```
gdata: read.xls support for 'XLS' (Excel 97-2004) files ENABLED.

gdata: read.xls support for 'XLSX' (Excel 2007+) files ENABLED.
```

Working Paper: Sequential Parameter Optimization and Optimal Computational Budget Allocation for Noisy Optimization Problems

CIOP Report 01/11
ISSN 2191-365X

Thomas Bartz-Beielstein Martina Friese

February 21, 2011

Abstract

Sequential parameter optimization (SPO) is a heuristic that combines classical and modern statistical techniques to improve the performance of search algorithms. It includes a broad variety of meta models, e.g., linear models, random forest, and Gaussian process models (Kriging). The selection of an adequate meta model can have significant impact on SPO's performance. A comparison of different meta models is of great importance. A recent study indicated that random forest based meta models might be a good choice. This rather surprising result will be analyzed in this paper.

Moreover, *Optimal Computing Budget Allocation* (OCBA), which is an enhanced method for handling the computational budget spent for selecting new design points, is presented. The OCBA approach can intelligently determine the most efficient replication numbers. We propose the integration of OCBA into SPO.

In this study, SPO is directly used as an optimization method on different noisy mathematical test functions. This is differs from the standard way of using SPO for tuning algorithm parameters in the context of complex real-world applications. Using SPO this way allows for a comparison to other optimization algorithms.

Our results reveal that the incorporation of OCBA and the selection of Gaussian process models are highly beneficial. Moreover, SPO outperformed three different alternative optimization algorithms on a set of five noisy mathematical test functions.

1 Introduction

Sequential parameter optimization (SPO) is a heuristic that combines classical and modern statistical techniques. It was originally developed for the analysis of search algorithms [6]. Here, we will use SPO itself as a search algorithm, i.e., SPO is applied to the objective function directly. An introduction to the state-of-the-art R implementation of SPO, the so-called *sequential parameter optimization toolbox* (SPOT), is presented in [3, 2].

This paper focuses on some internal aspects of SPOT such as the class of meta models used for generating new design points. Generally, two classes of meta models have been proven useful in the SPOT framework: (i) tree-based models such as random forest and (ii) stochastic process models (Gaussian processes, Kriging). Another aspect is the computational budget (number of function evaluations) that is spent for selecting new design points. Here, we propose the integration of a control-theoretic simulation technique called *optimal computing budget allocation* (OCBA) into SPOT. The OCBA approach can intelligently determine the most efficient replication numbers.[11] The goal is to obtain the highest decision quality using a fixed computing budget or to attain a desired simulation decision quality using a minimum computing budget. The approach presented in our study relies on ideas developed by Lasarczyk [16]. This SPOT-OCBA variant is compared to SPOT's standard technique of increasing the number of repeats.

The experimental study presented in this paper enables a comparison of SPOT with prominent search algorithms such as *covariance matrix adaptation evolution strategy* (CMA-ES), *Nelder Mead* (NM), and *simulated annealing* (SANN). Summing up, the following research questions are investigated:

Q-1. Does OCBA improve SPOT?
Q-2. How do random-forest based meta models perform in comparison to Kriging-based meta models?
Q-3. Regarding classical optimization algorithms: Does SPOT show a competitive performance on standard test problems?

This paper is organized as follows. SPOT and OCBA are introduced in Section 2. SPOT provides several meta models, which can be used for estimating objective function values. The meta models used for experiments described in this document are also presented in this section.

Test functions considered in this study are presented in Sect. 3. An overview of the general experiment setup is given in Sect. 4. Section 5 presents the results to our corresponding research questions and their analysis. Finally, Sect. 6 presents a summary and an outlook.

2 Sequential Parameter Optimization

2.1 SPOT in a Nutshell

SPOT uses the available budget (e.g., simulator runs, number of function evaluations) sequentially, i.e., it uses information from the exploration of the search space to guide the search by building one or several meta models. Predictions from meta models are used to select new design points. Meta models are refined to improve knowledge about the search space. SPOT provides tools to cope with noise, which typically occurs when real world applications, e.g., stochastic simulations, are run. It guarantees comparable confidence for search points. Users can collect information to learn from this optimization process, e.g., by applying *exploratory data analysis* (EDA) [20, 10]. Last, but not least, SPOT provides mechanisms both for interactive and automated tuning [7, 5]. An R version of this toolbox for interactive and automatic optimization of algorithms can be downloaded from CRAN.[1] Programs and files from this study can be requested from the author.

As can be seen from Algorithm 1, SPOT requires the generation of an initial design. Additionally, SPOT generates new design points during the sequential step. *Latin hypercube sampling* was chosen as the generator of design points during the initial and sequential SPOT steps. They were chosen, because they are easy to implement and understand. Many design point generators are available in R, see, e.g., the *CRAN Task View: Design of Experiments (DoE) & Analysis of Experimental Data.*[2]

There is a strong interaction between design generators and meta models, because the optimality of a design point depends on the meta model [18, 19]. This paper modifies SPOT's meta models, while design generators remain unchanged. The impact of the variation of the design generators on the algorithm's performance will be subject of a forthcoming paper.

2.2 OCBA Introduction

SPOT provides tools for improving the confidence during the search. First approaches increase the number of repeats. An early SPOT implementation proceeded as follows [6]:

[1]`http://cran.r-project.org/web/packages/SPOT/index.html`
[2]`http://cran.r-project.org/web/views/ExperimentalDesign.html`

At each step, two new designs are generated and the best is re-evaluated. This is similar to the selection procedure in $(1+2)$-Evolution Strategies. The number of repeat runs, k, of the algorithm designs is increased (doubled), if a design has performed best twice or more. A starting value of $k = 2$ was chosen.

This simple approach did not use any information about the variance.

Lasarczyk was the first who combined SPOT and OCBA [16]. OCBA was developed to ensure a high *probability of correct selection* (PCS). To maximize PCS, a larger portion of the available budget is allocated to those designs that are critical to the process of identifying the best candidates. OCBA uses sample means and variances in the budget allocation procedure in order to maximize PCS.

OCBA's central idea can be explained as follows. Consider a number of simulation replications, say T, which can be allocated to m competing design points with means $\overline{Y}_1, \overline{Y}_2, \ldots, \overline{Y}_m$ and finite variances $\sigma_1^2, \sigma_2^2, \ldots, \sigma_m^2$, respectively. The *Approximate Probability of Correct Selection* can be asymptotically maximized when

$$\frac{N_i}{N_j} = \left(\frac{\sigma_i/\delta_{b,i}}{\sigma_j/\delta_{b,j}}\right)^2, \quad i,j \in \{1,2,\ldots,m\}, \text{ and } i \neq j \neq b, \tag{1}$$

$$N_b = \sigma_b \sqrt{\sum_{i=1, i\neq b} \frac{N_i^2}{\sigma_i^2}},$$

where N_i is the number of replications allocated to design i, and $\delta_{b,j} = \overline{Y}_b - \overline{Y}_i$ denotes the difference of the i-th and b-th mean with $\overline{Y}_b \leq \min_{i\neq b} \overline{Y}_i$. As can be seen from (1), the allocated computing budget is proportional to variance and inversely proportional to the difference from the best design.

The OCBA implementation in our study is based on Lasarczyk's work [16]. SPOT with OCBA is shown in Algorithm 1. New design points which were proposed by the meta model are evaluated several times, e.g., twice [3]. During each SPOT step, a certain budget (here: `spot.ocba` = 3, as can be seen from Table 4) is allocated to the candidate solutions to ensure a high PCS for the best design point. Chen and Lee present a comprehensive coverage of the OCBA methodology [11].

2.3 Meta Models Used During SPOT Runs

SPOT processes data sequentially, i.e., starting from a small initial design, further design points are generated using a meta model. Many meta models are available in R. Similar as for the design generators the user has the option of choosing between state-of-the-art meta models for tuning his algorithm or writing his own meta model and use it as a plugin for SPOT. The default SPOT installation contains several meta models. The R implementation of `randomForest` was chosen as SPOT's default one. This is quite robust and can handle categorical and numerical values needing only a comparably small amount of computational resources. Table 1 summarizes meta models used for experiments described in this document.

2.3.1 Random Forest-based Parameter Tuning

The Random Forest method from the R package `randomForest` implements Breiman's algorithm, which is based on Breiman and Cutler's original Fortran code, for classification and regression [9]. It is implemented as a SPOT plugin, which can be selected via setting the command `seq.predictionModel.func` according to Table 1 in SPOT's configuration file.

Four different variations of the Random Forest plugin are used here.

- `spotPredictRandomForest` uses a random forest meta model, which will be evaluated based on the created sequential design to find good new design points. (package: `randomForest`).

