Time-Bounded Sequential Parameter Optimization

Conference Paper · September 2010

DOI: 10.1007/978-3-642-13800-3_30 · Source: DBLP

4 authors:

Frank Hutter
University of Freiburg
166 PUBLICATIONS **10,226** CITATIONS
SEE PROFILE

Holger H Hoos
Leiden University
328 PUBLICATIONS **19,684** CITATIONS
SEE PROFILE

Kevin Leyton-Brown
University of British Columbia - Vancouver
177 PUBLICATIONS **9,661** CITATIONS
SEE PROFILE

Kevin P. Murphy
Google Inc.
83 PUBLICATIONS **15,887** CITATIONS
SEE PROFILE

Time-Bounded Sequential Parameter Optimization

Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Kevin Murphy

Department of Computer Science, University of British Columbia, Vancouver, Canada
{hutter,hoos,kevinlb,murphyk}@cs.ubc.ca

Abstract. The optimization of algorithm performance by automatically identifying good parameter settings is an important problem that has recently attracted much attention in the discrete optimization community. One promising approach constructs predictive performance models and uses them to focus attention on promising regions of a design space. Such methods have become quite sophisticated and have achieved significant successes on other problems, particularly in experimental design applications. However, they have typically been designed to achieve good performance only under a budget expressed as a number of function evaluations (*e.g.*, target algorithm runs). In this work, we show how to extend the Sequential Parameter Optimization framework [SPO; see 5] to operate effectively under time bounds. Our methods take into account both the varying amount of time required for different algorithm runs and the complexity of model building and evaluation; they are particularly useful for minimizing target algorithm runtime. Specifically, we avoid the up-front cost of an initial design, introduce a time-bounded intensification mechanism, and show how to reduce the overhead incurred by constructing and using models. Overall, we show that our method represents a new state of the art in model-based optimization of algorithms with continuous parameters on single problem instances.

1 Introduction

Many high-performance algorithms have parameters whose settings largely determine their effectiveness in a particular application context. Optimizing these parameters is an important task for developers and end users of such algorithms. In general, given a parameterized *target algorithm* A, a set (or distribution) of problem instances I and a performance metric c, the goal is to find parameter settings of A that optimize c on I. Often, the performance metric c is based on either the running time of A on I or on the solution quality achieved by A on I within a given time budget.

Several automated approaches to solving this *parameter optimization* problem (also called *algorithm configuration* or *parameter tuning*) have been investigated in the literature. These differ in whether or not explicit models (so-called *response surfaces*) are used to describe the dependence of target algorithm performance on parameter settings, and they also vary in the type and number of target algorithm parameters considered. Much existing work deals with relatively small numbers of numerical (often continuous) parameters [see, *e.g.*, 8, 3, 1]). Existing model-based methods also typically fall into this category [5, 9]. Some relatively recent model-free approaches permit both larger numbers of parameters and categorical domains, in particular F-Race [7, 4] and ParamILS [13, 12].

Model-based approaches are appealing since they can be used to quantify the importance of each parameter, as well as interactions between parameters, and—in the context of multiple instances—between parameters and instance characteristics. They can support interpolation of performance between parameter settings, and, in principle, extrapolation to previously-unseen regions of the parameter space. Thereby, they can provide intuition about the parameter response that cannot be gathered from model-free methods. This can be a substantial help to designers seeking to understand or improve an algorithm.

This paper extends the Sequential Parameter Optimization (SPO) approach by Bartz-Beielstein et al. [6, 5]. As in previous work on SPO [6, 5, 10], we consider randomized target algorithms with numerical parameters, and limit ourselves to the simple case of optimizing performance for a single problem instance at a time. (Such an instance may be chosen as representative of a heterogeneous set or distribution.) This allows us to focus on core conceptual issues that remain important when taking into account a set or distribution of problem instances, while still retaining significant practical relevance. We are actively investigating the issue of multiple instances in our ongoing work.

Here we propose three key improvements to SPO, to allow it to operate effectively within a given time budget: (1) interleaved random parameter settings that eliminate the need for a costly initial design; (2) a time-bounded intensification mechanism; and (3) replacing Gaussian process (GP) models with a computationally-efficient approximation called projected process (PP) models. These improvements are particularly useful for parameter optimization tasks with the objective of minimizing algorithm runtime. However, our methods can also be applied to other objectives (in fact, in one of our experiments we minimize the number of search steps required to find a solution). In experiments for optimizing a high-performance local search solver for various SAT instances, we demonstrate that improvements (1) and (2) substantially reduce the amount of time required to find good parameter settings; that the PP models yield much better predictions than the previously-used noise-free GP models while only taking about 1/30 of their construction time; and that our resulting parameter optimization procedure—dubbed TB-SPO(PP)—significantly outperforms the previous state of the art.

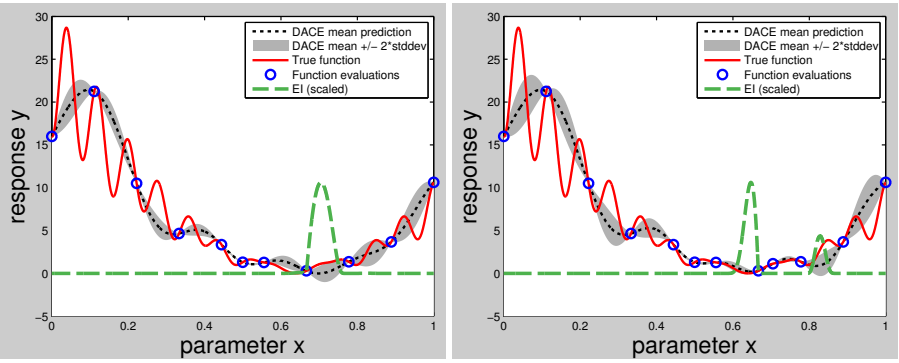
This paper is organized as follows. In Section 2, we build intuition for sequential model-based optimization (SMBO) and discuss existing SMBO approaches. In Section 3, we introduce our extensions of this framework to time-bounded SPO. In Section 4, we empirically study our new methods. We conclude the paper in Section 5.

2 Sequential Model-Based Optimization: Existing Work

Model-based optimization methods fit a regression model to given training data and use this model for optimization. In the context of parameter optimization, a *response surface model* is fitted to a training set $\{(\boldsymbol{\theta}_i, o_i)\}_{i=1}^{n}$, where parameter setting $\boldsymbol{\theta}_i = (\boldsymbol{\theta}_{i1}, \ldots, \boldsymbol{\theta}_{id})^{\intercal}$ is a complete instantiation of the d parameters of the given target algorithm, and o_i is the observed cost of the target algorithm run with parameter setting $\boldsymbol{\theta}_i$. In *sequential* model-based optimization, the selection of new settings to be evaluated can depend on previous observations.

