SPOT: A Toolbox for Interactive and Automatic Tuning in the R Environment

Thomas Bartz-Beielstein, Oliver Flasch, Patrick Koch, and Wolfgang Konen

Fakultät für Informatik und Ingenieurwissenschaften, Fachhochschule Köln
E-Mail: {thomas.bartz-beielstein | oliver.flasch | patrick.koch | wolfgang.konen}@fh-koeln.de

Abstract

Sequential parameter optimization is a heuristic that combines classical and modern statistical techniques to improve the performance of search algorithms. It includes methods for tuning based on classical regression and analysis of variance techniques; tree-based models such as CART and random forest; Gaussian process models (Kriging), and combinations of different meta-modeling approaches. The suitability of these different meta models for parameter tuning is analyzed in this article. Automated and interactive approaches are compared..

1 Introduction

Sequential parameter optimization (SPO) is a heuristic that combines classical and modern statistical techniques to improve the performance of search algorithms. Bartz-Beielstein [1] presents an introduction to the state-of-the-art R implementation of SPO, the so-called *sequential parameter optimization toolbox* (SPOT). This article describes important aspects of parameter tuning related to algorithm designs and meta models. A short case study illustrates our considerations. This paper is organized as follows. The sequential parameter optimization is introduced in Section 2. Exogenous parameters of algorithms such as population size define an algorithm design. Algorithm design used in our experiments are described in Sect. 3. SPOT uses several prediction models (meta models) in order to predict the performance of an algorithm. Meta models are introduced in Sect. 4. In order to compare different meta models, a case study is presented in Section 5. Tuning procedures considered so far use SPOT in an automated manner. Section 6 presents an interactive approach, which is related to classical *response surface methodology* (`rsm`). Section 7 presents a summary and an outlook.

2 Sequential Parameter Optimization

Sequential parameter optimization 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 the meta models are used to select new design points. The meta model is refined to improve knowledge about the search space. SPOT provides tools to cope with noise, which typically occurs when randomized search

Algorithm 1: Sequential parameter optimization toolbox (SPOT)

```
// phase 1, building the model:
let $A$ be the tuned algorithm;
// design considerations necessary:
generate an initial population $X = \{\bar{x}^1, \ldots, \bar{x}^m\}$ of $m$ parameter vectors;
let $k = k_0$ be the initial number of tests for determining estimated utilities;
foreach $\bar{x} \in X$ do
    run $A$ with $\bar{x}$ $k$ times to determine the estimated utility $y$ of $\bar{x}$;
end
// phase 2, using and improving the model:
while termination criterion not true do
    let $\bar{a}$ denote the parameter vector from $X$ with best estimated utility;
    let $k$ the number of repeats already computed for $\bar{a}$;
    // model considerations necessary:
    build meta model $f$ based on $X$ and $\{y^1, \ldots, y^{|X|}\}$;
    // 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 predicted utility $f(\bar{x})$ of $\bar{x}$;
    end
    select set $X''$ of $d$ parameter vectors from $X'$ with best predicted utility ($d \ll l$);
    run $A$ with $\bar{a}$ once and recalculate its estimated utility using all $k+1$ test
        results; // (improve confidence)
    let $k = k + 1$;
    run $A$ $k$ times with each $\bar{x} \in X''$ to determine the estimated utility $\bar{x}$;
    extend the population by $X = X \cup X''$;
end
```

heuristics are run. It guarantees comparable confidence for search points. Users can collect information to learn from this tuning process, e.g., by applying exploratory data analysis. Last, but not least, SPOT provides mechanisms both for interactive and automated tuning. SPOT is described in [2, 3]. 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 downloaded from the author's WWW page.[2] As can be seen from Algorithm 1, meta models are used to determine new algorithm designs. This article compares several standard and non-standard meta-modeling approaches.

3 Algorithm Designs

There is a strong interaction between meta models and algorithm designs, because the optimality of an algorithm design depends on the meta model [4, 5].