[3]This value can be modified using the `init.design.repeats` variable in SPOT's config file

Algorithm 1: SPOT-OCBA.
t_0 = `init.design.repeats`, t = `seq.ocba.budget`,
l = `seq.design.size`, d = `seq.design.new.size`

```
// phase 1, building the model:
let $F$ be the tuned algorithm;
// design considerations necessary:
generate an initial population $X = \{\bar{x}^1, \dots, \bar{x}^m\}$ of $m$ parameter vectors;
let $t_0$ be the initial number of tests for determining estimated function values;
foreach $\bar{x} \in X$ do
    evaluate $F$ with $\bar{x}$ $t_0$ times to determine the estimated function value $\hat{y}$ of $\bar{x}$;
end
// phase 2, using and improving the model:
while termination criterion not true do
    // OCBA:
    let $B \subseteq X$ denote the subset of candidate solutions with best estimated function value $\hat{y}$;
    let $t$ denote the OCBA budget;
    distribute $t$ among $B$, i.e., generate OCBA distribution $\mathcal{O}$;
    // model considerations necessary:
    build meta model $f$ based on $X$ and $\{\hat{y}^1, \dots, \hat{y}^{|X|}\}$;
    // design considerations necessary:
    generate a set $X'$ of $l$ new parameter vectors by random sampling;
    foreach $\bar{x} \in X'$ do
        calculate $f(\bar{x})$ to determine the estimated function value $f(\bar{x})$ of $\bar{x}$;
    end
    select set $X''$ of $d$ parameter vectors from $X'$ with best predicted utility ($d \ll l$);
    evaluate $F$ with $B$ following the OCBA distribution $\mathcal{O}$; // (improve confidence)
    evaluate $F$ $t_0$ times with each $\bar{x} \in X''$ to determine the estimated function values $\hat{y}$;
    extend the population by $X = X \cup X''$;
end
```

Table 1: Six SPOT meta models used in this study

Type	Name of the SPOT plugin	Abbreviation
Gaussian processes (Kriging)	`spotPredictMlegp`	sMl
Gaussian processes (Kriging) with Quasi Newton	`spotPredictMlegpOptim`	sMlO
Random forest	`spotPredictRandomForest`	sRF
Random forest with Gaussian processes (Kriging)	`spotPredictRandomForestMlegp`	sRfM
Random forest with Particle Swarm Optimization	`spotPredictRandomForestPSO`	sPS
Random forest with Quasi Newton	`spotPredictRandomForestOptim`	sRfO

- `spotPredictRandomForestOptim` uses a random forest meta model which will be optimized by the R-internal `optim` function with the method BFGS (Quasi-Newton) to determine promising design points (R packages: `randomForest`, `base`).
- `spotPredictRandomForestPSO` uses random forest and Particle Swarm Optimization (R packages: `randomForest`, `pso`).
- `spotPredictRandomForestMlegp` uses random forest and MLEGP, see Sect. 2.3.2. (R packages: `randomForest`, `mlegp`). The set of new design points is distributed among random forest and MLEGP, which are evaluated in parallel.

2.3.2 Maximum Likelihood Estimates of Gaussian Processes

SPOT provides a plugin for the *Maximum Likelihood Estimation of Gaussian process* (`mlegp`) package which is available in R. The package `mlegp` finds maximum likelihood estimates of Gaussian processes for univariate and multi-dimensional responses, for Gaussian processes with product exponential correlation structures; constant or linear regression mean functions; no nugget term, constant nugget terms, or a nugget matrix that can be specified up to a multiplicative constant [12].

`mlegp` is implemented as a SPOT plugin, which can again be selected via setting `seq.prediction-`

Table 2: Coverage of difficulty criteria by test functions

Function	1	2	3	4
Branin	+	+	+	-
Six Hump	+	-	+	+
Mexican Hat	+	-	+	+
Rosenbrock	+	-	-	-
Rastrigin	-	-	+	+

`Model.func` according to Table 1 in SPOT's configuration file. Two different variations of the `mlegp` plugin are used here.

- `spotPredictMlegp`. The model is evaluated based on the created sequential design to find good new design points (package: `mlegp`).
- `spotPredictMlegpOptim` uses a MLEGP meta model which will be optimized by the R-internal `optim` function with the method BFGS to find a good new design point (R packages: `mlegp`, `base`).

3 Test Functions

3.1 Considerations

Our main goal when choosing the test functions was to obtain a preferably small number of these, which cover a variety of different difficulty criteria. The following criteria were chosen beforehand:

1. The function's optimum does not lie at the origin.
2. The function is not symmetric.
3. The function is multi-modal.
4. The function has many local minima.

In addition, the functions should be well known in the optimization community to improve reproducibility and comparability of results.

The chosen test functions cover the different difficulty criteria as shown in Table 2. To gain some additional difficulty and stay consistent with SPOT's original area of application, we added fitness-proportional noise to all test functions. This is the most common case for real-world settings: values and variability both change together. We restricted ourselves to the two-dimensional instances of the test functions as the higher dimensional instances require a much higher budget of target function evaluations and thus a modified setup. The number of function evaluations was chosen as the termination criterion.

3.2 Function Definitions

3.2.1 Branin

The Branin function

$$f(x_1, x_2) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10,$$

with region of interest $x_1 \in [-5, 10]$ and $x_2 \in [0, 15]$ was chosen as a test function, because it is multimodal and not symmetric. It has three global minima, $\vec{x}_1^* = (3.1416, 2.2750)$, $\vec{x}_2^* = (9.4248, 2.4750)$ and $\vec{x}_3^* = (-3.1416, 12.2750)$ with
$y^* = f(\vec{x}_i^*) = 0.3979$, $(i = 1, 2, 3)$.

3.2.2 Six Hump

The Six Hump function

$$f\left(x_1, x_2\right) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1 x_2 + \left(-4 + 4x_2^2\right) x_2^2,$$

with region of interest $x_1 \in [-1.9, 1.9]$ and $x_2 \in [-1.1, 1.1]$ was chosen as a test function, because it is multimodal with many local minima. It is also not rotationally, but point symmetric around the origin. It has two global minima, $\vec{x}_1^* = (0.089842, -0.712656)$ and $\vec{x}_2^* = (-0.089842, -0.712656)$ with
$y^* = f\left(\vec{x}_i^*\right) = -1.031628$, $(i = 1, 2)$.

3.2.3 Mexican Hat

The Mexican Hat function

$$f(x_1, x_2) = \frac{\sin\left(\sqrt{x_1^2 + x_2^2}\right)}{\sqrt{x_1^2 + x_2^2}},$$

with region of interest $x_1 \in [-8, 8]$ and $x_2 \in [-8, 8]$ was chosen as a test function, because it is multimodal with many local minima and rotationally symmetric. It has its global optima at $\vec{x}^* \in \left\{\vec{x} \in \mathbb{R}^2 / \left\|\vec{x}^*\right\|_2 = 4.493409\right\}$ with $y^* = f\left(\vec{x}^*\right) = -0.217233$.

3.2.4 Rosenbrock

The Rosenbrock function

$$f\left(x_1, x_2\right) = (1 - x)^2 + 100\left(y - x^2\right)^2$$

with region of interest $x_1 \in [-2, 2]$ and $x_2 \in [-2, 2]$ was chosen as a test function, because it is unimodal; it has its global minimum at $\vec{x}^* = (1, 1)$ with $y^* = f(\vec{x}^*) = 0$. The global minimum lies inside a long, narrow, parabolic shaped, slowly descending valley, what makes it even harder to find. The function is not rotationally but axially symmetric.

3.2.5 Rastrigin

The Rastrigin function

$$f\left(x_1, x_2\right) = 20 + \sum_{i=1}^{2}\left(x_i^2 - 10\cos\left(2\pi x_i\right)\right)$$

with region of interest $x_1 \in [-5.12, 5.12]$ and $x_2 \in [-5.12, 5.12]$ was chosen as a test function, because it has a large number of local minima and only one global minimum at $\vec{x}^* = (0, 0)$ with $y^* = f\left(\vec{x}^*\right) = 0$. The function is not rotationally but axially symmetric.

4 Statistical Comparison

4.1 General Setup

All experiments share the general setup summarized in Table 3. In order to obtain reliable results, each algorithm is run ten times, with varying seeds. Fitness-proportionate noise, calculated as follows, was added to all objective function values:

$$\text{noise} = (y - y_{\text{opt}}) \times \sigma_\epsilon \times \frac{\texttt{rnorm}(1)}{100},$$

where y is the function value at the current position, y_{opt} is the value of the functions global optimum, σ_ϵ describes the noise level factor, $\sigma_\epsilon \in \{1.0, 10.0\}$, whereas $\texttt{rnorm}(1)$ is R's random

Table 3: General Setup

Generic Parameter	Value
Number of Function Evaluations	100
Initialization	LHD
Number of Algorithm Runs	10

Table 4: SPOT Setup

SPOT Setup Parameter	Value
`auto.loop.nevals`	100
`init.design.size`	10
`init.design.repeats`	2
`init.design.func`	”spotCreateDesignLhd”
`init.design.retries`	100
`spot.ocba`	TRUE | FALSE
`seq.ocba.budget`	3
`seq.design.size`	200
`seq.design.oldBest.size`	3
`seq.design.new.size`	3
`seq.design.func`	”spotCreateDesignLhd”

number generator for the normal distribution. The final best solution is evaluated on the noise free test function, i.e., we calculate $f(\vec{x})$ based on the parameters $\vec{x}$ determined by the algorithm.

SPOT uses a budget of one hundred target function evaluations and an initial design size of ten. As our target functions are noisy, each initial design point is evaluated twice. So the first twenty of one hundred function evaluations are spend on the initial design, which is created by the SPOT internal Latin Hypercube Design function. Each sequential step is then allowed to use two hundred evaluations of the meta model to detect good new design points. The best three design points will be used as the new design and evaluated by the target function. To deal with noise, there will also be repeated evaluations of the old design, depending on the chosen sequential step method (with or without OCBA). OCBA is used to adapt the number of repeats for each design point more efficiently and is allowed to use a budget of three design points for repeated evaluation in each sequential step. If OCBA is not used, the three best points of the old design will always be repeated.

The settings according to the description above can also be find in Table 4. However, the prediction model for the sequential step in SPOT (`seq.predictionModel.func`) is not mentioned in this table, since it is listed in Table 1.

4.2 Statistical Analysis

We are comparing nine algorithms (six SPOT variants and three optimization algorithms) on five test functions with two different noise levels. A typical question at this point is ”Which comparison method should be used?”

The first step of our analysis relies on EDA. EDA comprehends methods such as plotting the raw data, e.g., histograms, plotting simple statistics such as mean plots, standard deviation plots, box plots, and main effects plots of the raw data. We will use Trellis plots which position the graphical output so as to maximize our natural pattern-recognition abilities, such as using multiple plots per page.