2.1 An illustrative example

We demonstrate the generic SMBO framework with a simple example. Consider optimizing a deterministic algorithm with a single continuous parameter, where the algorithm's



(a) SMBO, step 1 (b) SMBO, step 2

Fig. 1. Two steps of SMBO for the optimization of a 1D function. The true function is shown as a solid line, the circles denote our observations. The dotted line denotes the mean prediction of the DACE model (a noise-free Gaussian process model, see Section 2.2), with the grey area denoting its uncertainty. Expected improvement (scaled for visualization) is shown as a dashed line.

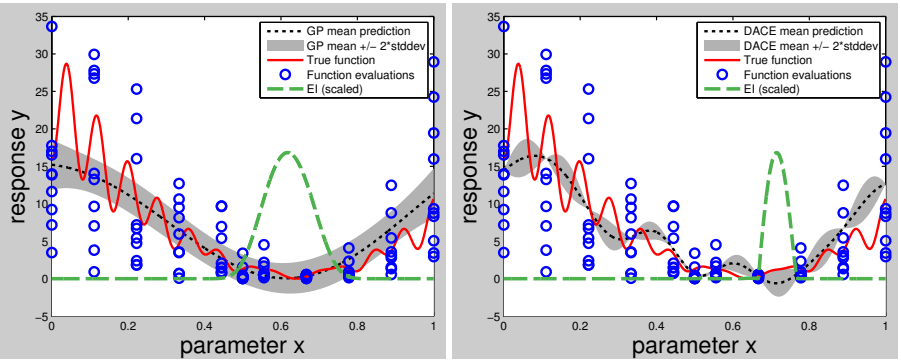
performance as a function of its parameter is described by the solid line in Figure 1(a). SMBO algorithms start by running the algorithm with a number of parameter settings defined by a so-called *initial design*, typically based on a Latin hypercube design [LHD; see 17] in the region of interest (the Cartesian product of the intervals considered for each parameter). In Figure 1, circles denote the settings in the initial design. Next, SMBO fits a response surface model to the data gathered. We discuss Gaussian process models, the most commonly-used type of models, in Section 2.2. SMBO uses its predictive model of performance to select settings for the next target algorithm run. In particular, it computes a so-called *expected improvement criterion* to trade off learning about new, unknown parts of the parameter space and intensifying the search locally in the best known region. In Figure 1(a), the dashed line denotes one possible EIC, $E[I_{exp}(\boldsymbol{\theta})]$ [see 11], evaluated throughout the region of interest. Note that EIC is high in regions of low predictive mean and high predictive variance; these regions are most likely to contain parameter settings $\boldsymbol{\theta}$ with cost lower than that of the incumbent, $\boldsymbol{\theta}_{inc}$.

SMBO next identifies a set of parameter settings with promising EIC values, runs the target algorithm on them, and updates its model based on the results. In our example, we chose the single setting with maximal EIC; in Figure 1(b), note the additional data point at $x = 0.705$. This data point changes the model, greatly reducing the uncertainty around the new data point and splitting the next promising region in two.

2.2 Gaussian Process Regression Models

In most recent work on sequential model-based optimization, the model takes the form of a Gaussian stochastic process [GP; 16]. To construct a GP model, first we need to select a parameterized kernel function $k_\lambda : \boldsymbol{\Theta} \times \boldsymbol{\Theta} \mapsto \mathbb{R}^+$, specifying the similarity between two parameter settings. We use the most commonly-used kernel function

$$K(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \exp\left[\sum_{l=1}^{d}(-\lambda_l \cdot (\boldsymbol{\theta}_{il} - \boldsymbol{\theta}_{jl})^2)\right], \quad (1)$$



(a) Standard GP fit (b) Noise-free GP fit on cost statistics

Fig. 2. Alternative ways of fitting a response surface to noisy observations. Note that we plot mean $\pm$ two standard deviations of the *predictive mean*; the predicted observation noise would need to be added to this (but is zero for the noise-free model in (b)).

where $\lambda_1, \ldots, \lambda_d$ are the kernel parameters. We also need to set the variance σ^2 of Gaussian-distributed observation noise (in parameter optimization, this corresponds to the variance of the target algorithm's runtime distribution). The predictive distribution of a zero-mean GP for response o_{n+1} at input $\boldsymbol{\theta}_{n+1}$, given training data $\mathcal{D} = \{(\boldsymbol{\theta}_1, o_1), \ldots, (\boldsymbol{\theta}_n, o_n)\}$, is then

$$p(o_{n+1}|\boldsymbol{\theta}_{n+1}, \boldsymbol{\theta}_{1:n}, \boldsymbol{o}_{1:n}) = \mathcal{N}(o_{n+1}|\boldsymbol{k}_*{}^\intercal \cdot [\boldsymbol{K}+\sigma^2\boldsymbol{I}]^{-1} \cdot \boldsymbol{o}_{1:n}, k_{**} - \boldsymbol{k}_*{}^\intercal \cdot [\boldsymbol{K}+\sigma^2\mathbf{I}]^{-1}), \tag{2}$$

where

$$\boldsymbol{K} = \begin{pmatrix} k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_1) & \ldots & k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_n) \\ & \ddots & \\ k(\boldsymbol{\theta}_n, \boldsymbol{\theta}_1) & \ldots & k(\boldsymbol{\theta}_n, \boldsymbol{\theta}_n) \end{pmatrix}$$
$$\boldsymbol{k}_* = (k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_{n+1}), \ldots, k(\boldsymbol{\theta}_n, \boldsymbol{\theta}_{n+1}))$$
$$k_{**} = k(\boldsymbol{\theta}_{n+1}, \boldsymbol{\theta}_{n+1}) + \sigma^2,$$

$\mathbf{I}$ is the n-dimensional identity matrix, and $p(a|b) = \mathcal{N}(a|\mu, \Sigma)$ denotes that the conditional distribution of a given b is a Gaussian with mean μ and covariance matrix Σ. [See, *e.g.*, 16, for a derivation.] The kernel parameters are typically set by maximizing the *marginal likelihood* $p(\boldsymbol{o_{1:N}})$ with a gradient-based optimizer.

In the presence of observation noise (*e.g.*, stemming from randomization in the algorithm), we can use GP models in two possible ways. The standard approach is simply to construct a model using the noisy data, treating the observation noise variance σ^2 as an additional kernel parameter to be optimized. Another possibility is to first compute a (user-defined) *empirical cost statistic*, $\hat{c}(\boldsymbol{\theta})$, of the empirical distribution of observed costs for each encountered parameter setting, $\boldsymbol{\theta}$, and to then fit a noise-free GP model to these empirical cost statistics (fixing $\sigma^2 = 0$).