SPOT generates new design points during the init and during the sequential step. Many algorithm designs generators are available in R, see, e.g., the *CRAN Task*

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

[2]`http://www.gm.fh-koeln.de/campus/personen/lehrende/thomas.bartz-beielstein/00489/`

Table 1: SPOT initial design plugins

Type	Name of the SPOT plugin
Fractional factorial design (Resolution III)	`spotCreateDesignDoe3`
Factorial design	`spotCreateDesignBasicDoe`
Latin hypercube	`spotCreateDesignLhd`
Latin hypercube	`spotCreateDesignLhs`

Table 2: SPOT meta models

Type	Name of the SPOT plugin	R package
Linear model	`spotPredictLm`	base
Response surface methodology	`spotPredictLmOptim`	`rsm`
Regression trees	`spotPredictTree`	`rpart`
Random forest	`spotPredictRandomForest`	`random forest`
Gaussian processes (Kriging)	`spotPredictMlegp`	`mlegp`
Tree based Gaussian processes	`spotPredictTgp`	`tgp`

View: Design of Experiments (DoE) & Analysis of Experimental Data.[3] This is one of the main reasons why SPOT is implemented in R. The user can use state of the art design generators for tuning his algorithm. Or, he can write his own design generator and use it as a *plugin* for SPOT. The default SPOT installation contains several design plugins (and further design plugins will be added in forthcoming versions). Table 1 summarizes design plugins from the current SPOT version (0.1.888).[4]

A *Latin hypercube design* (LHD) was chosen as the default initial design in Sect. 5, because it is easy to implement and understand. Section 6 uses fractional-factorial designs, because classical RSM techniques are used. 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.

4 Meta Models

SPOT processes data sequentially, i.e., starting from a small initial design, further design points (parameter settings for the algorithm) are generated using a meta model. Many meta models are available in R. The user can use state of the art meta models for tuning his algorithm. Or, he can write his own meta model and use it as a plugin for SPOT. The default SPOT installation contains several meta models (and further meta models will be added in forthcoming versions). Table 2 summarizes meta models from the current SPOT version (0.1.888).

The R implementation of `randomForest` was chosen as SPOT's default meta model, because it is quite robust and can handle categorical and numerical values.

[3]`http://cran.r-project.org/web/views/ExperimentalDesign.html`

[4]The command `spotVersion()` displays the actual version of your local SPOT package.

Note, these meta models and design plugins should be considered as templates. They were implemented in order to demonstrate how the interfaces should look like. We strongly recommend an adaptation of these plugins to your specific needs, if real-world tuning tasks are performed. However, we will use these simple and generic plugins for the following case study.

5 Case Study: Tuning Simulated Annealing

5.1 Goals of this Study

The following case study is devoted to the question: "Which meta models can be recommended for tuning algorithms?" The experimental setup is generic, i.e., we have chosen a set of meta models from statistics (linear models, tree based models, Gaussian processes or Kriging) which were used for tuning an algorithm (stochastic search heuristic), i.e., simulated annealing.

5.2 Simulated Annealing

Simulated annealing (SANN) belongs to the class of stochastic global optimization methods. The R implementation, which was investigated in our study, uses the Metropolis function for the acceptance probability. It is a variant of the simulated annealing algorithm given in [6]. SANN uses only function values but is relatively slow. It will also work for non-differentiable functions. By default the next candidate point is generated from a Gaussian Markov kernel with scale proportional to the actual temperature. Temperatures are decreased according to the logarithmic cooling schedule as given in [6]; specifically, the temperature is set to $\texttt{temp}/\log(((t-1)/\texttt{tmax}) \times \texttt{tmax} + \exp(1))$, where t is the current iteration step and `temp` and `tmax` are specifiable via control. SANN is not a general-purpose method but can be very useful in getting to a good value on a very rough surface.

SANN uses two design variables, which were tuned during our study:

`temp` is the starting temperature for the cooling schedule. Defaults to 10.

`tmax` is the number of function evaluations at each temperature. Defaults to 10.