The second step comprehends statistical tools such as *analysis of variance* (ANOVA). First, we have to decide whether we want to compare results with a reference algorithm. This procedure is adequate if one well established algorithm is the gold standard. Given n algorithms, this techniques requires $n-1$ comparisons only. Otherwise, pairwise comparisons can be used. Note, the combinatorial complexity of pairwise comparisons is large, i.e., n algorithms require

$$C(n) = n \times (n-1)/2 \tag{2}$$

comparisons. A standard approach from statistics reads as follows.

S-1. Use classical analysis of variance to determine whether there are differences between the treatment means. Under normality assumptions, use ANOVA for performing one-way location analysis. Otherwise, Kruskal-Wallis Rank Sum Test or its equivalent for two groups, the Wilcoxon rank sum test can be used. [15]

S-2. Next, if the answer from the first step is positive, analyze *which* means differ using *multiple comparison methods*. Under normality assumptions, *Tukey Honest Significant Differences* (TukeyHSD) can be used. Otherwise, the *Dunnett-Tukey-Kramer Pairwise Multiple Comparison tests* is recommended. [13]

5 Research Questions

We are following an approach for performing the experimental analysis and reporting results which has been proposed in [6, 1]. Due to the limited space in this paper, the corresponding twelve steps proposed are condensed to four steps: (i) research question, (ii) experimental setup, (iii) analysis, and (iv) scientific relevance.

5.1 Q-1: Does OCBA improve SPOT's performance?

5.1.1 Research Question

This research question is devoted to the influence of OCBA on SPOT's performance. Does the integration of OCBA improve SPOT's performance?

5.1.2 Experimental Setup

The set of five well known test functions as described in Sect. 3 was used for this comparison. Two different noise levels ($\sigma_\epsilon \in \{1.0, 10.0\}$) are used. Two variants of the SPOT implementation are compared in this experiment. SPOT version 0.1.1065 as available on the Comprehensive R Archive Network (CRAN), and SPOT with OCBA (Algorithm 1) as presented in this paper.

5.1.3 Analysis

Figures 1 and 2 present an overview. The noise level was set to $\sigma_\epsilon = 1.0$ and $\sigma_\epsilon = 10.0$, respectively. Each panel illustrates SPOT's results for one objective function. This simple visual inspection provides a good starting point for a deeper statistical analysis. Figures 1 and 2 clearly indicate that OCBA improves SPOT performance.

Since the objective function values are distributed non normally, rank-based tests will be used. While optimizing Branin, SixHump and Mexican Hat, statistical significant differences can be detected (Table 5). Wilcoxon rank sum tests reveal that OCBA does improve SPOT's performance significantly on three of the five test functions.

```
> kruskal.test(Y ~ ocba, data = df.spot.noise1.Branin)

        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 7.4779, df = 1, p-value = 0.006246

> kruskal.test(Y ~ ocba, data = df.spot.noise1.MexicanHat)
```

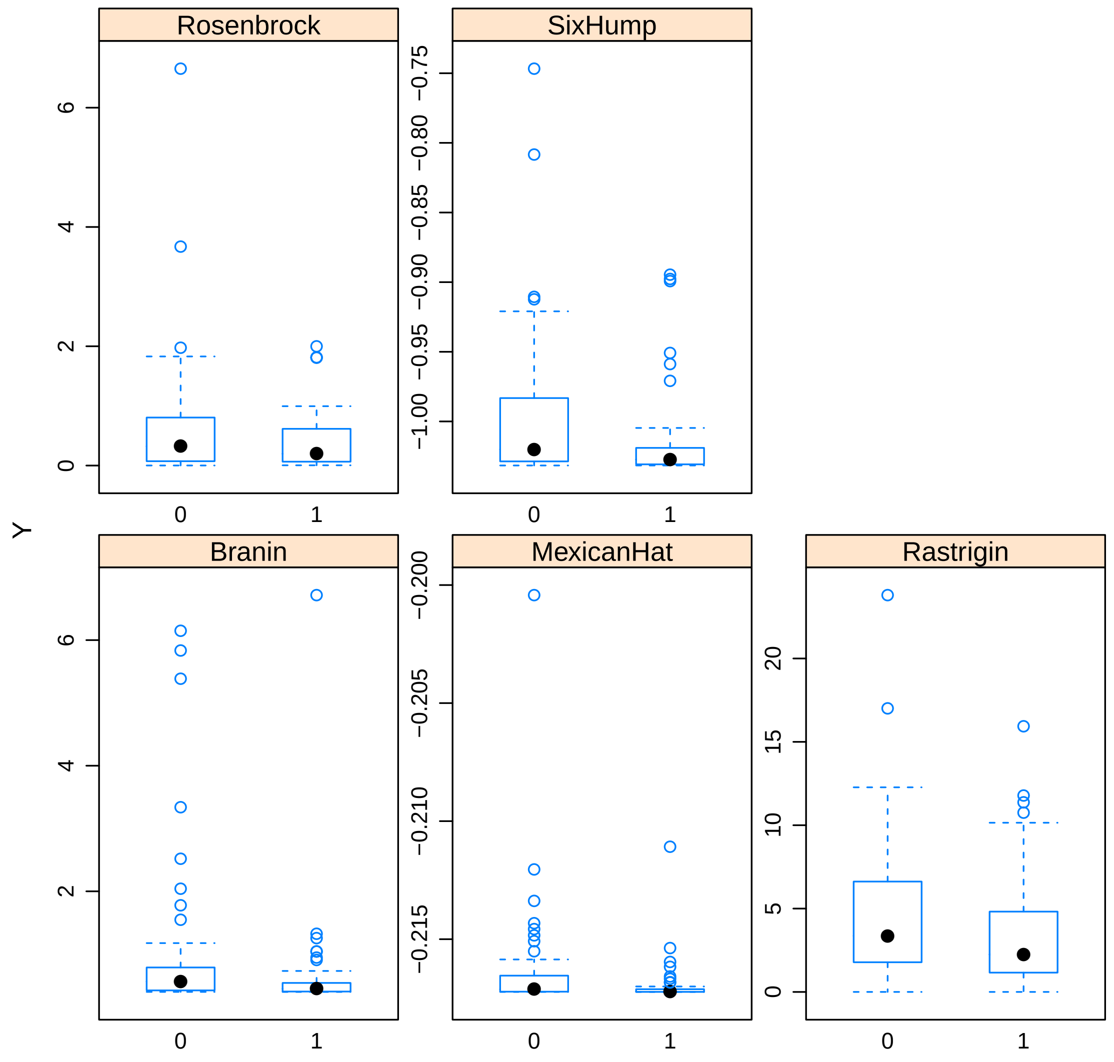

Figure 1: Trellis plots visualizing SPOT's performance without and with OCBA, 0 respectively 1. Noise level $\sigma_\epsilon = 1.0$, fitness proportionate. Smaller values are better

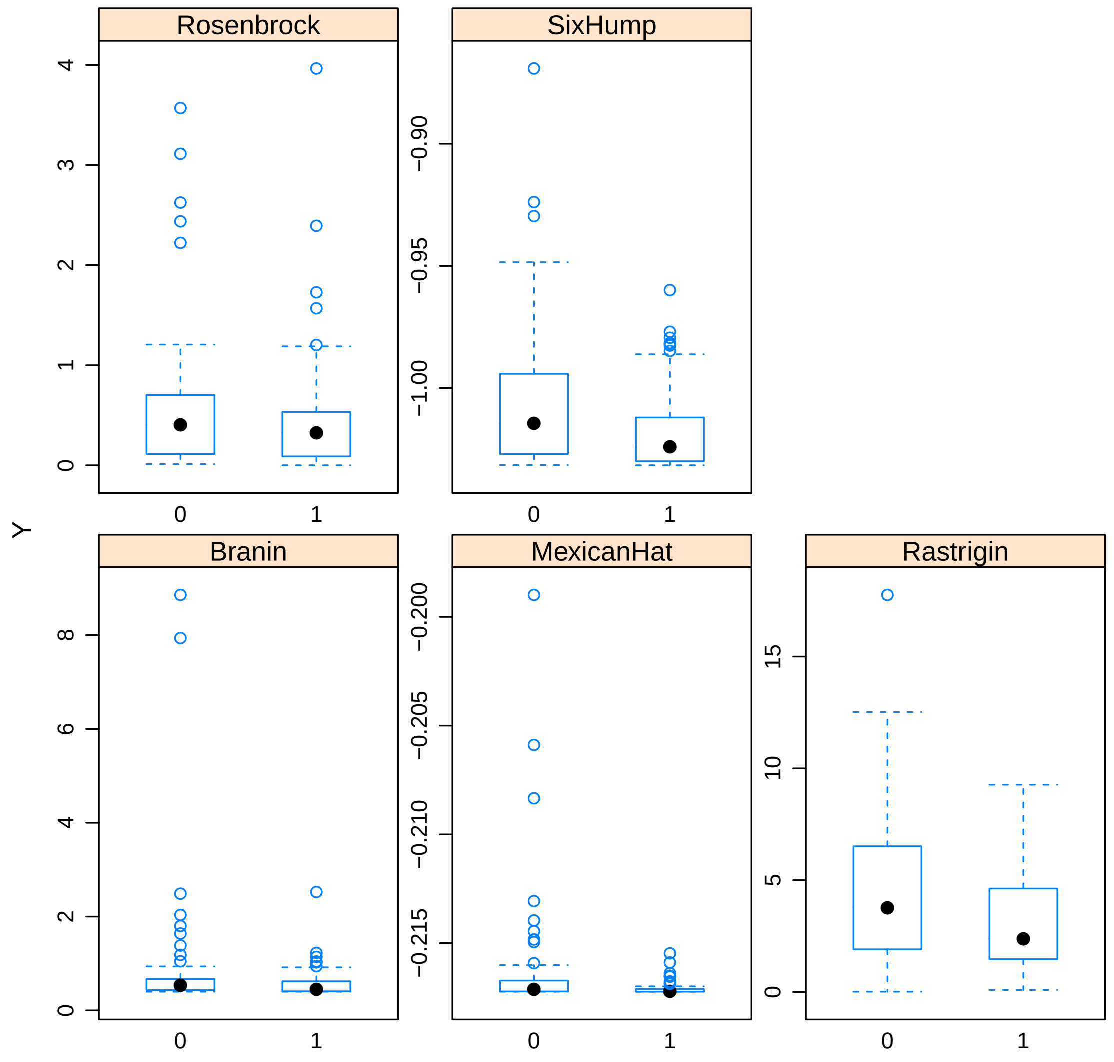

Figure 2: Trellis plots visualizing SPOT's performance without and with OCBA, 0 respectively 1. Noise level $\sigma_\epsilon = 10.0$, fitness proportionate. Smaller values are better