We illustrate these two approaches in Figure 2. In that figure, the solid line is the same true function as in the noise-free example in Figure 1(a), but we now have ten noisy observations for each of the design points considered in the previous example, depicted

by circles. Note that at training data points $\boldsymbol{\theta}$, the noise-free GP model's predictions perfectly match the empirical cost statistic, $\hat{c}(\boldsymbol{\theta})$, with zero uncertainty. This is despite the fact that the true cost often differs substantially from the empirical cost statistic (especially if the latter is based on few runs and the observation noise is high).

2.3 Existing State of the Art

In our previous work [11], we studied further components of SMBO and compared two prominent SMBO procedures: the sequential kriging optimization (SKO) algorithm by Huang et al. [9] and the sequential parameter optimization (SPO) procedure by Bartz-Beielstein et al. [6, 5]. We showed that SPO was more robust than SKO and primarily attributed this to what we call SPO's *intensification mechanism*, the strategy by which it determines the number of runs executed for each parameter setting. We also developed a new intensification mechanism, and demonstrated that the resulting algorithm, SPO$^+$, performed more robustly than SPO.

Besides the intensification mechanism, we identified two other components which are likely to strongly affect performance. Firstly, we left open the question how to choose the size of the initial design in SPO; there seemed to be no uniformly good choice. Secondly, we found evidence that SPO and especially SKO benefited from a log transformation of the data. We hypothesized that SKO's standard GP models are superior to SPO's noise-free GP models under that transformation; however, these standard GPs are computationally too expensive for the parameter optimization tasks we are interested in. In the following, we address both of these issues and also improve our intensification mechanism further, thereby substantially advancing the state of the art.

3 Time-Bounded SPO

The standard measure of a computational budget in blackbox function optimization is the number of "function evaluations" (in our case, runs of the target algorithm). This is meaningful if all target algorithm runs take the same amount of time and clearly dominate any computational costs incurred by the parameter optimization procedure. However, these conditions do not apply for the optimization of algorithm parameters, where (1) some target algorithm runs can be much faster than others, and (2) in some optimization scenarios target algorithm runs are fast compared to the overhead of the optimization procedure. For example, an experiment of running SPO$^+$ on SAPS-QWH we described in our previous work [11] required about 10 CPU hours to perform its allowed 20 000 target algorithms runs. Only about 10 minutes of this time was spent actually running SAPS, for an average runtime of 30 milliseconds per target algorithm run. Thus, in that case the computational overhead was a factor of 60. This paper introduces approaches for reducing that overhead.

Algorithm Framework 1 gives the outline of a generic SMBO procedure with a time budget. It initializes by running the target algorithm with the parameter settings in an initial design, and then iterates three steps: (1) fitting a response surface model using the existing data; (2) selecting a set of parameter settings to evaluate; and (3) running the target algorithm on (a subset of) the selected settings until a given time bound is reached. This time bound is related to the combined overhead, $t_{model} + t_{ei}$, due to fitting the model and selecting a set of promising parameter settings.

In the following three sections, we will introduce an instantiation of this algorithm framework corresponding to a time-bounded version of SPO, abbreviated TB-SPO. In

Algorithm Framework 1: Sequential Model-Based Optimization With a Time Budget.

Input : Target algorithm A
Output: Incumbent parameter setting, $\boldsymbol{\theta}_{inc}$
$[\mathbf{R}, \boldsymbol{\theta}_{inc}] \leftarrow$ Initialize()
repeat
  $[\mathcal{M}, t_{model}] \leftarrow$ FitModel($\mathbf{R}$)
  $[\vec{\boldsymbol{\Theta}}_{new}, t_{ei}] \leftarrow$ SelectNewParameterSettings($\mathcal{M}$, $\boldsymbol{\theta}_{inc}$)
  $[\mathbf{R}, \boldsymbol{\theta}_{inc}] \leftarrow$ Intensify($\vec{\boldsymbol{\Theta}}_{new}$, $\boldsymbol{\theta}_{inc}$, $\mathcal{M}$, $\mathbf{R}$, $t_{model} + t_{ei}$)
until *total time budget for tuning exhausted*
return $\boldsymbol{\theta}_{inc}$

Procedure 2: SelectNewParameterSettings($\mathcal{M}, \boldsymbol{\theta}_{inc}$) in TB-SPO

Input : Model, $\mathcal{M}$; incumbent configuration, $\boldsymbol{\theta}_{inc}$
Output: Sequence of parameter configurations to evaluate, $\vec{\boldsymbol{\Theta}}_{new}$

// ===== *Select parameter configurations with expected improvement*
$\boldsymbol{\Theta}_{rand} \leftarrow$ set of 10 000 elements drawn uniformly at random from $\boldsymbol{\Theta}$
$\vec{\boldsymbol{\Theta}}_{ei} \leftarrow \boldsymbol{\Theta}_{rand}$, sorted by decreasing $E[I_{exp}(\boldsymbol{\theta})]$ [for the formula, see 11]
Let t_{ei} denote the total time spent computing expected improvement
// ===== *Interleave configurations with high EI, and random configurations*
$\vec{\boldsymbol{\Theta}}_{new} \leftarrow []$
for $i = 1, \ldots, 10\,000$ **do**
  Append $\vec{\boldsymbol{\Theta}}_{ei}[i]$ to $\vec{\boldsymbol{\Theta}}_{new}$
  Draw a parameter configuration $\boldsymbol{\theta}$ uniformly at random from $\boldsymbol{\Theta}$ and append it to $\vec{\boldsymbol{\Theta}}_{new}$
return $[\vec{\boldsymbol{\Theta}}_{new}, t_{ei}]$

particular, we interleave random parameter settings to remove the need for a costly initial design, introduce a time-bounded intensification mechanism, and discuss an approximate version of standard GP models that reduces the computational overheads due to the construction and use of models by orders of magnitude.

3.1 Interleaving Random Parameter Settings

Procedure 2 details how TB-SPO selects parameter settings to be considered next. It iteratively interleaves settings selected by expected improvement and settings chosen uniformly at random. Due to this interleaving of random settings in each iteration, we no longer rely on a costly initialization using a Latin hypercube. Instead, we initialize the model based on a single run using the algorithm's default setting and can therefore start the sequential optimization process much more quickly.

3.2 A Time-Bounded Intensification Mechanism

TB-SPO's intensification mechanism iteratively performs runs of the target algorithm until the time spent for doing so equals the combined overhead incurred by the model construction and the search for promising parameter settings. This guarantees that at least 50% of the total time spent is used for runs of the target algorithm. Obviously, this percentage can be adjusted (in particular, that bound can be driven to 0% in applications where the computational overhead is not of concern).

