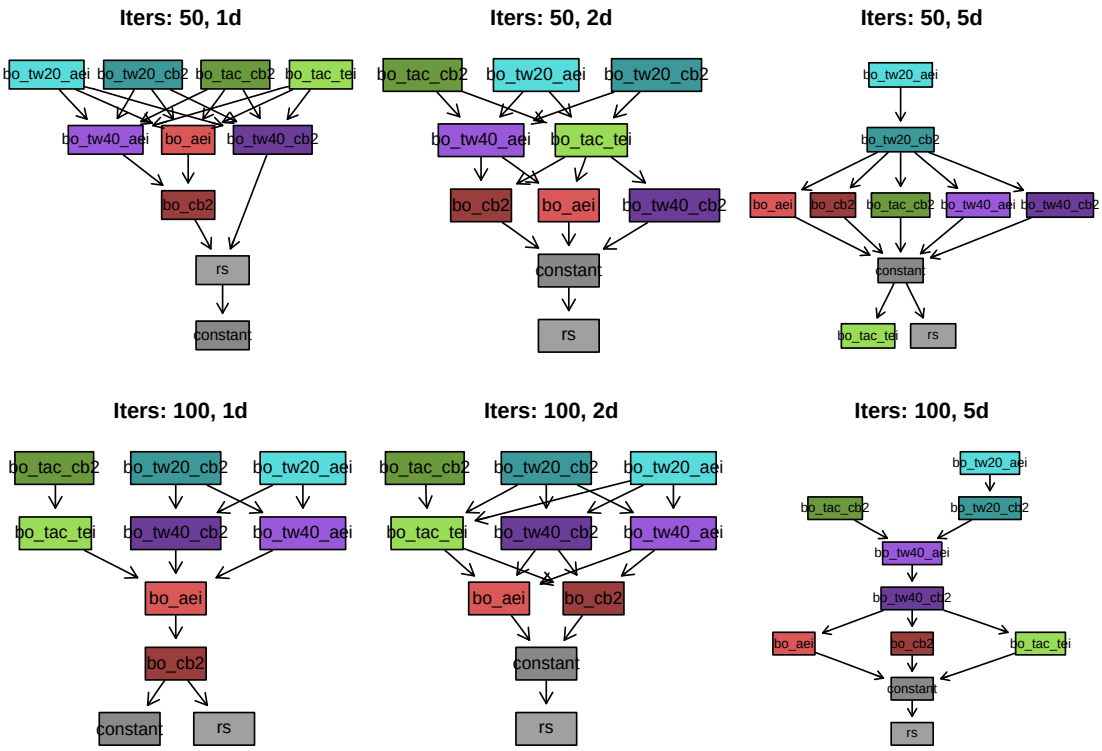

Figure 7.13: Preference graphs for each problem group of set 1 with incremental drifts.

to the drift-unaware `bo` methods for the 50 iteration setting, since the design is identical until iteration 40. But also the smaller window size is not sufficient to make the optimizer aware of the continuous change of the objective function within 50 iterations on set 1. The direct comparison between the two acquisition functions shows that `bo_tac_cb2` is better able to follow the optimum than `bo_tac_aei` on the `Ackley` and `Griewank` functions. We can assume that the exploitative characteristic of the confidence bound is beneficial in this scenario as it allows the surrogate to better fit in the region of the current global optima.

The individual preference plots in Figure 7.13 show that in all cases for the window-based optimizers the smaller window performs better, even independent of the acquisition function. A specific preference for an acquisition function is less clear, although `bo_tw20_aei` is never beat by `bo_tw20_cb2` and for $5d$ problems

Table 7.5: Rankings of the optimization methods on each subset of problems with incremental drifts.

			AEI				CB2					
Set	d	Iterations	bo_aei	bo_tac_tei	bo_tw20_aei	bo_tw40_aei	bo_cb2	bo_tac_cb2	bo_tw20_cb2	bo_tw40_cb2	rs	constant
Set 1	1	50	6.05 (6)	3.77 (4)	3.62 (3)	5.72 (5)	6.35 (8)	3.58 (2)	**3.49** (1)	6.07 (7)	8.21 (10)	8.15 (9)
		100	6.74 (7)	4.11 (5)	3.37 (2)	4.05 (3)	7.28 (8)	4.05 (3)	**3.12** (1)	4.17 (6)	9.25 (10)	8.87 (9)
	2	50	6.17 (7)	4.95 (4)	**3.27** (1)	5.55 (5)	6.26 (8)	3.67 (3)	3.43 (2)	5.89 (6)	8.53 (10)	7.28 (9)
		100	7.29 (7)	4.23 (5)	**2.94** (1)	4.25 (6)	7.55 (8)	3.59 (3)	3.17 (2)	4.19 (4)	9.35 (10)	8.43 (9)
	5	50	5.27 (5)	7.59 (9)	**3.03** (1)	5.45 (7)	5.05 (3)	5.11 (4)	3.55 (2)	5.29 (6)	8.01 (10)	6.65 (8)
		100	6.75 (8)	6.26 (6)	**2.65** (1)	4.21 (4)	6.59 (7)	3.85 (3)	3.09 (2)	4.37 (5)	9.01 (10)	8.21 (9)
Set 2	2	50	**3.54** (1)	3.64 (2)	5.41 (6)	5.25 (5)	4.10 (3)	4.52 (4)	7.69 (9)	5.98 (7)	8.79 (10)	6.08 (8)
		100	4.02 (4)	**2.61** (1)	6.22 (6)	4.65 (5)	4.00 (3)	3.21 (2)	8.15 (9)	6.25 (7)	9.32 (10)	6.56 (8)

`bo_tac_cb2` clearly outperforms `bo_tac_tei`. The time-as-covariate approach only outperforms the `bo_tw20` approaches on problems of set 2 (see Figure B.12). This indicates that the time-as-covariate approach only works well, if the surrogate can reliably model the response surface.

The averaged rankings in Table 7.5 show that apart from some cases `bo_tw20_aei` and `bo_tw20_cb2` are promising optimizers although their performance is lacking on problems of set 2.

The results of the time-as-covariate approaches are inconclusive. On set 2 `bo_tac_tei` performs best, whereas on set 1 `bo_tac_cb2` works comparably well. The number of iterations does not have a notable effect on the rankings of the optimizers.

7.3.4 Drifts combined

If the drift type is not known in advance, the question remains, which optimizer is an overall good choice. This is very hard to answer, as we saw that different effects can determine the success of a specific optimizer and there is no general winner. Additionally, in a real application, some prior knowledge about the drift

Table 7.6: Rankings of the optimization methods on each subset of problems with all drift types combined and on all problems averaged.

			AEI				CB2					
Set	d	Iterations	bo_aei	bo_tac_tei	bo_tw20_aei	bo_tw40_aei	bo_cb2	bo_tac_cb2	bo_tw20_cb2	bo_tw40_cb2	rs	constant
Set 1	1	50	6.42 (7)	3.46 (2)	4.98 (4)	6.46 (8)	5.18 (6)	**3.44** (1)	3.92 (3)	5.16 (5)	9.21 (10)	6.78 (9)
		100	6.93 (8)	**3.41** (1)	5.23 (5)	5.77 (7)	5.60 (6)	3.45 (2)	3.80 (4)	3.60 (3)	9.65 (10)	7.56 (9)
	2	50	5.44 (6)	6.16 (8)	**3.41** (1)	5.26 (5)	4.85 (3)	5.52 (7)	3.53 (2)	4.90 (4)	9.22 (10)	6.72 (9)
		100	6.83 (8)	5.73 (7)	3.22 (2)	4.61 (4)	5.58 (6)	4.84 (5)	**3.11** (1)	3.69 (3)	9.67 (10)	7.72 (9)
	5	50	4.02 (2)	7.88 (9)	**3.35** (1)	4.42 (5)	4.38 (4)	6.69 (8)	4.28 (3)	4.73 (6)	8.72 (10)	6.54 (7)
		100	5.08 (6)	7.54 (8)	**3.04** (1)	3.81 (3)	4.78 (5)	6.03 (7)	3.71 (2)	4.00 (4)	9.30 (10)	7.71 (9)
Set 2	2	50	**3.44** (1)	3.91 (2)	5.97 (7)	5.16 (6)	4.23 (3)	5.15 (5)	7.65 (9)	6.10 (8)	8.77 (10)	4.62 (4)
		100	3.59 (2)	**3.28** (1)	6.47 (8)	5.17 (6)	3.87 (3)	4.07 (4)	8.04 (9)	6.23 (7)	9.31 (10)	4.96 (5)
All			5.22 (8)	5.17 (7)	**4.46** (1)	5.08 (6)	4.81 (4)	4.90 (5)	4.76 (2)	4.80 (3)	9.23 (10)	6.57 (9)

might be available. This would heavily influence the choice, as one of the presented drift scenarios might be more probable. However, to obtain an overview, which optimizers performed generally well, the averaged ranks over all drift types are given in Table 7.6. The averaging implies that an equal probability of each scenario is assumed. Also the average ranks do not reflect the absolute differences, which have been of a higher magnitude for scenarios with drift than for the *no drift* scenario. Therefore, the meaningfulness of the given averaged ranks is very limited. For the average across all problems in the last row no significance test was conducted. Note that `constant` is underlined for set 2 because the sign test ignores the value of the difference. Therefore, the number of cases where `constant` placed first in the *no drift* scenario are sufficient for the test to not reject the hypothesis that `constant` is worse than the best performing optimizer. Since there is no clear winner for all drift types or across all different functions no clear recommendation can be given if the drift type and the function structure is unknown beforehand. Even if a drift of any kind is expected it is probably beneficial to be aware of the optimizers behavior in the *no drift* scenario. A longer time frame without a drift can cause behaviors, such as the saturation effect for optimizers that use the `tei` and `aei` acquisition function or the re-exploration effect of the `tw` approaches, that we both

observed in the *no drift* scenario. In general, for lower dimensional problems with a simple response surface `bo_tac_tei` seems to be a promising candidate. For higher dimensional problems `bo_tw20_aei` works best in average.

7.4 Conclusion

This benchmark compared two new Bayesian optimization approaches for dynamic optimization problems (DOPs) against three baselines. To adapt Bayesian optimization for problems, where the positions of the optima change over time, we proposed two approaches: First, the time-as-covariate approach includes the time as an additional covariate into the surrogate, which allows the surrogate to directly model the influence of the time on the function outcomes. Second, the window approach reduces the design of the surrogate to observations that are made within a certain recent time frame, which allows the surrogate to forget outdated concepts of the objective function. Both approaches were benchmarked with the lower confidence bound acquisition function and special adaptations of the expected improvement.

To obtain a comprehensive understanding of the optimizer's behavior we benchmarked our adaptations against the drift-unaware default Bayesian optimization on the three scenarios: no drift, sudden drift and incremental drift. All scenarios are created by a benchmark setup that was designed to convert common synthetic benchmark functions to DOPs with a desired drift. We defined a performance measure that includes every evaluation into the calculation of the average error, which is uncommon for DOPs, where usually just the best of a batch is taken into consideration.

The results show that no optimizer is a clear winner but that each method has its strengths and weaknesses. In general, the proposed methods are able to beat the baseline of the ordinary Bayesian optimization, whereas the window-based Bayesian optimization with a smaller window is the most promising candidate for the $5d$ problems and the time-as-covariate approach performed best on the $1d$, and

simpler $2d$ problems. The number of iterations did not play an important role for the final rankings. For most cases the augmented expected improvement appears to be the preferable acquisition function.

Altogether, the results also show that a real application of MBO on dynamic optimization problems faces many challenges, since there is no clear winner in this benchmark. They can serve as preliminary results that help to develop a more flexible MBO-CD framework. The first thing to mention is that the defined error measure influences how an MBO-CD algorithm should be designed. In our scenario we argue that every black-box evaluation counts towards the performance measurement. A more realistic scenario might be that we have two separated evaluations with individual budgets. One budget is for the production-evaluation, where we want to minimize the error and a second budget is for exploration-evaluations where we allow evaluations with a high error, so that the surrogate can obtain information about uncertain regions or the drift type. This could be a setting in a factory with a production line, where we want to produce items with the highest possible quality. For the exploration-evaluations a second smaller production line would be available, where we produce items with different production settings that can be discarded if the quality is not sufficient. If these exploration-evaluations produce items of higher quality, we can adjust the production parameters. Another practical example could be a machine-learning method that is set to optimal hyper-parameters to obtain the best predictions while in online operation. As soon as there are free computational resources (e.g. at night), these resources can be used to evaluate other hyper-parameters. Afterwards, the MBO-CD framework can adapt the online hyper-parameters to improve the prediction quality of the online operation. An additional challenge is to find an optimal strategy, if these exploration-evaluations are obtained parallel or even parallel to the production-evaluations.

Such a scenario would allow us to distinguish between exploratory and exploitative evaluations. With this distinction we could measure the performance only of the exploitative evaluations, i.e. the configurations where the model expects a good performance. This would allow us to conclude if the surrogate correctly models

the current state of the true objective function. In the current setting, where each evaluation is included in the error, we punish exploratory evaluations. Also in this current setting we can adapt the optimizers to better balance exploration and exploitation.

The analysis of the results showed that in some cases potentially unwanted behaviors of the optimizers can be observed. Such issues are the saturation effect, which was especially notable for optimizers that use the `aei` or `tei` acquisition function. Here, the acquisition function prevents the optimizers from evaluating regions where we expect a good outcome. Another issue is the failing adaption after a sudden change.

This allows two conclusions: Fist, the methods should be improved on a mathematical level, e.g. by adapting the acquisition function, the Gaussian process regression or the surrogate in general. Second, the optimization process should be supervised, either by a human or by another algorithm that detects unwanted behaviors of the optimizer. For online machine learning it is common to detect concept drifts with a method that is independent from the machine learning method itself. Such external concept drift detectors could help to determine the window size for the incremental drift or trigger a deletion of the outdated design if a sudden drift is detected. We also noticed that for certain functions different acquisition functions perform better. This is a common problem in MBO and can potentially be solved using a portfolio of acquisition functions.