Table 5: Q-1. Results from Wilcoxon Rank Sum Tests, noise level $\sigma_\epsilon = 1.0$ and 10.0. The same data as in Fig. 1 were used for these tests. Yes/no indicators refer to a significance level of 0.05

	$\sigma_\epsilon = 1.0$		$\sigma_\epsilon = 10.0$	
Function	p-value	sign.	p-value	sign.
Branin	0.006246	yes	0.02049	yes
Mexican Hat	0.00233	yes	0.007972	yes
Rastrigin	0.1388	no	0.02210	yes
Rosenbrock	0.2547	no	0.1228	no
Six Hump	0.005859	yes	0.006347	yes

```
        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 9.2697, df = 1, p-value = 0.00233

> kruskal.test(Y ~ ocba, data = df.spot.noise1.Rastrigin)

        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 2.1911, df = 1, p-value = 0.1388

> kruskal.test(Y ~ ocba, data = df.spot.noise1.Rosenbrock)

        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 1.2974, df = 1, p-value = 0.2547

> kruskal.test(Y ~ ocba, data = df.spot.noise1.SixHump)

        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 7.5931, df = 1, p-value = 0.005859
```

Rosenbrock and Rastrigin do not show a significant improvement. However, OCBA does not lead to a performance degression in any case.

Similar results were obtained with the increased noise level, i.e., $\sigma_\epsilon = 10.0$.

```
> kruskal.test(Y ~ ocba, data = df.spot.noise10.Branin)

        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 5.37, df = 1, p-value = 0.02049

> kruskal.test(Y ~ ocba, data = df.spot.noise10.MexicanHat)

        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 7.0398, df = 1, p-value = 0.007972
```

```
> kruskal.test(Y ~ ocba, data = df.spot.noise10.Rastrigin)

        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 5.2376, df = 1, p-value = 0.02210

> kruskal.test(Y ~ ocba, data = df.spot.noise10.Rosenbrock)

        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 2.3813, df = 1, p-value = 0.1228

> kruskal.test(Y ~ ocba, data = df.spot.noise10.SixHump)

        Kruskal-Wallis rank sum test

data:  Y by ocba
Kruskal-Wallis chi-squared = 7.4491, df = 1, p-value = 0.006347
```

The overall analysis shows that SPOT can be improved by integrating OCBA. Table 5 summarizes theses results.

5.1.4 Scientific Relevance

Results from this experimental study are statistically significant. However, the reader might consider the small set of objective functions. Hence, these results can be seen as an indicator. Further experiments are necessary. Taking these preliminaries into consideration, we recommend using OCBA.

5.2 Q-2: How do Tree-based Models perform compared to Kriging models?

A recent study demonstrated that random forest performs surprisingly well compared to other meta models in the SPOT framework [4]. A stochastic search algorithm was optimized in this study. Can this result be generalized for other settings? Here, we will consider classical test functions with varying noise strengths.

5.2.1 Research Question

Will tree-based meta models outperform Kriging model based approaches?

5.2.2 Experimental Setup

The set of five well known test functions as described in Sect. 3 was used for this comparison. As a consequence from Q-1, we will restrict our analysis to OCBA-based approaches.

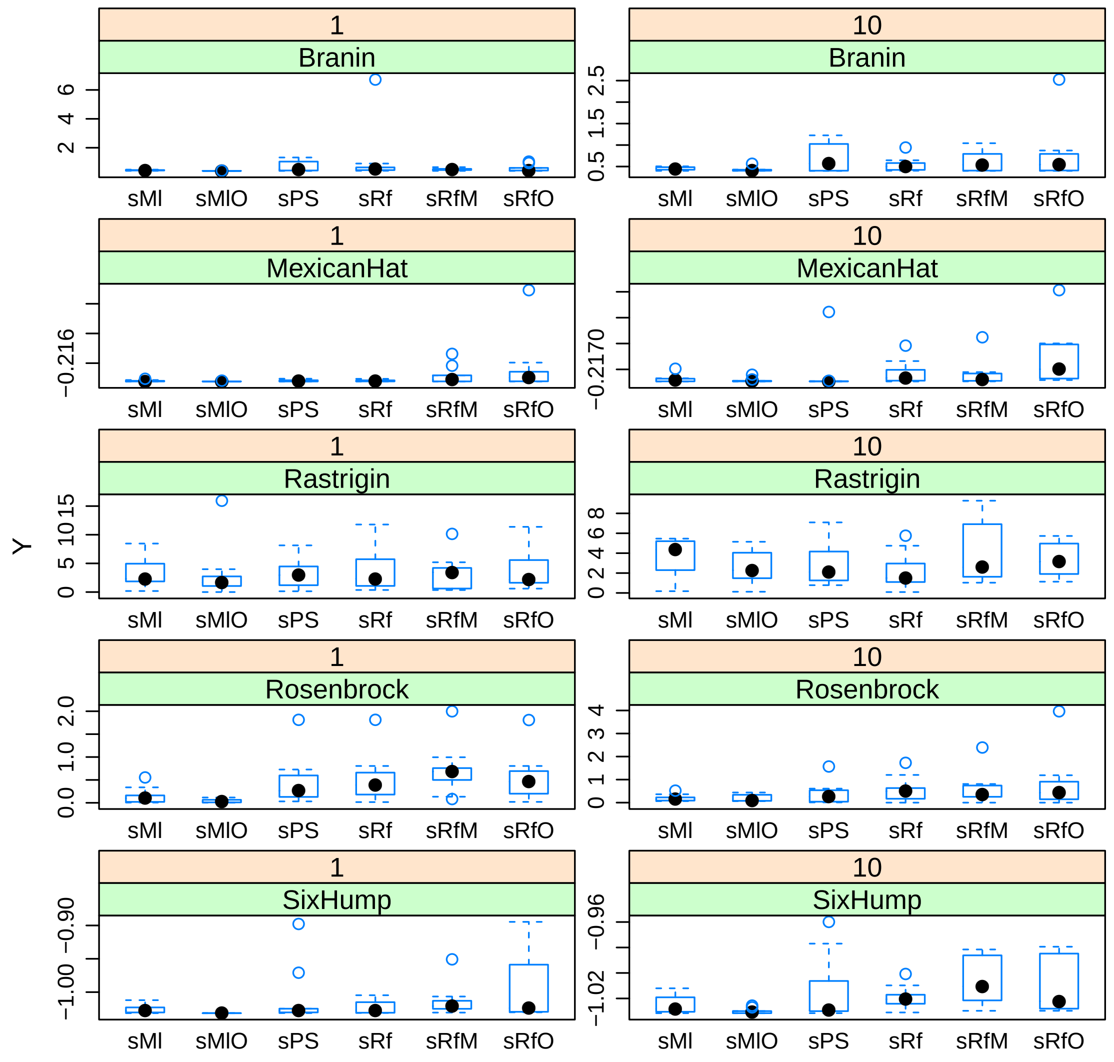

Figure 3: Comparison of SPOT runs with respect to noise level (first row in each panel, 1 or 10) and objective function (second row in each panel). OCBA was used in every run for each of the six SPOT variants, see Table 1

5.2.3 Analysis

Figure 3 presents a graphical overview. These Trellis plots reveal that Kriging-based models, i.e., `sM1` and `sM10`, perform best (acronyms are explained in Table 1). One exception form this rule can be observed for Rastrigin with $\sigma_\epsilon = 10.0$: The Kriging based approaches are outperformed by random forest (`sRf`).

Following the methodology introduced in Sect. 4.2, we will perform a Kruskal-Wallis test first. The six SPOT meta models are compared on every test function and for every noise level separately.

```
> kruskal.test(Y ~ Method, data = df.spot.ocba.Branin.noise1)

        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 23.5189, df = 5, p-value = 0.0002686

> kruskal.test(Y ~ Method, data = df.spot.ocba.MexicanHat.noise1)

        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 18.3345, df = 5, p-value = 0.002555

> kruskal.test(Y ~ Method, data = df.spot.ocba.Rastrigin.noise1)

        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 1.9093, df = 5, p-value = 0.8615

> kruskal.test(Y ~ Method, data = df.spot.ocba.Rosenbrock.noise1)

        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 26.1354, df = 5, p-value = 8.4e-05

> kruskal.test(Y ~ Method, data = df.spot.ocba.SixHump.noise1)

        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 27.7019, df = 5, p-value = 4.162e-05

> kruskal.test(Y ~ Method, data = df.spot.ocba.Branin.noise10)

        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 5.7041, df = 5, p-value = 0.3361

> kruskal.test(Y ~ Method, data = df.spot.ocba.MexicanHat.noise10)
```

Table 6: Q-2. Results from Kruskal-Wallis Tests, noise levels $\sigma_\epsilon = 1.0$ and $\sigma_\epsilon = 10.0$. The same data as in Fig. 3 were used for these tests

	$\sigma_\epsilon = 1.0$		$\sigma_\epsilon = 10.0$	
Function	p-value	sign.	p-value	sign.
Branin	0.0002686	yes	0.3361	no
Mexican Hat	0.002555	yes	0.0009408	yes
Rastrigin	0.8615	no	0.2724	no
Rosenbrock	8.4e-05	yes	0.2172	no
Six Hump	4.162e-05	yes	0.0006707	yes

```
        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 20.6556, df = 5, p-value = 0.0009408

> kruskal.test(Y ~ Method, data = df.spot.ocba.Rastrigin.noise10)

        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 6.3637, df = 5, p-value = 0.2724

> kruskal.test(Y ~ Method, data = df.spot.ocba.Rosenbrock.noise10)

        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 7.0468, df = 5, p-value = 0.2172

> kruskal.test(Y ~ Method, data = df.spot.ocba.SixHump.noise10)

        Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 21.4334, df = 5, p-value = 0.0006707
```

A Kruskal-Wallis rank sum test revealed a significant effect of the meta model on performance Y for Branin, Six Hump, Mexican Hat, and Rosenbrock, but no significant difference for Rastrigin. This might be an explanation for the seemingly better performance of the random forest models—it might be only an artefact caused by noise. The box plots in the first column from Fig. 3 illustrate this result. Now, that we have detected a difference, we are interested which meta model performs best. Equation 2 reveals that a pairwise comparison of the six models results in 15 combinations. To reduce complexity, we decided to split the set of meta models into two subsets. The first subset includes the Kriging-based models, whereas random forest based models can be found in the second subset. In the following, we will determine the best model from each subset. These two models will be compared in a second step.