The interval from 1 to 50 was chosen as the *region of interest* (ROI) for both design variables in our experiments. The total number of function evaluations (of the Branin function, see Sect. 5.3) was set to $\texttt{maxit} = 250$ for all experiments. The starting point, i.e., the initial value for the parameters to be optimized over, was $\vec{x}_0 = (10, 10)$.

5.3 Description of the Objective Function

The Branin function

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

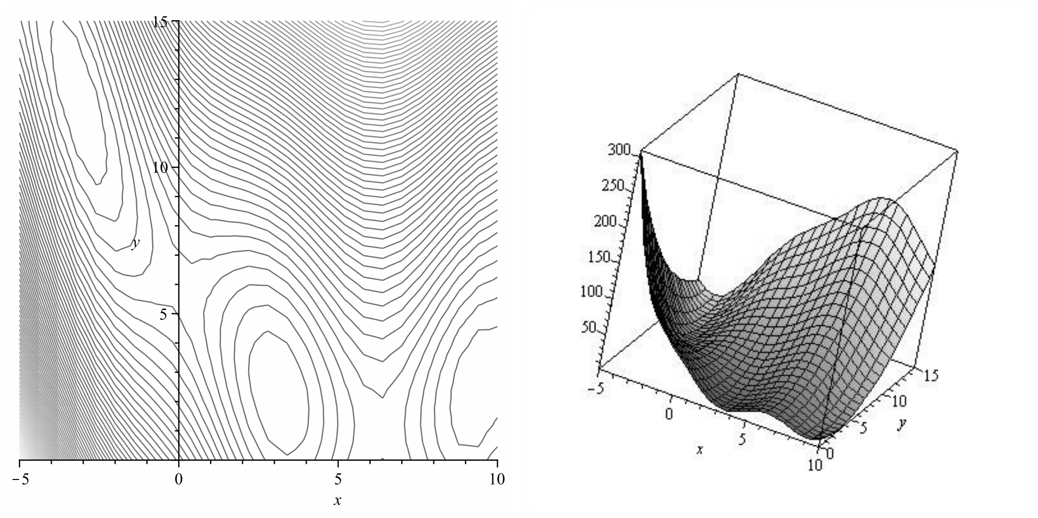

Figure 1: Plots of the Branin function. The contour plot (*left*) shows the location of the three global minima

with

$$x_1 \in [-5, 10] \text{ and } x_2 \in [0, 15]. \quad (1)$$

was chosen as a test function, because it is well-known in the global optimization community, so results are comparable. 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 $y^* = f(\vec{x}_i^*) = 0.3979$, $(i = 1, 2, 3)$, see Fig. 1.

5.4 Results from Default and Random Settings

As a baseline for our experiments, we run SANN one hundred times—first, with default parameters (`tmax` = `temp` = 10), and second, with randomly chosen parameter values from the interval $[1, 50]$.[5] These experiments were performed to quantify the benefit of tuning for our experiments. Results from theses two experiments are shown in the first and second result row from Tab. 3. SANN was not able to determine the optimal function value with these settings. Now that the baseline results are available, we can examine SANN's tunability.

5.5 SPOT Setup

The SPOT based tuning procedure uses the following parameter settings: SPOT uses a budget of one hundred algorithm runs (`auto.loop.nevals = 100`) and an initial design size of ten (`init.design.size = 10`). Each initial design point was evaluated twice (`init.design.repeats = 2`). A Latin hypercube design was chosen as the initial design. The interval from 1 to 50 was chosen as the ROI for both design variables, i.e., `temp` and `tmax`, respectively, in our experiments.

[5]SPOT can generate one hundred randomly chosen design points of the SANN by using the following setting in the CONF file: `init.design.size = 100` and `init.design.repeats = 1`.