We have to highlight that in this work we are restricted to problems where the optimal value stays unchanged and only the position of the optimum changes. This makes it easier to detect unwanted behavior of the optimizer, since an increased error directly implies that the evaluated values move away from the optimum. In other settings it could happen that the optimal value changes. If in such a situation the overall outcomes of the function increase during optimization we can no longer be sure that an increased error is the result of a change of the location of the optimum. The observed effect could also be an upwards shift in the function value space. On the other hand, if the outcomes during the optimization are continuously low, it does not imply a good performing optimization anymore. It

could happen that a new optimum appears in an undetected area with even lower function outcomes. As explained, the presented benchmark is easily expandable to introduce a controllable shift in the function value space. This shift likely poses a completely new challenge for the optimizers.

8 Summary

Bayesian optimization is a very current and active field of research. Many extensions, adaptations and use cases are presented regularly. On the one hand, it is a challenge to form a common and concise understanding of the state-of-the-art, but on the other hand it opens up many possibilities to combine newest research results and ideas to create new algorithms and applications. Within this work two novel extensions of the model-based optimization framework are presented that are each on its own independent developments.

The first extension is *RAMBO* which is a framework for applying MBO in parallel in a resource-aware fashion. We benchmarked state-of-the-art asynchronous and synchronous strategies against our newly proposed scheduled synchronous approach. The scheduling approach uses a second regression model to predict the runtime of each evaluation of the expensive black-box. With this information, we were able to avoid the parallel evaluation of black-box configurations that have heterogeneous runtimes and lead to idle times. Also, the algorithm was able to schedule multiple evaluations with short runtimes on one worker, while other workers evaluate long-running black-box configurations. If the runtimes are heterogeneous and well predictable by the regression model, the newly proposed scheduling approach is able to reach the optimum faster than their competitors. On problems with an unpredictable runtime RAMBO was still able to obtain comparable optimization performance to the state-of-the-art parallel MBO methods but was inferior to asynchronous optimizers. Especially for a high degree of parallelization with 16 workers the asynchronous approach performed well if the single proposal can be obtained comparably fast. A slow asynchronous point proposal due to either a large design or a computationally intensive acquisition function can decrease the

overall optimization performance of an asynchronous optimizer. In contrast to a synchronous method, where the proposals are only generated once in each iteration on the master, the asynchronous method obtains each proposal individually on each worker. The asynchronous methods incorporate the ongoing evaluations in their proposal. The problem how to deal with ongoing proposal generations seems to be neglected in the literature so far. A promising approach could be a hybrid optimizer that uses asynchronous and synchronous methods combined. As soon as multiple ongoing proposals are detected the hybrid optimizer could switch to a synchronous proposal for the workers in question.

The second contribution *MBO-CD* is an extension of MBO towards dynamic optimization problems. We present two approaches how MBO can be taught to handle black-box functions where the relation between input and output changes over time, i.e. where a concept drift occurs. The *window approach* trains the surrogate only on the most recent observations. The *time-as-covariate approach* includes the time as an additional input variable in the surrogate, giving it the ability to learn the effect of the time. For the latter, a special acquisition function, the *temporal expected improvement*, has been proposed. Both approaches were benchmarked against the drift-unaware classic Bayesian optimization. A special benchmark framework was designed to allow the simulation of different drift types on static objective functions. The chosen drift scenarios were *no drift*, *sudden drift* and *incremental drift*. To measure the performance of the optimizers, the *mean fitness error* was chosen, where each evaluation during the optimization counts equally towards the error. Therefore, any exploratory evaluation possibly increases the overall error, although the evaluation might be necessary to detect the drift or find better performing regions of the objective function. The benchmark showed that both proposed concept drift aware optimizers were able to beat the baseline in the majority of cases. However, there is no clear winner for all drift types across all problems. The results suggest that for easily modeled objectives of lower dimensionality the *time-as-covariate approach* obtains good results, whereas for higher dimensional objectives of higher complexity the *window approach* performs better. Furthermore, we observed that the acquisition function can heavily influence the average optimization performance. In general the newly suggested *temporal*

expected improvement for the *time-as-covariate approach* or the *augmented expected improvement* for the *window approaches* obtains better rankings than the *lower confidence bound*, although the EI-based optimizers suffer from a saturation effect when the global optimum is fitted accurately by the surrogate. In those cases, the EI-based acquisition function forbids exploitative reevaluations of configurations close to the optima, leading to exploratory evaluations in bad performing areas which negatively affects the average performance. Therefore, a potential further development could target two aspects. First, separate exploratory and exploitative observations and control the balance of both, so that an error measure that counts every evaluation can be minimized. Second, create a scenario where we allow exploratory evaluations without counting them towards the error. This could include a parallel setup where one part of the workers is assigned to do exploration evaluations and another part of workers is assigned to do the evaluations that count towards the final performance.

All optimization benchmarks in this thesis were conducted on synthetic functions with a purely real-valued domain. These synthetic benchmarks are mathematical functions that are commonly used in the optimization literature. The advantages are that the true optimum location and value is known which is not the case for most machine-learning hyper-parameter optimization problems and most real-world optimization problems in general. Additionally, the results of synthetic benchmarks are deterministic, so it is less likely to overinterpret results that are just artifacts of noise. Both facts increase the overall comparability and interpretability of the results across different problems.

However, the question if our observations will also apply to real-world problems remains open and should be the focus of an independent benchmark. Many real-world optimization problems are not defined on a purely numeric search space and include categorical or even hierarchical parameters. Such complex search spaces form a dedicated current research topic. Accordingly, an extension of RAMBO towards categorical search spaces creates a new challenge since not only the surrogate but also the model that predicts the runtimes has to be adapted. Hierarchical or categorical search spaces usually imply that certain search regions

are uncorrelated. This opens up an opportunity to use this structural knowledge to distribute evaluations of uncorrelated configurations across the workers to maximize the knowledge that can be obtained in parallel. Also for MBO-CD hierarchical or categorical search spaces create a new challenge. While the application of the *window approach* might be straightforward, the *time-as-covariate approach* has to deal with an even more complex surrogate that can extrapolate and also deal with categorical variables. It is questionable if the default random forest, which usually is the go-to solution if MBO is applied on mixed-valued search spaces, will suffice as a surrogate to obtain a good optimization performance.

Methodically, non-deterministic functions as they are found in real-life, might not be a particular problem for the presented methods. Obviously, the optimization problem will be harder and require more evaluations and the final point proposal has to be adapted as well as the choice of the acquisition function. For RAMBO we could argue that stochasticity is even beneficial, because evaluating configurations that are too similar is not a waste of resources anymore, it is even necessary to improve the accuracy of the surrogate. For MBO-CD we already assume that the function is not deterministic because it changes over time. However, the problem will become more complex as now the insecurity can be attributed to the noise and to the time.

The results in this work further demonstrate that the MBO framework is highly customizable and there hardly is a configuration that works best on multiple scenarios. Either a human expert has to supervise the MBO configuration or the self-configuration of MBO with reasonable settings for a given scenario has to be improved. This can be achieved either by simple heuristics or by applying techniques of meta-learning and AutoML to obtain a better surrogate and automatically configure MBO reasonably.

Bibliography

Assael, John-Alexander M., Ziyu Wang, Bobak Shahriari, and Nando de Freitas (Mar. 4, 2015). *Heteroscedastic Treed Bayesian Optimisation*, pp. 1–9. arXiv: 1410.7172. URL: http://arxiv.org/abs/1410.7172.

Benassi, Romain, Julien Bect, and Emmanuel Vazquez (2011). "Robust Gaussian Process-Based Global Optimization Using a Fully Bayesian Expected Improvement Criterion". In: *Learning and Intelligent Optimization*. Ed. by Carlos A. Coello Coello. Vol. 6683. Berlin, Heidelberg: Springer, pp. 176–190. DOI: 10.1007/978-3-642-25566-3_13.

Bergstra, J., D. Yamins, and D. D. Cox (2013). "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28* (Atlanta, GA, USA). ICML'13. JMLR.org, pp. I-115–I-123. URL: http://dl.acm.org/citation.cfm?id=3042817.3042832.

Bischl, Bernd, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones (2016). "Mlr: Machine Learning in R". In: *Journal of Machine Learning Research* 17.170, pp. 1–5. URL: http://jmlr.org/papers/v17/15-066.html.

Bischl, Bernd, Jakob Richter, Jakob Bossek, Daniel Horn, Janek Thomas, and Michel Lang (Mar. 9, 2017). *mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions*, pp. 1–26. arXiv: 1703.03373. URL: http://arxiv.org/abs/1703.03373.

Bischl, Bernd, Simon Wessing, Nadja Bauer, Klaus Friedrichs, and Claus Weihs (2014). "MOI-MBO: Multiobjective Infill for Parallel Model-Based Optimization". In: *Learning and Intelligent Optimization Conference*. LION. Lecture Notes in Computer Science. Florida: Springer, pp. 173–186. DOI: 10.1007/978-3-319-09584-4_17.

Borchers, Hans Werner (2018). *Adagio: Discrete and Global Optimization Routines*. Version 0.7.1.

Bossek, Jakob (2017). "Smoof: Single- and Multi-Objective Optimization Test Functions". In: *The R Journal* 9.1, pp. 103–113. URL: https://journal.r-project.org/archive/2017/RJ-2017-004/index.html.

Branke, and J. (June 2005). "Evolutionary Optimization in Uncertain Environments-a Survey". In: *IEEE Transactions on Evolutionary Computation* 9.3, pp. 303–317. DOI: 10.1109/TEVC.2005.846356.

Chevalier, Clément and David Ginsbourger (Jan. 7, 2013). "Fast Computation of the Multi-Points Expected Improvement with Applications in Batch Selection". In: *Learning and Intelligent Optimization*. International Conference on Learning and Intelligent Optimization. Springer Berlin Heidelberg, pp. 59–69. DOI: 10.1007/978-3-642-44973-4_7.

Cruz, Carlos, Juan R. González, and David A. Pelta (July 1, 2011). "Optimization in Dynamic Environments: A Survey on Problems, Methods and Measures". In: *Soft Computing* 15.7, pp. 1427–1448. DOI: 10.1007/s00500-010-0681-0.

Demšar, Janez (2006). "Statistical Comparisons of Classifiers over Multiple Data Sets". In: *Journal of Machine Learning Research* 7 (Jan), pp. 1–30. ISSN: ISSN 1533-7928. URL: http://www.jmlr.org/papers/v7/demsar06a.html.

Forrester, Alexander I. J. and Andy J. Keane (Jan. 1, 2009). "Recent Advances in Surrogate-Based Optimization". In: *Progress in Aerospace Sciences* 45.1, pp. 50–79. DOI: 10.1016/j.paerosci.2008.11.001.

Forrester, Alexander I. J., András Sóbester, and Andy J. Keane (Dec. 8, 2007). "Multi-Fidelity Optimization via Surrogate Modelling". In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 463.2088, pp. 3251–3269. DOI: 10.1098/rspa.2007.1900.

Gama, João, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia (Mar. 2014). "A Survey on Concept Drift Adaptation". In: *ACM Comput. Surv.* 46.4, 44:1–44:37. DOI: 10.1145/2523813.

Ginsbourger, David, Janis Janusevskis, and Rodolphe Le Riche (2011). *Dealing with Asynchronicity in Parallel Gaussian Process Based Global Optimization*, pp. 1–27. URL: https://hal.archives-ouvertes.fr/hal-00507632.

Ginsbourger, David, Rodolphe Le Riche, and Laurent Carraro (Mar. 2008). *A Multi-Points Criterion for Deterministic Parallel Global Optimization Based on Gaussian Processes*, pp. 1–30. URL: https://hal.archives-ouvertes.fr/hal-00260579.

– (2010). "Kriging Is Well-Suited to Parallelize Optimization". In: *Computational Intelligence in Expensive Optimization Problems*. Springer, pp. 131–162. ISBN: 978-3-642-10701-6.

Golovin, Daniel, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley (2017). "Google Vizier: A Service for Black-Box Optimization". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '17*. The 23rd ACM SIGKDD International Conference. Halifax, NS, Canada: ACM Press, pp. 1487–1495. DOI: 10.1145/3097983.3098043.

Gramacy, Robert B and Herbert K. H Lee (Sept. 1, 2008). "Bayesian Treed Gaussian Process Models With an Application to Computer Modeling". In: *Journal of the American Statistical Association* 103.483, pp. 1119–1130. DOI: 10.1198/016214508000000689.

Hansen, Nikolaus and Andreas Ostermeier (June 2001). "Completely Derandomized Self-Adaptation in Evolution Strategies". In: *Evolutionary Computation* 9.2, pp. 159–195. DOI: 10.1162/106365601750190398.

Horn, Daniel, Jörg Stork, Nils-Jannik Schüßler, and Martin Zaefferer (2019). "Surrogates for Hierarchical Search Spaces: The Wedge-Kernel and an Automated Analysis". In: *Proceedings of the Genetic and Evolutionary Computation Conference* (Prague, Czech Republic). GECCO '19. New York, NY, USA: ACM, pp. 916–924. DOI: 10.1145/3321707.3321765.

Huang, D., T. T. Allen, W. I. Notz, and N. Zeng (Mar. 1, 2006). "Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models". In: *Journal of Global Optimization* 34.3, pp. 441–466. DOI: 10.1007/s10898-005-2454-3.

Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown (Jan. 17, 2011). "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization*. Ed. by Carlos A. Coello Coello. Lecture Notes in Computer Science 6683. Springer Berlin Heidelberg, pp. 507–523. DOI: 10.1007/978-3-642-25566-3_40.

– (2012). "Parallel Algorithm Configuration". In: *Learning and Intelligent Optimization*. Ed. by Youssef Hamadi and Marc Schoenauer. Lecture Notes in Computer Science 7219. Springer Berlin Heidelberg, pp. 55–70. DOI: 10.1007/978-3-642-34413-8_5.

