Sequential Model-Based Parameter Optimization: an Experimental Investigation of Automated and Interactive Approaches

Frank Hutter† **Thomas Bartz-Beielstein‡** **Holger H. Hoos†**

Kevin Leyton-Brown† **Kevin P. Murphy†**

†Department of Computer Science
University of British Columbia
201-2366 Main Mall, Vancouver BC, V6T 1Z4, Canada
{hutter,hoos,kevinlb,murphyk}@cs.ubc.ca

‡Institute of Computer Science
Cologne University of Applied Sciences
51643 Gummersbach, Germany
thomas.bartz-beielstein@fh-koeln.de

Abstract

This work experimentally investigates model-based approaches for optimizing the performance of parameterized randomized algorithms. Such approaches build a response surface model and use this model for finding good parameter settings of the given algorithm. We evaluated two methods from the literature that are based on Gaussian process models: *sequential parameter optimization* (SPO) (Bartz-Beielstein et al, 2005) and *sequential Kriging optimization* (SKO) (Huang et al, 2006). SPO performed better "out-of-the-box," whereas SKO was competitive when response values were log transformed. We then investigated key design decisions within the SPO paradigm, characterizing the performance consequences of each. Based on these findings, we propose a new version of SPO, dubbed SPO$^+$, which extends SPO with a novel intensification procedure and a log-transformed objective function. In a domain for which performance results for other (model-free) parameter optimization approaches are available, we demonstrate that SPO$^+$ achieves state-of-the-art performance. Finally, we compare this automated parameter tuning approach to an interactive, manual process that makes use of classical regression techniques. This interactive approach is particularly useful when only a relatively small number of parameter configurations can be evaluated. Because it can relatively quickly draw attention to important parameters and parameter interactions, it can help experts gain insights into the parameter response of a given algorithm and identify reasonable parameter settings.

1 Introduction

Many high-performance algorithms—and, in particular, many heuristic solvers for computationally challenging problems—expose parameters to allow end users to adapt the algorithm to target applications. Optimizing parameter settings is thus an important task in the context of developing,

evaluating and applying such algorithms. Recently, a substantial amount of research has been aimed at defining effective, automated procedures for parameter optimization (also called *algorithm configuration* or *parameter tuning*). More specifically, the goal is to find parameter settings of a given target algorithm that optimize a given performance metric on a given set (or distribution) of problem instances. The performance metric is usually based on the runtime required to solve a problem instance or—in the case of optimization problems—on the solution quality achieved within a given time budget.

Several variations of this problem have been investigated in the literature. These formulations vary in the number and type of target algorithm parameters allowed. Much existing work deals with relatively small numbers of numerical (often continuous) parameters; see, e.g., Coy et al (2001); Audet and Orban (2006); Adenso-Diaz and Laguna (2006). Some relatively recent approaches permit both larger numbers of parameters and categorical domains; see, e.g., Birattari et al (2002); Beielstein (2003); Bartz-Beielstein and Markon (2004); Bartz-Beielstein et al (2004c); Hutter et al (2007, 2009b). A different problem formulation also permits parameter adaptation on a per-instance basis; see, e.g., Hutter et al (2006).

Approaches also differ in whether or not explicit models (so-called *response surfaces*) are used to describe the dependence of target algorithm performance on parameter settings. There has been a substantial amount of work on both model-free and model-based approaches. Some notable model-free approaches include F-Raceby Birattari et al (2002); Balaprakash et al (2007), CALIBRA by Adenso-Diaz and Laguna (2006), and ParamILS by Hutter et al (2007). State-of-the-art model-based approaches use Gaussian stochastic processes (also known as Kriging models) to fit a response surface model. These models aim to minimize the mean squared error between predicted and actual responses, using a nonparametric function to represent the mean response. One particularly popular and widely studied version of Gaussian stochastic process models in the statistics literature is known under the acronym DACE, for *design and analysis of computer experiments* (Sacks et al, 1989). Combining such a predictive model with sequential decisions about the most promising next design point (often based on a so-called *expected improvement criterion*) gives rise to a sequential optimization approach. An influential contribution in this field was the *efficient global optimization* (EGO) procedure by Jones et al (1998), which addressed the optimization of deterministic black-box functions. In the context of parameter optimization, EGO could be used to optimize deterministic algorithms with continuous parameters on a single problem instance. Two independent lines of work extended EGO to noisy functions, which in the context of parameter optimization, allow the consideration of randomized algorithms: the *sequential Kriging optimization* (SKO) algorithm by Huang et al (2006), and the *sequential parameter optimization* (SPO) procedure by Bartz-Beielstein et al(2004b; 2005).

In the first part of this chapter, we maintain this focus on *Gaussian process* (GP) models; while in the second part, we consider the use of classical models in an interactive approach. Throughout, we limit ourselves to the simple case of only one problem instance. Such an instance may be chosen as representative of a set or distribution of similar instances. This restriction allows us to sidestep problems arising from performance variation across a set or distribution of problem instances and to focus on other core conceptual issues, while retaining significant practical relevance. (The management of such variation is an interesting and important topic of study; indeed, we have already begun to investigate it in our ongoing work. We note that this problem can be addressed by the algorithm of Williams et al (2000), though only in the case of deterministic target algorithms.)

This chapter leverages our own previous work in a number of ways. In particular, a short version of Sects. 1–5 was published by Hutter et al (2009a). Here, we formalize mathematical and algorithmic concepts from that paper much more thoroughly and provide more details throughout. The general sequential optimization approach using Kriging models has a long tradition in the statistics literature (Mockus et al, 1978; Jones et al, 1998) and was adapted to the optimization of algorithm parameters by Bartz-Beielstein et al (2004a,b). Bartz-Beielstein et al (2005) summarized results from SPO applications in various problem domains and Bartz-Beielstein (2006) described

SPO's methodology in detail. Bartz-Beielstein and Preuss (2006) and Bartz-Beielstein et al (2008b) studied the allocation of a fixed computational budget to SPO's initial design and to the evaluation of each parameter setting. The interactive approach used in Sect. 6 was first presented by Beielstein (2003); Bartz-Beielstein (2003), and Bartz-Beielstein and Markon (2004).

In the following, we use the term *sequential parameter optimization* (SPO) to refer to the methodological framework, i.e., a sequential approach to improve and understand an algorithm's performance by optimizing its parameters. The *sequential parameter optimization toolbox* (SPOT) is a software framework supporting both automated and interactive applications of this approach. Further discussion of SPOT is offered in Chap. **??**, while the general SPO framework is presented in Chap. **??**. Finally, we study three fully-automatic SPO procedures, which we refer to as SPO 0.3, SPO 0.4, and SPO$^+$. In the first part of the chapter, where we study these procedures in detail, we sometimes also use the term SPO to refer to the algorithmic framework of which SPO 0.3, SPO 0.4, and SPO$^+$ are instantiations.

The first part of our study thoroughly investigates the two fundamental components of any model-based optimization approach in this setting: the procedure for building the predictive model and the sequential procedure that uses this model to find performance-optimizing parameter settings of the target algorithm. We begin in Sect. 2 by describing our experimental setup, focusing especially on the two target algorithms we consider: CMA-ES (Hansen and Ostermeier, 1996; Hansen and Kern, 2004), a prominent gradient-free numerical optimization algorithm, and SAPS (Hutter et al, 2002), a high-performance local search algorithm for the propositional satisfiability problem. In Sect. 3, we compare the model-based optimization procedures SKO and SPO. Overall, we found that SPO produced more robust results than SKO in terms of the final target algorithm performance achieved. Consequently, the remainder of our study focuses on the mechanisms that underly SPO.

In Sect. 4, we investigate the effectiveness of various methods for determining the set of parameter settings used for building the initial parameter response model. Here we found that using more complex initial designs did not consistently lead to improvements over more naïve methods. More importantly, we also found that parameter response models built from log-transformed performance measurements tended to be substantially more accurate than those built from raw data (as used by SPO). In Sect. 5, we turn to the sequential experimental design procedure. We introduce a simple variation in SPO's intensification mechanism which led to significant and substantial performance improvements. Next, we consider two previous expected improvement criteria for selecting the next parameter setting to evaluate, and derive a new expected improvement criterion specifically for optimization based on predictive models trained on log-transformed data. These theoretical improvements, however, did not consistently correspond to improvements in observed algorithm performance. Nevertheless, we demonstrate that overall, our novel variant of SPO—dubbed SPO$^+$—achieved an improvement over the best previously known results on the SAPS parameter optimization benchmark.

Section 6 presents the second part of our study, in which we investigate a different perspective on optimizing the parameters of an algorithm. Specifically, we explore the interactive use of statistical tools in parameter optimization, which (like automated tuning) is also supported by the SPOT. Our approach is based on *response surface methodology* (RSM), which can be characterized as a collection of mathematical and statistical techniques that are useful for the modeling and analysis of problems in which a response of interest is influenced by several variables and the objective is to maximize or minimize this response (Montgomery, 2001; Beielstein, 2003). We show how to assess the relative importance of each parameter as well as interactions between parameters, and to manually refine the region of interest based on this knowledge. While the automated tuning option requires nearly no statistical knowledge, it can be computationally expensive. In contrast, the interactive option requires some knowledge about statistics (especially classical regression analysis), but can reduce computational costs and help the algorithm designer to learn which parameters deserve attention. This is particularly useful in cases where target algorithm evaluations are costly compared to the overall computational resources and time available for the parameter optimization process.

Table 1: Target algorithms, parameters, and the regions of interest (parameter domains) considered

Target algorithm	Parameter	Domain	Type
CMA-ES	NPARENTS	[1, 50]	integer
	NU	[2, 10]	continuous
	CS	(0, 1]	continuous
	DAMPS	[0.25, 0.99]	continuous
SAPS	α	(1, 1.4]	continuous
	ρ	[0, 1]	continuous
	P_{smooth}	[0, 0.2]	continuous
	wp	[0, 0.06]	continuous

Finally, Sect. 7 concludes the chapter with a discussion of advantages and drawbacks of automated and interactive approaches and the identification of interesting directions for future work.

2 Target Algorithms and Experimental Setup

Our first target algorithm was the *covariance matrix adaptation evolution strategy* (CMA-ES). CMA-ES is a prominent gradient-free global optimization algorithm for continuous functions (Hansen and Ostermeier, 1996; Hansen and Kern, 2004). It is based on an evolutionary strategy that uses a covariance matrix adaptation scheme. We used the Matlab implementation CMA-ES 2.54,[1] which is integrated into the SPO toolbox version 0.4 and was used as an example application for parameter optimization in the SPOT manual (Bartz-Beielstein et al, 2008a). CMA-ES has two obvious parameters: the number of parents, NPARENTS, and a factor $\texttt{NU} \geq 1$ relating the number of parents to the population size. (The population size is defined as $\lfloor \texttt{NPARENTS} \times \texttt{NU} + 0.5 \rfloor$.) Bartz-Beielstein et al (2008a) modified CMA-ES's interface to expose two additional parameters: the "learning rate for the cumulation for the step size control," CS, and the damping parameter, DAMPS (for details, see Hansen (2006), where NPARENTS, NU, CS, and DAMPS are called N, ν, c_σ, and d_σ, respectively). We used exactly the same *region of interest* (ROI; also called *experimental region*) considered in Bartz-Beielstein et al (2008a)'s SPOT example based on CMA-ES. Table 1 provides a summary of the target algorithms, parameters, and regions of interest we used.

For each run of CMA-ES, we allowed a limited number of function evaluations and used the resulting solution quality (i.e., the minimal function value found) as the response variable to be optimized. We considered four canonical 10-dimensional test functions with a global minimum function value of zero that were previously used in published evaluations of CMA-ES. Specifically, we considered the Sphere function (used in the SPOT example mentioned above) and the Ackley, Griewangk, and Rastrigin functions (used by Hansen and Kern (2004)). Following Bartz-Beielstein et al (2008a), for the Sphere function we initialized CMA-ES at the point $[10, ..., 10]^T \in \mathbb{R}^{10}$. To test global search performance, in the other test functions we initialized CMA-ES further away from the optima, at the point $[20, ..., 20]^T \in \mathbb{R}^{10}$. For the first two functions, we optimized mean solution quality reached by CMA-ES within $1,000$ function evaluations. For the latter two functions, which are more challenging, we set a limit of $10,000$ function evaluations. This setup is summarized in Table 2.

The second target algorithm we considered was *Scaling And Probabilistic Smoothing* (SAPS) (Hutter et al, 2002), a high-performance dynamic local search algorithm for the propositional satisfiability problem. We used the standard UBCSAT implementation (Tompkins and Hoos, 2004) of SAPS and

[1] The newest CMA-ES version, 3.0, differs mostly in its interface and its support of "separable" CMA (see the change log at http://www.lri.fr/~hansen/cmaes_inmatlab.html).

Table 2: Experimental setup for the CMA-ES test cases

Test function	Dimensionality	Initial point [$\cdot\mathbf{1} \in \mathbb{R}^{10}$]	# Function evaluations allowed
Sphere	10	10	1 000
Ackley	10	20	1 000
Griewangk	10	20	10 000
Rastrigin	10	20	10 000

defined the region of interest (Table 1) to closely follow an earlier parameter optimization study of SAPS by Hutter et al (2007), with the difference that we did not discretize parameter values. (Hutter et aldid so because the parameter optimization procedure used in that work required it.) For SAPS, our goal was to minimize median runtime (measured in local search steps) for solving the "quasigroups with holes" (QWH) SAT instance used by Hutter et al (2007). This instance belongs to a family of distributions that has received considerable interest in the SAT community. We chose this particular instance to facilitate direct comparison of the performance achieved by the parameter optimization procedures considered here and by Hutter et al (2007).

To evaluate the quality $Q(\boldsymbol{\theta})$ of a proposed parameter setting $\boldsymbol{\theta}$ in an offline evaluation stage of the algorithm, we always performed additional test runs of the target algorithm with setting $\boldsymbol{\theta}$. In particular, for the CMA-ES test cases, we computed $Q(\boldsymbol{\theta})$ as the mean solution cost achieved by CMA-ES using $\boldsymbol{\theta}$ across 100 test runs. For the higher-variance SAPS domain, we computed $Q(\boldsymbol{\theta})$ as the median runtime achieved by SAPS with setting $\boldsymbol{\theta}$ across 1,000 test runs. We define the *solution cost c_k of a parameter optimization run* after k runs of the target algorithm as the quality $Q(\boldsymbol{\theta})$ of the parameter setting $\boldsymbol{\theta}$ the method would output if terminated at that point. The *final performance* of a parameter optimization run is the performance at the end of the run. In order to measure the performance of a parameter optimization method, we performed 25 runs of the method with different random seeds, reporting mean and standard deviation of the final performance across these 25 repetitions. We also performed paired Max-Wilcoxon signed rank tests for differences in final performance. We chose a paired test because, using identical random seeds, the ith repetition of every parameter optimization method used the same initial design and response values. For the experiments in Sect. 5.3 this pairing did not apply, and consequently, we used the (unpaired) Mann–Whitney U test instead.

3 Existing Methods for Sequential Model-Based Optimization of Noisy Functions

In this section, we review two existing model-based optimization methods for noisy responses: the sequential Kriging optimization (SKO) algorithm by Huang et al (2006), and the sequential parameter optimization (SPO) procedure by Bartz-Beielstein et al(2004b; 2005).

3.1 General Gaussian Process Regression

In order to model the dependence of a response variable (in our case, the performance of a given target algorithm) on parameter settings, both SKO and SPO use a form of Gaussian stochastic process, a nonparametric regression method that predicts both the mean and variance of a response variable. We first give the basic equations for Gaussian process regression and then describe the differences in the ways this approach is used by SKO and SPO.