Table 3: SANN results. Results from $n = 100$ repeats. Smaller values are better. The optimal function value is $y^* = 0.3979$

Model	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Default	0.3982	0.4037	0.4130	0.8281	0.5032	6.1120
Random	0.3988	0.5326	1.2160	2.0720	2.9820	8.8800
Tree (`rpart`)	**0.3979**	0.4001	0.4044	0.4184	0.4106	0.8576
RF	**0.3979**	**0.3987**	0.4000	**0.4010**	**0.4022**	**0.4184**
lm	0.3981	0.4034	0.4077	0.5325	0.4789	4.1570
`mlegp`	0.3980	0.4022	0.4089	0.4691	0.4349	2.7180
`tgp`	**0.3979**	0.3991	0.4002	0.4030	0.4034	0.4530
`rsm` + `rpart`	**0.3979**	**0.3987**	**0.3997**	0.4014	0.4027	0.4203

5.6 Tree-based Parameter Tuning

The first tuning procedure uses a (simple) regression tree, which is implemented in SPOT as a plugin (`seq.predictionModel.func = "spotPredictTree"`). This meta model plugin uses R's `rpart` package. The function `rpart` follows [7] very closely. The final best solution from the SPOT tuning run reads `temp` = 3 and `tmax` = 31.

Finally, one hundred SANN runs were performed with this design point. Results are shown in the third result row in Tab. 3. The tree-tuned SANN algorithm was able to detect the global minimum.

5.7 Random Forest-based Parameter Tuning

Next, a random forest based meta model was used in the SPOT. `randomForest` from the R package `random forest` implements Breiman's algorithm, which is based on Breiman and Cutler's original Fortran code, for classification and regression. It is implemented as a SPOT plugin which can be selected via the command `seq.predictionModel.func = "spotPredictRandomForest"` in SPOT's configuration file. The best algorithm design point determined by random forest is `temp` = 1.116 and `tmax` = 38. Finally, one hundred SANN runs were performed with this design point. Results are shown in the fourth result row in Tab. 3. The SANN algorithm with SPOT tuned parameters was able to detect the global minimum.

5.8 Linear Model-based Parameter Tuning

A standard linear regression model was used as a meta model in the SPOT. It is implemented as a SPOT plugin, which can be selected via the command `seq.predictionModel.func = "spotPredictLm"` in SPOT's configuration file. The best algorithm design point determined by this linear model is `temp` = 5.0664 and `tmax` = 49. Finally, one hundred SANN runs were performed with this design point. Results are shown in the fifth result row in Tab. 3. The SANN algorithm with tuned parameters, which were generated with a linear regression model was not able to detect the global minimum. However, SANN with these tuned algorithm design obtained better function value than the SANN algorithm with standard or the randomized algorithm design points.

5.9 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 [8].

`mlegp` is implemented as a SPOT plugin, which can be selected via the command `seq.predictionModel.func = "spotPredictMlegp"` in SPOT's configuration file. The best algorithm design point determined by `mlegp` is `temp` = 5.3510 and `tmax` = 37. Finally, one hundred SANN runs were performed with this design point. Results are shown in the sixth result row in Tab. 3. The SANN algorithm with `mlegp`-tuned parameters was not able to detect the global minimum. However, SANN with these tuned algorithm design obtained better function value than the SANN algorithm with standard or the randomized algorithm design points.

5.10 Treed Gaussian Process Models

`tgp` is an R package for fully Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian processes with jumps to the limiting linear model [9]. The SPOT plugin to `tgp` can be invoked via `seq.predictionModel.func = "spotPredictTgp"` in SPOT's configuration file. The best algorithm design point determined by `tgp` is `temp` = 1.4692 and `tmax` = 34. The SANN algorithm with SPOT tuned parameters was able to detect the global minimum.

5.11 Comparison of the Meta Model Results

Random forest was able to determine an algorithms design point which reduced the variance between different SANN runs and enables SANN to determine the optimal function value, y^*, or a value close to y^*. The `tgp` tuned SANN shows a similar performance. Random forest is preferred, because `tgp` runs require significantly more CPU time than random forest runs, or, as stated by Gramacy [10]: "Fully Bayesian analyses with MCMC are not the super-speediest of all statistical models."