Procedure 3: Intensify($\vec{\boldsymbol{\Theta}}_{new}$, $\boldsymbol{\theta}_{inc}$, $\mathcal{M}$, $\mathbf{R}$, $t_{overhead}$) in TB-SPO

Input : Sequence of parameter settings to evaluate, $\vec{\boldsymbol{\Theta}}_{new}$; incumbent parameter setting, $\boldsymbol{\theta}_{inc}$; model, $\mathcal{M}$; sequence of target algorithm runs, $\mathbf{R}$; time bound, $t_{overhead}$
Output: Sequence of target algorithm runs, $\mathbf{R}$; incumbent parameter setting, $\boldsymbol{\theta}_{inc}$

for $i = 1, \ldots,$ *length*($\vec{\boldsymbol{\Theta}}_{new}$) **do**
  $[\boldsymbol{\theta}_{inc}, \mathbf{R}] \leftarrow$ Compare($\vec{\boldsymbol{\Theta}}_{new}[i]$, $\boldsymbol{\theta}_{inc}$, $\mathbf{R}$)
  if *time spent in this call to this procedure exceeds* $t_{overhead}$ *and* $i \geq 2$ **then break**
return $[\mathbf{R}, \boldsymbol{\theta}_{inc}]$

We now describe our intensification mechanism in detail; Procedure 3 provides pseudocode. Given a list of parameter settings, $\vec{\boldsymbol{\Theta}}_{new} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots)$, to be considered, and a time bound for the current iteration, this mechanism iteratively compares each parameter setting $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots$ to the current incumbent parameter setting, $\boldsymbol{\theta}_{inc}$. It stops once the time bound is reached and at least two new parameter settings (*i.e.*, at least one randomly-sampled parameter setting) have been evaluated. Pseudocode for the comparison procedure is given in Procedure 4; it is identical to the corresponding subroutine in SPO$^+$. When comparing a parameter setting $\boldsymbol{\theta}$ to the current incumbent, $\boldsymbol{\theta}_{inc}$, we incrementally increase its number of runs $N(\boldsymbol{\theta})$ until either its empirical cost statistic exceeds the one of the incumbent (*i.e.*, $\hat{c}(\boldsymbol{\theta}) > \hat{c}(\boldsymbol{\theta}_{inc})$), or we have executed at least as many runs for it as for the incumbent (*i.e.*, $N(\boldsymbol{\theta}_i) \geq N(\boldsymbol{\theta}_{inc})$). In the former case, we keep the same incumbent and perform as many additional algorithm runs for it as we just performed for $\boldsymbol{\theta}_i$. In the latter case, we make $\boldsymbol{\theta}_i$ our new incumbent.

These changes result in a new parameter optimization procedure we call TB-SPO. Further differences between SPO$^+$ and TB-SPO include that we use expected improvement criterion (EIC) $E[I_{exp}(\boldsymbol{\theta})]$ [see 11]. In our experiments, we did not observe significant differences due to the choice of EIC but selected $E[I_{exp}(\boldsymbol{\theta})]$ because it is theoretically better justified than the EIC used in SPO$^+$. Also, while SPO$^+$ revisits a set of good previously-encountered parameter settings in each iteration to avoid missing the optimal one, the random interleaved settings remove the need for this mechanism.

3.3 Using an Approximate Gaussian Process Model

Learning a GP model imposes prohibitive computational overhead in many parameter optimization scenarios of interest. Inverting the $n \times n$ matrix $[\boldsymbol{K} + \sigma^2\mathbf{I}]$ in Equation (2) takes time $O(n^3)$, where n is the number of training data points (*i.e.*, the number of target algorithm runs performed). This inversion has to be done in each of h steps of the kernel parameter optimization, leading to a computational complexity of $O(h \cdot n^3)$. Once the kernel parameters are optimized and the inverse has been computed, subsequent predictions are relatively cheap, only requiring matrix-vector multiplications and thus time $O(n^2)$. In our typical parameter optimization scenarios, h is typically around 50 and the number of target algorithm runs we can perform within the given time budget is in the tenthousands; thus, standard GP models are clearly not feasible.

Various approximations exist to reduce the complexity of GP models [see, *e.g.*, 15]. Here, we use the projected process (PP) approximation. We only give the final equations for predictive mean and variance [for a derivation, see 16]. The PP approximation to GPs uses a subset of p of the n training data points, the so-called *active set*. Let v be a vector holding the indices of these p data points. Let $k(\cdot, \cdot)$ denote the GP covariance function

Procedure 4: Compare($\boldsymbol{\theta}$, $\boldsymbol{\theta}_{inc}$, $\mathbf{R}$)
Recall that $N(\boldsymbol{\theta})$ denotes the number of algorithm runs which have been performed for $\boldsymbol{\theta}$. The maximal number of runs to perform with a parameter setting, maxR, is a parameter; in all our experiments, we set it to 2 000.

Input : Challenger parameter setting, $\boldsymbol{\theta}$; incumbent parameter setting, $\boldsymbol{\theta}_{inc}$; sequence of target algorithm runs, $\mathbf{R}$
Output: Sequence of target algorithm runs, $\mathbf{R}$; incumbent parameter setting, $\boldsymbol{\theta}_{inc}$

$r \leftarrow 1$; numBonus $\leftarrow 1$
$\mathbf{R} \leftarrow$ ExecuteRuns($\mathbf{R}$, $\boldsymbol{\theta}$, 1)
if $N(\boldsymbol{\theta}) > N(\boldsymbol{\theta}_{inc})$ **then**
    $\mathbf{R} \leftarrow$ ExecuteRuns($\mathbf{R}$, $\boldsymbol{\theta}_{inc}$, 1)
    numBonus $\leftarrow 0$
while *true* **do**
    if $\hat{c}(\boldsymbol{\theta}) > \hat{c}(\boldsymbol{\theta}_{inc})$ **then**
        // ===== Reject challenger, perform bonus runs for $\boldsymbol{\theta}_{inc}$
        $\mathbf{R} \leftarrow$ ExecuteRuns($\mathbf{R}$, $\boldsymbol{\theta}_{inc}$, $\min(\text{numBonus}, \text{maxR} - N(\boldsymbol{\theta}_{inc}))$); **break**
    if $N(\boldsymbol{\theta}) \geq N(\boldsymbol{\theta}_{inc})$ **then**
        // ===== Challenger becomes incumbent
        $\boldsymbol{\theta}_{inc} \leftarrow \boldsymbol{\theta}$; **break**
    $r \leftarrow \min(2r, N(\boldsymbol{\theta}_{inc}) - N(\boldsymbol{\theta}))$
    $\mathbf{R} \leftarrow$ ExecuteRuns($\mathbf{R}$, $\boldsymbol{\theta}$, r)
    numBonus $\leftarrow$ numBonus $+r$
return $[\mathbf{R}, \boldsymbol{\theta}_{inc}]$

and let K_{pp} denote the p by p matrix with $K_{pp}(i,j) = k(\boldsymbol{\theta}_{v(i)}, \boldsymbol{\theta}_{v(j)})$. Similarly, let $\boldsymbol{K_{pn}}$ denote the p by n matrix with $\boldsymbol{K_{pn}}(i,j) = k(\boldsymbol{\theta}_{v(i)}, \boldsymbol{\theta}_j)$; finally, let $\boldsymbol{K_{np}}$ denote the transpose of $\boldsymbol{K_{pn}}$. We then have