To apply Gaussian process regression, first we need to select a parameterized kernel function $k_\lambda : \Theta \times \Theta \to \mathbb{R}^+$ that quantifies the similarity between two parameter settings. We also need to set the variance σ^2 of Gaussian-distributed measurement noise. The predictive distribution of a

Table 3: Summary of notation. The upper part of the table contains symbols used in the pseudocode, the middle part contains SPO parameters, and the lower part general notation

Symbol	Meaning
Θ	Space of allowable parameter settings (region of interest)
θ	Parameter setting, element of Θ
$\theta_{i:j}$	Vector of parameter settings, $[\theta_i, \theta_{i+1}, \ldots, \theta_j]$
D	Dimensionality of Θ (number of parameters to be tuned)
y	Response variable (performance of target algorithm)
history	Structure keeping track of target runs executed and responses, as well as incumbents
$N(\theta)$	Number of previous target algorithm runs with setting θ; depends on history
$\hat{c}(\theta)$	Empirical cost statistic over the $N(\theta)$ runs with θ (e.g., mean runtime); depends on history
$\mathscr{M}$	Predictive model
r	Number of repeats in SPO (increases over time). Initial value: SPO parameter
maxR	Maximal number of repeats in SPO
d	Size of initial design in SPO
m	Number of parameter settings to evaluate in each iteration in SPO
p	Number of previous parameter setting to evaluate in each iteration of SPO$^+$
$Q(\theta)$	Test quality (cost) of parameter setting θ
c_k	Solution cost $Q(\theta)$ of incumbent parameter setting θ at step k

zero-mean Gaussian stochastic process for response y_{n+1} at input θ_{n+1} given training data $\mathscr{D} = \{(\theta_1, y_1), \ldots, (\theta_n, y_n)\}$, measurement noise with variance σ^2, and kernel function k is then

$$p(y_{n+1}|\theta_{n+1}, \theta_{1:n}, \mathbf{y}_{1:n}) = \mathscr{N}(y_{n+1}|\boldsymbol{k}_*{}^{\mathrm{T}}[\boldsymbol{K} + \sigma^2\boldsymbol{I}]^{-1}\mathbf{y}_{1:n}), k_{**} - \boldsymbol{k}_*{}^{\mathrm{T}}(\boldsymbol{K} + \sigma^2\mathbf{I})^{-1}, \tag{1}$$

where

$$\boldsymbol{K} = \begin{pmatrix} k(\theta_1, \theta_1) & \ldots & k(\theta_1, \theta_n) \\ & \ddots & \\ k(\theta_n, \theta_1) & \ldots & k(\theta_n, \theta_n) \end{pmatrix} \tag{2}$$

$$\boldsymbol{k}_* = (k(\theta_1, \theta_{n+1}), \ldots, k(\theta_n, \theta_{n+1})) \tag{3}$$

$$k_{**} = k(\theta_{n+1}, \theta_{n+1}) + \sigma^2, \tag{4}$$

$\mathbf{I}$ is the n-dimensional identity matrix, and $p(a|b) = \mathscr{N}(a|c,d)$ denotes that the conditional distribution of a given b is a Gaussian with mean b and variance c; see, e.g., Rasmussen and Williams (2006) for a derivation. A variety of kernel functions can be used, the most common of which is of the form

$$K(\theta_i, \theta_j) = \sum_{k=1}^{d} \exp(-\lambda_k(\theta_{ik} - \theta_{jk})^2), \tag{5}$$

where $\lambda_1, \ldots, \lambda_d$ are the kernel parameters. This kernel is most appropriate if the response is expected to vary smoothly in the input parameters θ. The kernel parameters and the observation noise variance σ^2 constitute the *hyper-parameters* $\boldsymbol{\phi}$, which are typically set by maximizing the *marginal likelihood* $p(\mathbf{y}_{1:N})$ with a gradient-based optimizer. Using the chain rule, the gradient is

$$\frac{\partial \log p(\mathbf{y}_{1:N})}{\partial \phi_j} = \frac{\partial \log p(\mathbf{y}_{1:N})}{\partial (\boldsymbol{K} + \sigma^2\boldsymbol{I})} \frac{\partial (\boldsymbol{K} + \sigma^2\boldsymbol{I})}{\partial \phi_j}.$$

In *noise-free* Gaussian process models, the observation noise variance is fixed to $\sigma^2 = 0$.

Algorithm Framework 1: Sequential model-based Optimization

Input : Target algorithm A
Output: Incumbent parameter setting θ_{inc}

```
[history, θ_inc] ← Initialize();
M ← ∅;
repeat
    [M, θ_inc] ← FitModel(M, history, θ_inc);
    Θ_new ← SelectNewParameterSettings(M, θ_inc, history);
    [history, θ_inc] ← Intensify(Θ_new, θ_inc, M, history);
until TerminationCriterion() ;
return θ_inc;
```

Procedure 2: ExecuteRuns(history, θ, numRuns)

```
for i = 1, ..., numRuns do
    Execute target algorithm A with parameter setting θ, store response in y;
    Append θ to history.θ;
    Append y to history.y;
return history
```

Learning a Gaussian stochastic process model from data can be computationally expensive. Inverting the $n \times n$ matrix $[\boldsymbol{K} + \sigma^2 \mathbf{I}]$ takes time $O(n^3)$, and has to be done in every step of the hyper-parameter optimization. Various approximations can be used to reduce this to $O(n^2)$; see, e.g., Quinonero-Candela et al (2007). We refer to the process of optimizing hyper-parameters and computing the inverse as *fitting* the model. Subsequent predictions are cheaper than fitting the model, requiring matrix-vector multiplications and thus time $O(n^2)$. This is still substantially slower than prediction with a parametric model, the time complexity of which is independent of n.

3.2 A Common Framework for Sequential Model-Based Optimization

In this section, we describe SKO and SPO in a unified framework, which is outlined in the form of pseudocode in Algorithm Framework 1. One common building block of all parameter-optimization algorithms is a procedure that executes the target algorithm with a given parameter setting and stores the response. This building block is described in our Procedure 2, ExecuteRuns. On the first invocation, history.θ and history.y are initialized to be empty lists.

Table 3 summarizes our notation and defines some global variables used in SKO and SPO. Note in particular $N(\theta)$ and $\hat{c}(\theta)$; $N(\theta)$ denotes the number of runs we have so far executed with a parameter setting θ, and $\hat{c}(\theta)$ denotes the empirical performance across the $N(\theta)$ runs that have been performed for θ.

3.2.1 Initialization

We outline the initialization of SKO and SPO in Procedures 3 and 4, respectively. Procedure Initialize() is called as

$$[\text{history}, \theta_{inc}] = \text{Initialize}().$$

SKO starts with a Latin hypercube design of $10 \cdot D$ parameter settings, where D is the number of parameters to be optimized. It executes the target algorithm at these settings and then performs an additional run for the D settings with the lowest response. The incumbent θ_{inc} is chosen as the setting with the lowest empirical cost $\hat{c}(\theta)$ out of these D settings. In SPO, d parameter settings are

Procedure 3: Initialize() in SKO
Θ denotes the space of allowable parameter settings (the region of interest); D denotes the number of parameters to be tuned.

```
k ← 10D;
θ_{1:k} ← LatinHypercubeDesign(Θ, k);
for i = 1,...,k do
    history ← ExecuteRuns(history, θ_i, 1);
θ_{k+1:k+D} ← the D settings θ_j out of θ_{1:k} with smallest ĉ(θ_j);
for i = 1,...,D do
    history ← ExecuteRuns(history, θ_{k+i}, 1);
θ_inc ← random element of arg min_{θ∈{θ_1,...,θ_d}} ĉ(θ);
return [history, θ_inc];
```

Procedure 4: Initialize() in SPO
Θ denotes the space of allowable parameter settings (the region of interest). The number of LHD parameter settings, d, and the number of repetitions, r, are parameters of SPO

```
θ_{1:d} ← LatinHypercubeDesign(Θ, d);
for i = 1,...,d do
    history ← ExecuteRuns(history, θ_i, r);
θ_inc ← random element of arg min_{θ∈{θ_1,...,θ_d}} ĉ(θ);
history.Θ_hist ← {θ_inc};
return [history, θ_inc]
```

chosen with a Latin hypercube design, and the target algorithm is executed r times for each of them; d and r are algorithm parameters. In practice, the initial design size d is chosen as the minimum number of points required to fit a reasonably accurate model. (Here, we used $d = 250$ and $r = 2$ as discussed in Sect. 4.1; however, in the direct comparison to SKO, we used the initial design of SKO within SPO to limit the number of confounding factors in the comparison.) The incumbent θ_{inc} is chosen as the parameter setting with the lowest empirical cost $\hat{c}(\theta)$.

3.2.2 Fit of Response Surface Model

Both SKO and SPO base their predictions on a combination of a linear model and a Gaussian process (GP) model fit on the residual errors of the linear model. However, both of them default to using only a single constant basis function in the linear model, thus reducing the linear model component to a constant offset term, the mean response value. Throughout this chapter, we use these defaults; the model we use is thus an offset term plus a zero-mean GP model. Our implementation of SPO uses the DACE Matlab toolbox to construct this predictive model, while SKO implements the respective equations itself. The exact equations used in both SKO and the DACE toolbox implement methods to deal with ill-conditioning; we refer the reader to the original publications for details (Huang et al, 2006; Bartz-Beielstein, 2006; Lophaven et al, 2002).

Procedure FitModel is called as

$$[\mathscr{M}, \theta_{inc}] = \text{FitModel}(\mathscr{M}, \text{history}, \theta_{inc}). \tag{6}$$

When it is first called, $\mathscr{M} = \emptyset$. Note that FitModel may update the incumbent configuration, θ_{inc}. SKO makes use of this, while SPO does not. (Instead, SPO updates θ_{inc} in Procedure Intensify.)

Procedure 5: FitModel($\mathscr{M}$, history, θ_{inc}) in SKO
SKO sets its incumbent θ_{inc} based on the learnt model

if $\mathscr{M} == \emptyset$ *or last hyper-parameter optimization occurred more than n steps ago* **then**
  Fit GP model $\mathscr{M}$ and hyper-parameters for data $\{(\text{history}.\theta_i, \text{history}.y_i)\}_{i\in\{1,\ldots,n\}}$;
else
  Fit GP model $\mathscr{M}$ for data $\{(\text{history}.\theta_i, \text{history}.y_i)\}_{i\in\{1,\ldots,n\}}$, re-using hyper-parameters saved in previous $\mathscr{M}$;
$\Theta_{seen} \leftarrow \bigcup_{i=1}^{n}\{\text{history}.\theta_i\}$;
for all $\theta \in \Theta_{seen}$ **do**
  $[\mu_\theta, \sigma^2_\theta] \leftarrow \text{Predict}(\mathscr{M}, \theta)$;
$\theta_{inc} \leftarrow$ random element of $\arg\min_{\theta\in\Theta}(\mu_\theta + \sigma_\theta)$;
return $[\mathscr{M}, \theta_{inc}]$;

Procedure 6: FitModel($\mathscr{M}$, history, θ_{inc}) in SPO
Recall that $\hat{c}(\theta)$ is the cost statistic across *all* runs with setting θ. SPO does not update its incumbent θ_{inc} in this procedure

$\Theta_{seen} \leftarrow \bigcup_{i=1}^{n}\{\text{history}.\theta_i\}$;
Fit GP model $\mathscr{M}$ and hyper-parameters for data $\{\theta, \hat{c}(\theta)\}_{\theta\in\Theta_{seen}}$, using fixed $\sigma^2 = 0$;
return $[\mathscr{M}, \theta_{inc}]$;

SKO uses Gaussian process regression in the conventional manner to fit noisy response data directly; we describe this in Procedure 5. Note that when a GP model is trained directly on noisy response data, measurement noise is assumed to be normally distributed, an assumption that is violated in many applications of parameter optimization. While distributions of solution qualities across multiple runs of a randomized heuristic algorithm can often be approximated quite well with a Gaussian distribution, it is well known that the distributions of runtimes of randomized heuristic algorithms for solving hard combinatorial problems tend to exhibit substantially fatter tails; see, e.g., Chap. 4 of Hoos and Stützle (2005). Also note that Gaussian process regression assumes symmetric noise and fits the *arithmetic mean* of the response that is used as input. Thus, if the inputs are raw response values, Gaussian processes model mean response. Commonly, the response value is transformed in order to yield a better model fit (see Sect. 4.2 for a more detailed discussion of transformations). In particular, a log transformation appears to be reasonable in many circumstances. However, note that under a log transformation, Gaussian process regression fits the mean of the *transformed* response, which corresponds to the geometric mean rather than the arithmetic mean of the true response.

As also described in Procedure 9, after fitting its model, SKO updates the new configuration, θ_{inc}, based on the new model. The parameter setting that minimizes a GP model's mean prediction is not necessarily the best choice for the incumbent, because it may be based on dramatically fewer function evaluations than other, similar-scoring configurations. Recognizing this fact, SKO implements a risk-averse strategy: it picks the previously evaluated parameter setting that minimizes predicted mean plus one predicted standard deviation.

SPO uses Gaussian process regression in a very different, nonstandard way, described in Procedure 6. It first computes the user-defined empirical performance metric $\hat{c}(\theta)$ for each parameter setting θ evaluated so far, and then fits a *noise-free* GP model to learn a mapping from parameter settings to the performance metric. This approach has several benefits and drawbacks. If we have executed a single run with parameter setting θ_1 and many runs with setting θ_2, naturally our confidence in performance estimate $\hat{c}(\theta_1)$ will be lower than our confidence in $\hat{c}(\theta_2)$. SPO ignores

Procedure 7: SelectNewParameterSettings($\mathscr{M}, \theta_{inc}$, history) in SKO

$\Theta_{new} \leftarrow$ the single parameter setting found by optimizing the augmented expected improvement criterion from (Huang et al, 2006) using the Nelder–Mead simplex method.
return Θ_{new}

Procedure 8: SelectNewParameterSettings($\mathscr{M}, \theta_{inc}$, history) in SPO
Note that m is a parameter of SPO

// ===== Select m parameter settings with expected improvement
$\Theta_{rand} \leftarrow$ set of 10,000 elements drawn uniformly at random from Θ;
for all $\theta \in \Theta_{rand}$ **do**
  $[\mu_\theta, \sigma^2_\theta] \leftarrow$ Predict($\mathscr{M}$, θ);
  $EI(\theta) \leftarrow$ Compute expected improvement criterion $E[I^2(\theta)]$ (see Sect. 5.2) given μ_θ and σ^2_θ;
$\Theta_{new} \leftarrow$ the m elements of Θ_{rand} with highest $EI(\theta)$;
return Θ_{new};

this fact and collapses all information for a parameter setting θ to a single value $\hat{c}(\theta)$, discarding all information on variance. On the other hand, fitting the GP model to the performance metric directly enables SPO to optimize almost arbitrary user-defined performance metrics, which could not be done with standard GP models. Examples include median performance, variance across runs, and trade-offs between mean and variance. To our best knowledge, SPO is the only existing model-based method with such flexibility in the objective function being optimized. Another benefit is that the assumption of normally-distributed response values is dropped. The final benefit of collapsing the data to a single point per parameter setting lies in the reduction in computational complexity thus achieved. While SKO has cubic scaling behavior in the number of target algorithm runs performed, SPO only takes time cubic in the number of *disjoint* parameter settings evaluated.

3.2.3 Selection of New Parameter Settings to Evaluate

Following Jones et al (1998), both SKO and SPO use an expected improvement criterion (EIC) to determine which parameter settings to investigate next, thereby drawing on both the mean and variance predictions of the GP model. This criterion trades off learning about new, unknown parts of the parameter space and intensifying the search locally in the best known region (a so-called exploration/exploitation trade-off).

Procedure SelectNewParameterSettings is called as

$$\Theta_{new} = \text{SelectNewParameterSettings}(\mathscr{M}, \theta_{inc}, \text{history}). \tag{7}$$

SKO selects a single new parameter setting by maximizing an augmented expected improvement criterion using the Nelder-Mead simplex method. The augmentation adds a bias away from parameter settings for which predictive variance is low; see Huang et al (2006). SPO, on the other hand, evaluates the $E[I^2]$ expected improvement criterion (Schonlau et al, 1998) at 10,000 randomly-selected parameter settings, and chooses the m with the highest expected improvement; see Sect. 5.2 for more details. In this work, we use the default $m = 1$. For completeness, we give the simple pseudocode for these methods in Procedures 8 and 7.

Procedure 9: Intensify(Θ_{new}, θ_{inc}, $\mathcal{M}$, history) in SKO
SKO does not update its incumbent in this procedure

$\theta \leftarrow$ the single element of Θ_{new};
history $\leftarrow$ ExecuteRuns(history, θ, 1);
return [history, θ_{inc}];

Procedure 10: Intensify(Θ_{new}, θ_{inc}, $\mathcal{M}$, history) in SPO 0.3
After performing runs for the incumbent and the new parameter settings, SPO updates the incumbent. Side effect: may increase the global number of repeats, r

for all $\theta \in \Theta_{new}$ **do**
    history $\leftarrow$ ExecuteRuns(history, θ, r);
history $\leftarrow$ ExecuteRuns(history, θ_{inc}, $r/2$);
$\Theta_{seen} \leftarrow \bigcup_{i=1}^{n} \{\text{history}.\theta_i\}$;
$\theta_{inc} \leftarrow$ random element of $\arg\min_{\theta \in \Theta_{seen}} \hat{c}(\theta)$;
if $\theta_{inc} \in$ *history*.Θ_{hist} **then** $r \leftarrow \min\{2r, \text{maxR}\}$;
history.Θ_{hist} $\leftarrow$ history.$\Theta_{hist} \cup \{\theta_{inc}\}$;
return [history, θ_{inc}];

3.2.4 Intensification

Any parameter optimization method must make decisions about which parameter setting θ_{inc} to return to the user as its incumbent solution, both if interrupted during the search progress and (especially) upon completion. Candidate parameter settings are suggested by Procedure SelectNewParameterSettings, and in order to decide whether they, the current incumbent, or even another parameter setting should become the new incumbent, we need to perform additional runs of the target algorithm. Which parameter settings to use, how many runs to execute with each of them, and how to determine the new incumbent based on those runs is specified in Procedure Intensify, which is called as

$$[\text{history}, \theta_{inc}] = \text{Intensify}(\Theta_{new}, \theta_{inc}, \mathcal{M}, \text{history}). \tag{8}$$

Note that this procedure allows an update of the incumbent, θ_{inc}. SPO makes use of this option, while SKO updates its incumbent in Procedure FitModel (see Procedure 5).