Jones, Donald R. (2001). "A Taxonomy of Global Optimization Methods Based on Response Surfaces". In: *Journal of Global Optimization* 21.4, pp. 345–383. DOI: 10.1023/A:1012771025575.

Jones, Donald R., Matthias Schonlau, and William J. Welch (Dec. 1, 1998). "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13.4, pp. 455–492. DOI: 10.1023/A:1008306431147.

Kandasamy, Kirthevasan, Akshay Krishnamurthy, Jeff Schneider, and Barnabas Poczos (Mar. 31, 2018). "Parallelised Bayesian Optimisation via Thompson Sampling". In: *International Conference on Artificial Intelligence and Statistics*. International Conference on Artificial Intelligence and Statistics, pp. 133–142. URL: http://proceedings.mlr.press/v84/kandasamy18a.html.

Kotthaus, Helena (2018). "Methods for Efficient Resource Utilization in Statistical Machine Learning Algorithms". TU Dortmund. 153 pp. URL: http://dx.doi.org/10.17877/DE290R-18928 (visited on 01/07/2020).

Krige, D. G. (Dec. 1, 1951). "A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand". In: *Journal of the Southern African Institute of Mining and Metallurgy* 52.6, pp. 119–139. ISSN: 0038-223X. URL: https://journals.co.za/content/saimm/52/6/AJA0038223X_4792.

Lang, Michel, Bernd Bischl, and Dirk Surmann (Feb. 22, 2017). "Batchtools: Tools for R to Work on Batch Systems". In: *Journal of Open Source Software* 2.10, p. 135. DOI: 10.21105/joss.00135.

McKay, M. D., R. J. Beckman, and W. J. Conover (May 1, 1979). "Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code". In: *Technometrics* 21.2, pp. 239–245. DOI: 10.1080/00401706.1979.10489755.

Močkus, J. (1975). "On Bayesian Methods for Seeking the Extremum". In: *Optimization Techniques IFIP Technical Conference*. Springer, pp. 400–404. URL: http://link.springer.com/content/pdf/10.1007/978-3-662-38527-2_55.pdf.

Morar, Marius Tudor, Joshua Knowles, and Sandra Sampaio (May 1, 2017). "Initialization of Bayesian Optimization Viewed as Part of a Larger Algorithm Portfolio". In: *Data Science meets Optimization Workshop: CEC2017 & CPAIOR 2017: DSO 2017*, pp. 1–6. URL: https://www.research.manchester.ac.uk/portal/en/publications/initialization-of-bayesian-optimization-viewed-as-part-of-a-larger-algorithm-portfolio(128f1b9a-5717-4458-9423-89386cb1d103).html.

Munteanu, Alexander, Amin Nayebi, and Matthias Poloczek (2019). "A Framework for Bayesian Optimization in Embedded Subspaces". In: *International Conference on Machine Learning*, pp. 4752–4761.

Nyikosa, Favour M., Michael A. Osborne, and Stephen J. Roberts (Mar. 9, 2018). *Bayesian Optimization for Dynamic Problems*, pp. 1–10. arXiv: 1803.03432. URL: http://arxiv.org/abs/1803.03432.

Pintér, János D. (Mar. 14, 2013). *Global Optimization in Action: Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications*. Springer Science & Business Media. 481 pp. ISBN: 978-1-4757-2502-5. Google Books: uv7lBwAAQBAJ.

Rasmussen, Carl Edward and Christopher K. I. Williams (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press. 248 pp. ISBN: 978-0-262-18253-9.

Richter, Jakob (2015). "Modellbasierte Hyperparameteroptimierung für maschinelle Lernverfahren auf großen Daten". TU Dortmund. 81 pp.

Richter, Jakob, Helena Kotthaus, Bernd Bischl, Peter Marwedel, Jörg Rahnenführer, and Michel Lang (May 29, 2016). "Faster Model-Based Optimization Through Resource-Aware Scheduling Strategies". In: *Learning and Intelligent Optimization*. LION. Springer International Publishing, pp. 267–273. DOI: 10.1007/978-3-319-50349-3_22.

Richter, Jakob, Helena Kotthaus, Andreas Lang, Janek Thomas, Bernd Bischl, Peter Marwedel, Jörg Rahnenführer, and Michel Lang (June 19, 2017). "RAMBO: Resource-Aware Model-Based Optimization with Scheduling for Heterogeneous Runtimes and a Comparison with Asynchronous Model-Based Optimization". In: *Learning and Intelligent Optimization*. LION. Lecture Notes in Computer Science. Springer, Cham, pp. 180–195. DOI: 10.1007/978-3-319-69404-7_13.

Richter, Jakob, Katrin Madjar, and Jörg Rahnenführer (July 15, 2019). "Model-Based Optimization of Subgroup Weights for Survival Analysis". In: *Bioinformatics* 35.14, pp. i484–i491. DOI: 10.1093/bioinformatics/btz361.

Ru, Binxin, Ahsan S. Alvi, Vu Nguyen, Michael A. Osborne, and Stephen J. Roberts (June 20, 2019). *Bayesian Optimisation over Multiple Continuous and Categorical Inputs*, pp. 1–15. arXiv: 1906.08878. URL: http://arxiv.org/abs/1906.08878.

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas (Jan. 2016). "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1, pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.

Snoek, Jasper (2013). "Bayesian Optimization and Semiparametric Models with Applications to Assistive Technology". Toronto: University of Toronto. 129 pp.

Snoek, Jasper, Hugo Larochelle, and Ryan P Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

Srinivas, N., A. Krause, S. M. Kakade, and M. W. Seeger (May 2012). "Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit

Setting". In: *IEEE Transactions on Information Theory* 58.5, pp. 3250–3265. DOI: 10.1109/TIT.2011.2182033.

Swiler, Laura P., Patricia D. Hough, Peter Qian, Xu Xu, Curtis Storlie, and Herbert Lee (2014). "Surrogate Models for Mixed Discrete-Continuous Variables". In: *Constraint Programming and Decision Making*. Springer, pp. 181–202. URL: http://link.springer.com/chapter/10.1007/978-3-319-04280-0_21.

Thornton, Chris, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown (2013). "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 847–855. DOI: 10.1145/2487575.2487629.

Wang (2016). "Bayesian Optimization in a Billion Dimensions via Random Embeddings". In: *The Journal of artificial intelligence research* 55, pp. 361–387. DOI: 10.1613/jair.4806.

Wang, Ziyu, Masrour Zoghi, Frank Hutter, David Matheson, and Nando De Freitas (2013). "Bayesian Optimization in High Dimensions via Random Embeddings." In: *IJCAI*. International Joint Conference on Artificial Intelligence, pp. 1778–1784. ISBN: 978-1-57735-633-2.

Appendices

A Parallel MBO Benchmark

A.1 High Runtime Estimation Quality: $\texttt{rosenbrock}_d$

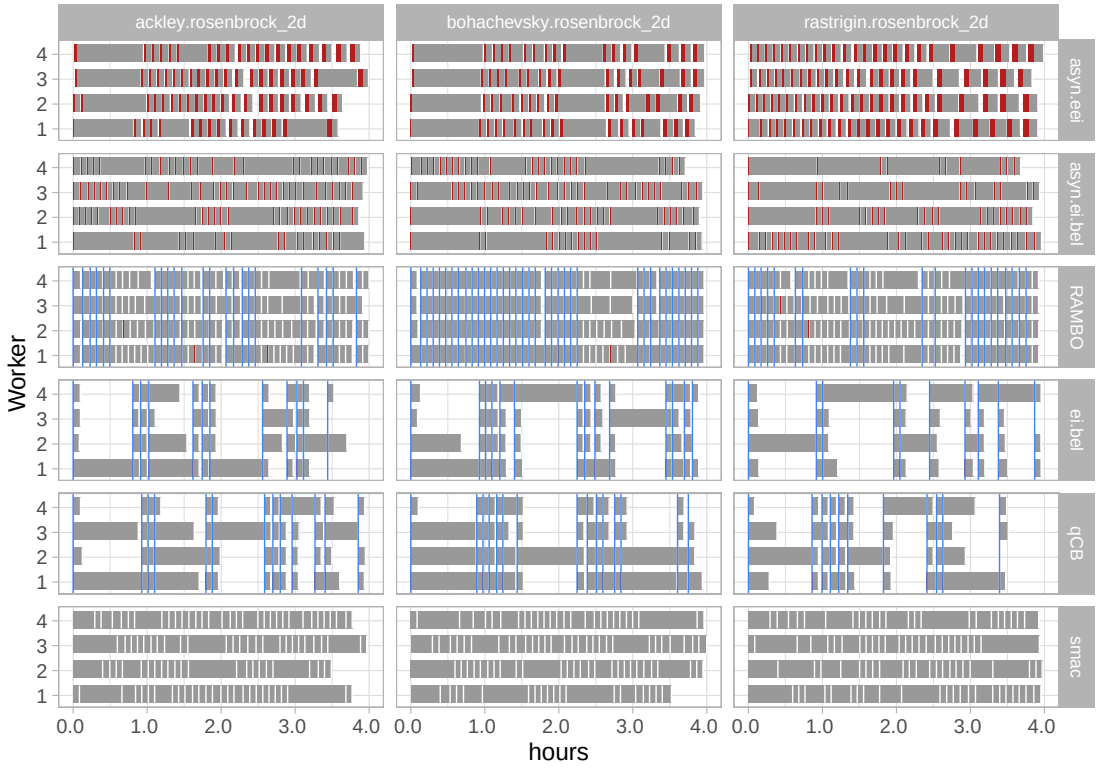

Figure A.1: Utilization of the $k = 4$ workers by the different optimization methods on the functions with the $\texttt{rosenbrock}_2$ time function, which can be modeled reliably by the resource estimator. Note that this just visualizes one of the ten stochastic repetitions.

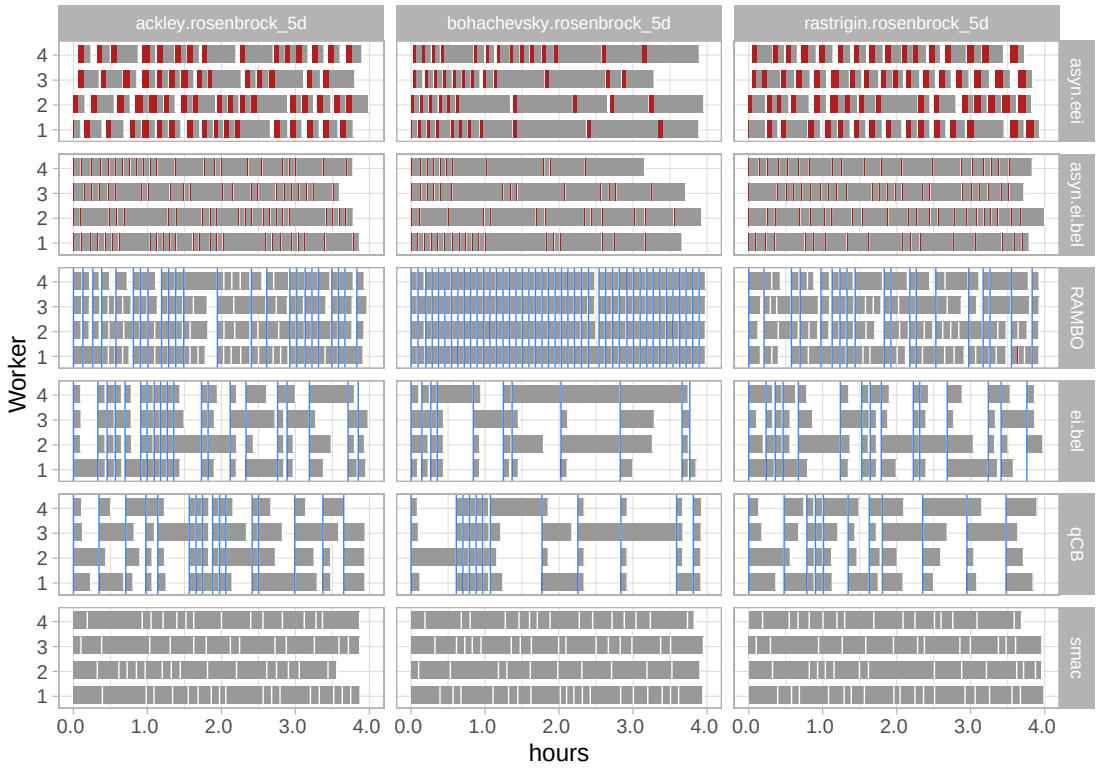

Figure A.2: Utilization of the $k = 4$ workers by the different optimization methods on the functions with the $\texttt{rosenbrock}_5$ time function, which can be modeled reliably by the resource estimator. Note that this just visualizes one of the ten stochastic repetitions.