The two Kriging models, namely `sM1` and `sM10`, will be analyzed first. Based on the methodology from Sect. 4.2, we can detect significant statistical differences between these models.

```
> generateBoxPlot(df.spot.ocba.Branin.noise1.ml, "Branin1Ml")
> generateBoxPlot(df.spot.ocba.MexicanHat.noise1.ml, "MexicanHat1Ml")
> generateBoxPlot(df.spot.ocba.SixHump.noise1.ml, "SixHump1Ml")
> generateBoxPlot(df.spot.ocba.Rastrigin.noise1.ml, "Rastrigin1Ml")
> generateBoxPlot(df.spot.ocba.Rosenbrock.noise1.ml, "Rosenbrock1Ml")
```

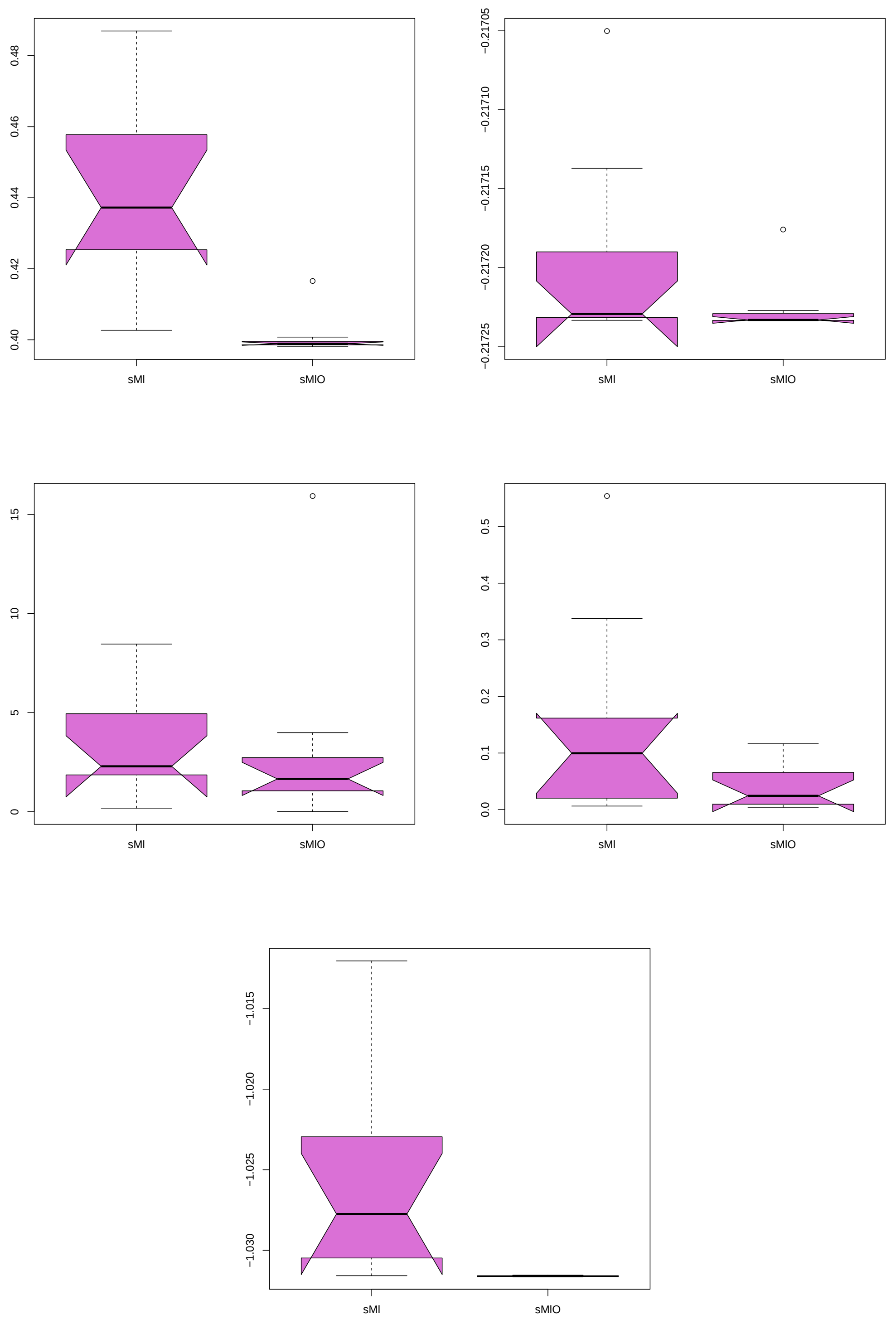

Figure 4: Comparison of Kriging based models. $\sigma_\epsilon = 1.0$. Significant difference

Visual inspection based on EDA (Fig. 3) supports this claim. Therefore, we conclude that sMlO outperforms sMl.

Next, we will consider random forest based approaches, asking the same question: which random forest based model performs best? Results are shown for Branin:

```
> attach(df.spot.ocba.Branin.noise1.rf)
> dtk1 <- DTK.test(Y, Method)
> print(dtk1)
```

```
[[1]]
[1] 0.05

[[2]]
                Diff   Lower CI  Upper CI
sRf-sPS    0.4709695 -1.4915670 2.4335060
sRfM-sPS  -0.2049333 -0.5766559 0.1667892
sRfO-sPS  -0.1338873 -0.5655580 0.2977833
sRfM-sRf  -0.6759028 -2.6058538 1.2540481
sRfO-sRf  -0.6048568 -2.5472452 1.3375315
sRfO-sRfM  0.0710460 -0.1728815 0.3149735
```

```
> detach(df.spot.ocba.Branin.noise1.rf)
```

```
> generateDkPlot(df.spot.ocba.Branin.noise1.rf, "Branin1Rf")
> generateDkPlot(df.spot.ocba.MexicanHat.noise1.rf, "MexicanHat1Rf")
> generateDkPlot(df.spot.ocba.Rastrigin.noise1.rf, "Rastrigin1Rf")
> generateDkPlot(df.spot.ocba.Rosenbrock.noise1.rf, "Rosenbrock1Rf")
> generateDkPlot(df.spot.ocba.SixHump.noise1.rf, "SixHump1Rf")
```

Figure 5 illustrates that there are no significant differences in the mean performance of the random forest based models. The statistical analysis reveals that there is no difference in the performance of the random forest based meta models. Therefore, the standard random forest (sRf) was chosen for the following comparisons. Finally, we have to compare sMlO with sRf. The analysis shows that the Kriging based model clearly outperforms random forest.

5.2.4 Scientific Relevance

Results from a recent study could not be transferred to our test set [4]. The Kriging based approach outperforms random forest model approaches in our scenario. However, [4] did not use OCBA. The combination of Kriging with OCBA might be the reason for this performance improvement.

5.3 Q-3: How does SPOT perform on standard test set compared to classical algorithms?

First, we will take a global view on the data, divided by algorithm type. Figure 6 indicates that SPOT based algorithms show at least a competitive performance compared to their classical counterparts. Results from answering research questions Q-1 and Q-2 lead to the conclusion that SPOT performs best with OCBA and sMlO (Kriging).

5.3.1 Research Question

Does SPOT show a competitive performance compared to standard optimization algorithms?