In order to provide more confident estimates for its incumbent, SPO implements an explicit intensification strategy. In SPO, predictive uncertainty cannot be used in the context of updating the incumbent parameter setting, because the underlying noise-free GP model predicts uncertainty to be exactly zero at all previously evaluated parameter settings. Instead, the number of evaluations performed for a parameter setting is used as a measure of confidence. SPO performs additional runs for its incumbent parameter setting θ_{inc} in order to challenge that configuration. This is done to make sure that θ_{inc} did not merely happen to yield low response values in the limited number of target algorithm runs previously performed with θ_{inc}. The number of evaluations used in this context differs between SPO versions 0.3 and 0.4; Procedures 10 and 11 reflect the differences between these two versions. In contrast, SKO does not implement an explicit intensification strategy; it only performs a single run with each parameter setting considered.

3.3 Empirical Comparison of SKO and SPO

We empirically compared SKO and SPO on the CMA-ES test cases, based on the same initial design (the one used by SKO) and with a limit of 200 runs of the target algorithm.

Procedure 11: Intensify(Θ_{new}, θ_{inc}, $\mathcal{M}$, history) in SPO 0.4
After performing runs for the incumbent and the new parameter settings, SPO updates the incumbent. Side effect: may increase the global number of repeats, r

for all $\theta \in \Theta_{new}$ **do**
  history $\leftarrow$ ExecuteRuns(history, θ, r);
history $\leftarrow$ ExecuteRuns(history, θ_{inc}, $r - N(\theta_{inc})$);
$\Theta_{seen} \leftarrow \bigcup_{i=1}^{n} \{\text{history}.\theta_i\}$;
$\theta_{inc} \leftarrow$ random element of $\arg\min_{\theta \in \Theta_{seen}} \hat{c}(\theta)$;
if $\theta_{inc} \in$ *history*.Θ_{hist} **then** $r \leftarrow \min\{r+1, \text{maxR}\}$;
history.$\Theta_{hist} \leftarrow$ history.$\Theta_{hist} \cup \{\theta_{inc}\}$;
return [history, θ_{inc}];

We chose this low limit on the number of runs because the original SKO implementation was very slow: even limited to as few as 200 runs of the target algorithm, SKO runs took about one hour.[2] SKO spent most of its time performing numerical optimization of the expected improvement criterion. The iterations of SKO became slower as a run progressed, and therefore we could only afford to perform tuning runs for 200 runs of the target algorithm. For this reason and because SKO can only optimize *mean* performance, we did not perform experiments with SKO for the SAPS scenario, which features quite high observation noise and in which the objective is to minimize *median* runtime.

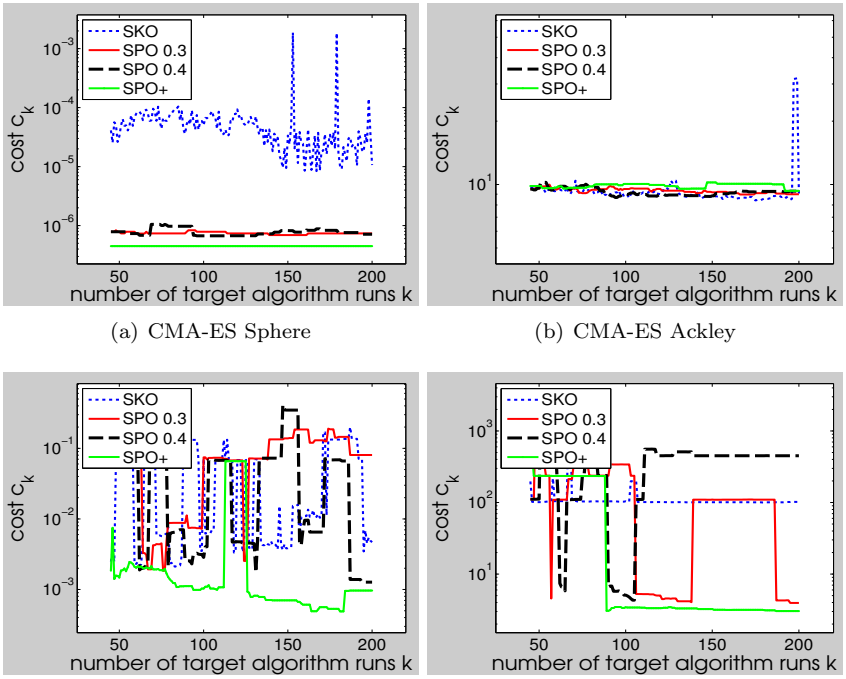
We reimplemented SPO 0.3 and 0.4, as well as a new version, SPO$^+$ (defined in Sect. 5.1). We verified that the performance of our reimplemented SPO 0.4 matched that of the original SPO 0.4 implementation. The SPO runs were substantially faster than those of SKO. They took about 2 minutes per repetition,[3] 85% of which was spent running the target algorithm.

Our first set of experiments used original, untransformed CMA-ES solution quality as the objective function to be minimized; we show the results in Fig. 1. On the Sphere function, the LHD already contained very good parameter settings, and the challenge was mostly to select the best of these and stick with it. From the figure, we observe that SPO largely succeeded in doing this, while SKO did not. On the Ackley function, SKO's performance was quite good, except for a drastic spike close to 200 runs of the target algorithm. On the Griewangk and Rastrigin functions, the variation of performance across multiple runs of CMA-ES was very high. Correspondingly, all approaches showed large variation in the quality of the parameter settings they selected over time. (Intuitively, the parameter optimization procedure detects a new region, which seems promising based on a few runs of the target algorithm. Then, after additional target algorithm runs, that region is discovered to be worse than initially thought, and the optimizer moves on. During the period it takes to discover the true, poor nature of the region, the search returns a poor incumbent.) SPO$^+$ showed the most robust performance of the four parameter optimization procedures.

Secondly, we experimented with log-transformed qualities (see Sect. 4.2 for a more detailed discussion of log-transformations). Figure 2 shows that this transformation improved SKO performance quite substantially and did not affect the SPO variants much. We attribute this to the fact that

[2] We ran SKO on a 3 GHz Pentium 4 with 4 GB RAM running Windows XP Professional, Service Pack 3. We report wall clock time on an otherwise idle system. (We did not use Linux machines for these experiments because SKO only compiled for Windows.) In order to ascertain that the target algorithm has exactly the same behavior as for other methods running under Linux, we gathered the function values SKO requested by means of a wrapper script that connected to the machines the SPO experiments were carried out on, performed a requested run of the target algorithm there, and returned the result of the run. This incurred very little overhead. Finally, some SKO runs failed due to problems in the numerical optimization of the expected improvement criterion. We repeated those runs until they completed. These repetitions were nondeterministic due to measurement noise in the objective function.

[3] These experiments were carried out on a cluster of 55 dual 3.2 GHz Intel Xeon CPUs with 2 GB RAM each and 2 MB cache per CPU, running OpenSuSE Linux 10.1. Each run only used one CPU and we report CPU time.



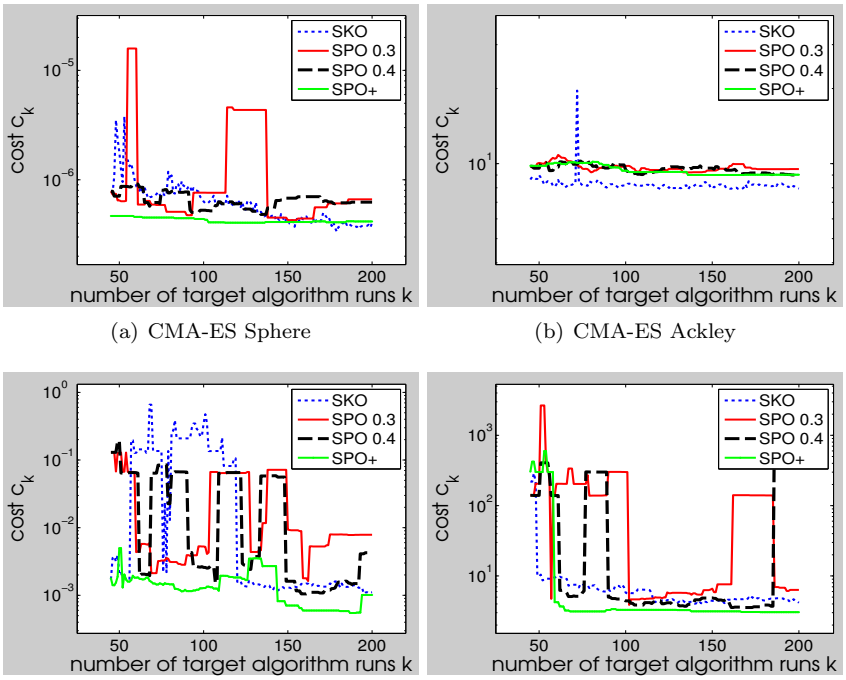
(a) CMA-ES Sphere

(b) CMA-ES Ackley

(c) CMA-ES Griewangk

(d) CMA-ES Rastrigin

Figure 1: Comparison of SKO and three variants of SPO for optimizing CMA-ES on the Sphere function. We plot the solution cost c_k of each method (mean solution quality CMA-ES achieved in 100 test runs on each of the 4 test functions using the method's chosen parameter settings) as a function of the number of algorithm runs, k, it was allowed to perform. These values are averaged across 25 runs of each method. All models were based on SKO's initial design and original untransformed data



(a) CMA-ES Sphere

(b) CMA-ES Ackley

(c) CMA-ES Griewangk

(d) CMA-ES Rastrigin

Figure 2: Comparison of SKO and three variants of SPO for optimizing CMA-ES on the Sphere function. We plot the solution cost c_k of each method (mean solution quality CMA-ES achieved in 100 test runs on each of the 4 test functions using the method's chosen parameter settings) as a function of the number of algorithm runs, k, it was allowed to perform. These values are averaged across 25 runs of each method. All models were based on SKO's initial design and log-transformed data (for SPO as discussed in Sect. 4.2)

the quality of the model is much more important in SKO. While SPO "only" uses the model in order to make decisions about the next parameter setting to evaluate, SKO also uses it in order to select its incumbent. After the log transformation, SKO and SPO$^+$ performed comparably. For three main reasons, we decided to focus the remainder of this study on various aspects of the SPO framework. Firstly, our main interest is in complex scenarios, in which the predictive model might not actually perform very well. In such scenarios, we believe it is important to employ an explicit intensification criterion instead of relying on the model alone to select incumbents. Secondly, SPO has the advantage of being able to optimize a variety of user-defined objective functions, while the Gaussian process model underlying SKO can only fit the mean of the data. In practice, users might be more interested in statistics other than mean performance, such as the best out of ten algorithm runs. SPO implements this performance metric and also allows for many other options. Finally, the SKO implementation we used was simply too slow to perform the type of experiments with tens of the thousands of target algorithm runs that interested us. Nevertheless, a worthwhile direction for future work would be to experiment with explicit intensification mechanisms in the SKO framework.

4 Model Quality

It is not obvious that a model-based parameter optimization procedure needs models that accurately predict target algorithm performance across all parameter settings, particularly including very bad ones. Nevertheless, all else being equal, models with good overall accuracy are generally helpful to such methods, and are furthermore essential to more general tasks such as performance robustness analysis. In this section, we investigate the effects of two key model-design choices on the accuracy of the GP models used by SPO.

4.1 Choosing the Initial Design

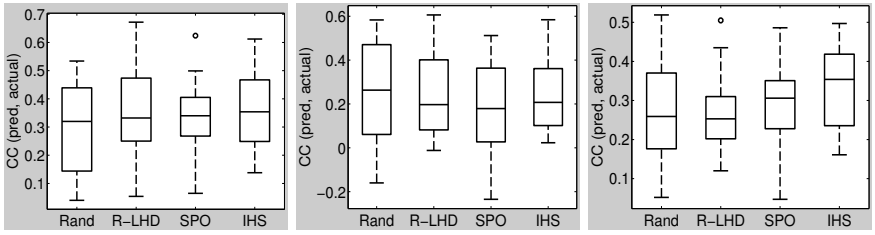
In the overall approach described in Algorithm Framework 1, an initial parameter response model is determined by constructing a GP model based on the target algorithm's performance on a given set of parameter settings (the *initial design*). This initial model is then subsequently updated based on runs of the target algorithm ith additional parameter settings. The decision about which additional parameter settings to select is based on the current model.

It is reasonable to expect that the quality of the final model (and the performance-optimizing parameter setting determined from it) would depend on the quality of the initial model. Therefore, we studied the overall accuracy of the initial parameter response models constructed based on various initial designs. The effect of the number of parameter settings in the initial design, d, as well as the number of repetitions for each parameter setting, r, has been studied before (Bartz-Beielstein and Preuss, 2006), and we thus fixed them in this study.[4] Specifically, we used $r = 2$ and $d = 250$, such that when methods were allowed $1{,}000$ runs of the target algorithm, half of them were chosen with the initial design.

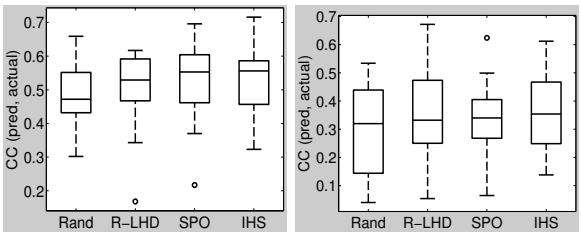
Here, we study the effect of the method for choosing *which* 250 parameter settings to include in the initial design, considering four methods: (1) a uniform random sample from the region of interest; (2) a random Latin hypercube design (LHD); (3) the LHD used in SPO; and (4) a more complex LHD based on *iterated distributed hypercube sampling* (IHS) (Beachkofski and Grandhi, 2002).

We evaluated the parameter response models obtained using these initialisation strategies by assessing how closely model predictions at previously unseen parameter settings matched the true performance achieved using these settings. In particular, we were interested in how useful the predictions were for determining whether a given parameter setting performed better or worse than

[4] We do note, however, that this previous work has not conclusively answered the question of how best to set the initial design size, and thus that this question continues to present an open research problem.

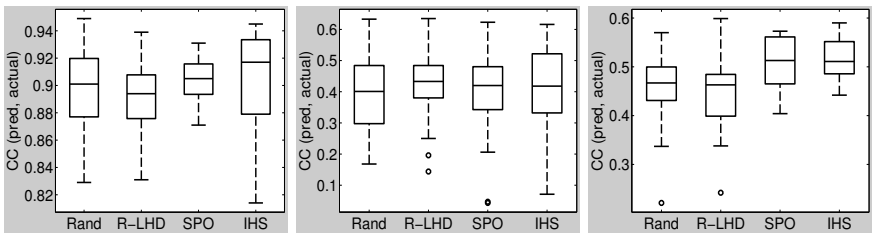


(a) CMAES-Sphere (b) CMAES-Ackley (c) CMAES-Griewangk

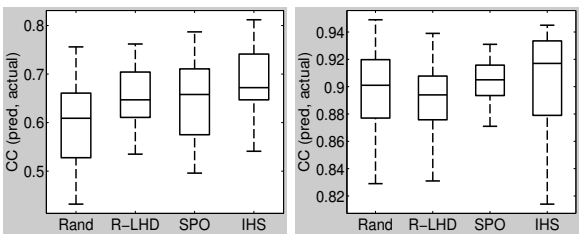


(d) CMAES-Rastrigin (e) SAPS-QWH

Figure 3: Performance of models based on different initial designs and raw, untransformed data. We computed the Spearman correlation coefficient (CC) between actual and predicted performance for 250 randomly selected test parameter settings and show boxplots across 25 independent repetitions. Each repetition used the same test parameter settings



(a) CMAES-Sphere (b) CMAES-Ackley (c) CMAES-Griewangk



(d) CMAES-Rastrigin (e) SAPS-QWH

Figure 4: Performance of models based on different initial designs and log-transformed data. We compute the Spearman correlation coefficient (CC) between actual and predicted performance for 250 randomly selected test parameter settings and show boxplots across 25 independent repetitions. Each repetition used the same test parameter settings

others. In order to answer this question, we evaluated the Spearman correlation coefficient between true and predicted performance for 250 randomly-sampled test parameter settings. This value lies on the interval [-1,1], with 1 indicating perfect correlation of the predicted and the true ranks, 0 indicating no correlation, and -1 perfect anti-correlation.

The results of this analysis for our five test cases are summarized in Fig. 3. Overall, for the original untransformed data we observed little variation in predictive quality due to the procedure used for constructing the initial design. This observation is consistent with others that have been made in the literature; for example, Santner et al (2003, p.149) state: "It has not been demonstrated that LHDs are superior to any designs other than simple random sampling (and they are only superior to simple random sampling in some cases)."