We used random forest to generate a sensitivity plot.[6] Figure 2 shows sensitivity plots based on results from different meta models used in the tuning process.

All plots show that `temp` affects SANN's performance significantly, whereas `tmax` has nearly no effect at all. Smaller `temp` values improve the algorithms performance.

[6]SPOT provides a report plugin to generate theses plots. The report plugin was written by Wolfgang Konen and can be invoked via `report.func = "spotReportSens"` in SPOT's CONF file.

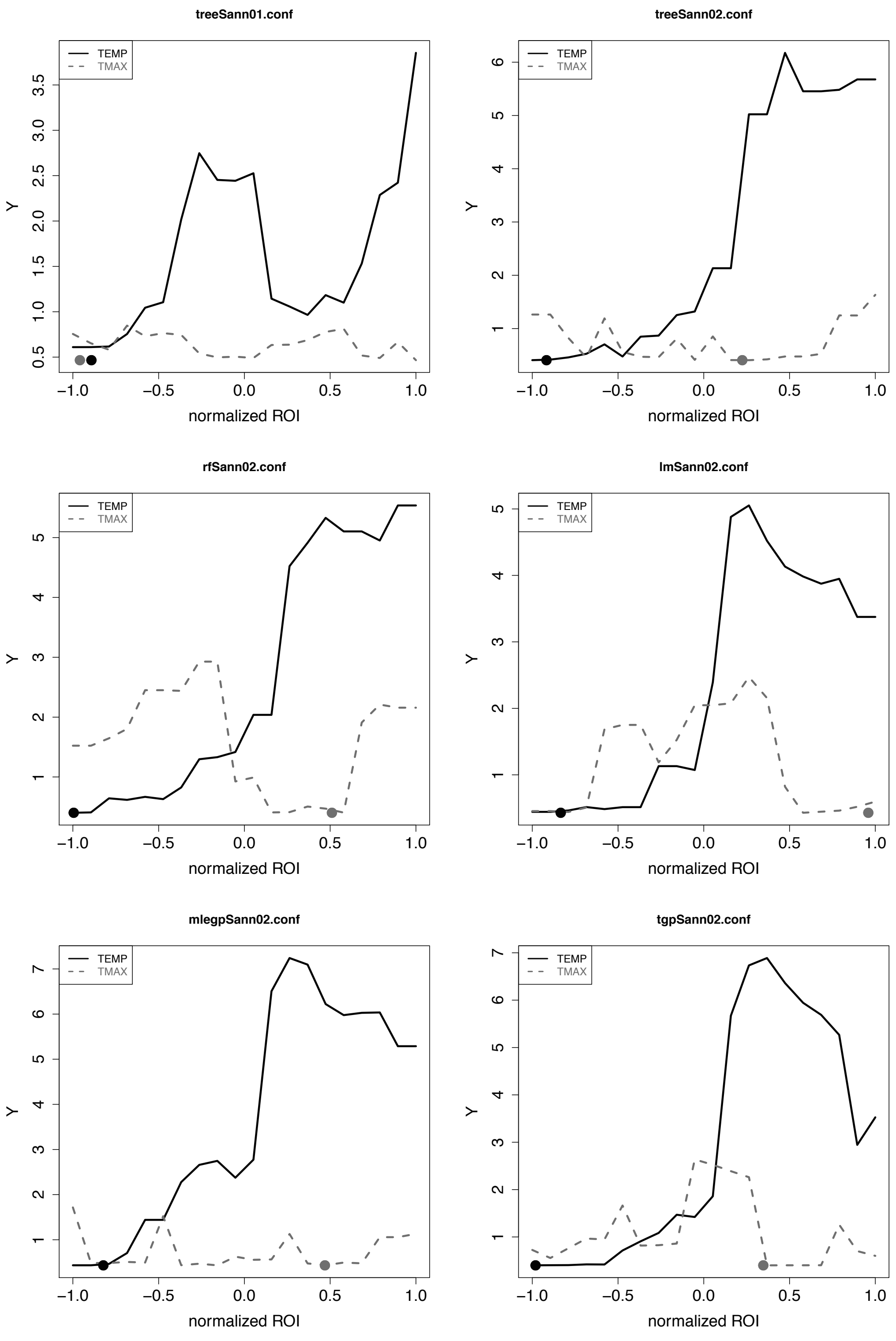

Figure 2: Sensitivity plots (generated by random forest). *Top left:* One hundred random samples, *Top right:* Regression tree, *Middle left:* Random forest. *Middle right:* Linear model, *Bottom left:* Mlegp, and *Bottom right:* Tree Gaussian process models

Dots denote the best design points with respect to the normalized ROI. These sensitivity plots reveal that random forest and `rpart` were able to explore the search space better than their competitors. The results from these sensitivity plots are in accordance with results from the response surface analysis which will be discussed later on, see Fig. 4.

6 Interactive Tuning

In Sect. 5, SPOT was run as an automatic tuner. Steps from the automatic mode can be used in an interactive manner: The user can perform one of the tasks `init`, `seq`, `run`, or `rep` sequentially, see [1]. She can also add and delete design points or modify the region of interest.

In the following, different meta models, namely tree based regression (`rpart`) and response surface modeling (`rsm`), are combined and the region of interest is adapted.

6.1 Designs

A *central composite design* (CCD) was chosen as the starting point of the tuning process. SPOT's `spotCreateDesignFrF2` plugin can be used to generate CCD points. After the first run is finished, we can use SPOT's report facility to analyze results. In the following, we will use RSM. [11] describes an implementation of RSM in `R`. This R package `rsm` has many useful tools for an analysis of the results from the SPOT runs. After evaluating the algorithm in these design points, a second order regression model with interactions is fitted to the data. Functions from the `rsm` package were used by the SPOT plugin `spotPredictLmOptim`. Before meta models are build, data are standardized. Data in the original units are mapped to coded data, i.e., data with values in the interval $[-1, 1]$.