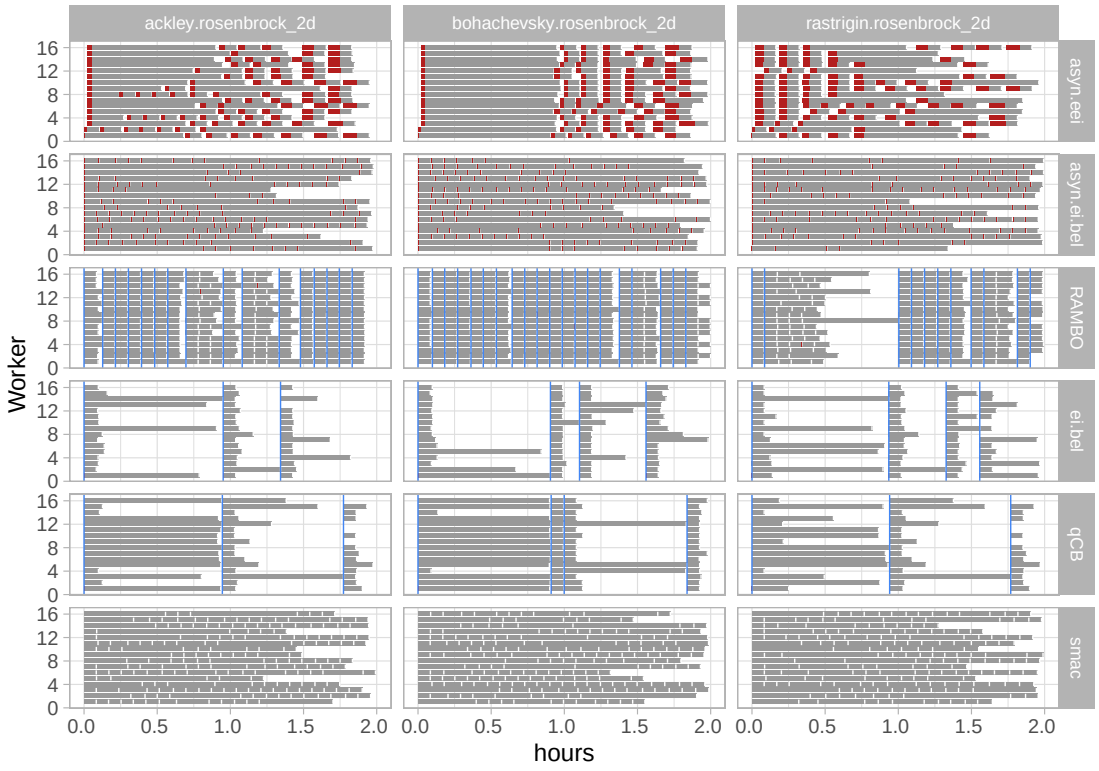

Figure A.3: Utilization of the $k = 16$ workers by the different optimization methods on the functions with the $\texttt{rosenbrock}_2$ time function, which can be modeled reliably by the resource estimator. Note that this just visualizes one of the ten stochastic repetitions.

A.2 Low Runtime Estimation Quality: $\texttt{rastrigin}_d$

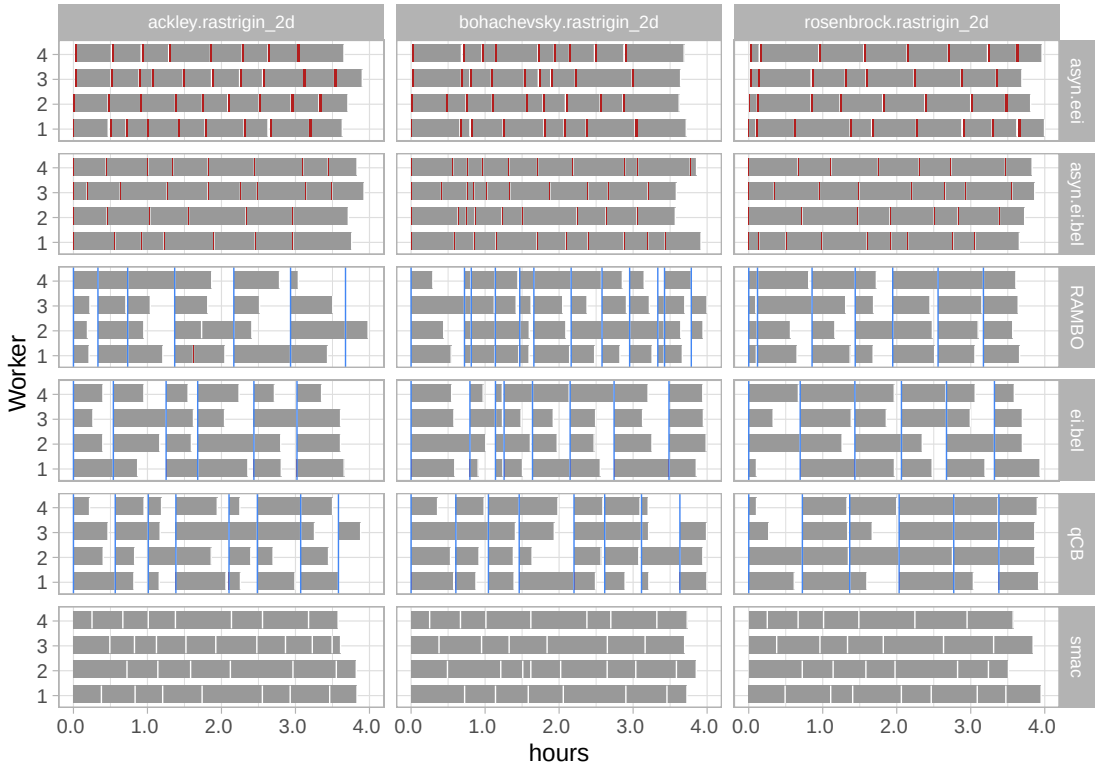

Figure A.4: Utilization of the $k = 4$ workers by the different optimization methods on the functions with the $\texttt{rastrigin}_2$ time function, which can hardly be modeled by the resource estimator. Note that this just visualizes one of the ten stochastic repetitions.

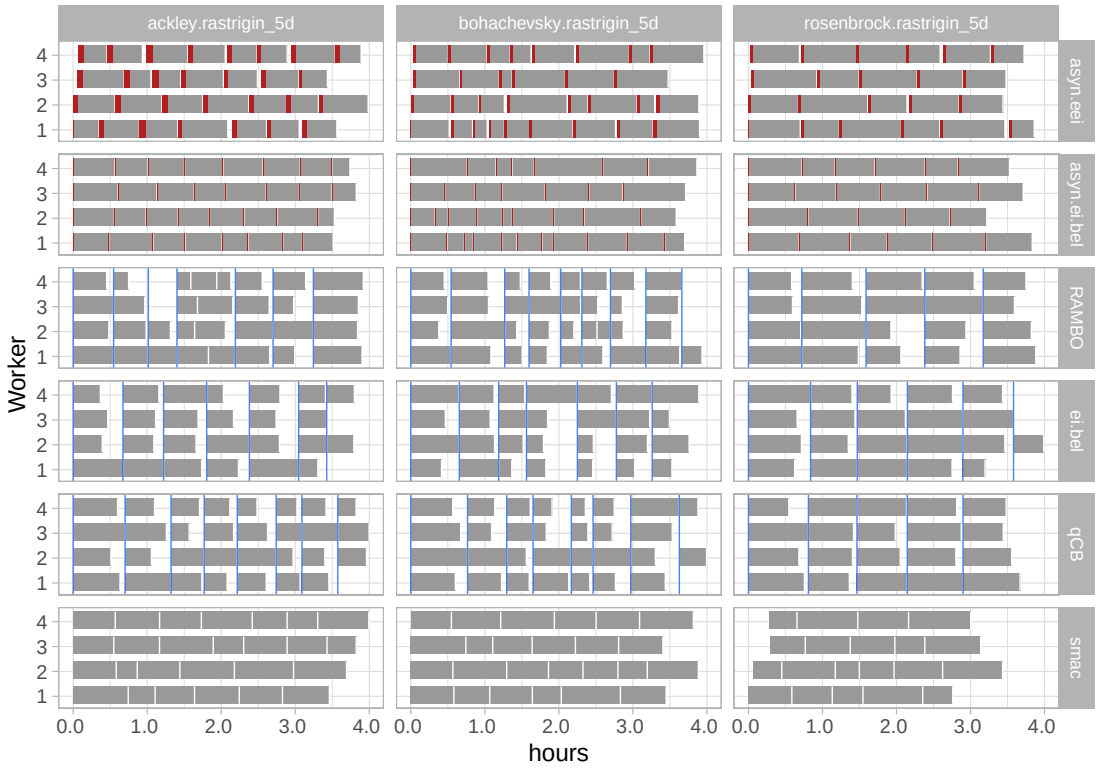

Figure A.5: Utilization of the $k = 4$ workers by the different optimization methods on the functions with the $\texttt{rastrigin}_5$ time function, which can hardly be modeled by the resource estimator. Note that this just visualizes one of the ten stochastic repetitions.

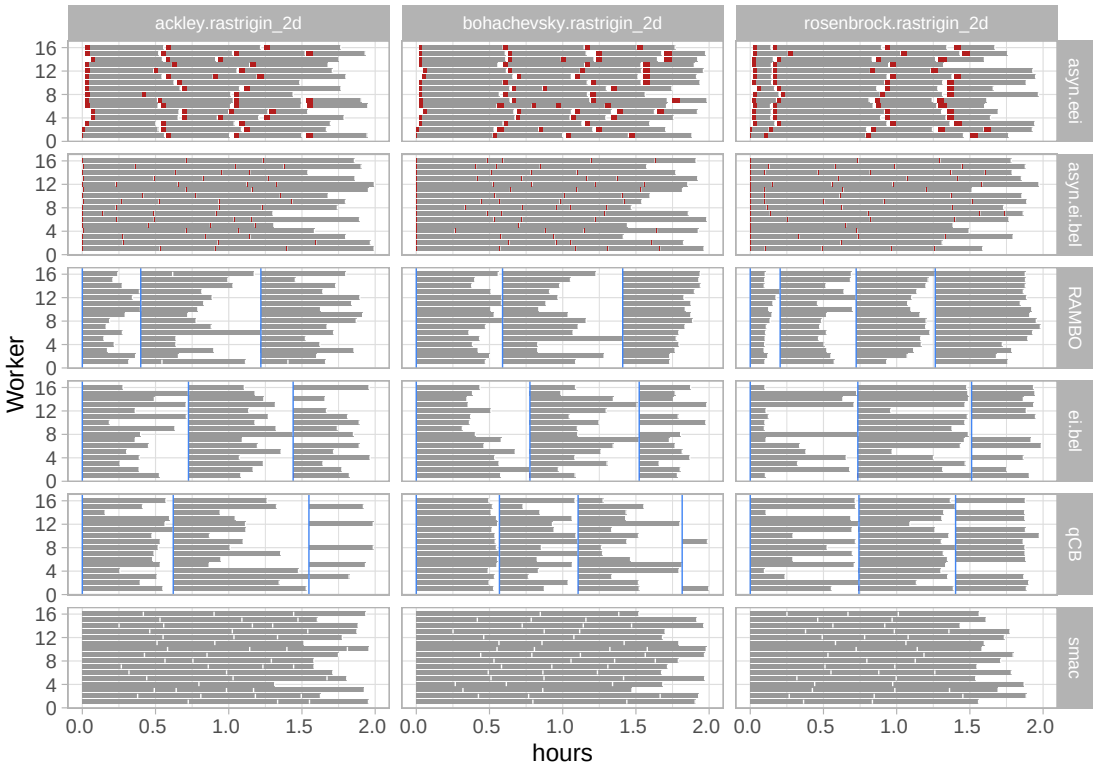

Figure A.6: Utilization of the $k = 16$ workers by the different optimization methods on the functions with the $\texttt{rastrigin}_2$ time function, which can hardly be modeled by the resource estimator. Note that this just visualizes one of the ten stochastic repetitions.

B MBO CD Benchmark

B.1 No Drift

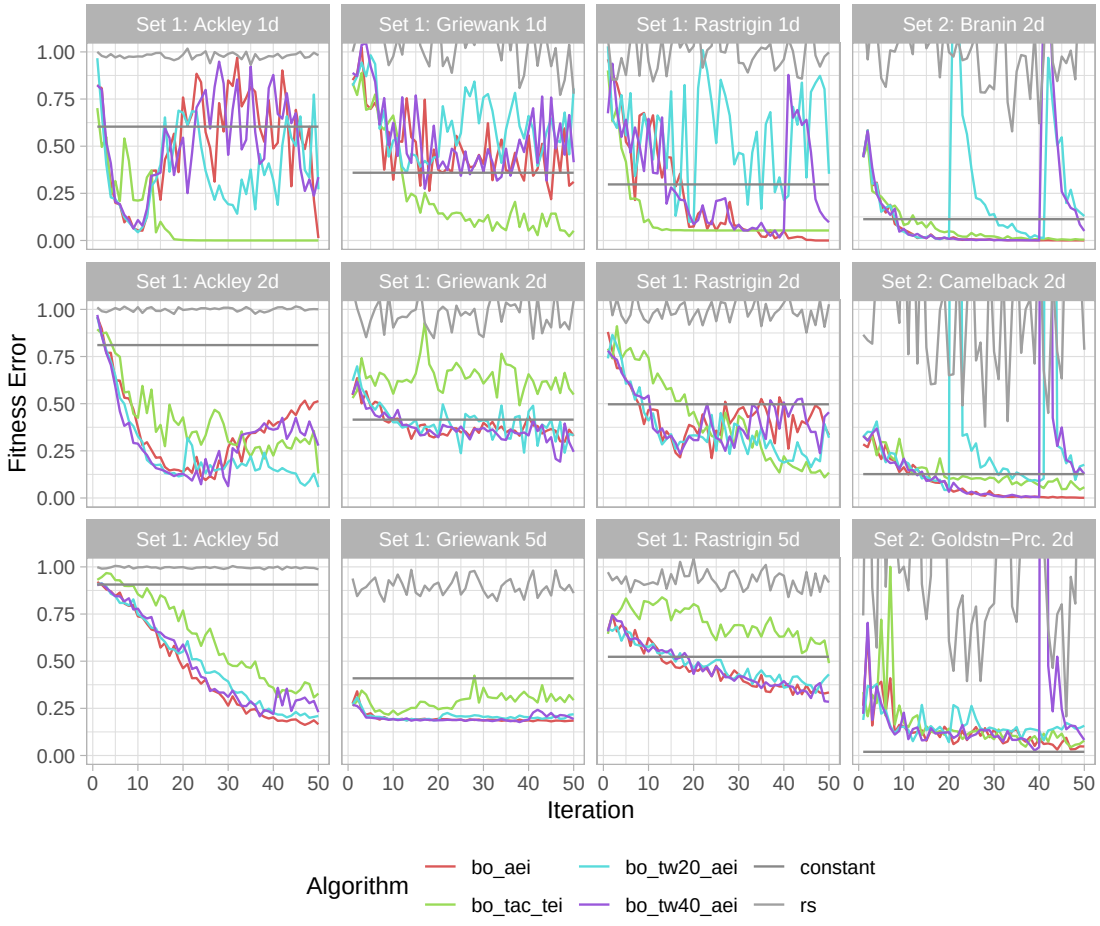

Figure B.1: Optimization curves for optimization strategies that use the `aei` acquisition function on problems with 50 optimization iterations and no drift.