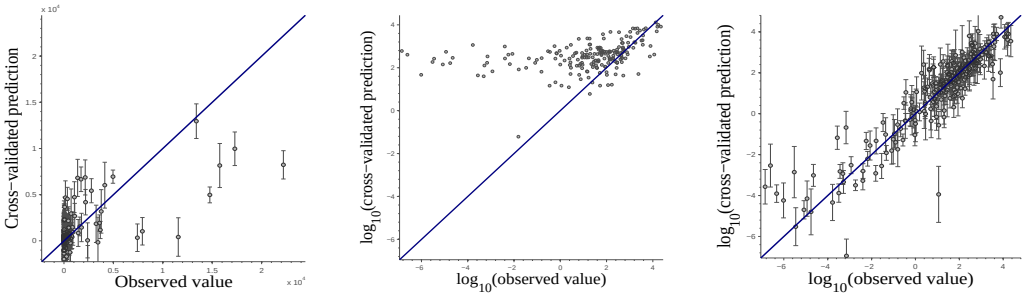
4.2 Transforming Performance Data

The second issue we investigated was whether more accurate models could be obtained by using log-transformed performance data for building and updating the model. Our consideration of this transformation was motivated by the fact that our main interest is in minimizing positive functions with spreads of several orders of magnitude that arise in the optimization of runtimes. Indeed, we have used log-transformations for predicting runtimes of algorithms in different contexts before (see., e.g., Leyton-Brown et al (2002)). All of the test functions we consider for CMA-ES are also positive functions; in general, non-positive functions can be transformed to positive functions by subtracting a lower bound. In the context of model-based optimization, log transformations were previously advocated by Jones et al (1998). The problem studied in that work was slightly different in that the functions considered were noise free. We adapt Jones et al's approach by first computing performance metrics (such as median runtime) and then fitting a GP model to the log-transformed metrics. Note that this is different from fitting a GP model to the log-transformed noisy data as done by Williams et al (2000) and Huang et al (2006). As discussed earlier, the performance metric that is implicitly optimized under this latter approach is geometric mean performance, which is a poor choice in situations where performance variations change considerably depending on the parameter setting. In contrast, when first computing performance metrics and then applying a transformation, any performance metric can be optimized, and we do not need to assume a Gaussian noise model. Finally, Bartz-Beielstein et al (2008b) and Konen et al (2009) report similar results based on square-root or cube-root transformations as indeed does Leyton-Brown (2003); see also the discussion of transformations in Chap. **??** of this book.

We experimentally evaluated the impact of log transformations. For the same data as in the previous section (initial designs with 250 parameter settings, 2 repetitions each), we fit noise-free GP models based on log-transformed cost statistics. That is, for each of the 250 parameter settings, $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{250}$, we first computed the mean of the two responses, $\hat{c}(\boldsymbol{\theta}_i)$, and then constructed a model using the data $\{(\boldsymbol{\theta}_1, \hat{c}(\boldsymbol{\theta}_1)), \ldots, (\boldsymbol{\theta}_{250}, \hat{c}(\boldsymbol{\theta}_{250}))\}$. We then computed predictions for the same 250 test parameters settings used in the previous section and evaluated the Spearman correlations between predicted and true responses. As can be seen from the results reported in Fig. 4, the use of log-transformed performance data tended to result in much better model accuracy than the use of raw performance data. In some cases, the improvements were quite drastic. For example, for CMA-ES-sphere, the Spearman correlation coefficient improved from below 0.4 to above 0.9 when using models based on log-transformed performance data. Figure 5 illustrates the predictive accuracy and predictive uncertainty of these two models.

5 Sequential Experimental Design

Having studied the initial design, we now turn our attention to the sequential search for performance-optimizing parameters. Since log transformations consistently led to improved performance, and



(a) Untransformed data.

(b) Untransformed data on log-log scale; only means are plotted.

(c) Log-transformed data on log-log scale.

Figure 5: Performance of noise-free GP models for CMA-ES-sphere based on an initial design using a random LHD; in (a) and (c) we plot mean ± one standard deviation of the prediction. For better visual comparison to (c), (b) shows exactly the same mean predictions as (a), but on a log–log scale, restricted to the 228/250 data points whose predicted response values were positive

since random LHDs yielded performance comparable to that of more complex designs, we fixed these two design choices.

5.1 Intensification Mechanism

In order to achieve good results when optimizing parameters based on a noisy performance metric such as runtime or solution quality achieved by a randomized algorithm, it is important to perform a sufficient number of runs for the parameter settings considered. However, runs of a given target algorithm on interesting problem instances are typically computationally expensive. There is thus a delicate trade-off between the number of algorithm runs performed on each parameter setting and the number of parameter settings considered over the course of the optimization process.

Realizing the importance of this trade-off, SPO implements a mechanism for gradually increasing the number of runs to be performed for each parameter setting during the parameter optimization process. In particular, SPO increases the number of runs to be performed for each subsequent parameter setting whenever the incumbent $\boldsymbol{\theta}_{inc}$ selected in an iteration was already the incumbent in some previous iteration. SPO 0.3 (Bartz-Beielstein et al, 2005; Bartz-Beielstein, 2006; Bartz-Beielstein and Preuss, 2006) doubles the number of subsequent target algorithm runs whenever this happens; SPO 0.4 only increments the number of runs by one each time; see Procedures 10 and 11 in Sect. 3.2.4. Both versions perform additional runs for the current incumbent, $\boldsymbol{\theta}_{inc}$, to make sure it is used in as many runs of the target algorithm as any new parameter setting. [5]

While the intensification mechanisms of SPO 0.3 and SPO 0.4 work well in most cases, we have encountered runs of SPO in high-noise scenarios in which there are many parameter settings with few, "lucky" target algorithm runs that allow them to become incumbents. In those runs of SPO, a new incumbent was picked in almost every iteration, because the previous incumbent had been found to be poor after additional runs were performed on it. This situation continued throughout the entire parameter optimization process, leading to a final choice of parameter settings that had only been evaluated using very few ("lucky") target algorithm runs and that performed poorly in

[5] Another approach for allocating an appropriate number of target algorithm runs to each parameter setting is Chen et al (2003)'s *optimal computational budget allocation* (OCBA). Lasarczyk (2007) implemented OCBA as an explicit intensification method to improve SPO's selection procedure, especially in high-noise scenarios. This implementation was done in R (Ihaka and Gentleman, 1996), and forthcoming versions of SPOT, which will also be based on R, will include OCBA.

Procedure 12: Intensify(Θ_{new}, θ_{inc}, $\mathscr{M}$, history) **in SPO$^+$**
Recall that $N(\theta)$ denotes the number of algorithm runs which have been performed for θ; it depends on the history

```
1  for all θ ∈ Θ_new do
2      r ← 1;
3      history ← ExecuteRuns(history, θ, 1);
4      numBonus ← 1;
5      if N(θ) > N(θ_inc) then
6          history ← ExecuteRuns(history, θ_inc, 1);
7          numBonus ← 0;
8      while true do
9          if ĉ(θ) > ĉ(θ_inc) then
               // ===== Reject challenger, perform bonus runs for θ_inc
10             history ← ExecuteRuns(history, θ_inc, min(numBonus, maxN − N(θ_inc)));
11             break;
12         if N(θ) ≥ N(θ_inc) then
               // ===== Challenger becomes incumbent
13             θ_inc ← θ;
14             break;
15         if TerminationCriterion() then
16             return [θ_inc, history];
17         r ← min(2r, N(θ_inc) − N(θ));
18         history ← ExecuteRuns(history, θ, r);
19         numBonus ← numBonus + r;
20 return [history, θ_inc];
```

independent test runs.[6]

This observation motivated us to introduce a different intensification mechanism that guarantees increasing confidence about the performance of the parameter settings we select as incumbents. In particular, inspired by the mechanism used in FocusedILS (Hutter et al, 2007), we maintain the invariant that we never choose an incumbent unless it is the parameter setting with the most function evaluations. Promising parameter settings are given additional function evaluations until either they cease to appear promising or they receive enough function evaluations to become the new incumbent. We provide pseudocode for this new intensification mechanism in Procedure 12.

In detail, our new intensification mechanism works as follows. In the first iteration, the incumbent is chosen exactly as in SPO, because all parameter settings receive the same number of function evaluations. From then on, in each iteration we select a set of parameter settings and compare them to the incumbent θ_{inc}. We denote the number of runs that have so far been executed with parameter setting θ as $N(\theta)$, and the corresponding empirical performance as $\hat{c}(\theta)$. For each selected setting θ, we iteratively perform runs until $N(\theta) \geq N(\theta_{inc})$ and/or $\hat{c}(\theta) > \hat{c}(\theta_{inc})$.[7] Whenever we reach a point where $N(\theta) \geq N(\theta_{inc})$ and $\hat{c}(\theta) \leq \hat{c}(\theta_{inc})$, we select θ as the new incumbent. On the other hand, if we

[6] One possible explanation of this scenario is that SPO has problems when the number of LHD points is large (e.g., $d = 250$ in our experiments) in high-noise scenarios. SPO selects its incumbent as the previously visited parameter setting with the best empirical performance across the runs performed with it. All parameter settings in the LHD count as "previously visited". Thus, the larger the LHD, the more runs need to be performed in order for decisions about the incumbent to be based on a minimum number of runs.

[7] We batch runs to reduce overhead, starting with a single new run for each θ and doubling the number of new runs iteratively up to a maximum of $N(\theta_{inc}) - N(\theta)$ runs.

Procedure 13: SelectNewParameterSettings($\mathscr{M}, \theta_{inc}$, history) in **SPO**$^+$
Recall that p and m are parameters of SPO$^+$. We use $m = 1$ and $p = 5$ in our experiments

```
// ===== Select m parameter settings with expected improvement
Θ_rand ← set of 10,000 elements drawn uniformly at random from Θ;
for all θ ∈ Θ_rand do
    [μ_θ, σ²_θ] ← Predict(M, θ);
    EI(θ) ← Compute expected improvement criterion given μ_θ and σ²_θ;
Θ_new ← the m elements θ of Θ_rand with highest EI(θ);
// ===== Select p previously used parameter settings
Θ ← ⋃_{i=1}^{n} {history.θ_i};
Θ_previous ← p elements θ ∈ Θ, drawn without repetitions with prob. proportional to 1/ĉ(θ);
return Θ_new ∪ Θ_previous;
```

Table 4: The p-values for pairwise comparisons of different intensification mechanisms for optimizing the performance of CMA-ES on our standard benchmarks. These p-values correspond to the data in Fig. 6 and are based on a pairwise Max-Wilcoxon test as described in Sect. 2

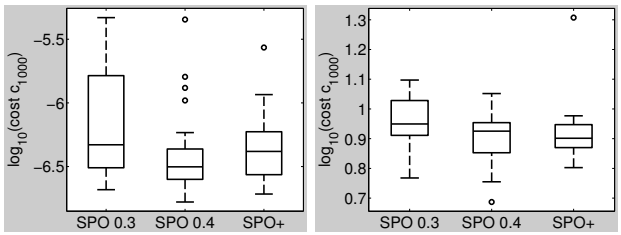
	Sphere	Ackley	Griewangk	Rastrigin
SPO 0.3 versus SPO 0.4	0.07	**0.015**	0.95	1
SPO 0.3 versus SPO$^+$	0.20	**0.020**	**0.00006**	**0.0005**
SPO 0.4 versus SPO$^+$	0.56	0.97	**0.00009**	**0.0014**

ever observe $\hat{c}(\theta) > \hat{c}(\theta_{inc})$, we reject θ. Note this criterion for rejection is very aggressive. Indeed, rejection frequently occurs after a single run, at a point where a statistical test would not be able to conclude that θ is worse than θ_{inc}. Upon rejecting a configuration θ, we also perform as many additional runs for θ_{inc} as were just performed for evaluating θ. This ensures that the number of runs used for intensification is comparable to that used for exploration of new parameter settings.

The parameter settings we evaluate against θ_{inc} at each iteration include one new parameter setting selected based on an expected improvement criterion (here $E[I^2]$, see Sect. 5.2). They also include p previously evaluated parameter settings $\theta_{1:p}$, where p is an algorithm parameter and in this work always set to 5. This set is constructed by selecting p previously evaluated settings θ with probability proportional to $1/\hat{c}(\theta)$, without replacement. Procedure 13 provides pseudocode for this selection of parameter settings to evaluate against θ_{inc}.

This mechanism guarantees that at each step there will be a positive probability of reevaluating a potentially optimal parameter setting after it has been rejected. It allows us to be aggressive in rejecting new candidates, since we can always get back to the most promising ones. Note that if the other SPO variants (0.3 and 0.4) discover the true optimal parameter setting θ_{best} but observe one or more very "unlucky" runs on it, θ_{best} will never be revisited, because the underlying noise-free GP model attributes a high mean and zero uncertainty to any previously visited parameter setting for which poor empirical performance has been observed across the target algorithm runs performed for it. Therefore, no expected improvement criterion will select such a parameter setting again in later iterations.

We denote as SPO$^+$ the variant of SPO that uses a random LHD, log-transformed data (for positive functions only; otherwise untransformed data), expected improvement criterion $E[I^2]$, and the new intensification criterion just described. We compared SPO 0.3, SPO 0.4, and SPO$^+$—all based on a random LHD and log-transformed data—for our CMA-ES test cases and summarize the results in Fig. 6 and Table 4. For the Ackley function, SPO 0.4 performed best on average, but



(a) CMAES-Sphere (b) CMAES-Ackley

(c) CMAES-Griewangk (d) CMAES-Rastrigin

Figure 6: Comparison of different intensification mechanisms for optimizing CMA-ES performance. We show boxplots of solution cost $c_{1,000}$ achieved in the 25 runs of each parameter optimizer for each test case

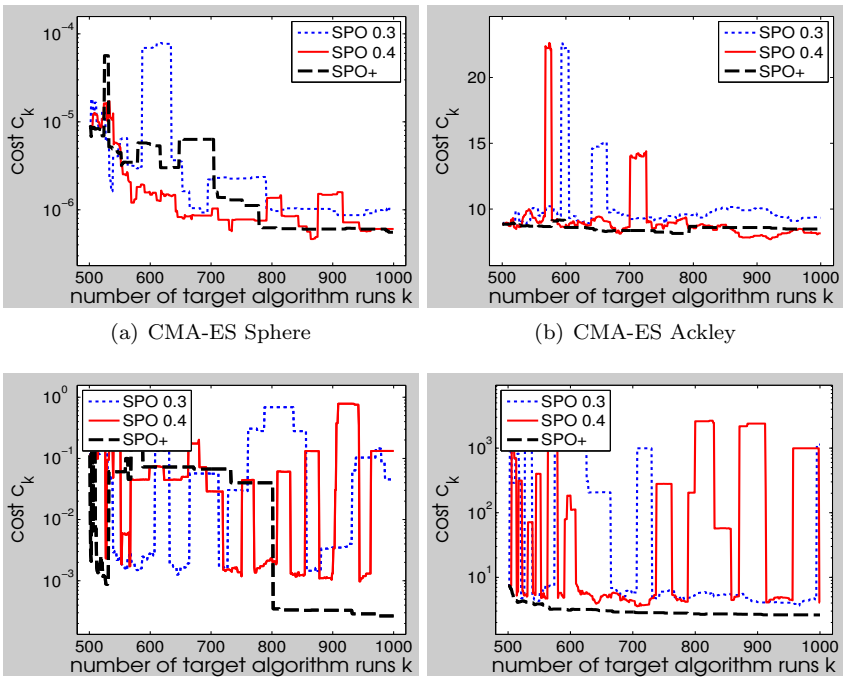
only insignificantly better than SPO$^+$, one of whose runs performed quite poorly. For the Sphere function, on average SPO$^+$ performed insignificantly better than the other SPO variants, showing better median performance and no poor outliers among its 25 runs. For the other two functions, SPO$^+$ performed both significantly and substantially better than either SPO 0.3 or SPO 0.4, finding parameter settings that led to CMA-ES performance orders of magnitude better than those obtained from SPO 0.3.

More importantly, as can be seen in Fig. 7, over the course of the optimization process, SPO$^+$ showed much less variation in the quality of the incumbent parameter setting than the other SPO variants did. This was the case even for the Ackley function, where SPO$^+$ did not perform best on average at the very end of its trajectory, and can also be seen on the Griewangk and Rastrigin functions, where SPO$^+$ clearly produced the best results.

We now take a more in-depth look at how the mean solution costs c_k come about. To do this, for each of the four test cases, we extracted the finally-chosen parameter settings from nine automated parameter optimization runs: the best, median, and worst settings of each of SPO 0.3, SPO 0.4, and SPO$^+$, all with respect to test set performance. Recall that this test set performance is the mean solution quality across 100 test runs of CMA-ES. To obtain an independent estimate of a parameter setting's true performance, we performed an additional 100 test runs of CMA-ES for each of these nine parameter settings, with a set of random number seeds disjoint from those used in the original test set.

In Fig. 8, we plot the performance of these nine parameter settings for the 100 new seeds. (The figure also shows a parameter setting IA, found by our interactive approach. We defer its discussion until Sect. 6.) For test case CMA-ES-Sphere, all selected parameter settings performed very similarly. In test case CMA-ES-Ackley, SPO$^+$ seemed to perform slightly better than SPO 0.3 and 0.4, especially in their respective worst runs.