$$p(o_{n+1}|\boldsymbol{\theta}_{n+1}, \boldsymbol{\theta}_{1:n}, o_{1:n}) = \mathcal{N}(o_{n+1}|\mu_{n+1}, \text{Var}_{n+1}),$$

where

$$\mu_{n+1} = \boldsymbol{k}_*{}^\intercal \cdot (\sigma^2 \boldsymbol{K_{pp}} + \boldsymbol{K_{pn}} \cdot \boldsymbol{K_{np}})^{-1} \cdot \boldsymbol{K_{pn}} \cdot o_{1:n}$$
$$\text{Var}_{n+1} = k_{**} - \boldsymbol{k}_*{}^\intercal \cdot \boldsymbol{K_{pp}}^{-1} \cdot \boldsymbol{k}_* + \sigma^2 \boldsymbol{k}_*{}^\intercal \cdot (\sigma^2 \boldsymbol{K_{pp}} + \boldsymbol{K_{pn}} \cdot \boldsymbol{K_{np}})^{-1} \cdot \boldsymbol{k}_*,$$

and $\boldsymbol{k}_*$ and k_{**} are defined as in Section 2.2.

These equations assume a kernel with fixed parameters. We optimize the kernel parameters using a set of p data points randomly sampled without repetitions from the n input data points. We then sample an independent set of p data points for the subsequent PP approximation; in both cases, if $p > n$, we use n data points.

This approximation leads to a substantial improvement in computational complexity. The kernel parameter optimization based on p data points takes time $O(h \cdot p^3)$. In addition, there is a one-time cost of $O(p^2 \cdot n)$ for the PP equations. Thus, the complexity for fitting the approximate GP model is $O([h \cdot p + n] \cdot p^2)$. The complexity for predictions with this PP approximation is $O(p)$ for the mean and $O(p^2)$ for the variance of the predictive distribution [16]. Throughout, we use $p = 300$. As mentioned above, n is often in the tenthousands, and h is typically around 50. Thus, in our typical scenarios,

it is dramatically faster to construct a PP model than a standard GP model (which is $O(h \cdot n^3)$).

To use the PP approximation in our SMBO framework, we simply change the function FitModel and the use of models inside Function SelectNewParameterSettings, resulting in a new procedure we call TB-SPO(PP).

4 Experimental Evaluation

In this section, we empirically study the effects of removing the initial design and of time-bounding the intensification mechanism; the quality of our improved models; and the performance of SMBO with these models. First, we describe our experimental setup.

4.1 Experimental Setup

We empirically evaluated our parameter optimization procedures using a set of seven different scenarios. These scenarios are concerned with the optimization of SAPS [14], a high-performance dynamic local search algorithm for the propositional satisfiability problem (SAT). SAPS is a good test case for parameter optimization since it is a prominent algorithm that shows state-of-the-art performance for certain types of SAT instances and has been used prominently to evaluate automated parameter optimization procedures [13, 11, 2].

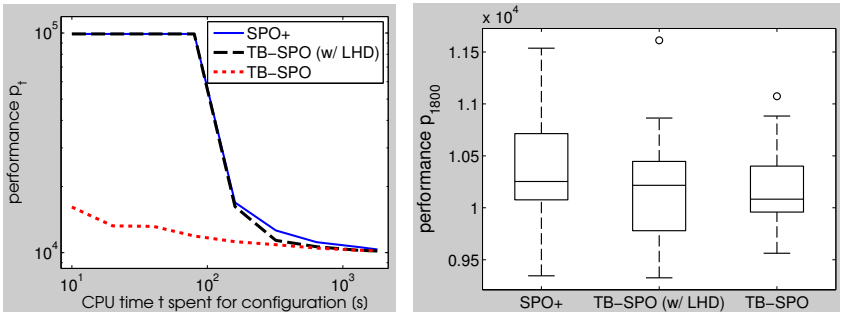
We used the standard UBCSAT implementation [18] of SAPS and considered the same four continuous parameters and the same region of interest as in previous work [11]. In order to allow for a direct comparison with that work, we used the same scenario SAPS-QWH used there. We also used six new scenarios, concerned with the optimization of SAPS for three SAT-encoded instances of each of the quasigroup completion problem (QCP) and the graph-colouring problem (for small-world graphs, SWGCP). For both QCP and SWGCP, these three instances were selected from previously-studied instance distributions [12], as the 50%, 75%, and 95% quantiles of hardness for SAPS, allowing us to assess scaling behaviour with instance hardness.

We used two different optimization objectives. For SAPS-QWH, to be consistent with past work, we aimed to minimize the median number of SAPS search steps required to solve the instance. For the other scenarios, we aimed to minimize mean runtime. In order to penalize timeouts at a cutoff time of $\kappa_{max} = 5$ seconds, we defined the *penalized average runtime* [PAR, see 12] of a set of runs to be the mean runtime over those runs, where unsuccessful runs are counted as $a \cdot \kappa_{max}$ with penalization constant $a = 10$.

We measured the performance of a parameter optimization run given a time budget t by evaluating its *incumbent* parameter setting at time t, the setting that would be returned if the procedure was to be terminated at time t. In particular, to approximate the true cost of a proposed parameter setting, $\boldsymbol{\theta}$, in an offline evaluation stage, we performed $1\,000$ test runs of $\boldsymbol{\theta}$ and used their empirical cost (for SAPS-QWH: median runlength; for the other scenarios: PAR) as our measure of *test performance*, $p_{test,t}$.

4.2 Experimental Evaluation of Comparison Mechanism

First, we compared the previous state-of-the-art SMBO method, SPO$^+$, to our new method, TB-SPO. We performed 25 runs of each parameter optimization procedure for each scenario and evaluated test performances $p_{test,t}$ for various time budgets, t. For fairness of comparison, we used the same LHD for SPO$^+$ that we used in our previous work [11] for scenario SAPS-QWH (500 points and 2 repetitions each). Table 1 shows the



(a) Performance over time (b) End result after one hour, p_{3600}

Fig. 3. Comparison of SPO$^+$ and TB-SPO. We carried out 25 runs of each procedure and show performance p_t (median number of SAPS search steps using the procedure's incumbent parameter setting). 'TB-SPO(w/ LHD)' denotes a version of TB-SPO that uses the same LHD as the SPO$^+$ variant shown ($500 \cdot 2$ data points). Subplot (a): p_t as a function of the time, t, the parameter optimization procedure is allowed (mean over 25 runs). Subplot (b): box plot of performance values p_{3600} achieved in the 25 runs.