6.2 Response Surface Models and Gradient Information

Based on the number of design points, SPOT automatically determines whether a first-order, two-way interaction, pure quadratic, or second order model can be fitted to the data. The CCD generated by `spotCreateDesignFrF2` allows the fit of an second-order model which can be summarized as shown in Fig. 3.

The response surface analysis (cf. Fig. 3) determines the following stationary point on the response surface: (0.48868220.2712242), or, in the original units `temp` $= 37.47271$ and `tmax` $= 32.14499$. The eigenanalysis shows that the eigenvalues ($\lambda_1 = 1.848425$; $\lambda_2 = -2.010494$) have different signs, so this is a saddle point, as can also be seen in Fig. 4. SPOT determines the most steeply rising ridge in both directions, see also [11] for details. In addition to the points from the steepest descent, the best point from the first design, i.e., (1,50) is evaluated again, see Fig. 5.

Now, these points are evaluated and a new `rsm` model is build. Summarizing, the SPOT tasks `init`, `run`, `seq`, `run`, `seq`, `run`, `seq`, `run`, `seq` were performed. Each `seq` step generates information as shown in Fig. 3 and a plot of the response surface as shown in Fig. 4. The resulting surface plots from four steps of these procedure are shown in Fig. 4.

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.7383     0.5075   5.396   0.0125 *
x1            2.0779     0.3995   5.201   0.0138 *
x2           -0.7664     0.3987  -1.922   0.1503
x1:x2        -0.4677     0.4467  -1.047   0.3720
x1^2         -1.9963     0.9581  -2.084   0.1286
x2^2          1.8342     0.9591   1.912   0.1518
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.8934 on 3 degrees of freedom
Multiple R-squared: 0.9242,       Adjusted R-squared: 0.798
F-statistic: 7.319 on 5 and 3 DF,  p-value: 0.06605