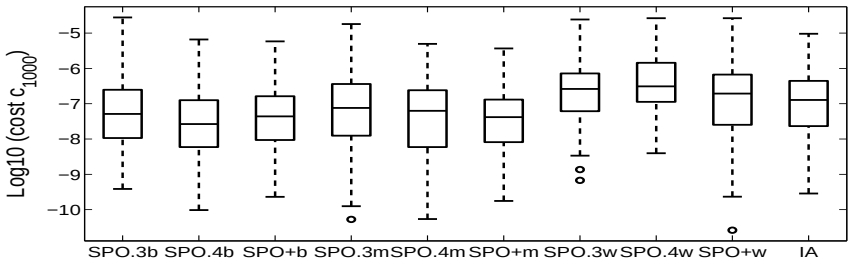
In test case CMA-ES-Rastrigin, note that most selected parameter settings yielded comparable performance, with the exception of the one identified in the worst repetition of SPO 0.3. With several runs whose performance was about five orders of magnitude worse than the remaining runs,



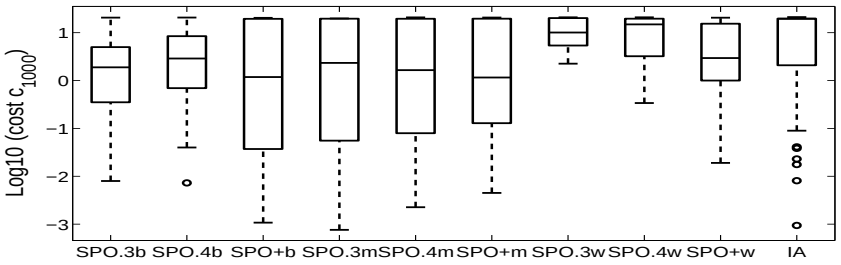
(a) CMA-ES Sphere (b) CMA-ES Ackley

(c) CMA-ES Griewangk (d) CMA-ES Rastrigin

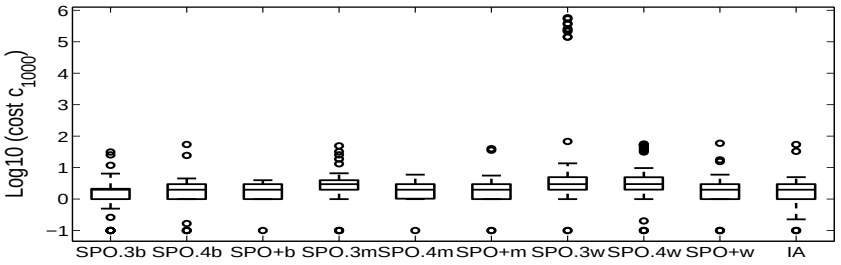
Figure 7: Solution cost c_k (mean solution quality CMA-ES achieved in 100 test runs using the method's chosen parameter settings) of SPO 0.3, SPO 0.4, and SPO$^+$, as a function of the number of target algorithm runs, k, the method is allowed. We plot means of c_k across 25 repetitions of each parameter optimization procedure



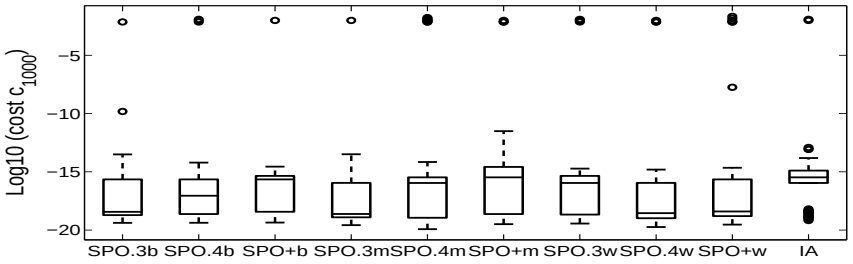
(a) CMAES-Sphere



(b) CMAES-Ackley

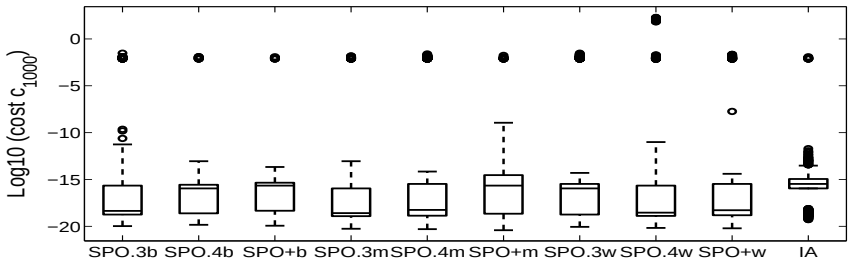


(c) CMAES-Rastrigin



(d) CMAES-Griewangk

Figure 8: Boxplots comparing automatically-identified parameter settings to those found with the interactive procedure. In the axis labels, "b" stands for the parameter setting of the best run of that parameter optimizer (with respect to "original" test performance), "m" stands for the median-best one, and "w" for the worst. IA stands for the interactively found setting. Each run of the interactive process used fewer than 200 function evaluations, whereas the automated optimization procedures were allowed 1,000 evaluations per run. In order to avoid clutter in subfigure (c), we plot data points below 0.1 as 0.1 (-1 after $\log_{10}$ transformation); note the poor performance of SPO 0.3's worst run



(a) CMAES-Griewangk, 1,000 test runs of CMA-ES used for evaluation

Figure 9: Same as Fig. 8(d), but using 1,000 instead of 100 runs of CMA-ES to evaluate each parameter setting. Note the very poor performance of some CMA-ES runs with the parameter setting from the worst run of SPO 0.4. For this parameter setting, the cloud of points with much higher solution cost than any of the other runs contains 14 points

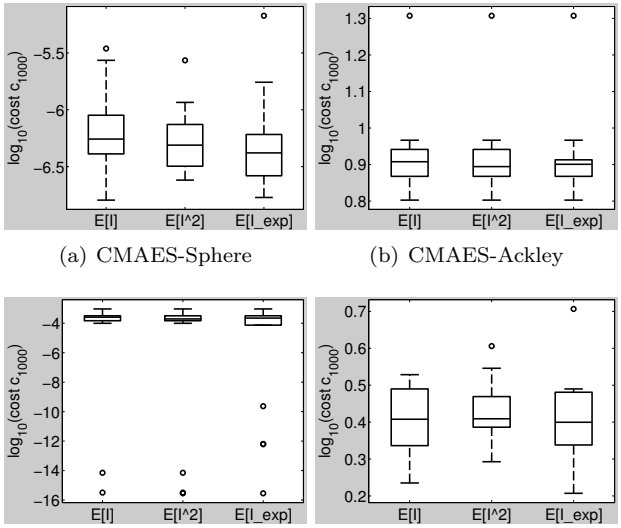
this parameter setting was responsible for the poor mean performance across the 25 runs indicated in Fig. 7(d). Note that the SPO 0.3 run returning this poor parameter setting selected it just before running out of its budget of 1,000 calls to CMA-ES. Likewise, one mostly poor SPO 0.4 run selected a good setting just before reaching the limit on target algorithm runs. As we can see in Fig. 7(d), SPO$^+$ performed much more robustly.

In test case CMA-ES-Griewangk, the situation is somewhat more complicated. While Fig. 7(c) shows very sensitive behavior of SPO 0.3 and SPO 0.4, we do not see any evidence for this in Fig. 8(d). Since the evaluation in Fig. 8(d) used 100 different random seeds (different from both the initial random seeds used during parameter optimization and from the "test" random seeds used to produce Fig. 8(d)) there was no guarantee of obtaining similar performance. Indeed, the measured performances for the parameter settings from the worst run of SPO 0.3 and 0.4 were quite different than previously observed; while they yielded poor mean performance before, they did quite well based on the new set of 100 test seeds. We repeated the evaluation with a larger set of 1,000 random seeds and show the result in Fig. 9. In this experiment, 14 of the 1,000 CMA-ES runs for the worst repetition of SPO 0.4 showed extremely poor performance. For the setting from the worst run of SPO 0.3, even these 1,000 test runs did not explain the poor performance in Fig. 7(c). To study this further, we performed $100,000$ additional test runs for this setting and found that nine of them yielded similarly poor performance as the 14/1000 poor CMA-ES for the setting above (best function values around 10^2). Another roughly 8500 runs yielded results around 10^{-2} and the rest (about $91\,500$ runs) yielded results $< 10^{-10}$. The optimization of target algorithms with such multimodal result distributions requires a large number of target algorithm runs: in the case above, a very sensitive setting performed just as well as a robust one based on as many as 1,000 runs of the target algorithm.

5.2 Expected Improvement Criterion

In sequential model-based optimization, parameter settings to be investigated are selected based on an expected improvement criterion (EIC). This aims to address the exploration/exploitation trade-off between learning about new, unknown parts of the parameter space and intensifying the search locally in the best known region. We briefly summarize two common versions of the EIC, and then describe a novel variation that we also investigated.

The classic expected improvement criterion used by Jones et al (1998) is defined as follows. Given a deterministic function f and the minimal value f_{min} seen so far, the improvement at a new



(a) CMAES-Sphere (b) CMAES-Ackley

(c) CMAES-Griewangk (d) CMAES-Rastrigin

Figure 10: Comparison of different expected improvement criteria for optimizing the performance of CMA-ES on our standard benchmarks. We show boxplots of performance c_{1000} achieved in the 25 runs of each optimizer for each test case

Table 5: The p-values for pairwise comparisons of different expected improvement criteria for optimizing the performance of CMA-ES on our standard benchmarks. These p-values correspond to the data in Fig. 10 and are based on a pairwise Max-Wilcoxon test as described in Sect. 2

	Sphere	Ackley	Griewangk	Rastrigin
$E[I]$ *vs* $E[I^2]$	0.29	0.55	**0.016**	0.90
$E[I]$ *vs* $E[I_{\text{exp}}]$	0.63	0.25	0.11	**0.030**
$E[I^2]$ *vs* $E[I_{\text{exp}}]$	0.54	0.32	0.77	0.38

parameter setting θ is defined as

$$I(\theta) := \max\{0, f_{min} - f(\theta)\}. \tag{9}$$

Of course, this quantity cannot be computed, since $f(\theta)$ is unknown. We therefore compute the expected improvement, $E[I(\theta)]$. To do so, we require a probabilistic model of f, in our case the GP model. Let $\mu_\theta := E[f(\theta)]$ be the mean and σ^2_θ be the variance predicted by our model, and define $u := (f_{min} - \mu_\theta)/\sigma_\theta$. Then one can show that $E[I(\theta)]$ has the following closed-form expression:

$$E[I(\theta)] = \sigma_\theta \cdot [u \cdot \Phi(u) + \varphi(u)], \tag{10}$$

where φ and Φ denote the probability density function and cumulative distribution function of a standard normal distribution, respectively.

A generalized expected improvement criterion was introduced by Schonlau et al (1998), who considered the quantity

$$I^g(\theta) := \max\{0, [f_{min} - f(\theta)]^g\} \tag{11}$$

for $g \in \{0, 1, 2, 3, \ldots\}$, with larger g encouraging more global search behavior. The value $g = 1$ corresponds to the classic EIC. SPO uses $g = 2$, which takes into account the uncertainty in our estimate of $I(\theta)$, since $E[I^2(\theta)] = (E[I(\theta)])^2 + \mathrm{Var}(I(\theta))$ and can be computed by the closed-form formula

$$E[I^2(\theta)] = \sigma^2_\theta \cdot [(u^2 + 1) \cdot \Phi(u) + u \cdot \varphi(u)]. \tag{12}$$

One issue that seems to have been overlooked in previous work is the interaction between log-transformations of the data and the EIC. When we use a log transformation, we do so in order to increase predictive accuracy, yet our loss function continues to be defined in terms of the untransformed data (e.g., actual runtimes). Hence we should optimize the criterion

$$I_{\exp}(\theta) := \max\{0, f_{min} - e^{h(\theta)}\}, \tag{13}$$

where $h(\cdot)$ predicts log performance and f_{min} is the untransformed best known function value.

Let $v := (\ln(f_{min}) - \mu_\theta)/\sigma_\theta$. Then, we have the following closed-form expression (see the appendix for the proof):

$$E[I_{\exp}(\theta)] \quad = \quad f_{min}\Phi(v) - e^{\frac{1}{2}\cdot\sigma^2_\theta + \mu_\theta} \cdot \Phi(v - \sigma_\theta). \tag{14}$$

In Fig. 10 and Table 5, we experimentally compare SPO$^+$ with these three expected improvement criteria on the CMA-ES test cases, based on a random LHD and log-transformed data. Overall, the differences are small. On average, $E[I^2]$ yielded the best results for test case CMA-ES-sphere, and our new criterion $E[I_{\exp}]$ performed best in the remaining cases. Even though not visually obvious from the boxplots, 2 of the 12 pairwise differences were statistically significant based on a Max-Wilcoxon test.

5.3 Overall Evaluation

In Sects. 5.1 and 5.2, we fixed the design choices of using log transformations and initial designs based on random LHDs. Now, we revisit these choices. Using our new SPO$^+$ intensification criterion and expected improvement criterion $E[I^2]$, we studied how much the final performance of SPO$^+$ changed when not using a log-transformation and when using different methods to create the initial design. Not surprisingly, none of the initial designs led to significantly better final performance than any of the others. The result for the log transformation was more surprising. Although we saw in Sect. 4 that the log transformation consistently improved predictive model performance, based on a Mann-Whitney U test it turned out to significantly improve *final* parameter optimization performance only for CMA-ES-sphere.

Table 6: Comparison of final performance of various parameter optimization procedures for optimizing SAPS on instance QWH. We report mean ± standard deviation of performance c_{20000} (median search steps SAPS required on instance QWH in 1,000 test runs using the parameter settings the method chose after 20,000 algorithm runs), across 25 repetitions of each method. Based on a Mann-Whitney U test, SPO$^+$ performed significantly better than CALIBRA, BasicILS, FocusedILS, and SPO 0.3 with p-values 0.015, 0.0002, 0.0009, and $4 \cdot 10^{-9}$, respectively. The p-value for a comparison against SPO 0.4 was 0.06

Procedure	SAPS median runtime [search steps]
SAPS default from Hutter et al (2002)	$85.5 \cdot 10^3$
CALIBRA(100) from Hutter et al (2007)	$10.7 \cdot 10^3 \pm 1.1 \cdot 10^3$
BasicILS(100) from Hutter et al (2007)	$10.9 \cdot 10^3 \pm 0.6 \cdot 10^3$
FocusedILS from Hutter et al (2007)	$10.6 \cdot 10^3 \pm 0.5 \cdot 10^3$
SPO 0.3	$18.3 \cdot 10^3 \pm 13.7 \cdot 10^3$
SPO 0.4	$10.4 \cdot 10^3 \pm 0.7 \cdot 10^3$
SPO$^+$	$\mathbf{10.0 \cdot 10^3 \pm 0.4 \cdot 10^3}$

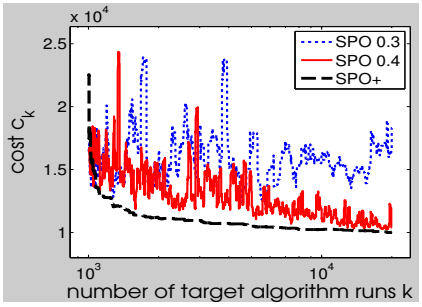


Figure 11: Comparison of SKO and two variants of SPO (discussed in Sect. 5.1) for optimizing CMA-ES on the Sphere function. Comparison of SPO variants (all based on a random LHD and log-transformed data) for minimizing SAPS median runtime on instance QWH. We plot the solution cost c_k of each method (median search steps SAPS required on instance QWH in 1,000 test runs using the parameter settings the method chose after k algorithm runs), as a function of the number of algorithm runs, k, it was allowed to perform. These values are averaged across 25 runs of each method

Finally, we compared the performance of SPO 0.3, 0.4, and SPO$^+$ (all based on random LHDs and using log-transformed data) to the parameter optimization methods studied by Hutter et al (2007). We summarize the results in Table 6. While SPO 0.3 performed worse than the other methods, SPO 0.4 performed comparably, and SPO$^+$ outperformed all methods with the exception of SPO 0.4 significantly. Figure 11 illustrates the difference between SPO 0.3, SPO 0.4, and SPO$^+$ for this SAPS benchmark. Similar to what we observed for CMA-ES (Fig. 7), SPO 0.3 and 0.4 changed their incumbents very frequently, with SPO 0.4 showing more robust behavior than SPO 0.3, and SPO$^+$ in turn much more robust behavior than SPO 0.4.

6 Interactive Exploration of Parameter Space

The automated methods discussed so far in this chapter can be very effective when it is possible to perform a relatively large number of evaluations of the given target algorithm to be optimized. However, there are cases in which target algorithm evaluations are overwhelmingly costly compared to the overall computational resources and time available for the parameter optimization process. Real-world applications of this nature can be found in the optimization of engineering designs in the aerospace and automotive industry (Alexandrov et al, 2001), in hydrological applications (Mayer et al, 2002), and in climate modeling. In those examples, the algorithms to be optimized control or model the behavior of complex systems, and evaluating their performance for a single parameter configuration can take hundreds of CPU hours. In such cases, the number of parameter configurations that can be evaluated is severely limited, and approaches that achieve good results based on a small number of evaluations are required. In the following, we explore an interactive approach for model-based parameter optimization that utilizes human judgement and insight in conjunction with statistical methods.

6.1 Using SPOT Interactively

The sequential parameter optimization toolbox (SPOT) was developed to improve the performance of algorithms and to gain insight into their working mechanisms (Beielstein, 2003). When using SPOT interactively, the experimenter makes use of the statistical analysis techniques supported by the toolbox to sequentially select new settings for the given target algorithm. The general flow of the interactive sequential parameter optimization process is illustrated in Fig. **??** on page **??**. We will illustrate this interactive approach by means of a case study, in which we focus primarily on the performance of CMA-ES on the Rastrigin function. This function was chosen because the previously studied, fully-automated SPO procedures SPO 0.3 and SPO 0.4 performed relatively poorly compared to SPO$^+$(see, e.g., Fig. 1). The interactive approach might shed some light on this poor performance, since it provides tools for understanding the structure of the search space (ROI) and the determination of factor effects. Later, we also report results for CMA-ES on the other three classical test problems from global optimization introduced in Sect. 2 (Sphere, Griewangk, and Ackley). We note that, while in principle, the interactive approach can be applied to the optimization of any objective function, the linear regression models used in the following fit arithmetic mean performance. Further, although experimenters working in the context of real-world performance optimization tasks often rely upon prior knowledge about the target algorithm, here we assume, for the sake of generality, that no prior knowledge is available regarding important parameters, optimal experimental designs or regression models (predictors) for the target algorithm. We do, however, assume some general knowledge about reasonable ranges for parameters of evolutionary strategies, such as CMA-ES.