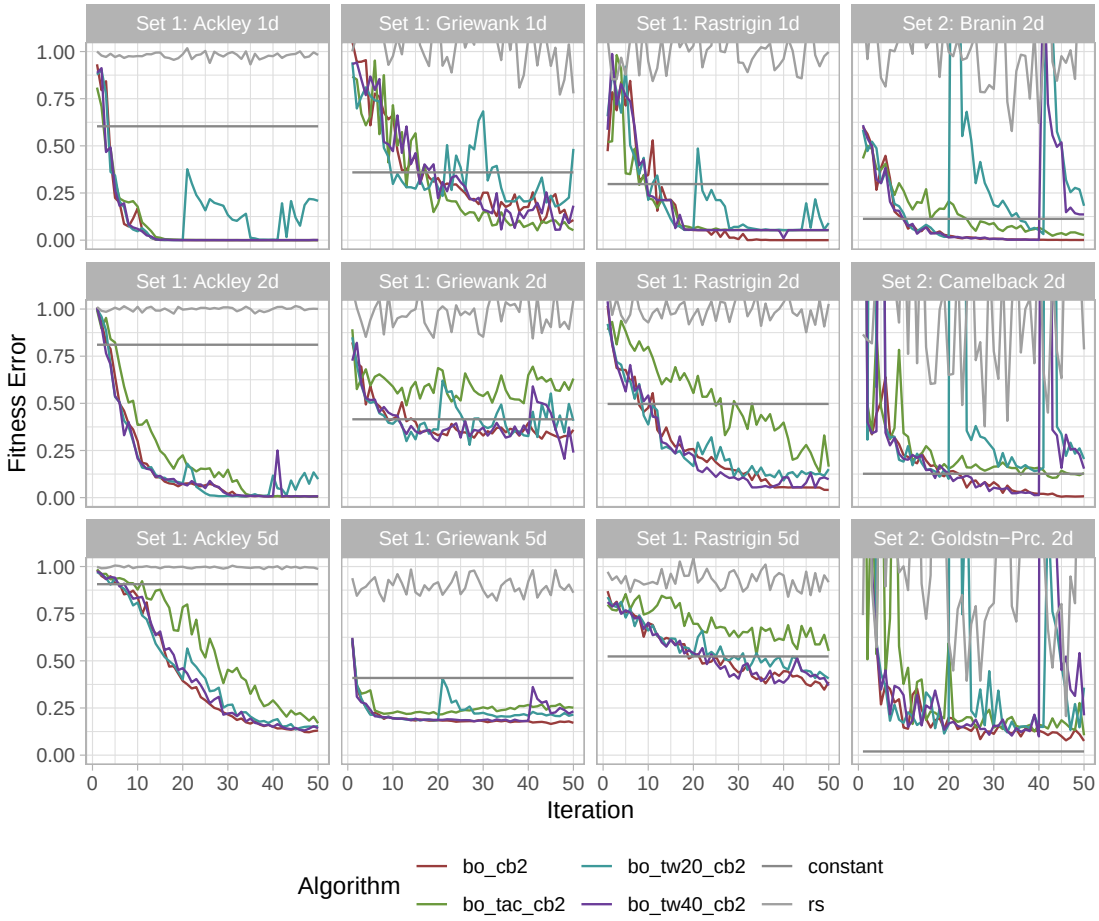

Figure B.2: Optimization curves for optimization strategies that use the `cb2` acquisition function on problems with 50 optimization iterations and no drift.

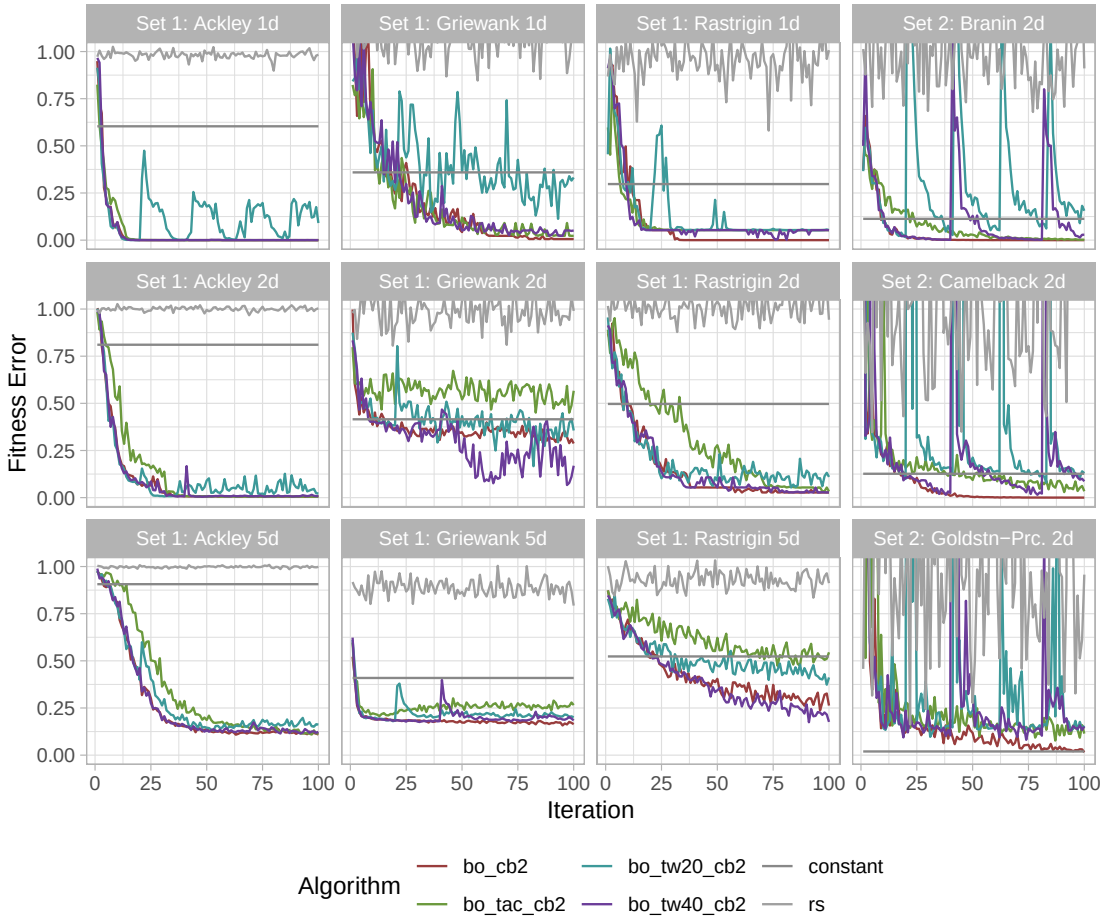

Figure B.3: Optimization curves for optimization strategies that use the `cb2` acquisition function on problems with 100 optimization iterations and no drift.

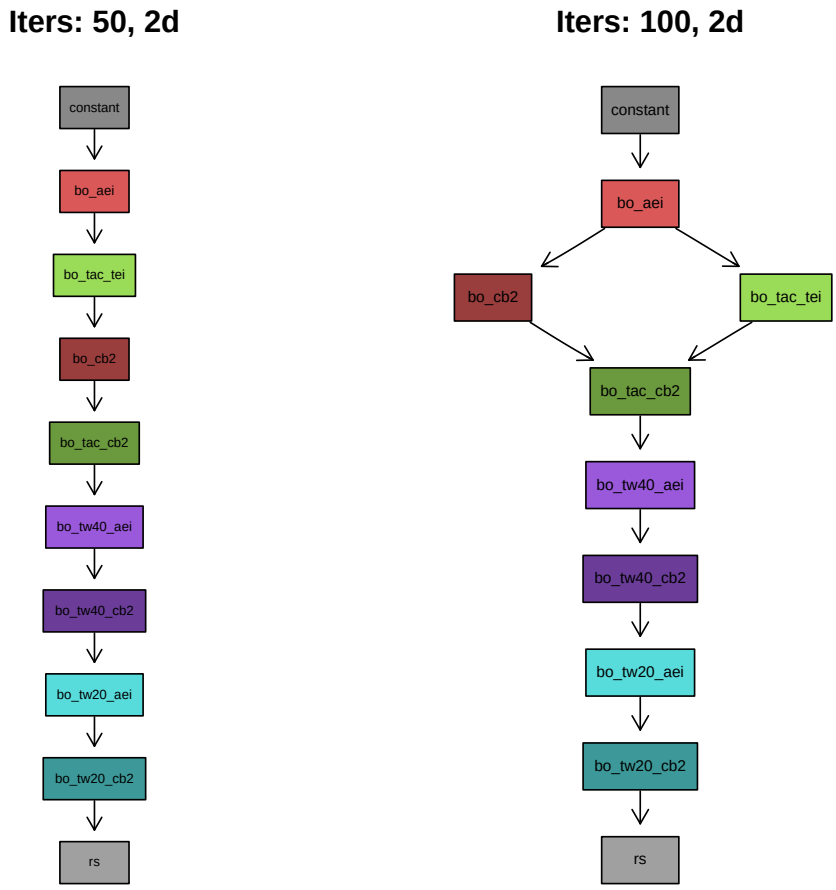

Figure B.4: Preference graphs for each problem subset of set 2 with no drift.

Table B.1: MFE and its standard deviation of each optimization method on each problem with no drift averaged over 50 stochastic repetitions.

				AEI				CB2					
Set	Function	d	Iterations	bo_aei	bo_tac_tei	bo_tw20_aei	bo_tw40_aei	bo_cb2	bo_tac_cb2	bo_tw20_cb2	bo_tw40_cb2	rs	constant
Set 1	Ackley	1	50	0.49 ±0.04	0.19 ±0.16	0.45 ±0.06	0.50 ±0.04	0.16 ±0.18	**0.12** ±0.09	0.24 ±0.06	0.12 ±0.03	0.89 ±0.03	0.59 ±0.28
			100	0.48 ±0.04	0.13 ±0.18	0.44 ±0.03	0.50 ±0.03	0.07 ±0.12	0.07 ±0.12	0.22 ±0.04	**0.07** ±0.03	0.90 ±0.02	0.59 ±0.28
		2	50	0.40 ±0.21	0.45 ±0.14	0.32 ±0.17	0.36 ±0.18	**0.22** ±0.14	0.30 ±0.12	0.24 ±0.12	0.26 ±0.22	0.96 ±0.02	0.75 ±0.16
			100	0.45 ±0.15	0.40 ±0.06	0.27 ±0.17	0.35 ±0.19	0.18 ±0.15	**0.17** ±0.07	0.19 ±0.09	0.17 ±0.11	0.96 ±0.01	0.75 ±0.16
		5	50	0.51 ±0.18	0.65 ±0.14	0.54 ±0.17	0.53 ±0.17	**0.45** ±0.13	0.61 ±0.12	0.46 ±0.10	0.48 ±0.16	0.99 ±0.01	0.88 ±0.07
			100	0.40 ±0.16	0.53 ±0.10	0.40 ±0.17	0.38 ±0.14	**0.30** ±0.13	0.39 ±0.08	0.35 ±0.07	0.34 ±0.16	0.99 ±0.00	0.88 ±0.07
	Griewank	1	50	0.62 ±0.12	0.46 ±0.20	0.67 ±0.09	0.63 ±0.10	0.50 ±0.16	0.49 ±0.21	0.57 ±0.20	0.50 ±0.19	1.05 ±0.09	**0.44** ±0.36
			100	0.57 ±0.07	0.33 ±0.18	0.65 ±0.05	0.62 ±0.07	**0.33** ±0.16	0.38 ±0.19	0.57 ±0.19	0.39 ±0.20	1.07 ±0.05	0.44 ±0.36
		2	50	0.41 ±0.07	0.76 ±0.12	0.44 ±0.11	**0.41** ±0.07	0.43 ±0.07	0.67 ±0.09	0.47 ±0.11	0.45 ±0.08	1.01 ±0.07	0.42 ±0.15
			100	0.39 ±0.05	0.71 ±0.09	0.43 ±0.07	0.38 ±0.06	0.39 ±0.05	0.65 ±0.07	0.45 ±0.08	**0.36** ±0.07	1.02 ±0.05	0.42 ±0.15
		5	50	**0.20** ±0.02	0.39 ±0.10	0.23 ±0.02	0.21 ±0.02	0.21 ±0.02	0.29 ±0.04	0.26 ±0.03	0.23 ±0.03	0.91 ±0.04	0.41 ±0.10
			100	**0.19** ±0.02	0.42 ±0.11	0.23 ±0.02	0.21 ±0.02	0.19 ±0.02	0.29 ±0.02	0.25 ±0.02	0.22 ±0.02	0.91 ±0.03	0.41 ±0.10
	Rastrigin	1	50	0.40 ±0.08	**0.20** ±0.11	0.66 ±0.06	0.51 ±0.09	0.26 ±0.10	0.26 ±0.11	0.45 ±0.19	0.30 ±0.12	0.99 ±0.07	0.36 ±0.28
			100	0.21 ±0.05	0.15 ±0.08	0.64 ±0.04	0.47 ±0.09	**0.15** ±0.05	0.15 ±0.07	0.37 ±0.16	0.17 ±0.09	0.97 ±0.06	0.36 ±0.28
		2	50	0.48 ±0.09	0.51 ±0.17	0.41 ±0.07	0.47 ±0.08	**0.32** ±0.10	0.57 ±0.15	0.36 ±0.12	0.33 ±0.12	1.00 ±0.05	0.47 ±0.18
			100	0.54 ±0.06	0.37 ±0.14	0.37 ±0.07	0.42 ±0.06	**0.23** ±0.11	0.38 ±0.13	0.29 ±0.11	0.24 ±0.09	1.01 ±0.05	0.47 ±0.18
		5	50	**0.47** ±0.06	0.73 ±0.14	0.50 ±0.07	0.47 ±0.07	0.54 ±0.09	0.71 ±0.08	0.58 ±0.09	0.54 ±0.07	0.93 ±0.03	0.52 ±0.10
			100	0.41 ±0.07	0.65 ±0.09	0.45 ±0.07	**0.41** ±0.08	0.43 ±0.07	0.63 ±0.06	0.53 ±0.07	0.41 ±0.08	0.93 ±0.02	0.52 ±0.10
Set 2	Branin		50	**0.14** ±0.07	0.18 ±0.11	0.52 ±0.11	0.31 ±0.07	0.20 ±0.09	0.30 ±0.14	0.70 ±0.12	0.35 ±0.09	1.54 ±0.25	0.20 ±0.19
			100	**0.07** ±0.03	0.11 ±0.07	0.50 ±0.07	0.27 ±0.04	0.09 ±0.04	0.15 ±0.06	0.64 ±0.09	0.30 ±0.06	1.54 ±0.15	0.20 ±0.19
	Camelback	2	50	0.47 ±0.23	0.74 ±0.71	1.73 ±0.20	1.04 ±0.23	0.75 ±0.30	0.97 ±0.51	2.06 ±0.21	1.28 ±0.22	2.16 ±0.39	**0.13** ±0.08
			100	0.26 ±0.12	0.40 ±0.34	1.58 ±0.13	0.94 ±0.11	0.35 ±0.16	0.54 ±0.35	1.80 ±0.17	1.09 ±0.16	2.23 ±0.28	**0.13** ±0.08
	Gldstn-Pr		50	1.77 ±0.95	2.02 ±2.11	3.60 ±1.41	2.60 ±1.15	2.36 ±1.22	2.92 ±2.12	6.17 ±1.91	4.63 ±1.83	8.39 ±2.77	**0.05** ±0.10
			100	0.94 ±0.64	0.96 ±0.82	3.22 ±0.98	2.47 ±0.84	1.32 ±0.63	2.05 ±1.33	5.97 ±1.81	3.71 ±1.08	7.92 ±1.66	**0.05** ±0.10