Stationary point of response surface:
       x1        x2
0.4886822 0.2712242

Stationary point in original units:
    TEMP     TMAX
37.47271 32.14499

Eigenanalysis:
$values
[1]  1.848425 -2.010494
```

Figure 3: Output from the first regression model in the interactive approach

6.3 Automatic Adaptation of the Region of Interest

SPOT modifies the region of interest, if `seq.useAdaptiveRoi = TRUE`. This procedure consists of two phases, which are repeated in an alternating manner.

During the *orientation* phase, the direction of the largest improvement is determined as described in Sect. 6.2. Based on an existing design and related function values, the path of the steepest descent is determined. A small number of points is chosen from this path. Optimization runs are performed on these design points. In some situations, where no gradient information is available, the best point from a large number of design points, which were evaluated on the regression model, is chosen as the set of improvement points.

The *recalibration* phase determines the best point $\vec{x}_b$. It can be selected from the complete set of evaluated design points or from the points along the steepest descent only. The best point $\vec{x}_b$ defines the new center point of a central composite design. The minimal distance of $\vec{x}_b$ to the borders of the actual region of interest defines the radius of this design. If $\vec{x}_b$ is located at (or very close to) the borders of the region of interest, a Latin hypercube design which covers the whole region of interest is determined. This can be interpreted as a restart. To prevent premature convergence of this procedure, one additional new design point is generated by a tree based model. Next, the orientation phase is repeated.

This tuning process, which is based on regression models (`rsm`) and tree based regression (`rpart`), results in a tuned algorithm design point with `temp` =1 and `tmax` = 50. As in Sect. 5, one hundred repeats of the best solution from this tuning process

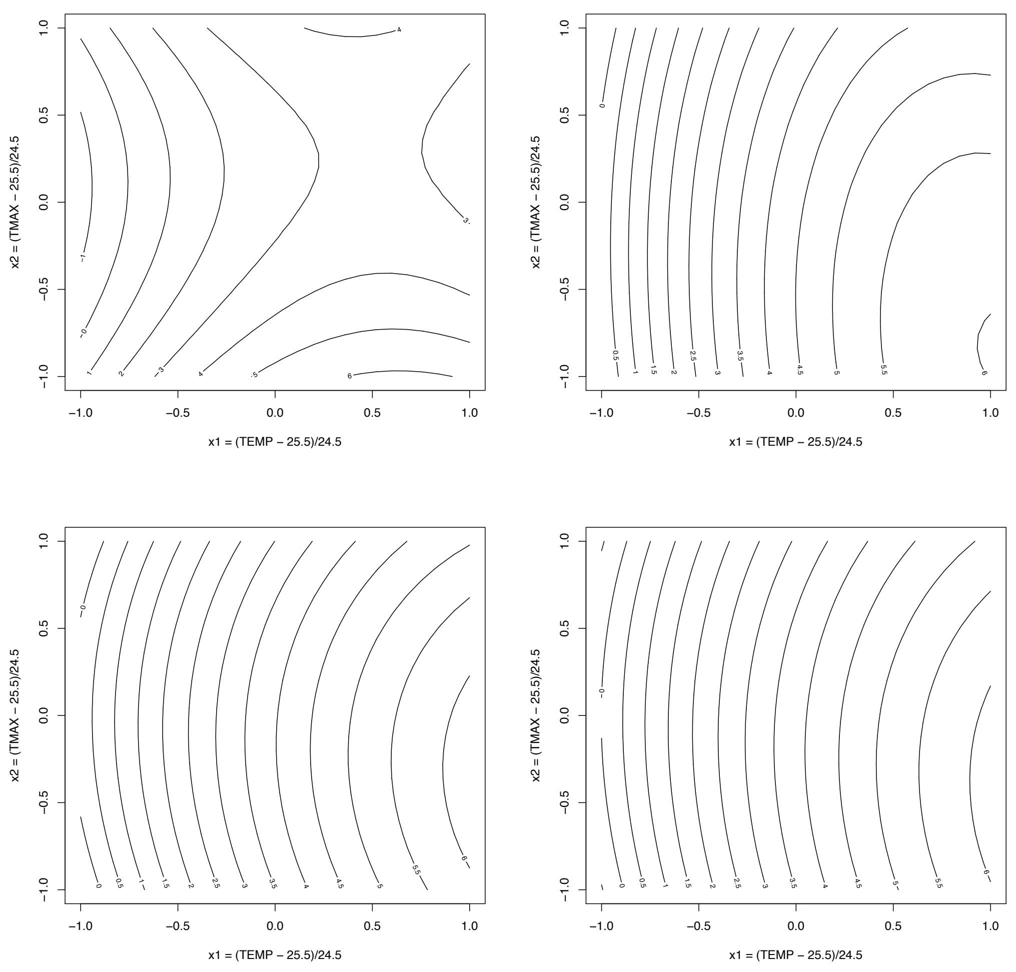

Figure 4: Response surface model based on the initial design. `rsm` was used to generate this plot

TEMP	TMAX
1.0000	50.0000
42.3560	32.4335
39.9060	32.2865
37.4805	32.1395
35.0305	31.9925
32.5805	31.8455

Figure 5: Second algorithm design. The best design point is evaluated again and five new design points are determined. Gradient information was used to generate this design

are generated. Results are shown in the last result row from Tab. 3.

7 Summary