6.1.1 Pre-experimental Planning

In the pre-experimental planning phase, we have to select an initial design, a predictor, and a quality measure. Our goal is to improve CMA-ES, which requires the specification of the four parameters `NPARENTS`, `NU`, `CS`, and `DAMPS`. As a rule of thumb, we assert that the experimenter should not invest more than about a quarter of the available budget (here the number of CMA-ES runs) in the first design.[8] In most circumstances, it is unwise to plan too comprehensive a design at this outset (Box et al, 1978).

In light of our goal of keeping the number of configurations to be evaluated relatively small, we have chosen a *Box–Behnken design* as the initial design (Box and Behnken, 1960; Pukelsheim, 1993). Box–Behnken designs are central composite designs that augment 2^k designs with center points (see also the discussion of designs in Chap. **??** of this book). Box–Behnken designs can be used to calibrate full quadratic models; furthermore, they are rotatable and, when the number of factors is four or fewer, require fewer runs than central composite designs. (Chapts. **??** and **??** in this book discuss further design considerations.) Box–Behnken designs avoid the corners of the region of interest and allow experimenters to work around extreme factor combinations. This is especially important for the optimization of CMA-ES, because large `NPARENTS` and large `NU` values at the same time are expected to produce poor results. The four-factor Box–Behnken design used in the first step of our analysis requires 27 runs (Pukelsheim, 1993).

We now discuss how the region of interest (ROI) was determined for our experiments. We note that this determination of the ROI settings is an integral part of the interactive approach and more elaborate than in the previously discussed automated approach. Imagine that a single run of CMA-ES has a budget of $t = 10,000$ function evaluations. (Note that here we discuss the number of function evaluations performed by *every CMA-ES run*, not the number of CMA-ES parameter settings that can be evaluated by the experimenter.) As a rule of thumb, the smallest number of generations for evolutionary algorithms that use some step-length adaptation is $g = 10$. This gives an upper limit for the population size of $\texttt{NPARENTS} \times \texttt{NU} = 1,000$, so $\texttt{NPARENTS}_{\max} = 100$ and $\texttt{NU}_{\max} = 10$ are reasonable values. The lower bounds of the region of interest for these two variables was chosen as $\texttt{NPARENTS}_{\min} = 2$ and $\texttt{NU}_{\min} = 2$. Since no information about reasonable settings for `CS` and `DAMPS` is known, we have chosen the maximum interval length, i.e., $\texttt{CS} \in [0.1; 1]$ and $\texttt{DAMPS} \in [0.25; 0.99]$. In general, we recommend liberally-chosen intervals in the absence of prior knowledge justifying tighter bounds.

6.1.2 Prediction Model

First, we consider a linear model without interactions. We apply a logarithmic transformation because it improves the fit. The transformation is motivated by model diagnostics, e.g., residuals plotted versus predicted values and histogram plots of the data. Our regression modeling is based on coded variables, i.e., $\{-1, 0, +1\}$, rather than on the natural variables of the target algorithm. For example, consider a region of interest for some variable in the range from one to ten. The coded variables $\{-1, 0, +1\}$ correspond to the natural variables $\{1, 5.5, 10\}$. The linear regression model is given by

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \varepsilon,$$

where the x_i's are explanatory variables (or predictors) representing `NPARENTS`, `NU`, `CS`, and `DAMPS`, and the β_i's can be estimated using the method of least squares. Further details are presented in the Appendix to this book. Here, y denotes the function value produced by the run of the target

[8] Note, however, that there are exceptions to this rule. For example, Bartz-Beielstein and Preuss (2006) state:

> The experimental analysis clearly demonstrated that the determination of a suitable initial design is of crucial importance for the second phase, which performs a local tuning. To play safe, we recommend increasing the number of initial design points. The number of sequential optimization steps could be reduced in many situations without a significant performance loss.

```
Stepwise Model Path
Analysis of Deviance Table
Initial Model:
Y ~ NPARENTS + NU + DAMPS + CS
Final Model:
Y ~ NPARENTS + NU + DAMPS + CS + NPARENTS:NU + DAMPS:CS +
    NU:CS
           Step Df Deviance Resid. Df Resid. Dev      AIC
1                                  22   57.95982 30.62566
2 + NPARENTS:NU  1 7.082627        21   50.87720 29.10660
3  + CS:DAMPS  1 5.881372        20   44.99582 27.78979
4     + NU:CS  1 4.350343        19   40.64548 27.04437
```

Figure 12: Output from R's `stepAIC()` procedure

algorithm (CMA-ES). The fitted least squares equation we obtained based on data from the first 27 runs was

$$\hat{y} = 3.8 + 0.93x_1 + 1.37x_2 - 1.84x_3 + 0.39x_4.$$

In addition, we also performed visual inspections, e.g., on the basis of added-variable plots (see also Chap. **??**). Box and Draper (1987) mention some elementary checks for interaction and curvature. For example, a comparison of the average y_c at the center of the design with the average of the remaining points of the Box–Behnken design y_{-c} gives a measure of the overall curvature of the response surface.

6.1.3 Model Selection

Next, we check whether interaction terms should be included into the model. This is done by increasing model complexity in a stepwise manner. R's `stepAIC()` function is used for performing model searches by the *Akaike information criterion* (AIC) (Venables and Ripley, 2002). Here, smaller AIC values are better. The function `stepAIC()` can be used for an automated stepwise selection procedure. It requires a fitted model to define the starting process and a list of two formulae defining the most complex and the simplest models. We have chosen the linear model with two-factor interactions as the most complex model, and the model which includes the four main factors only as the simplest model. The automated search leads to a model which, in addition to the four main factors, includes interactions between `NPARENTS` and `NU`, `DAMPS` and `CS`, and `NU` and `CS`. Figure 12 shows the output from this analysis. We note that it is easy to be misled by this automated model search, and experience shows that different variables could be selected if the `stepAIC()` procedure were repeated on a new, similar data set (Dalgaard, 2002). In many cases, the decision between models cannot be based on the data alone, but should take into consideration results from previous investigations or theoretical considerations. Therefore, it is recommended that users carefully evaluate results obtained by the `stepAIC()` procedure.

In the second step of the model selection process, we analyze the enhanced model from the point of view of regression-based significance. Here, we apply R's `dropterm()` function to the final model from the `stepAIC()` procedure. Venables and Ripley (2002) note that selecting the terms on the basis of the `stepAIC()` criterion can be somewhat permissive in its choice of terms. We also observed this in our own analysis (Fig. 13).

We observe that the regression coefficients for `NPARENTS`, `NU`, and `DAMPS` are large relative to their standard errors. Based on the conventional significance level from the regression analysis, i.e., $Pr(>|t|)$, we conclude that the model `Y` $\sim$ `NPARENTS + NU + DAMPS` should be used for the steepest descent. Predictions of this model are illustrated in Fig. 14. The contour lines can be tentatively accepted as a rough estimate of the underlying response function over the region of interest explored

```
Call: lm(formula = Y ~ NPARENTS + NU + DAMPS + CS + NPARENTS:NU,
    data = df0012normlogy)
Residuals:
     Min       1Q   Median       3Q      Max
-3.79993 -0.79931  0.02917  1.04553  2.12640
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.8001     0.2996  12.686 2.59e-11 ***
NPARENTS      0.9279     0.4493   2.065 0.051483 .
NU            1.3686     0.4493   3.046 0.006142 **
DAMPS        -1.8354     0.4493  -4.085 0.000531 ***
CS          0.3886     0.4493   0.865 0.396912
NPARENTS:NU   1.3307     0.7783   1.710 0.102040
---
Residual standard error: 1.557 on 21 degrees of freedom
Multiple R-squared: 0.6175,     Adjusted R-squared: 0.5264
F-statistic:  6.78 on 5 and 21 DF,  $p$-value: 0.0006601
```

Figure 13: Result from R's `dropterm()` analysis. Starting point for this analysis is the model proposed by the stepwise selection method `stepAIC()`

so far. (Further details of the model selection based on the t-statistics are discussed on p. **??** of the Appendix to this book.)

6.1.4 Steepest Descent

We proceed with the steepest descent based on the model `Y` $\sim$ `NPARENTS + NU + DAMPS + CS` (note that we included factor `CS`; although this factor was not judged to have a significant impact by the regression model, including it also did not require any additional effort). The procedure of steepest descent is performed as described on p. **??** in Chap. **??**. Again, as the largest step width we recommend the value that leads to the border of the ROI. Here, we obtained a step width of $\delta x_4 = 0.036$ in the natural variables for `DAMPS`, leading to 11 design points until we hit the border of the ROI (`DAMPS` reaches its maximal value 0.99). Data from the steepest descent experiments are shown in Table 7. A graph of these results to determine the new region of interest following the direction of the steepest descent is shown in Fig. 15.

Table 7: Steepest descent experiment

	Y	NPARENTS	NU	TCCS	DAMPS	CONFIG
1	39.29	51	6.00	0.55	0.62	26
2	24.31	49	5.70	0.54	0.66	27
3	18.63	46	5.40	0.53	0.69	28
4	13.49	44	5.11	0.52	0.73	29
5	27.61	41	4.81	0.51	0.77	30
6	2.00	39	4.51	0.50	0.81	31
7	0.01	36	4.21	0.49	0.84	32
8	0.00	34	3.91	0.48	0.88	33
9	1.99	31	3.61	0.47	0.92	34
10	0.99	29	3.32	0.46	0.95	35
11	1.99	26	3.02	0.45	0.99	36

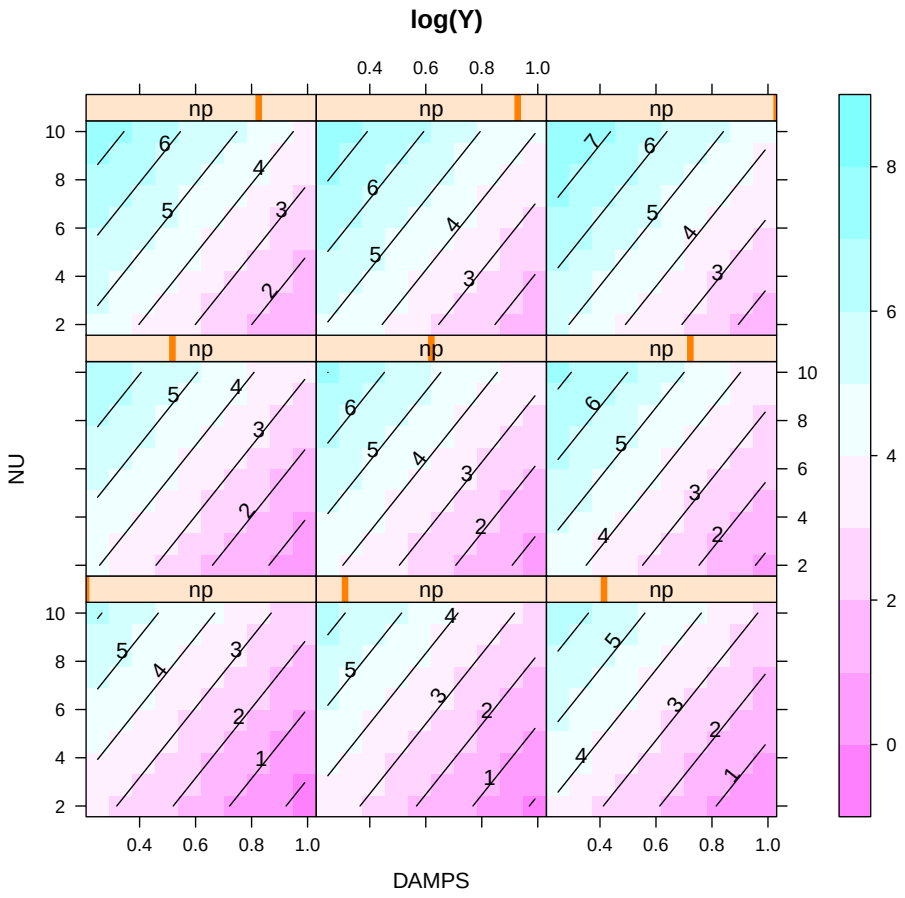


Figure 14: Data from the first design (27 runs). These data were used to determine the steepest descent. These contour plots support the assumption that the values for `DAMPS` should be increased, and values for `NU` should be decreased, whereas the impact of the population size (`NPARENTS`) is relatively small. Predicted values are based on the regression equation $\hat{y} = 3.8 + 0.93x_1 + 1.37x_2 - 1.84x_3$. Values for `NPARENTS` are taken from the interval $[2, 100]$, which was split into nine subintervals. The slider on top of each panel indicates the value of population size

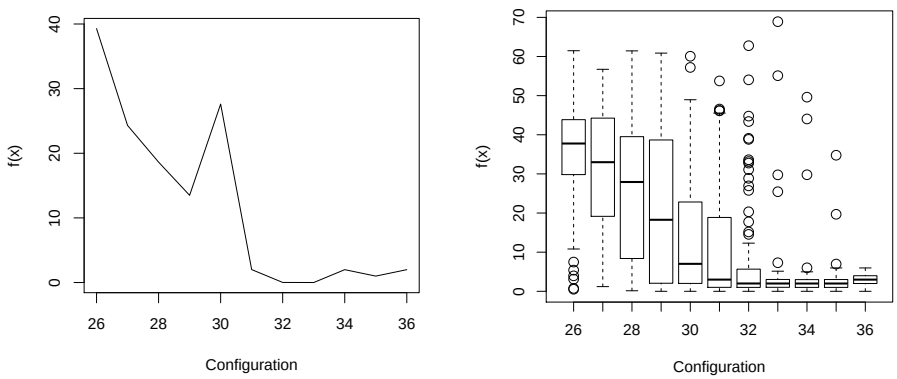


Figure 15: *Left:* Function values $f(x)$ (Rastrigin) versus steps along the path of the steepest descent. Indices denote the eleven steps from the steepest descent. Note that these values are based on one repeat only, so variation in the data, e.g., the peak (index 5), is not surprising. These data were used during the interactive approach. *Right:* Boxplots showing the variance of the data used for the steepest descent. Same situation as on the *left*, but 100 repeats of each CMA-ES configuration. These experiments were performed after the CMA-ES tuning was finished. Information from these runs was not used to determine the tuned CMA-ES parameter set

These settings are used for additional runs of CMA-ES. Figure 15 plots the yield at each step along the path of the steepest descent. Based on visual inspection of the yields in Fig. 15, the new central point was determined to be the eighth point of the steepest descent since no significant decrease occurred during steps nine through eleven. Furthermore, this new central point leaves some space for variation of the `DAMPS` values, say, in the interval $[0.8, 0.99]$.

6.1.5 Second Model and Steepest Descent

Based on the best value obtained with the steepest descent, we build a new model with center point

$$\vec{x}_c = [\texttt{NPARENTS}, \texttt{NU}, \texttt{CS}, \texttt{DAMPS}] = [34, 3.91, 0.48, 0.88].$$

The specification of the new region of interest requires user knowledge. The new center point was determined by interpreting graphical results based on the steepest descent. Next, we have to determine a new region around $\vec{x}_c$. Sometimes, especially when a classical factorial design is used during the first step, it can be useful to increase the region of interest at this stage. However, we have chosen a Box–Behnken design for the first step and therefore have to decrease the region of interest. As a rule of thumb, to be reconsidered on a case-by-case basis, we use at least $\pm$ 1/5th of the values at the new central point. For example, if the value of the new population size `NPARENTS` is 50, we define a new region of interest for this values as the interval $[40, 60]$. Here, the new region of interest reads as follows: $\texttt{NPARENTS} \in [24, 44]$, $\texttt{NU} \in [3, 5]$, $\texttt{CS} \in [0.4, 0.6]$, and $\texttt{DAMPS} \in [0.8, 0.99]$.