B.2 Sudden Drift

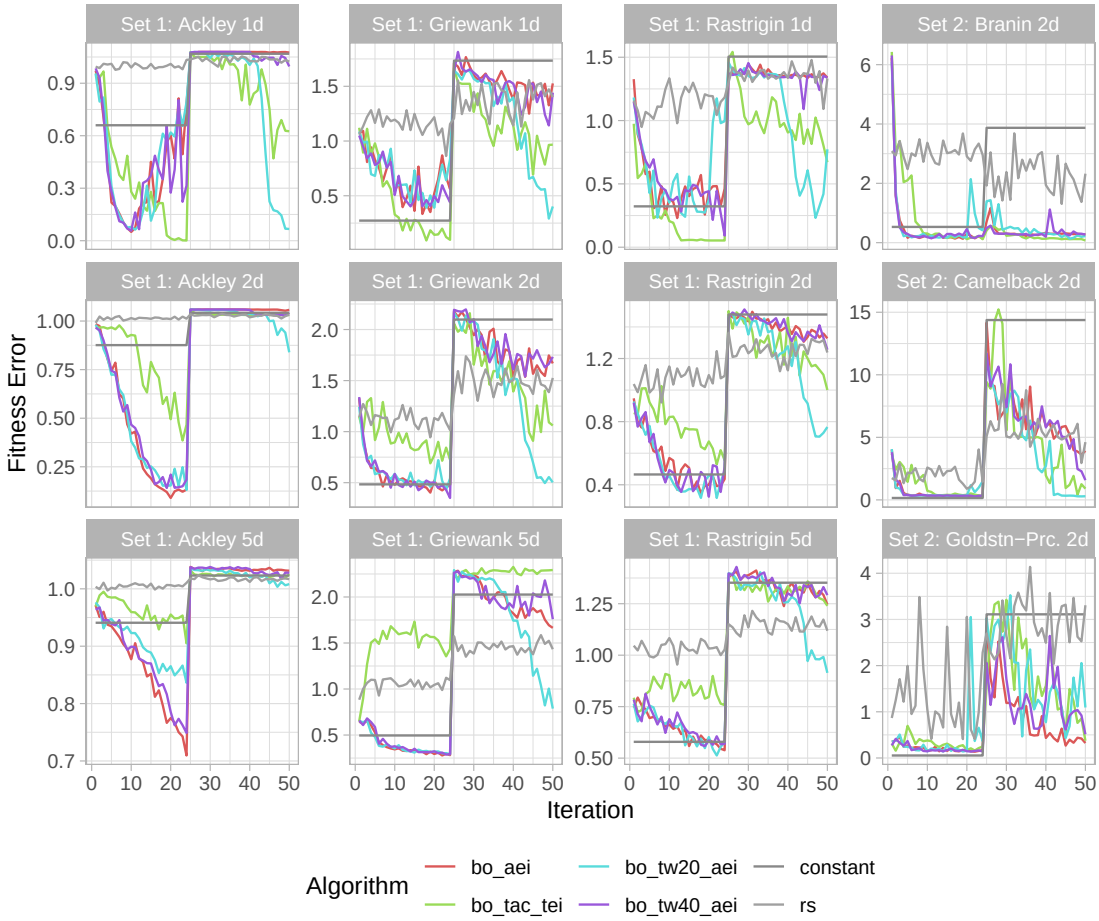

Figure B.5: Optimization curves for optimization strategies that use the `aei` acquisition function on problems with 50 optimization iterations and sudden drift.

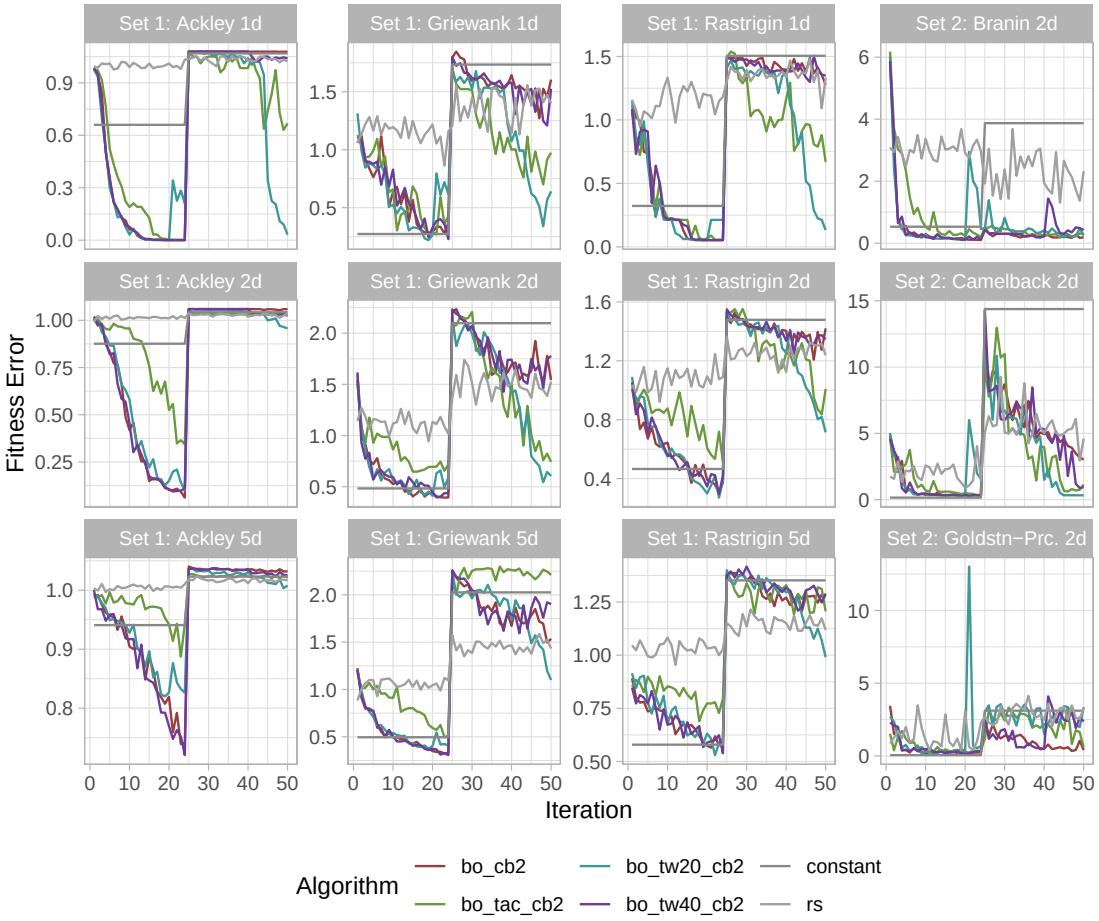

Figure B.6: Optimization curves for optimization strategies that use the `cb2` acquisition function on problems with 50 optimization iterations and sudden drift.

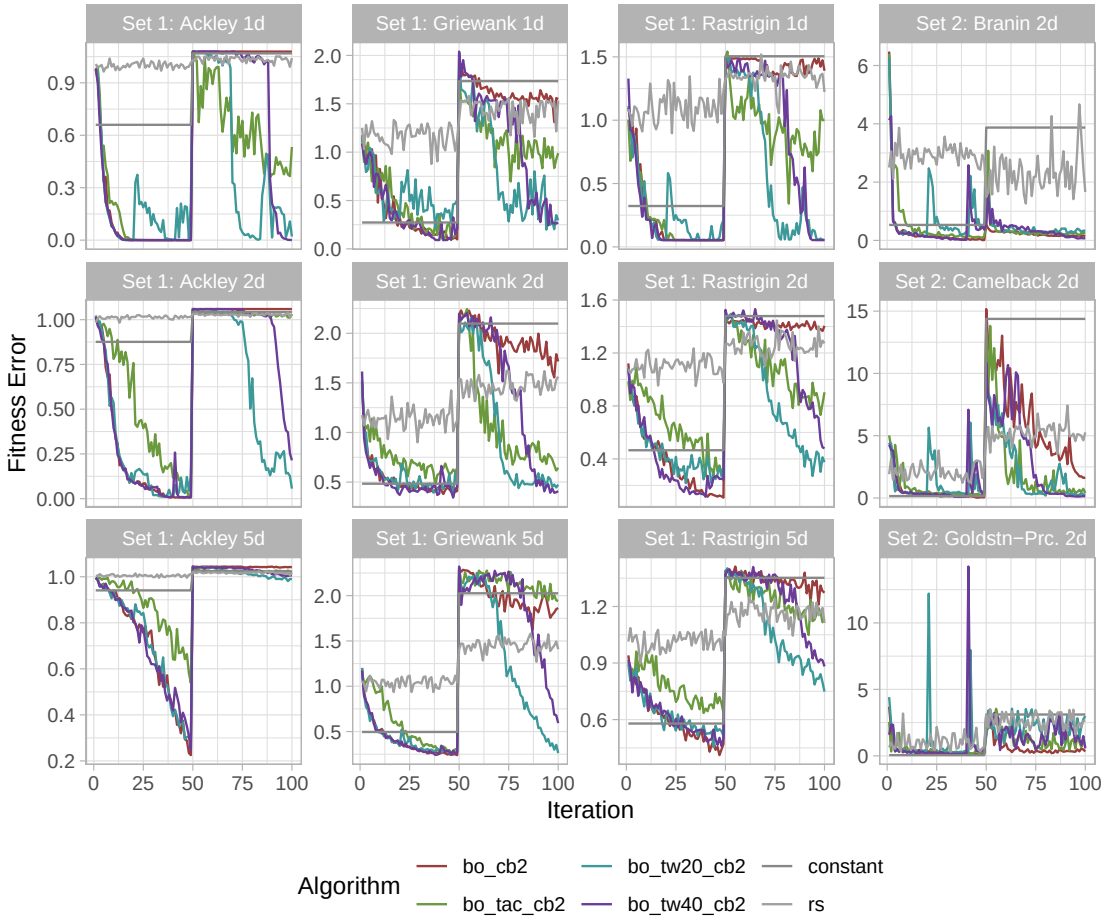

Figure B.7: Optimization curves for optimization strategies that use the `cb2` acquisition function on problems with 100 optimization iterations and sudden drift.

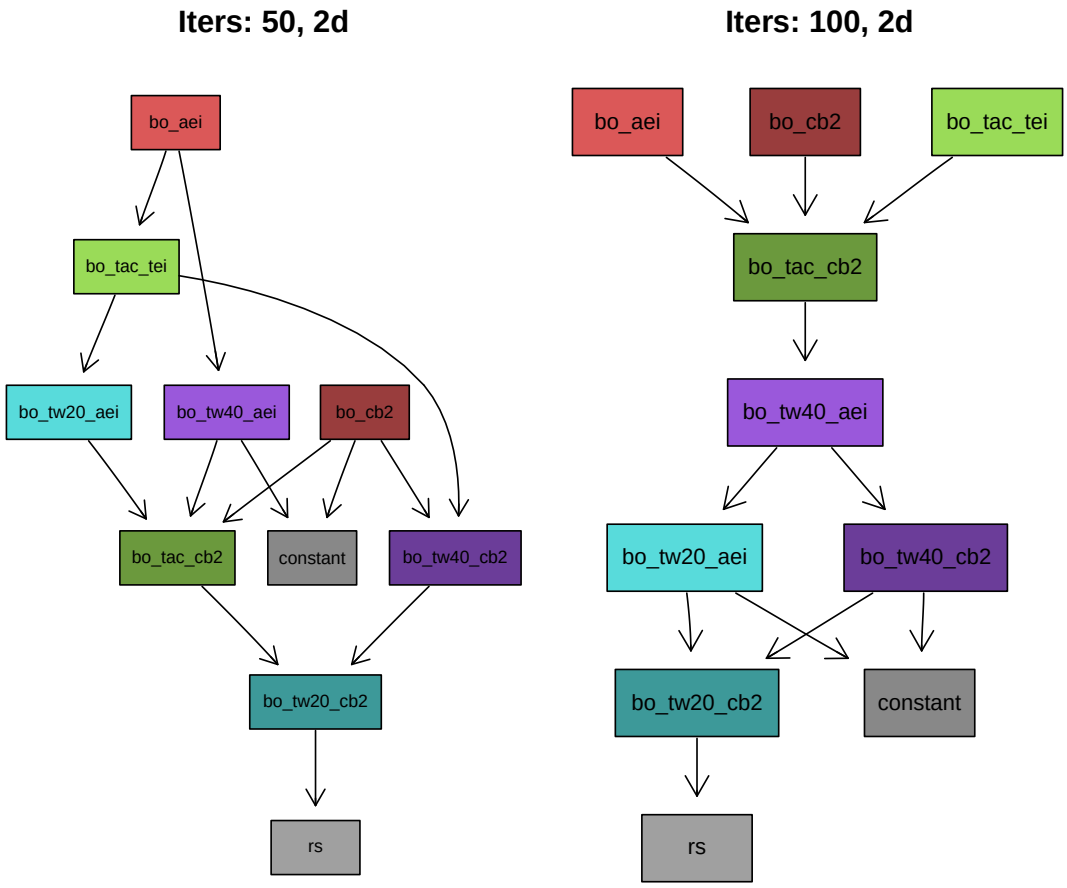

Figure B.8: Preference graphs for each problem subset of set 2 with sudden drift.