All meta model improved SANN's performance. Random forest perform very well compared to other meta models. The interactive approach provides additional information, e.g., gradient, eigenvalues, plots of the response surface, which might accelerate the tuning process. Users can add or delete design points manually, e.g., after studying the response surface or using additional tools from exploratory data analysis. However, this feature was not used in the current study. In comparison to the automated approaches, the interactive approach performed competitively. This short study recommends random forest meta models for "lazy" tuning of algorithms. As an additional interesting feature, random forests can handle categorical and numerical design variables in one model.

Note, 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 algorithm design generators as well, are necessary.

8 Acknowledgments

This work has been supported by the Bundesministerium für Bildung und Forschung (BMBF) under the grants FIWA and SOMA (AIF FKZ 17N1009, "Ingenieurnachwuchs") and by the Cologne University of Applied Sciences under the research focus grant COSA.

References

[1] Bartz-Beielstein, T.: SPOT: An R Package For Automatic and Interactive Tuning of Optimization Algorithms by Sequential Parameter Optimization. Techn. Ber. arXiv:1006.4645. CIOP Technical Report 05-10, Cologne University of Applied Sciences. URL `http://arxiv.org/abs/1006.4645`. Comments: Related software can be downloaded from http://cran.r-project.org/web/packages/SPOT/index.html. 2010.

[2] Bartz-Beielstein, T.; Preuss, M.: The Future of Experimental Research. In: *Experimental Methods for the Analysis of Optimization Algorithms* (Bartz-Beielstein, T.; Chiarandini, M.; Paquete, L.; Preuss, M., Hg.), S. 17–46. Berlin, Heidelberg, New York: Springer. 2010.

[3] Bartz-Beielstein, T.; Lasarczyk, C.; Preuss, M.: The Sequential Parameter Optimization Toolbox. In: *Experimental Methods for the Analysis of Optimization Algorithms* (Bartz-Beielstein, T.; Chiarandini, M.; Paquete, L.; Preuss, M., Hg.), S. 337–360. Berlin, Heidelberg, New York: Springer. 2010.

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

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

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

[7] Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J.: *Classification and Regression Trees*. Monterey CA: Wadsworth. 1984.

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

[9] Gramacy, R. B.; Taddy, M.: Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with tgp Version 2, an R Package for Treed Gaussian Process Models (2010). URL `http://www.jstatsoft.org/v33/i06/paper`.

[10] Gramacy, R. B.: tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models. *Journal of Statistical Software* 19 (2007) 9, S. 1–46. URL `http://www.jstatsoft.org/v19/i09`.

[11] Lenth, R. V.: Response-Surface Methods in `R`, Using `rsm`. *Journal of Statistical Software* 32 (2009) 7, S. 1–17.