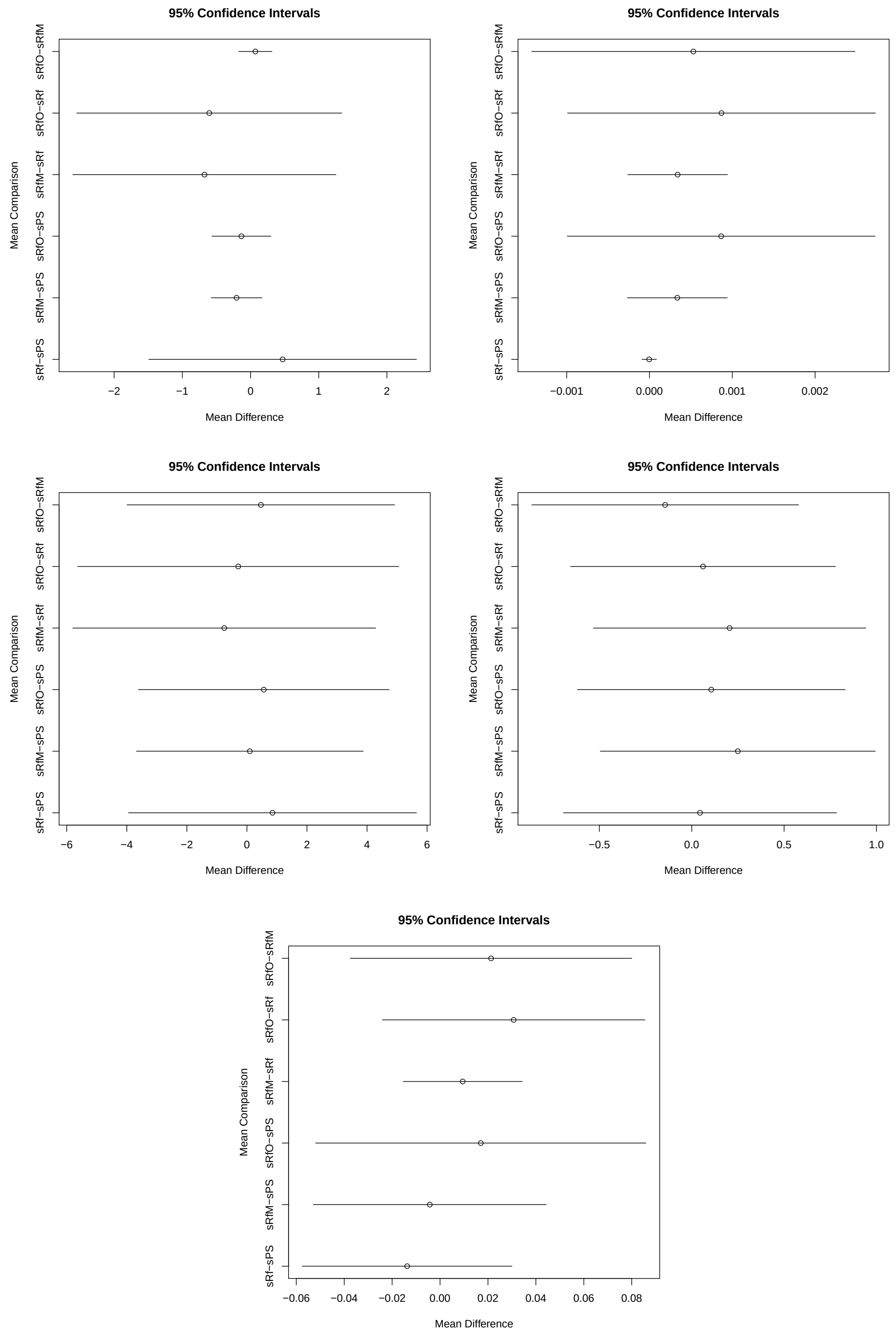

Figure 5: Comparison of random forest based models. $\sigma_\epsilon = 1.0$. No significant difference

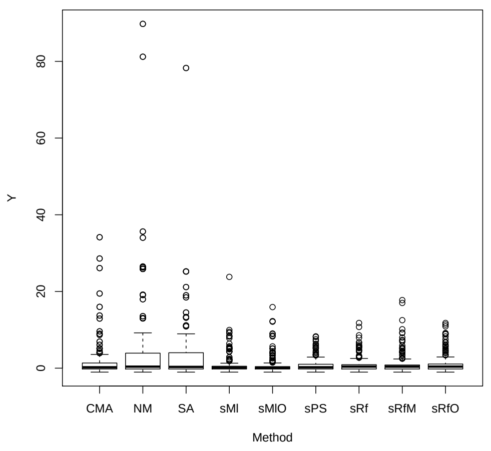

Figure 6: Results from a bird's eye perspective indicate competitive performances of the SPOT algorithms. Smaller values are better

5.3.2 Experimental Setup

The SPOT variant with OCBA and Kriging (`sMlO`) will be used in this comparison with standard algorithms from optimization. To improve reproducibility of our results, we have chosen two general-purpose optimization algorithms from R's `optim` package. Since `cmaes` is considered as state-of-the-art optimizer by several authors, an R implementation was included into our algorithm portfolio.[14] The following algorithms were chosen for our comparison:

1. Nelder-Mead Simplex (NM). This method is an implementation of that of Nelder and Mead [17].
2. Simulated Annealing (SANN). This is a variant of simulated annealing provided in Belisle [8]. Simulated-annealing belongs to the class of stochastic global optimization methods. It uses only function values but is relatively slow. It will also work for non-differentiable functions. The implementation at hand uses the Metropolis function for the acceptance probability.
3. Covariance Matrix Adaptation Evolution Strategy (CMA-ES). Mersmann's and Arnu's R implementation, which is available as an R package via CRAN, was used in our study. This variant is based on [14].

5.3.3 Analysis

First, a visual inspection is performed. Trellis plots provide a comprehensive overview and indicate that SPOT's performance is at least competitive. Next, we generate a normal QQ plot of the values in Y. As can be seen from Fig. 7, data are non normally distributed. A Kruskal-Wallis test (p-value = 0.001433) indicates that there is a difference in means.

```
          Kruskal-Wallis rank sum test

data:  Y by Method
Kruskal-Wallis chi-squared = 15.5035, df = 3, p-value = 0.001433
```

Because this first test is positive, we can analyze *which* means differ using Dunnett's Pairwise Multiple Comparison Test. This is a pairwise multiple comparison test for mean differences with unequal sample sizes and no assumption of equal population variances. Results from this test are shown in Fig. 8. Consider the first line (95 % confidence interval) which is labeled "sMlO-SA". Since this interval does not contain 0, it indicates that SPOT-OCBA (MLEGP plus optimization used as a meta model) outperforms simulated annealing. A numerical summary is given in Table 7. Similar results were obtained in the experiments with Six Hump, Branin, Rosenbrock, and Mexican hat. None of the classical algorithms outperformed SPOT.

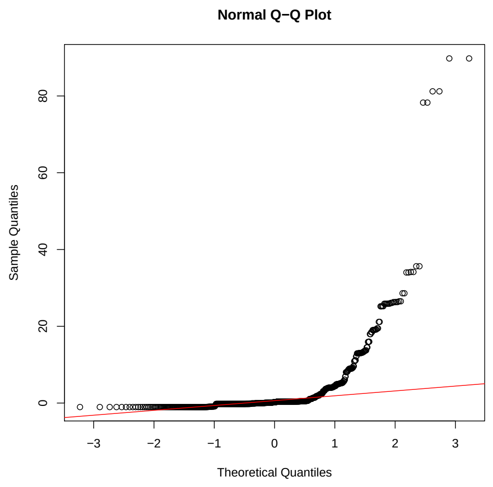

Figure 7: Quantile-quantile (QQ) plot. Since data coming from a normal distribution are expected to obtain a straight line, data from our experiments are non normal

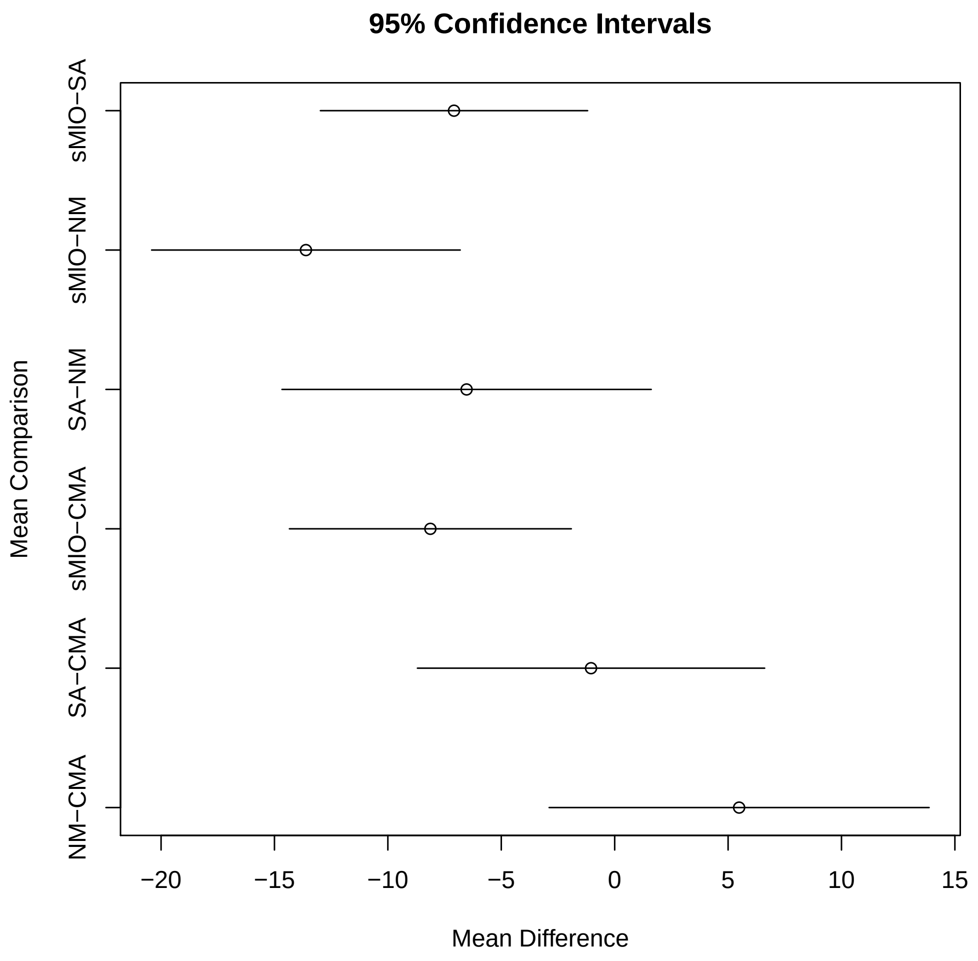

Figure 8: Dunnett's Pairwise Multiple Comparison Test. Rastrigin Function, noise level $\sigma_\epsilon = 1.0$. Numerical results are shown in Table 7