Table B.2: MFE and its standard deviation of each optimization method on each problem with sudden drift averaged over 50 stochastic repetitions.

				AEI				CB2					
Set	Function	d	Iterations	bo_aei	bo_tac_tei	bo_tw20_aei	bo_tw40_aei	bo_cb2	bo_tac_cb2	bo_tw20_cb2	bo_tw40_cb2	rs	constant
Set 1	Ackley	1	50	0.72 ±0.08	0.61 ±0.13	0.61 ±0.06	0.71 ±0.08	0.66 ±0.04	0.59 ±0.11	**0.55** ±0.07	0.64 ±0.04	0.95 ±0.03	0.83 ±0.13
			100	0.77 ±0.09	0.45 ±0.08	0.53 ±0.03	0.64 ±0.06	0.60 ±0.03	0.43 ±0.07	**0.38** ±0.05	0.47 ±0.05	0.95 ±0.02	0.83 ±0.13
		2	50	0.78 ±0.10	0.86 ±0.10	**0.74** ±0.10	0.79 ±0.11	0.79 ±0.11	0.87 ±0.08	0.77 ±0.10	0.78 ±0.10	1.00 ±0.01	0.93 ±0.07
			100	0.75 ±0.11	0.73 ±0.13	0.57 ±0.14	0.68 ±0.10	0.69 ±0.09	0.74 ±0.11	**0.54** ±0.12	0.65 ±0.12	1.00 ±0.01	0.92 ±0.08
		5	50	**0.93** ±0.05	0.98 ±0.03	0.95 ±0.05	0.94 ±0.04	0.95 ±0.04	0.98 ±0.03	0.95 ±0.04	0.95 ±0.04	1.01 ±0.00	0.97 ±0.03
			100	0.88 ±0.08	0.95 ±0.06	0.86 ±0.08	0.88 ±0.09	0.87 ±0.08	0.93 ±0.05	**0.85** ±0.07	0.87 ±0.08	1.01 ±0.00	0.97 ±0.03
	Griewank	1	50	1.12 ±0.12	**0.91** ±0.18	0.97 ±0.11	1.10 ±0.11	1.11 ±0.12	0.96 ±0.17	0.95 ±0.11	1.11 ±0.12	1.25 ±0.08	1.12 ±0.20
			100	1.11 ±0.11	**0.79** ±0.14	0.85 ±0.07	0.92 ±0.08	1.04 ±0.09	0.88 ±0.16	0.79 ±0.12	0.84 ±0.11	1.27 ±0.06	1.11 ±0.19
		2	50	1.19 ±0.07	1.28 ±0.26	**1.03** ±0.09	1.19 ±0.07	1.21 ±0.08	1.21 ±0.21	1.07 ±0.11	1.22 ±0.07	1.32 ±0.06	1.26 ±0.14
			100	1.16 ±0.06	1.14 ±0.20	**0.76** ±0.07	0.96 ±0.07	1.20 ±0.07	1.01 ±0.15	0.83 ±0.08	0.97 ±0.08	1.31 ±0.05	1.25 ±0.13
		5	50	1.19 ±0.06	1.82 ±0.16	**1.10** ±0.08	1.21 ±0.08	1.20 ±0.07	1.54 ±0.18	1.21 ±0.07	1.22 ±0.07	1.26 ±0.05	1.29 ±0.05
			100	1.20 ±0.05	1.71 ±0.20	**0.75** ±0.08	1.05 ±0.06	1.17 ±0.06	1.37 ±0.13	0.88 ±0.06	1.12 ±0.06	1.25 ±0.03	1.27 ±0.05
	Rastrigin	1	50	0.96 ±0.08	**0.69** ±0.16	0.85 ±0.08	0.97 ±0.07	0.91 ±0.09	0.74 ±0.14	0.76 ±0.11	0.91 ±0.08	1.16 ±0.06	0.95 ±0.18
			100	0.90 ±0.07	0.61 ±0.10	0.80 ±0.05	0.79 ±0.07	0.84 ±0.08	0.63 ±0.09	**0.59** ±0.08	0.62 ±0.12	1.15 ±0.05	0.94 ±0.18
		2	50	0.99 ±0.06	1.02 ±0.14	**0.86** ±0.10	0.99 ±0.06	0.99 ±0.07	1.03 ±0.13	0.93 ±0.07	1.00 ±0.07	1.16 ±0.06	0.97 ±0.14
			100	0.96 ±0.06	0.89 ±0.14	**0.67** ±0.07	0.84 ±0.06	0.92 ±0.07	0.89 ±0.13	0.69 ±0.07	0.83 ±0.08	1.16 ±0.04	0.96 ±0.14
		5	50	1.00 ±0.04	1.08 ±0.08	**0.94** ±0.05	1.00 ±0.05	1.01 ±0.04	1.06 ±0.07	0.99 ±0.05	1.01 ±0.04	1.09 ±0.03	0.97 ±0.09
			100	0.96 ±0.05	1.03 ±0.09	**0.80** ±0.06	0.89 ±0.04	0.97 ±0.05	1.01 ±0.07	0.86 ±0.05	0.93 ±0.04	1.09 ±0.02	0.96 ±0.09
Set 2	Branin	2	50	0.98 ±0.36	1.09 ±0.44	1.27 ±0.20	1.05 ±0.31	**0.89** ±0.41	1.30 ±0.46	1.25 ±0.22	1.10 ±0.33	3.05 ±0.30	2.65 ±1.87
			100	0.77 ±0.26	0.79 ±0.21	1.14 ±0.13	0.99 ±0.17	**0.64** ±0.34	0.88 ±0.34	1.23 ±0.16	0.96 ±0.20	3.00 ±0.26	2.61 ±1.84
	Camelback		50	4.50 ±0.54	4.64 ±1.97	**3.62** ±0.46	4.49 ±0.71	4.52 ±0.58	4.14 ±1.29	3.91 ±0.47	4.32 ±0.68	4.82 ±0.58	6.93 ±1.83
			100	4.08 ±0.62	3.24 ±1.34	**2.80** ±0.27	3.17 ±0.42	4.10 ±0.67	3.03 ±1.30	3.11 ±0.27	3.20 ±0.53	4.74 ±0.37	6.80 ±1.80
	Gldstn-Pr		50	3.02 ±0.87	4.89 ±1.99	5.56 ±1.68	4.18 ±1.15	4.12 ±1.59	6.03 ±1.86	8.61 ±2.12	5.25 ±1.84	6.39 ±1.88	**2.19** ±3.12
			100	**2.15** ±0.80	3.21 ±1.02	5.39 ±1.10	3.94 ±1.06	2.74 ±1.00	3.95 ±1.84	7.93 ±1.48	5.58 ±1.14	6.21 ±1.42	2.15 ±3.06

B.3 Incremental Drift

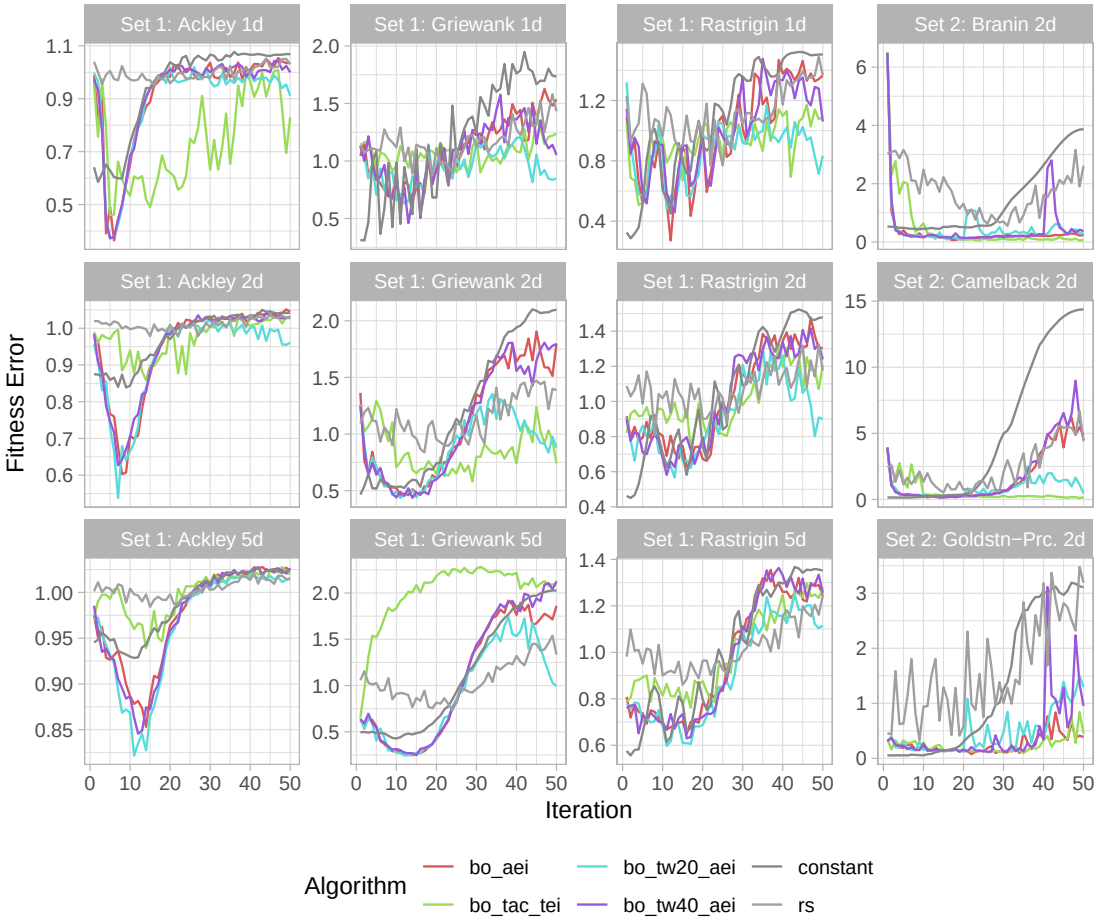

Figure B.9: Optimization curves for optimization strategies that use the `aei` acquisition function on problems with 50 optimization iterations and incremental drift.

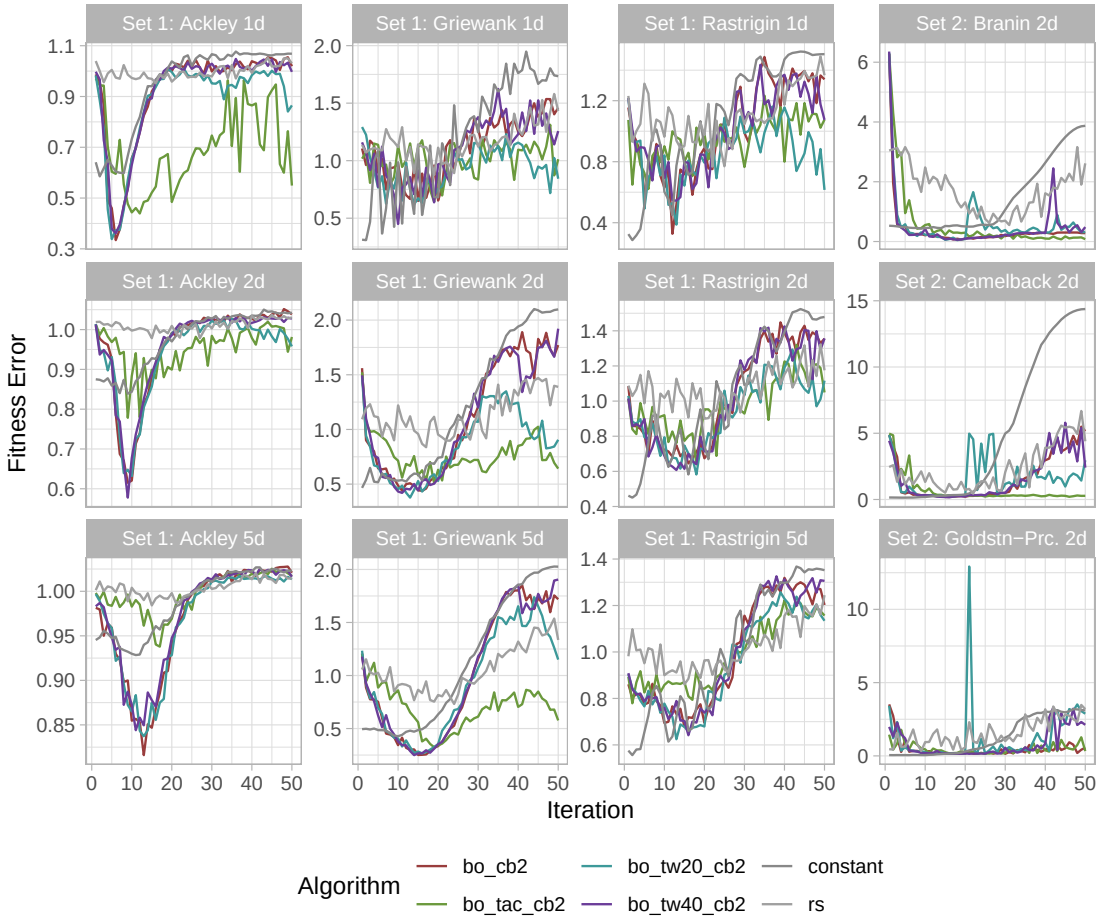

Figure B.10: Optimization curves for optimization strategies that use the `cb2` acquisition function on problems with 50 optimization iterations and incremental drift.