Again, a Box–Behnken design with 27 points is used to set up a regression model to determine the path of the steepest descent. Steps along this path are performed until no improvement is obtained. Note that five repeats are used now (previous experiments used only one repeat). The path among the steepest descents consists of 14 steps. Thus, 70 experiments are performed. The final configuration reads

$$[\texttt{NPARENTS}, \texttt{NU}, \texttt{CS}, \texttt{DAMPS}] = [34, 3, 0.43, 0.98].$$

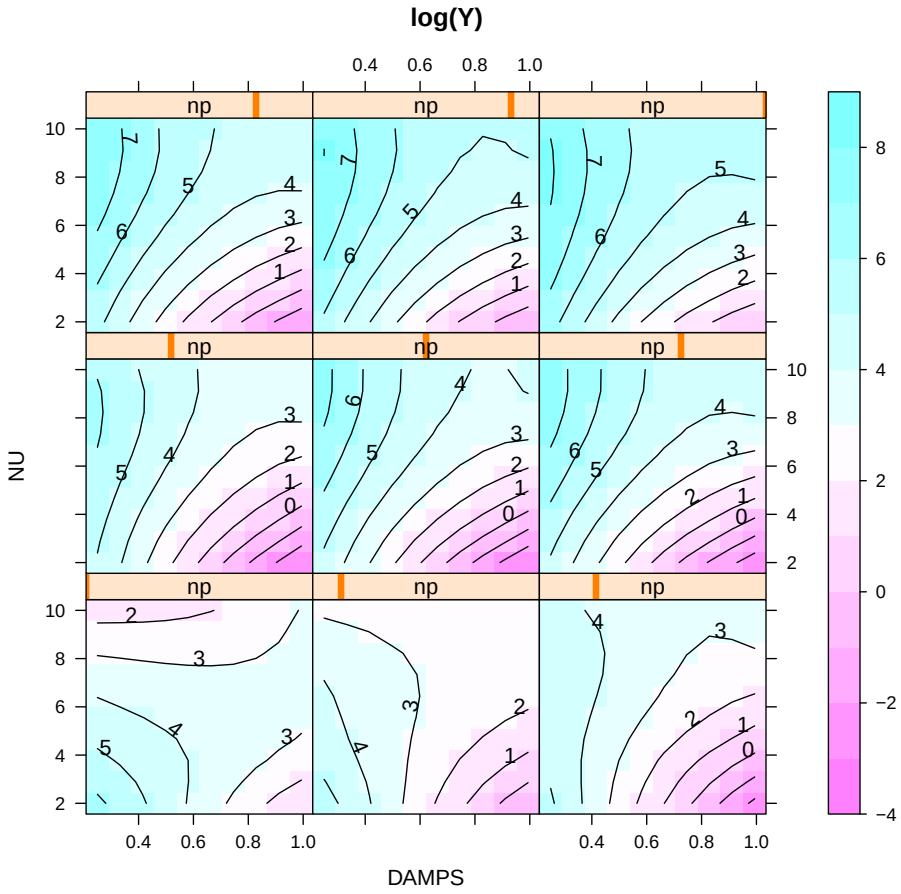


Figure 16: Contour plot based on the complete data set (CMAES-Rastrigin with 135 function evaluations). Smaller values are better. Better configurations are placed in the lower right corner of the panels. The factor which has the smallest effect, `CS`, is held constant. `NU` is plotted versus `DAMPS`, while values of the factor with the second smallest effect, namely `NPARENTS` (np), are varied with the slider on top of each panel

6.1.6 Final Exploration

Finally, we use graphical tools to get an overview of the experimental region used in our experiments. Figure 16 displays a fit of the response surface which is based on the complete data set. A local regression model based on R's `loess()` function is fitted to the data.

Altogether 135 (= 27 + 11 + 27 + 70) runs of CMA-ES were used. The experiments were performed on a 2.3 GHz Pentium 4 with 4 GB RAM running MatlabVersion 7.6 on a Linux system. The SPOT runs for the interactive exploration required 39 seconds. Writing the reports and setting up the R scripts for the interactive exploration took approximately one hour. Our experience from working on real-world problems indicates that one working day is necessary to perform the complete SPOT process if applied to a new simulation or optimization algorithm. This includes discussions with domain experts to define performance criteria and the specification and implementation of SPOT interfaces. Substantially more time might be required in cases where target algorithm runs are very costly.

6.2 Further Interactive Tuning Results

We also applied our interactive tuning approach to the other three CMA-ES scenarios introduced in Sect. 2. Here we briefly summarize the results.

6.2.1 Interactive Tuning of CMA-ES on the Sphere Function

Our experiments with CMA-ES on the Sphere function used a different setup (i.e., different values of starting point, dimension, and number of function evaluations); see Table 2. However, regarding the region of interest, the same initial settings as reported in Sect. 6.1.1 were used. Again, a Box–Behnken design was generated, and 27 runs of CMA-ES were performed. Based on the results from these runs, the following regression model was fitted:

$$\hat{y} = 2.13 + 9.25x_1 + 1.13x_2 + 0.003x_3 - 0.74x_4.$$

The regression analysis revealed that the value for `NPARENTS` should be decreased. The regression model as described in Sect. 6.1.2 and its refinement by steepest descent produced

$$[\texttt{NPARENTS}, \texttt{NU}, \texttt{CS}, \texttt{DAMPS}] = [2.0, 3.5, 0.6, 0.9].$$

Note the drastic change in the `NPARENTS` values, whereas other predictors are only slightly modified. Here, following the direction of the steepest descent resulted in a significant improvement of CMA-ES's performance. The function value could be reduced from 7132 to 1.40e-05, and—in a repeat of the steepest descent—from 5862 to 1.27e-06. Since we reached a region with small function values and relatively little variation, we decided to stop the procedure at this stage and perform a visual inspection based on contour plots (see Fig. 17). This required no additional function evaluations.

The final configuration for the sphere function reads

$$[\texttt{NPARENTS}, \texttt{NU}, \texttt{CS}, \texttt{DAMPS}] = [2.0, 3.5, 0.6, 0.9].$$

Altogether 49 ($= 27 + 2 \times 11$) runs of CMA-ES were used to produce this result.

6.2.2 Interactive Tuning of CMA-ES on the Ackley Function

The same initial settings as reported in Sect. 6.1.1 were used for the interactive tuning of the Ackley function. Again, we generated a Box–Behnken design and performed 27 runs of CMA-ES. Based on the results from these runs, the following regression model was obtained (through fitting, as previously described):

$$\hat{y} = 1.75 + 0.88x_2 + 0.21x_3 - 0.61x_4.$$

Compared to the fit of the functions considered so far, where the regression model showed p-values smaller than 0.01 (e.g., a value of 0.0008 in the first regression model), the fit of the regression model was relatively poor (p-value 0.2). This behavior can be explained as follows. Running CMA-ES on Ackley's function with parameters from the initial ROI produces many outliers which disturb the modeling process. CMA-ES generates good (< 5) and bad (> 15) solutions with the same parameter setting. The SPOT tuning procedure applied in this study is based on mean values. Figure 17 illustrates difficulties arising from this situation. The RSM is based on mean values and produces unhelpful gradient information. There is a relatively large gap between good and bad solutions. Note that the mean lies exactly in this gap, so it represents a value that is never realized. We believe that in this case tuning mean performance might not yield the most meaningful results. Instead, one could use the trimmed distribution; this will be subject of forthcoming studies.

We nevertheless present results from the interactive approach to complete our study. A contour plot of the predicted function values of the CMA-ES for parameter values from the region of interest

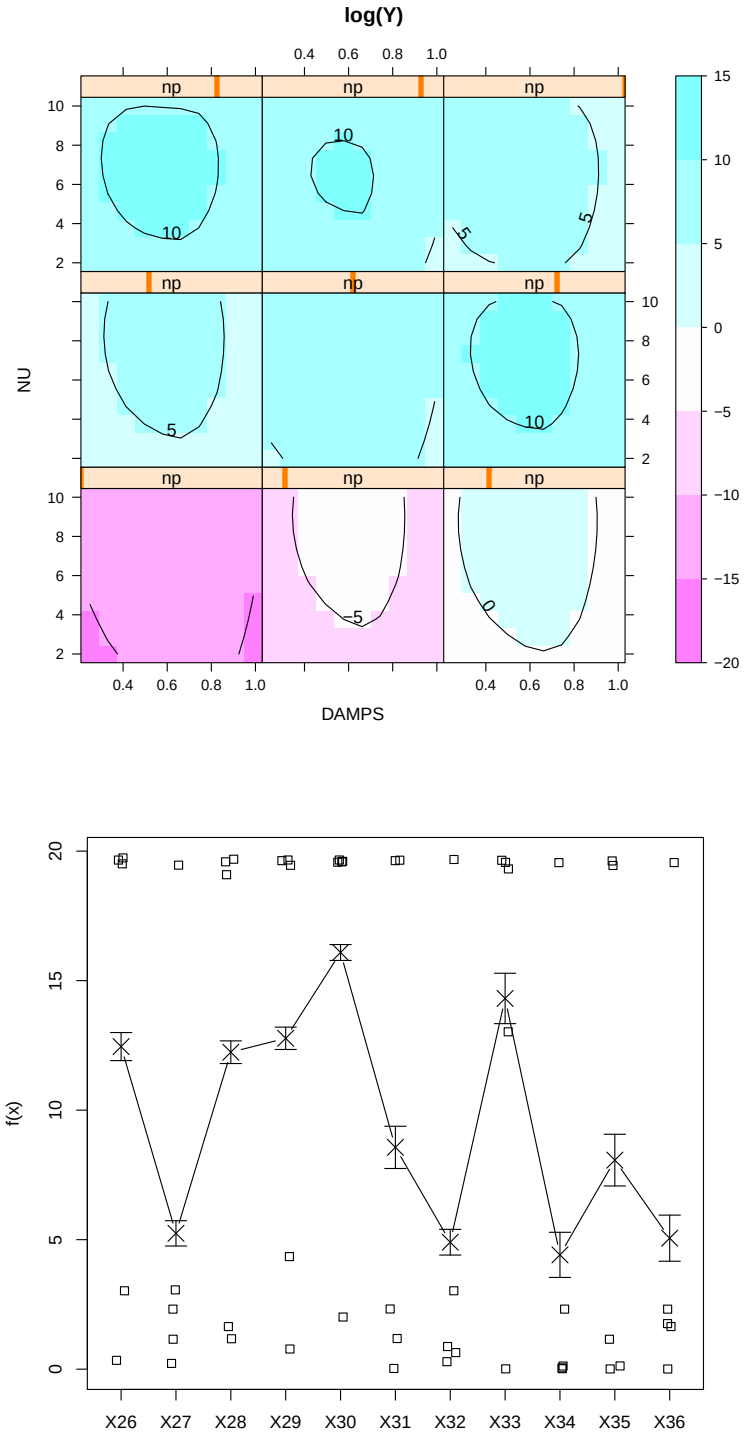


Figure 17: *Top:* Contour plot based on the complete data set (CMAES-Sphere, 49 data points). Smaller values are better. The factor which has the smallest effect, `CS`, is held constant. `NU` is plotted versus `DAMPS`, while values of the factor with the last but one effect, namely `NPARENTS` (np), are varied with the slider on top of each panel. *Bottom:* Function values $f(x)$ (Ackley) versus steps along the path of the steepest descent. Indices denote the eleven steps from the steepest descent. Note that these values are based on five repeats

Table 8: Performance comparison of the parameter settings found by our automatic and interactive approaches. We give median and mean of solution costs c_{1000} across the 25 repetitions of SPO$^+$. For each function, we also give the number of target algorithm runs, K, used in the interactive approach and the resulting solution cost, c_K, as well as the quantile corresponding to this value in the empirical distribution of SPO$^+$ results (e.g., 56% means that 14 of 25 SPO$^+$ runs yielded better results)

Test	**25 repetitions of SPO$^+$**		**Interactive approach**		
case	median	mean ± stddev	K	sol. cost	quantile of SPO$^+$ dist.
Sphere [$\times 10^{-7}$]	4.16	5.55 ± 5.15	49	4.65	56%
Ackley	7.97	8.48 ± 2.61	82	12.25	96%
Griewangk [$\times 10^{-4}$]	1.73	2.66 ± 2.53	69	2.22	56%
Rastrigin	2.50	2.62 ± 0.51	135	2.82	68%

used in the interactive tuning study of the Ackley function is shown in Fig. 18. The final configuration reads

$$[\texttt{NPARENTS}, \texttt{NU}, \texttt{CS}, \texttt{DAMPS}] = [2, 2, 0.11, 0.55].$$

As in the case of the Sphere function, 27 initial runs were performed, followed by the evaluations on the path of the steepest descent (2× 11). To obtain more insight, we performed three additional runs during the steepest descent (3 × 11). Therefore, a total of 82 runs (= 27 + 2×11 + 3 × 11) was used in this experiment.

6.2.3 Interactive Tuning of CMA-ES on the Griewangk Function

The same initial settings as reported in Sect. 6.1.1 was used. Again, a Box–Behnken design was generated, and 27 runs of CMA-ES were performed. Based on the results from these runs, the following regression model was fitted:

$$\hat{y} = -5.65 + 0.61x_1 + 9.07x_2 - 1.12x_3 - 4.30x_4.$$

A steepest descent (with two repeats) based on this first regression model led directly to an improved CMA-ES configuration. To validate the improvement following the path of the steepest descent, we repeated the corresponding runs twice. The improved configuration reads

$$[\texttt{NPARENTS}, \texttt{NU}, \texttt{CS}, \texttt{DAMPS}] = [48, 2, 0.61, 0.80].$$

Since $49 = 27 + (11 \times 2)$ runs of CMA-ES already resulted in an improved configuration, we used 20 additional CMA-ES runs to obtain an overview of the region of interest, scanning this region based on Latin hypercube sampling. The corresponding contour plot is shown in Fig. 18. As for the Sphere function, 27 initial runs were performed, followed by the evaluations on the path of the steepest descent (2× 11). Therefore, a total of 69 runs (27 + 2×11 + 20) was used in this experiment.

6.3 Comparison of Solutions Found Automatically and Interactively

Next, we evaluate how the interactively found parameter settings compare to the automatically found ones in terms of CMA-ES performance achieved. Table 8 compares the performance of the manually identified parameter settings against the distribution of performance achieved across 25 runs of SPO$^+$. In all test cases, the median-best SPO$^+$ run achieved better performance than the manual procedure; between 56% (14/25) and 96% (24/25) of the automatically found parameter settings performed better than the one identified manually. However, recall that the manual process used many function evaluations less than the automatic procedure. Nevertheless, on two test cases, the manually identified parameter settings performed better than the automatically found settings

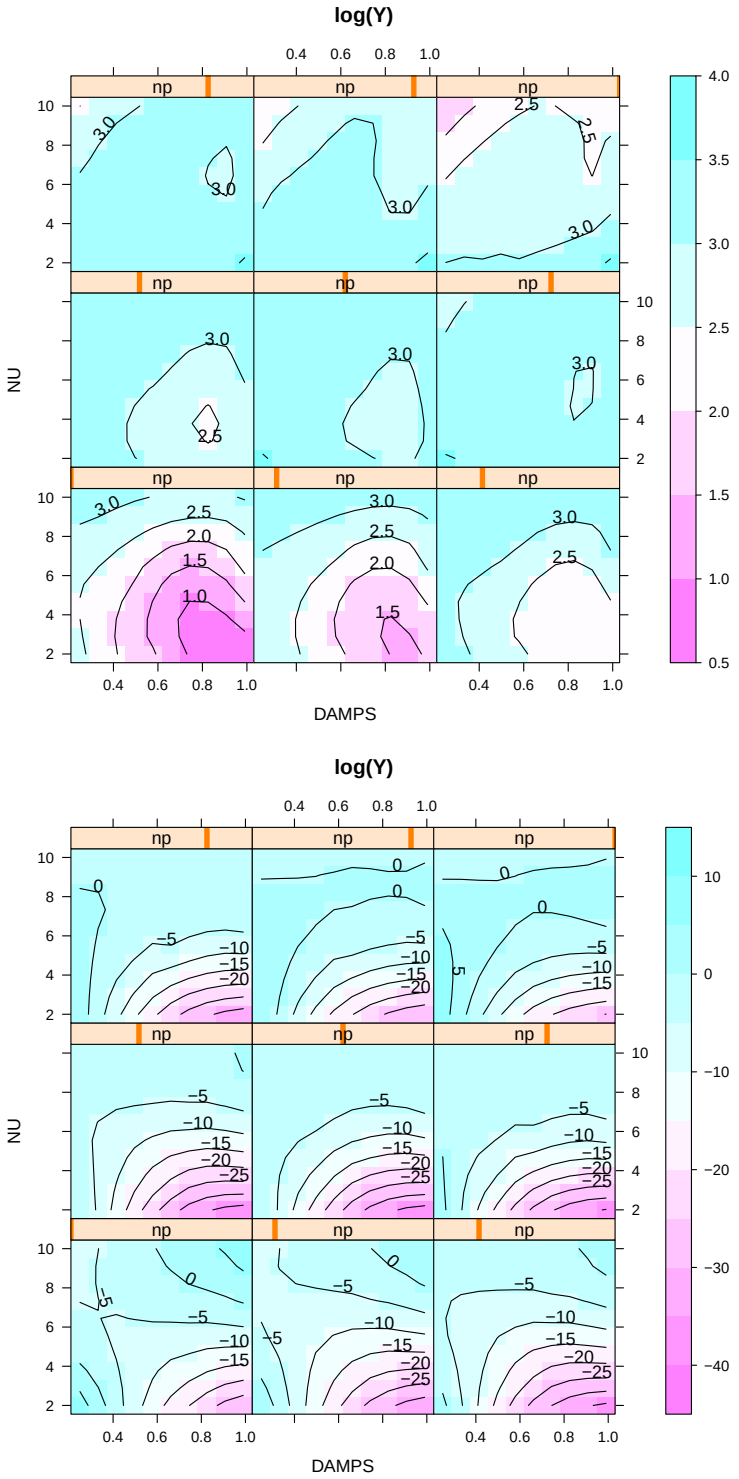


Figure 18: *Top:* Contour plots based on the complete data set (CMA-ES Ackley based on 82 function evaluations). Smaller values are better. Better configurations are placed in the lower area of the panels. The factor `CS` is held constant. `NU` is plotted versus `DAMPS`, while values of the factor `NPARENTS` (np), are varied with the slider on top of each panel. *Bottom:* Contour plots based on the complete data set (CMA-ES Griewangk based on 69 function evaluations). Smaller values are better. Better configurations are placed in the lower region, i.e., small `NU` values are important. The factor which has the smallest effect, `CS`, is held constant. `NU` is plotted versus `DAMPS`, while values of the factor with the last but one effect, namely `NPARENTS` (np), are varied with the slider on top of each panel

do on average. We also provide boxplots for the performance of the interactively found settings in Fig. 8; their performance is comparable to the one of the automatically found settings.