Scenario	SPO$^+$	TB-SPO	RANDOM*	*pval1*	*pval2*
SAPS-QCP-MED [$\cdot 10^{-2}$]	4.50 ± 0.31	4.32 ± 0.21	$\mathbf{4.23 \pm 0.15}$	$\mathbf{4 \cdot 10^{-3}}$	0.17
SAPS-QCP-Q075	3.77 ± 9.72	$\mathbf{0.19 \pm 0.02}$	$\mathbf{0.19 \pm 0.01}$	$\mathbf{2 \cdot 10^{-6}}$	0.78
SAPS-QCP-Q095	49.91 ± 0.00	$\mathbf{2.20 \pm 1.17}$	2.64 ± 1.24	$\mathbf{1 \cdot 10^{-10}}$	0.12
SAPS-QWH [$\cdot 10^3$]	10.7 ± 0.76	10.1 ± 0.58	$\mathbf{9.88 \pm 0.41}$	$\mathbf{6 \cdot 10^{-3}}$	0.14
SAPS-SWGCP-MED	49.95 ± 0.00	0.18 ± 0.03	$\mathbf{0.17 \pm 0.02}$	$\mathbf{1 \cdot 10^{-10}}$	0.37
SAPS-SWGCP-Q075	50 ± 0	0.24 ± 0.04	$\mathbf{0.22 \pm 0.03}$	$\mathbf{1 \cdot 10^{-10}}$	0.08
SAPS-SWGCP-Q095	50 ± 0	$\mathbf{0.25 \pm 0.05}$	0.28 ± 0.10	$\mathbf{1 \cdot 10^{-10}}$	0.89

Table 1. Performance comparison of SPO$^+$ (based on a LHD with 500 data points and 2 repetitions), TB-SPO, and RANDOM*. We performed 25 runs of each procedure and computed test performance $p_{test,t}$ (for SAPS-QWH: median number of SAPS search steps; for the other scenarios: SAPS penalized average runtime) of its incumbents at time $t = 1\,800$s (3 600s for SAPS-QWH). We give mean $\pm$ standard deviation across the 25 runs, with boldface indicating the best procedure for each scenario. (Note that the two entries 50 ± 0 reflect the worst possible result: all test runs timed out after $\kappa_{max} = 5s$ and were thus scored as $10 \cdot 5\text{s} = 50\text{s}$.) Column *pval1* gives p-values for a Mann Whitney U test between the performance of SPO$^+$ and TB-SPO; *pval2* gives these values for comparing TB-SPO and RANDOM*.

result of this comparison: TB-SPO performed substantially better than SPO$^+$. Figure 3 sheds some light on this result, showing that the evaluation of SPO$^+$'s initial LHD already required about 100 seconds (of the total time budget of 3 600 seconds) for the easiest problem instance. We also tested a TB-SPO variant using the same initial LHD as SPO$^+$; at the end of the time budget, this variant performed significantly better than SPO$^+$ and not significantly worse than the regular version of TB-SPO. For QCP-q095 and all three SWGCP instances (figures not shown), the LHD took the full time budget of 3 600 seconds and hence SPO$^+$ did not improve over the default at all. Overall, we conclude that the removal of the initial LHD phase helped SPO to find good parameter settings quickly, and that the time-bounded intensification mechanism led to significant performance improvements for larger time budgets.

Interestingly, our intensification criterion did not only improve SPO, it also transformed pure random search into a competitive parameter optimization procedure. Specif-

ically, we studied a simple procedure—dubbed RANDOM*—that samples the list of new promising parameter settings, $\vec{\Theta}_{new}$, uniformly at random from the region of interest but still applies our intensification mechanism. As we show in Table 1, RANDOM* never performed statistically-significantly better or worse than TB-SPO. Thus, we conclude that the GP model used in TB-SPO does not justify its (conceptual and computational) complexity in this setting. In the next section, we will study whether this conclusion still holds when we use approximate GP models instead.

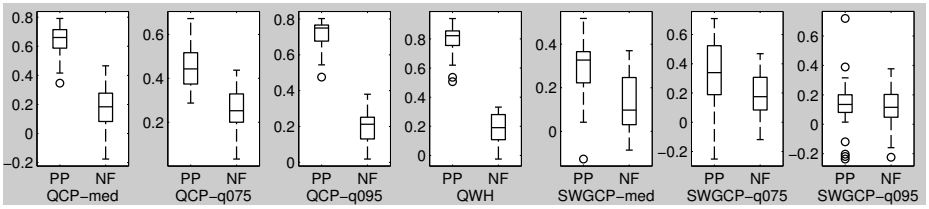
4.3 Experimental Evaluation of Model Performance

Next, we used our seven parameter optimization scenarios to compare the quality of the GP model used in all SPO variants so far, and the PP approximation to GP models. As training data for each model, we employed $1\,001$ data points: single runtimes of SAPS for its default and for $1\,000$ randomly-sampled parameter settings. Some of these runs timed out after $\kappa_{max} = 5$ seconds; according to our penalized average runtime (PAR) criterion with penalty constant 10, we counted these as 50 seconds, learning models that directly predict PAR.

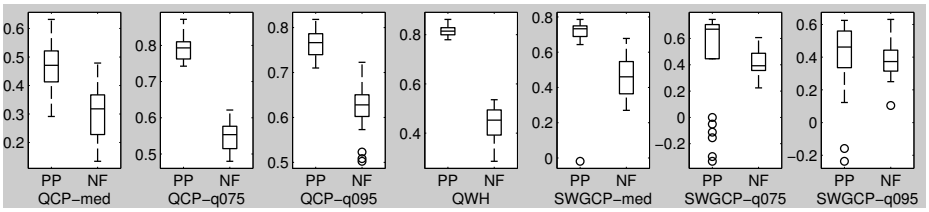
Since the main purpose of models in SMBO is to propose *good* parameter settings to be evaluated, we are mostly interested in their ability to accurately predict the performance of such good settings (as opposed to accurate predictions throughout the entire space). Thus, for each parameter optimization scenario, we employed a test set of 100 high-quality parameter settings, determined as follows. We executed 25 runs of SPO$^+$ and kept track of the set of parameter settings it ever labelled as incumbents during its optimization process. We then selected 100 of these settings uniformly at random.

We used this set of good parameter settings (unknown to the learning mechanisms) to evaluate three different performance measures. The *quality of predictive ranks* is the Spearman correlation coefficient between the *true* performance of our 100 test parameter settings and their *predicted* mean performance. The *EIC quality* is the Spearman correlation between the true performance of test settings and the expected improvement criterion (EIC), computed based on model predictions. Note that EIC depends both on the predictive mean *and* the predictive uncertainty; we chose this since the primary use of models in SMBO lies in selecting promising parameter settings based on their EIC. Finally, the *root mean squared error (RMSE)* is the square root of the mean difference between predicted and true performance.