Table 7: Q-3. Results from Dunnett's Pairwise Multiple Comparison Test (significance level = 0.05), Rastrigin function, noise level $\sigma_\epsilon = 1.0$. The same data as in Fig. 11 were used for these tests

Algorithms	Diff	Lower CI	Upper CI
NM-CMA	5.486697	-2.890826	13.864219
SA-CMA	-1.042051	-8.697670	6.613568
sMlO-CMA	-8.126343	-14.341739	-1.910948
SA-NM	-6.528748	-14.669059	1.611564
sMlO-NM	-13.613040	-20.416518	-6.809562
sMlO-SA	-7.084292	-12.976061	-1.192524

```
> attach(df.spotlessMlO.noise1.Branin)
> dtk1 <- DTK.test(Y, Method)
> print(dtk1)

[[1]]
[1] 0.05

[[2]]
                Diff      Lower CI    Upper CI
NM-CMA   -0.213466218 -5.504109e-01  0.12347846
SA-CMA    1.475710466  1.356843e-01  2.81573668
sMlO-CMA -0.207235944 -5.442388e-01  0.12976690
SA-NM     1.689176684  3.922038e-01  2.98614952
sMlO-NM   0.006230274 -3.052102e-05  0.01249107
sMlO-SA  -1.682946410 -2.979934e+00 -0.38595846

> detach(df.spotlessMlO.noise1.Branin)
```

```
> generateDkPlot(df.spotlessMlO.noise1.Branin, "Branin1spotlessMlO")
> generateDkPlot(df.spotlessMlO.noise1.MexicanHat, "MexicanHat1spotlessMlO")
> generateDkPlot(df.spotlessMlO.noise1.Rastrigin, "Rastrigin1spotlessMlO")
> generateDkPlot(df.spotlessMlO.noise1.Rosenbrock, "Rosenbrock1spotlessMlO")
> generateDkPlot(df.spotlessMlO.noise1.SixHump, "SixHump1spotlessMlO")
```

Figure 9 illustrates that there are significant differences in the mean performance of the classical optimization algorithms and SPOT.

```
> attach(df.spotlessMlO.noise1.Branin)
> dtk1 <- DTK.test(Y, Method)
> print(dtk1)

[[1]]
[1] 0.05

[[2]]
                Diff      Lower CI    Upper CI
NM-CMA   -0.213466218 -5.504109e-01  0.12347846
SA-CMA    1.475710466  1.356843e-01  2.81573668
sMlO-CMA -0.207235944 -5.442388e-01  0.12976690
SA-NM     1.689176684  3.922038e-01  2.98614952
sMlO-NM   0.006230274 -3.052102e-05  0.01249107
sMlO-SA  -1.682946410 -2.979934e+00 -0.38595846

> detach(df.spotlessMlO.noise1.Branin)
```

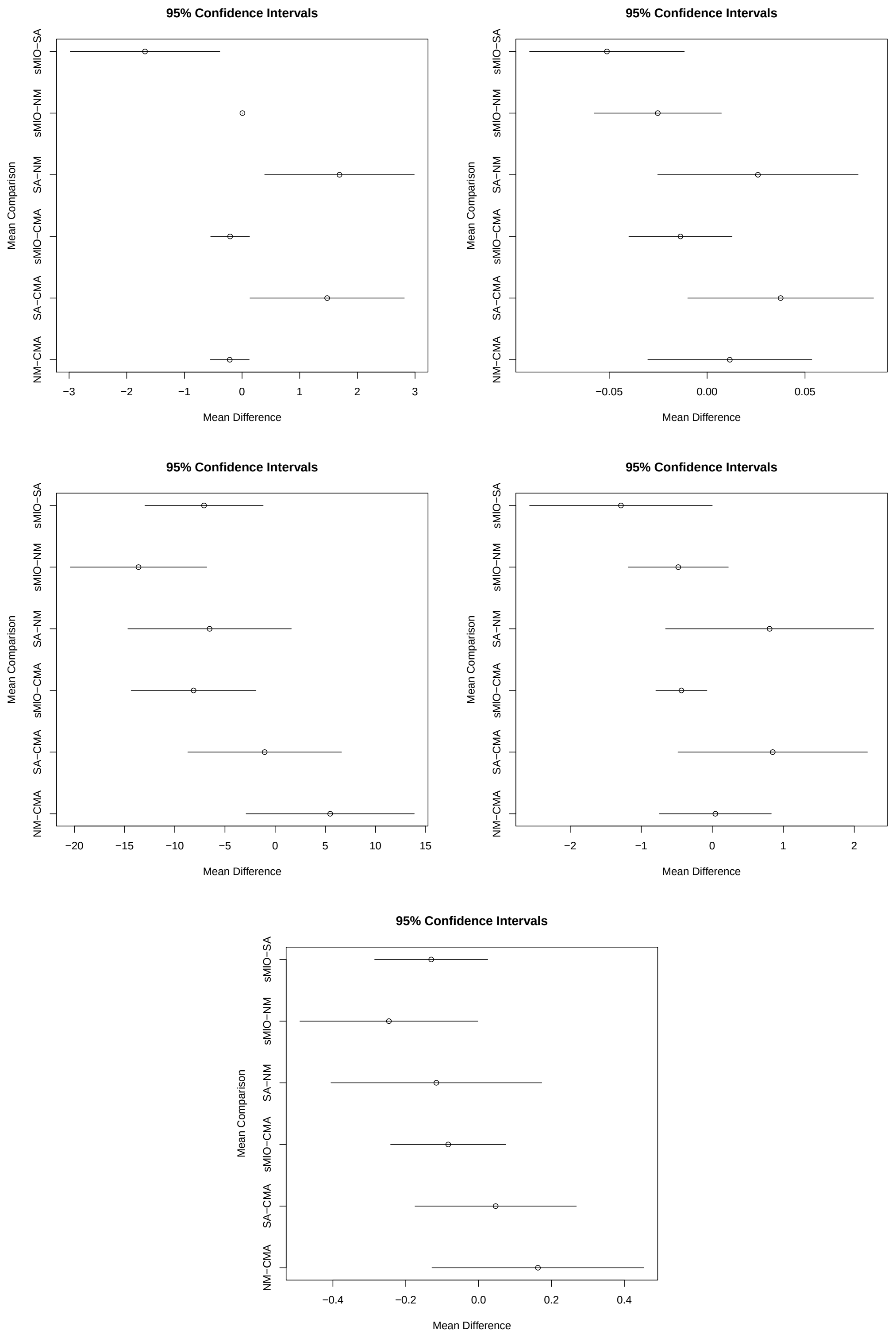

Figure 9: Comparison of classical optimization algorithms and SPOT. $\sigma_\epsilon = 1.0$. Significant differences can be detected

```
> generateDkPlot(df.spotlessMl0.noise10.Branin, "Branin10spotlessMl0")
> generateDkPlot(df.spotlessMl0.noise10.MexicanHat, "MexicanHat10spotlessMl0")
> generateDkPlot(df.spotlessMl0.noise10.Rastrigin, "Rastrigin10spotlessMl0")
> generateDkPlot(df.spotlessMl0.noise10.Rosenbrock, "Rosenbrock10spotlessMl0")
> generateDkPlot(df.spotlessMl0.noise10.SixHump, "SixHump10spotlessMl0")
```

Figure 10 illustrates that there are significant differences in the mean performance of the classical optimization algorithms and SPOT.

Results from Dunnett's Pairwise Multiple Comparison Test are in correspondence with Figure 11: SPOT outperforms the three classical optimization algorithms from this study.

5.3.4 Scientific Relevance

Summarizing, we can conclude that SPOT with OCBA and Kriging outperforms the other approaches. Figure 11 illustrates this result. SPOT with Kriging (`sMl0`) shows a robust behavior (only a few outliers). Results from this study can be seen as first indicators. Further studies are necessary.

6 Summary

The paper investigates some interesting questions concerning the parametrization of the sequential parameter optimization toolbox SPOT. To this end, we applied SPOT to noisy mathematical test functions to mimic the stochastic behavior of its natural application area, the parametrization of stochastic optimization methods.

First investigations focused on the budget allocation for new design points within SPOT. Our results show that incorporating OCBA really improves the quality of results. This is of course accounted to OCBA's way to distribute the budgets to different design points incorporating sample means as well as variances of the design points (approximated) quality.

Secondly, we examined the internal model that is used within SPOT to suggest new design points. Random forest models are compared to Gaussian process models (Kriging). The results received indicate an advantage of the latter models, however, the computational effort of these is higher than for the standard models. Based on these results we concentrated on the way to incorporate Kriging models and found out that an optimization on the models using simple gradient based techniques are beneficial.

Finally, we compared the resulting best SPOT variant to classical optimization techniques. Here, SPOT with OCBA, Kriging, and model-based optimization was able to outperform the other approaches in the study, i.e., a Nelder-Mead simplex method, the simulated annealing algorithm, and the CMA-ES. Note that results from this study shed some light on the behavior of meta models in the tuning process. However, we do not claim that these results are correct in every situation. Further studies, which include different design point generators as well, are necessary. We are going to expand our experiments to more test functions and, in particular, test functions with a higher dimensional search space.

References

[1] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation—The New Experimentalism.* Natural Computing Series. Springer, Berlin, Heidelberg, New York, 2006.

[2] T. Bartz-Beielstein. SPOT: An R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. CIOP Technical Report 05/10, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, Jun 2010.