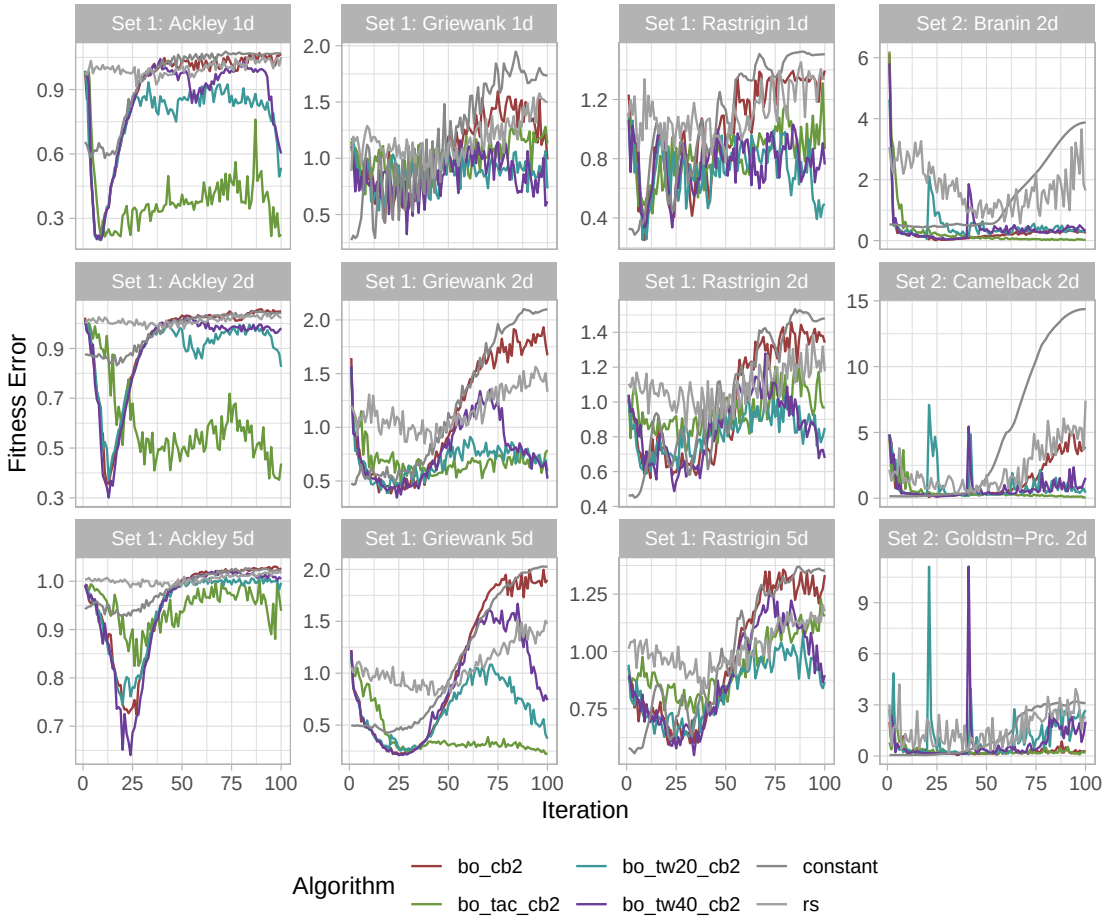

Figure B.11: Optimization curves for optimization strategies that use the `cb2` acquisition function on problems with 100 optimization iterations and incremental drift.

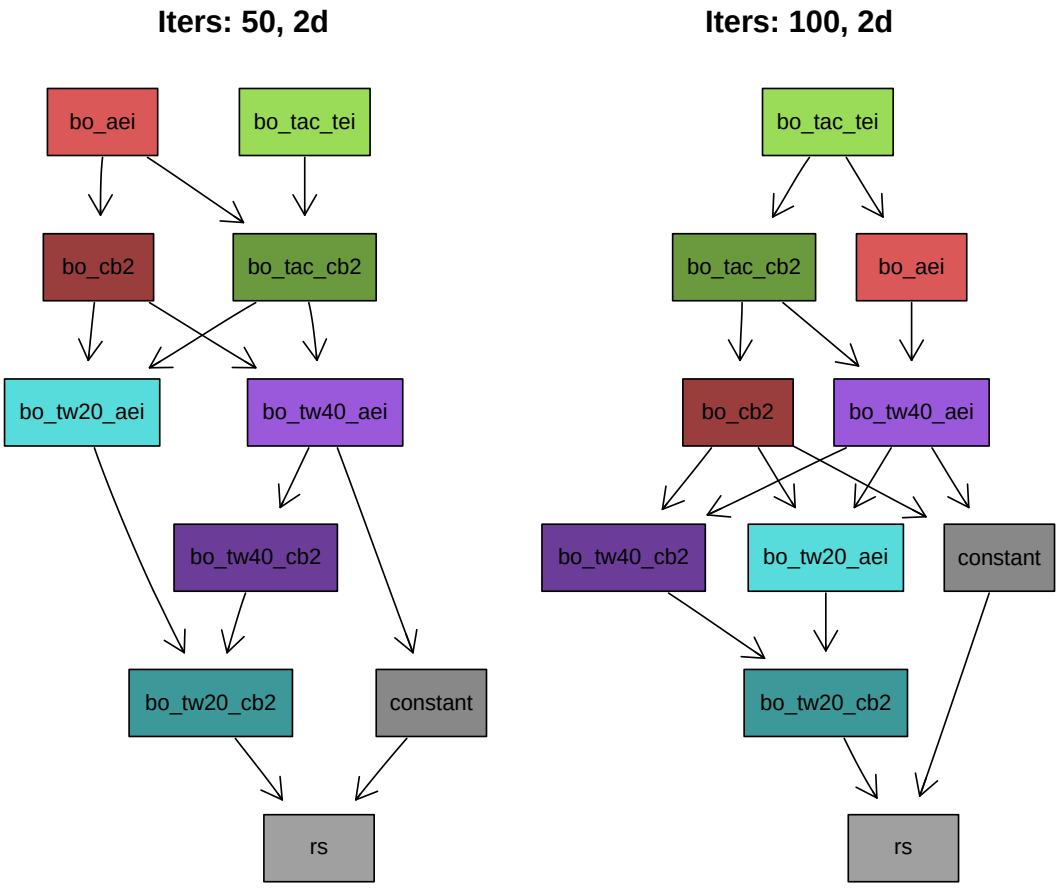

Figure B.12: Preference graphs for each problem subset of set 2 with incremental drift.

Table B.3: MFE and its standard deviation of each optimization method on each problem with incremental drift averaged over 50 stochastic repetitions.

Set	Function	d	Iterations	AEI				CB2					
				bo_aei	bo_tac_tei	bo_tw20_aei	bo_tw40_aei	bo_cb2	bo_tac_cb2	bo_tw20_cb2	bo_tw40_cb2	rs	constant
Set 1	Ackley	1	50	0.86 ±0.04	0.70 ±0.08	0.84 ±0.06	0.87 ±0.04	0.87 ±0.04	**0.67** ±0.09	0.83 ±0.04	0.87 ±0.04	0.92 ±0.03	0.93 ±0.07
			100	0.85 ±0.03	0.58 ±0.04	0.72 ±0.05	0.81 ±0.04	0.86 ±0.03	**0.48** ±0.04	0.71 ±0.06	0.81 ±0.04	0.92 ±0.02	0.92 ±0.07
		2	50	0.92 ±0.03	0.89 ±0.07	0.89 ±0.05	0.92 ±0.04	0.93 ±0.03	**0.87** ±0.09	0.90 ±0.05	0.92 ±0.03	0.98 ±0.01	0.95 ±0.05
			100	0.91 ±0.03	0.75 ±0.08	0.82 ±0.05	0.87 ±0.05	0.92 ±0.03	**0.63** ±0.09	0.83 ±0.05	0.85 ±0.05	0.98 ±0.01	0.95 ±0.05
		5	50	0.96 ±0.02	0.98 ±0.02	**0.95** ±0.02	0.96 ±0.02	0.96 ±0.02	0.98 ±0.02	0.96 ±0.02	0.96 ±0.02	1.00 ±0.00	0.98 ±0.02
			100	0.95 ±0.02	0.92 ±0.05	0.92 ±0.03	0.93 ±0.03	0.94 ±0.03	**0.91** ±0.07	0.93 ±0.03	0.93 ±0.02	1.00 ±0.00	0.98 ±0.02
	Griewank	1	50	1.08 ±0.10	1.05 ±0.13	**0.94** ±0.10	1.06 ±0.11	1.09 ±0.10	1.01 ±0.10	0.94 ±0.09	1.06 ±0.11	1.16 ±0.09	1.22 ±0.18
			100	1.04 ±0.13	1.02 ±0.08	0.87 ±0.08	**0.86** ±0.06	1.05 ±0.09	1.00 ±0.08	0.86 ±0.08	0.87 ±0.09	1.16 ±0.07	1.21 ±0.18
		2	50	1.10 ±0.09	1.00 ±0.23	0.90 ±0.10	1.08 ±0.08	1.10 ±0.10	**0.89** ±0.12	0.90 ±0.11	1.09 ±0.09	1.16 ±0.06	1.21 ±0.21
			100	1.07 ±0.08	0.89 ±0.15	0.77 ±0.09	0.81 ±0.10	1.08 ±0.08	0.79 ±0.08	**0.74** ±0.07	0.81 ±0.09	1.16 ±0.05	1.20 ±0.21
		5	50	1.03 ±0.04	1.77 ±0.46	0.91 ±0.10	1.07 ±0.07	1.00 ±0.05	**0.85** ±0.38	0.94 ±0.08	1.00 ±0.06	1.07 ±0.05	1.11 ±0.14
			100	1.02 ±0.05	1.24 ±0.43	0.64 ±0.10	0.83 ±0.07	1.02 ±0.04	**0.50** ±0.24	0.67 ±0.07	0.86 ±0.08	1.08 ±0.03	1.10 ±0.14
	Rastrigin	1	50	0.97 ±0.08	0.91 ±0.10	0.87 ±0.09	0.96 ±0.08	0.98 ±0.08	0.93 ±0.11	**0.86** ±0.10	0.96 ±0.10	1.07 ±0.07	1.09 ±0.14
			100	0.91 ±0.10	0.79 ±0.07	0.77 ±0.07	0.77 ±0.08	0.95 ±0.10	0.84 ±0.09	**0.75** ±0.06	0.78 ±0.10	1.07 ±0.06	1.08 ±0.14
		2	50	1.02 ±0.08	1.01 ±0.10	**0.93** ±0.08	1.00 ±0.08	1.02 ±0.06	0.98 ±0.10	0.94 ±0.08	1.01 ±0.06	1.08 ±0.05	1.04 ±0.10
			100	0.97 ±0.06	0.93 ±0.08	**0.82** ±0.06	0.85 ±0.08	0.98 ±0.05	0.95 ±0.08	0.85 ±0.05	0.87 ±0.07	1.08 ±0.03	1.03 ±0.09
		5	50	0.98 ±0.04	1.00 ±0.08	**0.91** ±0.05	0.97 ±0.05	0.99 ±0.03	0.99 ±0.10	0.94 ±0.04	0.99 ±0.04	1.01 ±0.03	0.99 ±0.07
			100	0.96 ±0.03	0.96 ±0.10	**0.83** ±0.05	0.89 ±0.05	0.97 ±0.05	0.94 ±0.08	0.84 ±0.05	0.89 ±0.04	1.01 ±0.02	0.99 ±0.07
Set 2	Branin	2	50	0.66 ±0.35	0.88 ±0.29	1.00 ±0.20	0.97 ±0.41	**0.63** ±0.30	0.96 ±0.40	1.10 ±0.25	0.90 ±0.33	2.27 ±0.27	2.21 ±1.85
			100	0.56 ±0.23	**0.54** ±0.20	0.91 ±0.12	0.80 ±0.16	0.56 ±0.26	0.57 ±0.26	1.01 ±0.11	0.83 ±0.15	2.27 ±0.18	2.19 ±1.83
	Camelback		50	2.61 ±0.53	**1.84** ±0.79	2.37 ±0.40	2.76 ±0.56	2.59 ±0.49	1.99 ±0.74	3.08 ±0.44	2.66 ±0.30	3.46 ±0.53	5.08 ±2.09
			100	2.23 ±0.37	**1.20** ±0.57	2.04 ±0.19	1.77 ±0.21	2.12 ±0.39	1.29 ±0.53	2.50 ±0.19	2.13 ±0.25	3.43 ±0.37	5.02 ±2.08
	Gldstn-Pr		50	**2.20** ±1.05	2.83 ±2.89	3.93 ±1.32	3.07 ±1.23	3.23 ±1.44	3.79 ±2.07	7.51 ±2.29	5.25 ±1.66	6.96 ±2.13	2.34 ±3.50
			100	**1.51** ±0.59	1.52 ±1.27	4.02 ±1.12	2.82 ±0.92	2.08 ±0.93	2.47 ±1.76	6.93 ±1.49	4.59 ±1.28	6.91 ±1.69	2.32 ±3.47