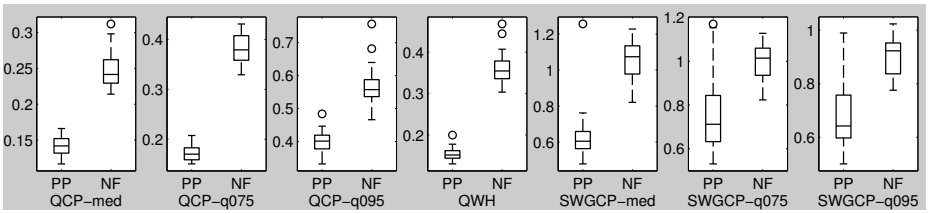
In Figure 4, we report these three measures of predictive quality for both the PP model and the noise-free GP model used in SPO, and for each of our seven parameter optimization scenarios. We also report the time required to construct the models. In this comparison, the PP model performed better with respect to almost all measures of model quality for all parameter optimization scenarios. It also took about 1.5 orders of magnitude less time to construct than the noise-free GP model. One may wonder how the PP model (an *approximate* GP model) can perform *better* than the full (noise-free) GP model. This is due to the fact that the latter clamps the observation noise to zero, asserting that the sometimes quite noisy empirical cost statistics are perfect. If we were to use a standard (noisy) GP model, we would expect it to perform somewhat better than the PP model. However, as discussed in Section 3.3, we cannot afford such a model in our sequential optimization procedure.



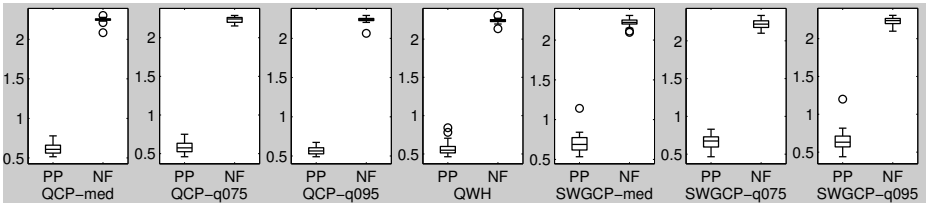
(a) Quality of predictive ranks (high is good, 1 is optimal)



(b) EIC quality (high is good, 1 is optimal)



(c) Root mean squared error (RMSE; low is good, 0 is optimal)



(d) Log_{10} of CPU time (in seconds)

Fig. 4. Comparison of models. We performed 25 runs for each model with different training but identical test data and show box plots for the respective quantities across the 25 runs. In each plot, "PP" denotes the projected process approximation of standard GP models and "NF" denotes the noise-free GP model.

We also identified an important failure mode of SPO's noise-free GP model for sequentially-gathered training data. This failure mode is due to the fact that SPO has to fit its training data perfectly instead of realizing that it is corrupted by noise—as a standard (noisy) GP or the PP approximation would. It arises when two parameter settings, $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, are explored that are very close in parameter space but—due to the randomness in the algorithm—have very different empirical cost statistics $\hat{c}(\boldsymbol{\theta}_1)$ and $\hat{c}(\boldsymbol{\theta}_2)$, even though their true costs may be very similar. The noise-free GP model—fitted on the empirical cost statistics—is forced to interpolate the two data points exactly and thus has to use very small length scales (the λ's in Equation 1). Such small length scales cause most data points to be considered "far away" from all training data points, meaning

Scenario	RANDOM*	TB-SPO	TB-SPO(PP)	FOCUSEDILS	Significantly-different pairs
SAPS-QCP-MED [$\cdot 10^{-2}$]	4.23 ± 0.15	4.32 ± 0.21	$\mathbf{4.13 \pm 0.14}$	5.12 ± 0.41	R/P, R/F, S/P, S/F, P/F
SAPS-QCP-Q075	0.19 ± 0.01	0.19 ± 0.02	$\mathbf{0.18 \pm 0.01}$	0.24 ± 0.02	R/P, R/F, S/P, S/F, P/F
SAPS-QCP-Q095	2.64 ± 1.24	2.20 ± 1.17	$\mathbf{1.44 \pm 0.53}$	2.99 ± 3.20	R/P, S/P, P/F
SAPS-QWH [$\cdot 10^{3}$]	9.88 ± 0.41	10.1 ± 0.58	$\mathbf{9.42 \pm 0.32}$	10.6 ± 0.49	R/P, R/F, S/P, S/F, P/F
SAPS-SWGCP-MED	0.17 ± 0.02	0.18 ± 0.03	$\mathbf{0.16 \pm 0.02}$	0.27 ± 0.12	R/P, R/F, S/P, S/F, P/F
SAPS-SWGCP-Q075	0.22 ± 0.03	0.24 ± 0.04	$\mathbf{0.21 \pm 0.02}$	0.35 ± 0.08	R/F, S/P, S/F, P/F
SAPS-SWGCP-Q095	0.28 ± 0.10	0.25 ± 0.05	$\mathbf{0.23 \pm 0.05}$	0.37 ± 0.16	R/F, S/P, S/F, P/F

Table 2. Quantitative comparison of parameter optimization procedures. We performed 25 runs of each procedure and computed their test performance $p_{test,t}$ (penalized average runtime, PAR, over $N = 1\,000$ test instances using the methods' final incumbents $\boldsymbol{\theta}_{inc}(t)$) for a time budget of $t = 1\,800$s. Here, we give mean $\pm$ standard deviation across the 25 runs. We performed pairwise Mann Whitney U tests and list pairs of parameter optimization procedures with significantly-different test performance; 'R', 'S', 'P', and 'F' denote RANDOM*, TB-SPO, TB-SPO(PP), and FOCUSEDILS, respectively. Figure 5 visualizes this data.

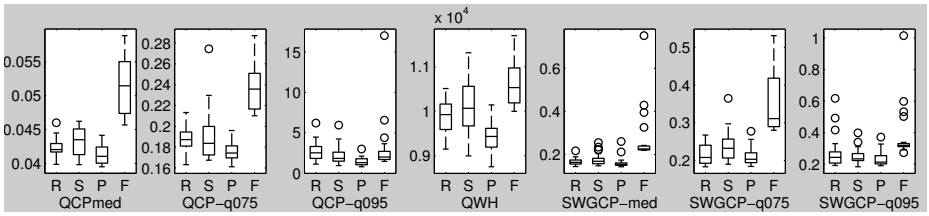


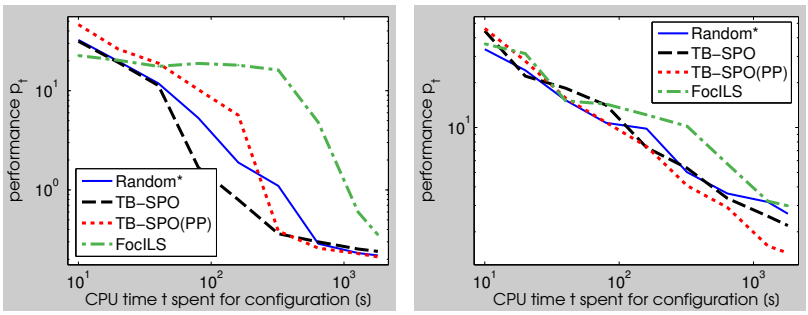
Fig. 5. Box plot comparison of optimization procedures. "R" stands for RANDOM*, "S" for TB-SPO, "P" for TB-SPO(PP), and "F" for FocusedILS. We show box plots for the data presented in Table 2; lower values are better.

that the predictions for these data points is simply the mean of the training data. This effect occurred frequently in our experiments: after a dozen iterations, in most runs of SPO variants using noise-free GP models, the model predicted the performance of all (or almost all) test parameter settings to be the mean performance of the training data points.