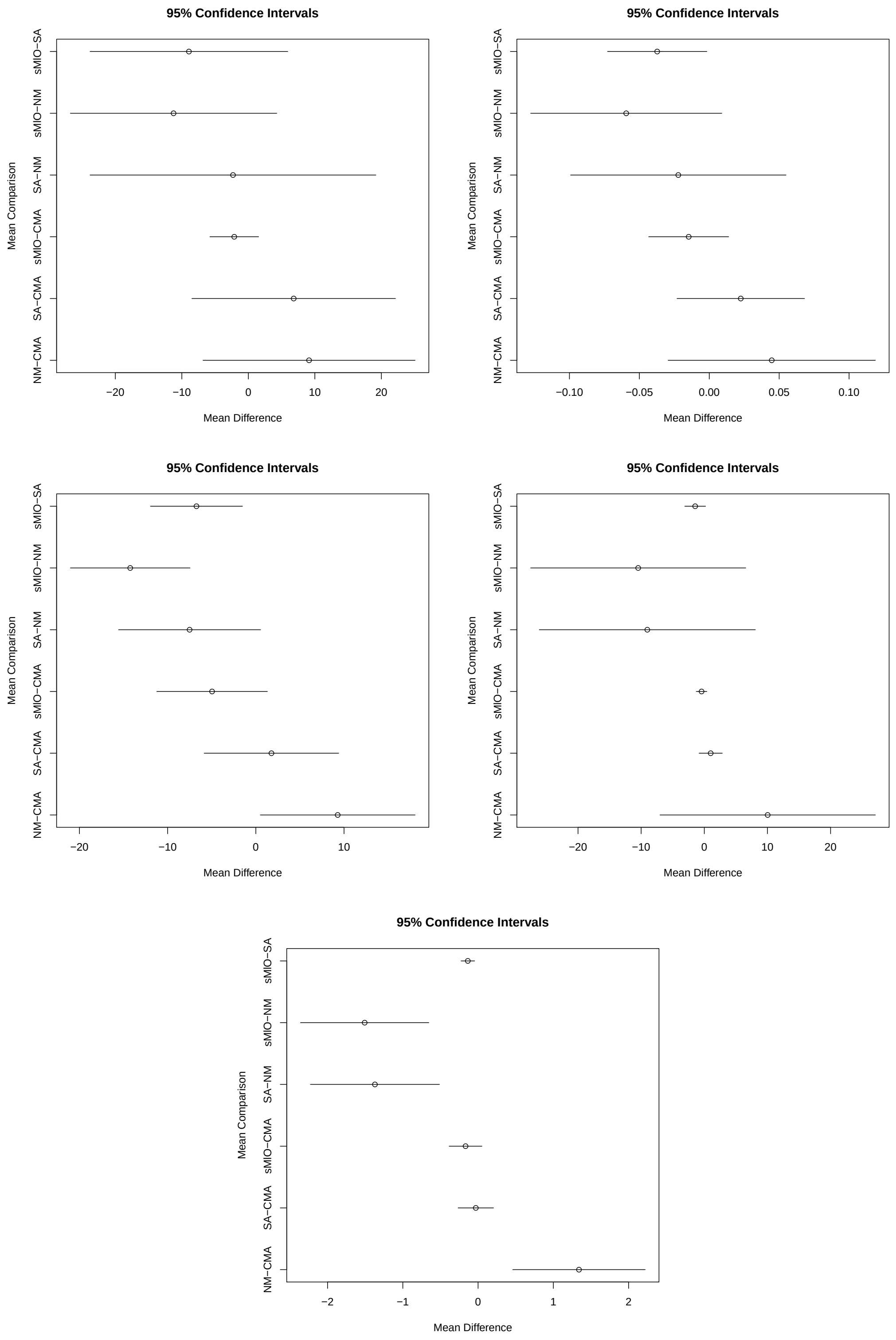

Figure 10: Comparison of classical optimization algorithms and SPOT. $\sigma_\epsilon = 10.0$. Significant differences can be detected

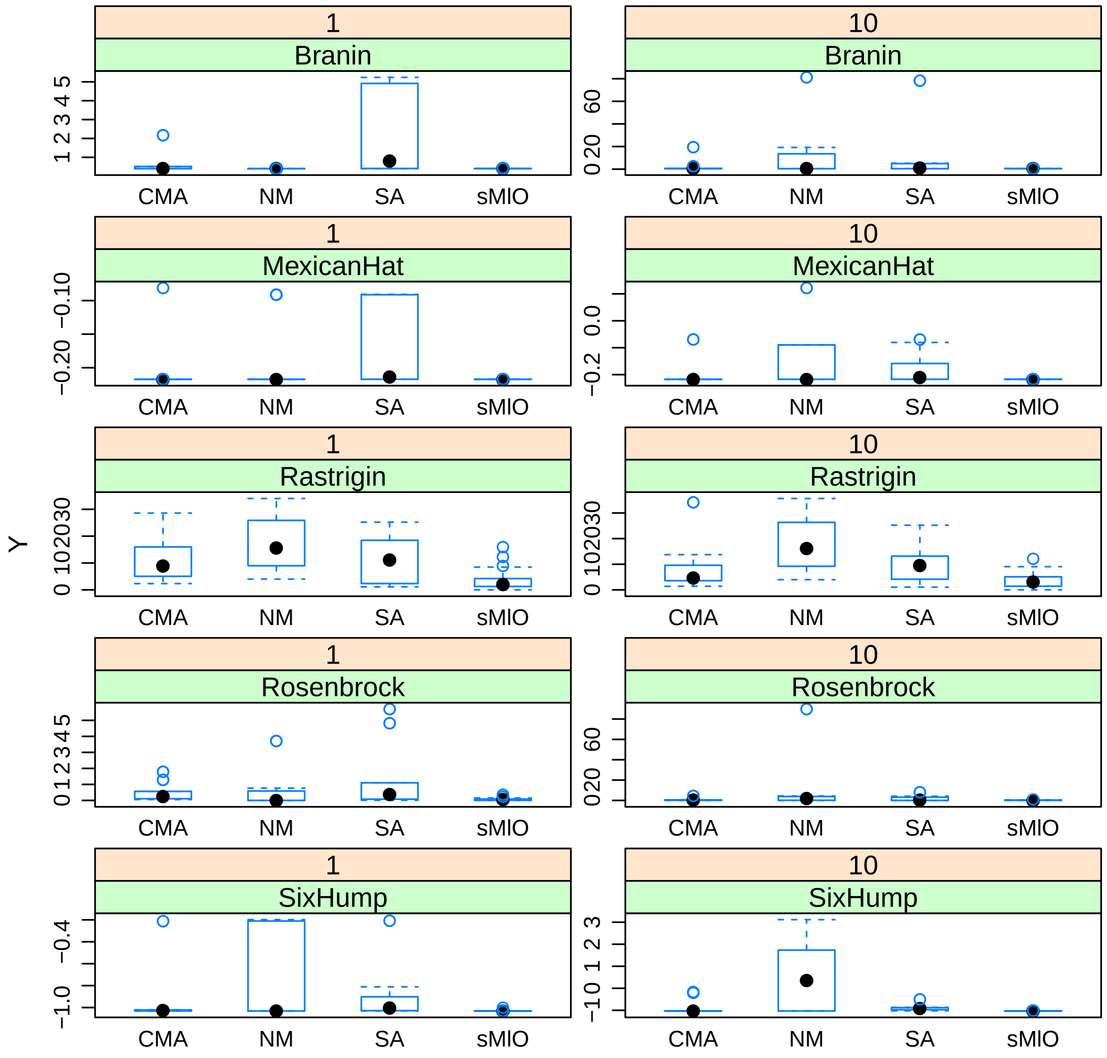

Figure 11: Comparison of classical optimization algorithms with SPOT

[3] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors. *Experimental Methods for the Analysis of Optimization Algorithms.* Springer, Berlin, Heidelberg, New York, 2010.

[4] T. Bartz-Beielstein, O. Flasch, P. Koch, and W. Konen. SPOT: A toolbox for interactive and automatic tuning in the R environment. In F. Hoffmann and E. Hüllermeier, editors, *Proceedings 20. Workshop Computational Intelligence*, pages 264–273. Universitätsverlag Karlsruhe, 2010.

[5] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. The sequential parameter optimization toolbox. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–360. Springer, Berlin, Heidelberg, New York, 2010.

[6] T. Bartz-Beielstein, K. E. Parsopoulos, and M. N. Vrahatis. Design and analysis of optimization algorithms using computational statistics. *Applied Numerical Analysis and Computational Mathematics (ANACM)*, 1(2):413–433, 2004.

[7] T. Bartz-Beielstein and M. Preuss. The future of experimental research. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 17–46. Springer, Berlin, Heidelberg, New York, 2010.

[8] C. J. P. Belisle. Convergence theorems for a class of simulated annealing algorithms. *Journal Applied Probability*, 29:885–895, 1992.

[9] L. Breiman. Random forests. *Machine Learning*, 45(1):5 –32, 2001.

[10] J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey. *Graphical Methods for Data Analysis.* Wadsworth, Belmont CA, 1983.

[11] C.-H. Chen and L. H. Lee. *Stochastic simulation optimization.* World Scientific, 2011.

[12] G. M. Dancik and K. S. Dorman. mlegp. *Bioinformatics*, 24(17):1966–1967, 2008.

[13] C. Dunnett. Pairwise multiple comparisons in the unequal variance case. *Journal of the American Statistical Association*, 75:796–800, 1980.

[14] N. Hansen. The CMA evolution strategy: a comparing review. In J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

[15] M. Hollander and D. A. Wolfe. *Nonparametric Statistical Methods.* John Wiley & Sons, 1973.

[16] C. W. G. Lasarczyk. *Genetische Programmierung einer algorithmischen Chemie.* PhD thesis, Technische Universität Dortmund, 2007.

[17] J. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[18] F. Pukelsheim. *Optimal Design of Experiments.* Wiley, New York NY, 1993.

[19] T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments.* Springer, Berlin, Heidelberg, New York, 2003.

[20] J. Tukey. The philosophy of multiple comparisons. *Statistical Science*, 6:100–116, 1991.