We conclude that often a manually executed classical regression analysis can yield well-performing parameter settings using a very limited number of runs of the target algorithm. The exception in our experiments was test case CMA-ES-Ackley. For this test case, we observed a pronounced multimodal distribution during the interactive tuning; we hypothesize that this caused problems for the classical regression analysis. Although the distributions for test cases CMA-ES-Rastrigin and CMA-ES-Griewangk were also multimodal, the poor runs were much rarer in these test cases and rather played the role of outliers. (In the boxplots of Fig. 8 on page 23, the poor runs in test cases CMA-ES-Rastrigin and CMA-ES-Griewangk were indeed marked as outliers, while the multimodal distributions for CMA-ES-Ackley were not seen as caused by outliers).

6.4 Discussion of the Interactive Approach

Similar to microscopes in biology, SPOT can be used as a "datascope" to gain insight into algorithm behavior, by revealing factor effects and their importance to the experimenter. Such insights can not only be used to guide the interactive parameter optimization process, but also be of intrinsic value to the developer or end user of a target algorithm.

The classical response surface methodology (as discussed in Chap. 15 of Box et al (1978)) underlying our interactive approach was developed not only for finding parameter settings that achieve improved performance, but also to provide insights into how the performance of a target algorithm is affected by parameter changes. This latter question is related to the analysis of the response surface in the region of interest, and contour plots as shown in Fig. 16 are useful tools to answer it.

In particular, from the results reported earlier in this section, we can conclude that CMA-ES performs robustly on the test functions we studied in the sense that its mean performance (e.g., as summarized by contour plots) varies only moderately with changes in the parameters. We furthermore observed that large `DAMPS` values and smaller `NU` values resulted in better CMA-ES performance, while the effect of `CS` was rather marginal. Finally, small population sizes improved CMA-ES's performance on the Sphere function, corresponding nicely with theoretical results for evolution strategies (Schwefel, 1995; Beyer, 2001). These statements can be understood as hypotheses derived from our experimental results, and each of them could be further studied by additional experiments, e.g., as described on p. **??** in Chap. **??** of this book.

In our case study illustrating the interactive approach, we used classical regression models, because these models can be interpreted quite easily; features of the response surface can be seen directly from the regression equation $Y = X\beta$. This is not the case for more sophisticated prediction models, such as neural networks or Gaussian process models. Furthermore, as demonstrated here in the case of CMA-ES, it is possible to obtain competitive results using such simple models. Nevertheless, in principle, more complex regression models could be used in the context of the interactive sequential parameter optimization approach. Furthermore, we note that observations and hypotheses regarding the dependence of a given target algorithm's performance on its parameter settings could also be obtained by analyzing more complex models, including the Gaussian process models constructed by the previously discussed, automatic sequential parameter optimization procedures.

Clearly, the interactive approach makes it easy to use results from early stages of the sequential parameter optimization process to effectively guide decisions made at later stages. For example, looking back at the initial stages of the process, the experimenter can detect that the set of variables studied at this stage was chosen poorly, or that inappropriate ranges were chosen for certain variables. Box et al (1978) state: "It is rather like looking at an old movie of a swimmer, who can now do back flips from a high diving board, when he was a young child making his first feeble attempts to keep his head above water. [...] The investigator must learn from the swimmer, who was prepared to begin by putting his foot in the water and was not afraid of getting wet." We note that the models used in early stages of the automated procedures discussed earlier in this chapter also provide guidance to

later stages of the process. However, the interactive process leaves room for expert human judgement, which can often be more effective in terms of the improvement achieved based on a small number of target algorithm runs.

The human expertise required to use the interactive approach successfully can be seen as a drawback compared to fully automated approaches. However, by providing dedicated support for the various operations that need to be carried out in this context, SPOT eases the burden on the experimenter and lowers the barrier to using the interactive approach effectively.

7 Conclusions and Future Work

In this work, we experimentally investigated model-based approaches for optimizing the performance of parametrized, randomized algorithms. First, we restricted our attention to procedures based on GP models, the most popular family of models for this problem. We evaluated two approaches from the literature, and found that "out-of-the-box" sequential parameter optimization (SPO) offered more robust performance than the sequential Kriging optimization (SKO) approach. However, when a log-transformation was used, SKO performed competitively. We then investigated key design decisions within the SPO paradigm: the initial design; whether to fit models to raw or log-transformed data; the expected improvement criterion; and the intensification criterion. Of these four, the log transformation and the intensification criterion substantially affected performance. Based on our findings, we proposed a new version of SPO, dubbed SPO$^+$, which yielded substantially better performance than SPO for optimizing the solution quality of CMA-ES (Hansen and Ostermeier, 1996; Hansen and Kern, 2004) on a number of test functions, as well as the runtime of SAPS (Hutter et al, 2002) on a SAT instance. In this latter domain, for which performance results for other (model-free) parameter optimization approaches are available, we demonstrated that SPO$^+$ achieved state-of-the-art performance.

We then contrasted this automated tuning approach with an interactive approach based on classical linear regression models. The interactive approach yielded well-performing parameter settings based on very few function evaluations, and also provided the basis for interesting hypotheses about CMA-ES's performance under different parameter settings. The interactive approach is particularly suitable in situations where the evaluation of individual configurations is computationally very expensive and therefore the overall number of parameter configurations evaluated has to be kept as low as possible.

In the future, we plan to extend our work to deal with optimization of runtime across a set of instances, along the lines of the approach of Williams et al (2000). We also plan to compare other types of models, such as random forests (Breiman, 2001), to the Gaussian process approach. SPOT and the interactive approach already support the optimization of categorical parameters using tree-based regression models (Chap. **??**). We further plan to develop automated methods for the sequential optimization of categorical variables.

Acknowledgements

We thank Theodore Allen for making the original SKO code available to us. This work was supported by the collaborative research project COSA.

Appendix

We show that for a random variable X distributed according to a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, it is the case that $E[\max(f_{min} - \exp(X), 0)] = f_{min}\Phi(v) - e^{\frac{1}{2}\sigma^2+\mu} \cdot \Phi(v-\sigma)$, where $v = \frac{\ln(f_{min})-\mu}{\sigma}$. We

denote the probability density function and cumulative distribution function of a standard normal distribution as φ and Φ, respectively.

$$
\begin{aligned}
& E[\max(f_{min} - \exp(X), 0)] \\
= & \int_{-\infty}^{\infty} \max(f_{min} - \exp(x), 0) p(x) dx \\
= & \int_{-\infty}^{\ln(f_{min})} (f_{min} - \exp(x)) \frac{1}{\sigma} \varphi(\frac{x-\mu}{\sigma}) dx \\
= & f_{min} \Phi(\frac{\ln(f_{min}) - \mu}{\sigma}) - \int_{-\infty}^{\frac{\ln(f_{min})-\mu}{\sigma}} \exp[x\sigma + \mu] \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}x^2\right] dx \\
= & f_{min} \Phi(\frac{\ln(f_{min}) - \mu}{\sigma}) - \int_{-\infty}^{\frac{\ln(f_{min})-\mu}{\sigma}} \exp[\frac{1}{2}\sigma^2 + \mu] \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}(x-\sigma)^2\right] dx \\
= & f_{min} \Phi(\frac{\ln(f_{min}) - \mu}{\sigma}) - \exp[\frac{1}{2}\sigma^2 + \mu] \Phi(\frac{\ln(f_{min}) - \mu}{\sigma} - \sigma).
\end{aligned}
$$

References

Adenso-Diaz B, Laguna M (2006) Fine-tuning of algorithms using fractional experimental design and local search. Operations Research 54(1):99–114

Alexandrov NM, Lewis RM, Gumbert CR, Green LL, Newman PA (2001) Approximation and model management in aerodynamic optimization with variable-fidelity models. Journal of Aircraft 38(6):1093–1101

Audet C, Orban D (2006) Finding optimal algorithmic parameters using the mesh adaptive direct search algorithm. SIAM Journal on Optimization 17(3):642–664

Balaprakash P, Birattari M, Stützle T (2007) Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein T, Aguilera MJB, Blum C, Naujoks B, Roli A, Rudolph G, Sampels M (eds) 4th International Workshop on Hybrid Metaheuristics (HM'07), pp 108–122

Bartz-Beielstein T (2003) Experimental analysis of evolution strategies—overview and comprehensive introduction. Interner Bericht des Sonderforschungsbereichs 531 Computational Intelligence CI–157/03, Universität Dortmund, Germany

Bartz-Beielstein T (2006) Experimental Research in Evolutionary Computation—The New Experimentalism. Natural Computing Series, Springer, Berlin, Heidelberg, New York

Bartz-Beielstein T, Markon S (2004) Tuning search algorithms for real-world applications: A regression tree based approach. In: Greenwood GW (ed) Proceedings 2004 Congress on Evolutionary Computation (CEC'04), Portland OR, IEEE, Piscataway NJ, vol 1, pp 1111–1118

Bartz-Beielstein T, Preuss M (2006) Considerations of budget allocation for sequential parameter optimization (SPO). In: Paquete L, et al (eds) Workshop on Empirical Methods for the Analysis of Algorithms, Proceedings, Reykjavik, Iceland, pp 35–40

Bartz-Beielstein T, Parsopoulos KE, Vrahatis MN (2004a) Analysis of particle swarm optimization using computational statistics. In: Simos TE, Tsitouras C (eds) Proceedings International Conference Numerical Analysis and Applied Mathematics (ICNAAM), Wiley-VCH, Weinheim, Germany, pp 34–37

Bartz-Beielstein T, Parsopoulos KE, Vrahatis MN (2004b) Design and analysis of optimization algorithms using computational statistics. Applied Numerical Analysis and Computational Mathematics (ANACM) 1(2):413–433

Bartz-Beielstein T, de Vegt M, Parsopoulos KE, Vrahatis MN (2004c) Designing particle swarm optimization with regression trees. Interner Bericht des Sonderforschungsbereichs 531 Computational Intelligence CI–173/04, Universität Dortmund, Germany

Bartz-Beielstein T, Lasarczyk C, Preuß M (2005) Sequential parameter optimization. In: McKay B, et al (eds) Proceedings 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland, IEEE Press, Piscataway NJ, vol 1, pp 773–780

Bartz-Beielstein T, Lasarczyk C, Preuss M (2008a) Sequential parameter optimization toolbox, manual version 0.5, September 2008, available at `http://www.gm.fh-koeln.de/imperia/md/content/personen/lehrende/bartz_beielstein_thomas/ spotdoc.pdf`

Bartz-Beielstein T, Zimmer T, Konen W (2008b) Parameterselektion für komplexe modellierungsaufgaben der wasserwirtschaft – moderne CI-verfahren zur zeitreihenanalyse. In: Mikut R, Reischl M (eds) Proc. 18th Workshop Computational Intelligence, Universitätsverlag, Karlsruhe, pp 136–150

Beachkofski B, Grandhi R (2002) Improved distributed hypercube sampling. American Institute of Aeronautics and Astronautics Paper 2002-1274

Beielstein T (2003) Tuning evolutionary algorithms—overview and comprehensive introduction. Interner Bericht des Sonderforschungsbereichs 531 *Computational Intelligence* CI–148/03, Universität Dortmund, Germany

Beyer HG (2001) The Theory of Evolution Strategies. Springer, Berlin, Heidelberg, New York

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Proc. of GECCO-02, pp 11–18

Box G, Behnken D (1960) Some new three level designs for the study of quantitative variables. Technometrics 2:455–475

Box GEP, Draper NR (1987) Empirical Model Building and Response Surfaces. Wiley, New York NY

Box GEP, Hunter WG, Hunter JS (1978) Statistics for Experimenters. Wiley, New York NY

Breiman L (2001) Random forests. Machine Learning 45(1):5–32

Chen J, Chen C, Kelton D (2003) Optimal computing budget allocation of indifference-zone-selection procedures, working paper, taken from `http://www.cba.uc.edu/faculty/keltonwd`. Cited 6 January 2005

Coy SP, Golden BL, Runger GC, Wasil EA (2001) Using experimental design to find effective parameter settings for heuristics. Journal of Heuristics 7(1):77–97

Dalgaard P (2002) Introductory Statistics with R. Springer, Berlin, Heidelberg, New York

Hansen N (2006) The CMA evolution strategy: a comparing review. In: Lozano J, Larranaga P, Inza I, Bengoetxea E (eds) Towards a new evolutionary computation. Advances on estimation of distribution algorithms, Springer, pp 75–102

Hansen N, Kern S (2004) Evaluating the CMA evolution strategy on multimodal test functions. In: Yao X, et al (eds) Parallel Problem Solving from Nature PPSN VIII, Springer, LNCS, vol 3242, pp 282–291

Hansen N, Ostermeier A (1996) Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proc. of CEC-96, Morgan Kaufmann, pp 312–317

Hoos HH, Stützle T (2005) Stochastic Local Search – Foundations & Applications. Morgan Kaufmann

Huang D, Allen TT, Notz WI, Zeng N (2006) Global optimization of stochastic black-box systems via sequential kriging meta-models. Journal of Global Optimization 34(3):441–466

Hutter F, Tompkins DAD, Hoos HH (2002) Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In: Proc. of CP-02, pp 233–248

Hutter F, Hamadi Y, Hoos HH, Leyton-Brown K (2006) Performance prediction and automated tuning of randomized and parametric algorithms. In: Proc. of CP-06, pp 213–228

Hutter F, Hoos HH, Stützle T (2007) Automatic algorithm configuration based on local search. In: Proc. of AAAI-07, pp 1152–1157

Hutter F, Hoos HH, Leyton-Brown K, Murphy KP (2009a) An experimental investigation of model-based parameter optimisation: SPO and beyond. In: Proc. of GECCO-09, pp 271–278

Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009b) ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36:267–306

Ihaka R, Gentleman R (1996) R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics 5(3):299–314

Jones DR, Schonlau M, Welch WJ (1998) Efficient global optimization of expensive black box functions. Journal of Global Optimization 13:455–492

Konen W, Zimmer T, Bartz-Beielstein T (2009) Optimierte modellierung von füllständen in regenüberlaufbecken mittels ci-basierter parameterselektion. at – Automatisierungstechnik Im Druck

Lasarczyk CWG (2007) Genetische programmierung einer algorithmischen chemie. PhD thesis, Technische Universitï£¡t Dortmund

Leyton-Brown K (2003) Resource allocation in competitive multiagent systems. PhD thesis, Stanford University

Leyton-Brown K, Nudelman E, Shoham Y (2002) Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Proc. of CP-02

Lophaven SN, Nielsen HB, Sondergaard J (2002) Aspects of the Matlab toolbox DACE. Tech. Rep. IMM-REP-2002-13, Informatics and Mathematical Modelling, Technical University of Denmark, DK-2800 Kongens Lyngby, Denmark

Mayer AS, Kelley C, Miller CT (2002) Optimal design for problems involving flow and transport phenomena in saturated subsurface systems. Advances in Water Resources 12:1233–1256

Mockus J, Tiesis V, Zilinskas A (1978) The application of bayesian methods for seeking the extremum. Towards Global Optimisation 2:117–129, north Holland, Amsterdam

Montgomery DC (2001) Design and Analysis of Experiments, 5th edn. Wiley, New York NY

Pukelsheim F (1993) Optimal Design of Experiments. Wiley, New York NY

Quinonero-Candela J, Rasmussen CE, Williams CK (2007) Approximation methods for gaussian process regression. In: Large-Scale Kernel Machines, Neural Information Processing, MIT Press, Cambridge, MA, USA, pp 203–223, URL http://mitpress.mit.edu/9780262026253

Rasmussen CE, Williams CKI (2006) Gaussian Processes for Machine Learning. The MIT Press

Sacks J, Welch WJ, Welch TJ, Wynn HP (1989) Design and analysis of computer experiments. Statistical Science 4(4):409–423

Santner TJ, Williams BJ, Notz WI (2003) The Design and Analysis of Computer Experiments. Springer Verlag, New York

Schonlau M, Welch WJ, Jones DR (1998) Global versus local search in constrained optimization of computer models. In: Flournoy N, Rosenberger W, Wong W (eds) New Developments and Applications in Experimental Design, vol 34, Institute of Mathematical Statistics, Hayward, California, pp 11–25

Schwefel HP (1995) Evolution and Optimum Seeking. Sixth-Generation Computer Technology, Wiley, New York NY

Tompkins DAD, Hoos HH (2004) UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In: Proc. of SAT-04

Venables WN, Ripley BD (2002) Modern Applied Statistics with S-PLUS, 4th edn. Springer, Berlin, Heidelberg, New York

Williams BJ, Santner TJ, Notz WI (2000) Sequential design of computer experiments to minimize integrated response functions. Statistica Sinica 10:1133–1152