4.4 Final Experimental Evaluation

Finally, we experimentally compared our various parameter optimization procedures: RANDOM*, TB-SPO, TB-SPO(PP), and, for reference, FOCUSEDILS [13]. As before, we performed 25 runs for each procedure and scenario and evaluated their test performances.

In Table 2 and Figure 5, we summarize test performance at the end of the time budget. Figure 5 provides box plots, while Table 2 lists means and standard deviations across the 25 runs, as well as the result of pairwise significance tests. First, we note that RANDOM*, TB-SPO, and TB-SPO(PP) all yielded very competitive performance. In particular, in all 7 optimization scenarios, all of them yielded better mean test performance than FOCUSEDILS, and in 6 of these 7 the differences were statistically significant (in all 7 for TB-SPO(PP)). This is not entirely surprising, since FOCUSEDILS has been developed to optimize algorithms with many discrete parameters and is restricted to a discretized subspace, while TB-SPO can search the whole continuous space. It is, however, noteworthy, since FOCUSEDILS has in the past shown state-of-the-art performance for optimizing SAPS [13].



(a) SAPS-SWGCP-Q075 (b) SAPS-QCP-Q095

Fig. 6. Comparison of optimization procedures over time for two parameter optimization scenarios. We performed 25 runs of the procedures and computed their test performance $p_{test,t}$ at time steps $t = 10, 20, 40, \ldots, 1280, 1800$ seconds; we plot mean $p_{test,t}$ across the 25 runs.

Here, we use exactly the same discretization and demonstrate that our new methods—based on searching the whole continuous space—perform substantially better. Overall, TB-SPO(PP) clearly found the best parameter settings. It yielded the best mean performance in all scenarios and significantly outperformed TB-SPO in all 7 scenarios and RANDOM* in 5 of the 7 scenarios. (For the remaining 2 scenarios, SAPS-SWGCP-Q075 and SAPS-SWGCP-Q095, TB-SPO(PP) only performed insignificantly better than RANDOM*; we attribute this to the relatively hard modelling tasks for those problems—see the two rightmost columns of Figure 4).

In Figure 6, for two representative scenarios, we show mean test performance as the time budget increases. Figure 6(a) shows that TB-SPO(PP) did not always dominate all other methods for all time budgets. We observed a similar pattern in 3 other scenarios, but in those 3, TB-SPO(PP) started performing best after less than 100 seconds. Figure 6(b) shows performance for scenario SAPS-QCP-Q095, the scenario with the hardest instance, which also resulted in the largest differences between the parameter optimization procedures. For this scenario, TB-SPO(PP) yielded a more than 1.5-fold improvement over TB-SPO. It also found a solution matching the best found by TB-SPO in roughly half the time.

5 Conclusions

In this paper, we improved sequential model-based techniques for optimizing algorithms with continuous parameters on single problem instances. We paid special attention to the issue of time, taking into account both the overhead incurred by the use of models and the fact that the target algorithm's runtime often varies considerably across different parameter settings, especially if our objective is to minimize algorithm runtime. In particular, we augmented the Sequential Parameter Optimization (SPO) framework to (1) avoid performing a costly initial design and (2) introduce a time-bounded intensification strategy; we dubbed the resulting time-bounded SPO version TB-SPO. We also (3) employed an approximate version of Gaussian process (GP) models to reduce the computational complexity of constructing and using models. Our experiments for optimizing a local search algorithm for seven different SAT instances demonstrated that mechanisms (1) and (2) substantially sped up SPO. The approximate GP model performed much better than the previously-used noise-free GP model, while only imposing about 1/30 of the

overhead. Consequently, this model led to significant performance improvements of the TB-SPO framework.

In future work, we plan to extend our techniques to include the optimization of algorithms with categorical parameters, as well as optimization across multiple instances. In order to further reduce the computational time required for parameter optimization, we plan to develop approaches that actively select the cutoff time to be used for each run of the target algorithm.

References

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

[2] Ansotegui, C., Sellmann, M., and Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of solvers. In *Proc. of CP-09*, pages 142–157.

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

[4] Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In *Proc. of MH-07*, pages 108–122.

[5] Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation: The New Experimentalism*. Natural Computing Series. Springer Verlag, Berlin.

[6] Bartz-Beielstein, T., Lasarczyk, C., and Preuss, M. (2005). Sequential parameter optimization. In B. McKay et al, editor, *Proc. of CEC-05*, pages 773–780. IEEE Press.

[7] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proc. of GECCO-02*, pages 11–18.

[8] Coy, S. P., Golden, B. L., Runger, G. C., and Wasil, E. A. (2001). Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97.

[9] Huang, D., Allen, T. T., Notz, W. I., and Zeng, N. (2006). Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3):441–466.

[10] Hutter, F., Bartz-Beielstein, T., Hoos, H. H., Leyton-Brown, K., and Murphy, K. P. (2009a). Sequential model-based parameter optimisation: an experimental investigation of automated and interactive approaches. In *Empirical Methods for the Analysis of Optimization Algorithms*. Springer Verlag. To appear.

[11] Hutter, F., Hoos, H. H., Leyton-Brown, K., and Murphy, K. P. (2009b). An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proc. of GECCO-09*, pages 271–278.

[12] Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009c). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306.

[13] Hutter, F., Hoos, H. H., and Stützle, T. (2007). Automatic algorithm configuration based on local search. In *Proc. of AAAI-07*, pages 1152–1157.

[14] Hutter, F., Tompkins, D. A. D., and Hoos, H. H. (2002). Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proc. of CP-02*, pages 233–248.

[15] Quinonero-Candela, J., Rasmussen, C. E., and Williams, C. K. (2007). Approximation methods for gaussian process regression. In *Large-Scale Kernel Machines*, Neural Information Processing, pages 203–223. MIT Press, Cambridge, MA, USA.

[16] Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

[17] Santner, T. J., Williams, B. J., and Notz, W. I. (2003). *The Design and Analysis of Computer Experiments*. Springer Verlag, New York.

[18] Tompkins, D. A. D. and Hoos, H. H. (2004). UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In *Proc. of SAT-04*, pages 306–320.