Tuning metaheuristics by sequential optimisation of regression models.[☆]

Áthila R. Trindade[a,b], Felipe Campelo[c,d,*]

[a] *Graduate Program in Electrical Engineering, Universidade Federal de Minas Gerais*
Av. Antônio Carlos 6627, Belo Horizonte 31270-010, Brazil
[b] *Department of Computing, Universidade Federal dos Vales do Jequitinhonha e Mucuri*
Rua da Glória 187, Diamantina 39100-000, Brazil
[c] *Department of Electrical Engineering, Universidade Federal de Minas Gerais*
Av. Antônio Carlos 6627, Belo Horizonte 31270-010, Brazil
[d] *School of Engineering and Applied Sciences, Aston University, Birmingham B4 7ET, UK*

Abstract

Tuning parameters is an important step for the application of metaheuristics to specific problem classes. In this work we present a tuning framework based on the sequential optimisation of perturbed regression models. Besides providing algorithm configurations with good expected performance, the proposed methodology can also provide insights on the relevance of each parameter and their interactions, as well as models of expected algorithm performance for a given problem class, conditional on the parameter values. A number of test cases are presented, including the use of a simulation model in which the true optimal parameters of a hypothetical algorithm are known, as well as usual tuning scenarios for different problem classes. Comparative analyses are presented against Iterated Racing, SMAC, and ParamILS. The results suggest that the proposed approach returns high quality solutions in terms of mean performance of the algorithms equipped with the resulting configurations, with the advantage of providing additional information on the relevance and effect of each parameter on the expected performance.

[☆]This work has been partially funded by Brazilian research agencies CNPq (grant: 404988/2016-4), CAPES, and Fapemig (grant: APQ-01099-16).

[*]Corresponding author

Email addresses: rochaathila@gmail.com (Áthila R. Trindade), f.campelo@aston.ac.uk (Felipe Campelo)

Keywords: Parameter Tuning, Metaheuristics, Regression modelling

1. Introduction

Metaheuristics such as evolutionary algorithms [1, 2] represent a class of computational problem solvers subject to stochastic behaviour, determined in part by the values of user-defined parameters. These parameters are responsible for determining the global-local exploration profile, solution quality, and efficiency of the algorithm when searching for solutions in the objective space. Poor choices of parameter values can result in low performance of the method, even if the implementation is done properly, while well-chosen values can lead the algorithm to consistently return high-quality solutions. Moreover, good parameter configurations are often problem-dependent [3], which limits the utility of looking for one-size-fits-all configurations and requires the development of efficient strategies for tuning parameters based on a limited sample of representative instances of the problem class of interest.

Assuming that parameters can assume several (sometimes infinitely many) values, a possibly very large number of combinations of parameter values – called here *candidate configurations*, or simply *configurations* – can be considered for an algorithm when solving a given problem. There are two sources of random variation in the expected performance of an algorithm (equipped with a given configuration) when solving instances of a given problem class: the uncertainty due to the instance being solved, which gives rise to an *across-instances variance*; and the uncertainty due to the stochastic behaviour of the metaheuristic itself, which results in a *within-instance variance* [3]. Due to these random influences on the observed performance of a given algorithm configuration, several researchers have proposed strategies for recommending candidate configurations based on statistical concepts, in a process commonly referred to as *parameter tuning* [4, 5].

This work is focused on the application of statistical modelling to the development of tuning approaches. More specifically, we present a modular frame-

work for implementing parameter tuning methods, which is based on concepts drawn from Sequential Model Based Optimisation (SMBO) strategies [6, 7]. The proposed framework is aimed not only at returning algorithm configurations that are well-adjusted for particular problem classes, but also to provide statistical models capable of supporting further investigations on the relative relevance of algorithm parameters and interaction effects, as well as estimations of expected algorithm performance given new sets of parameter values.

The remainder of this paper is organised as follows. We start by formally stating the *Parameter Tuning Problem* that we are attempting to solve (Section 2), and briefly reviewing the most widely used parameter tuning methods (Section 3). The proposed tuning framework is introduced in Section 4. To illustrate its use, we consider three tuning experiments, and contrast the results obtained by the proposed method against those returned by Iterated Racing (Irace) [8], SMAC [9] and ParamILS [10]. Finally, some conclusions and possibilities of future works are explored in Section 6.

2. The Parameter Tuning Problem

In this work we are interested in tuning algorithm parameters for a given problem class of interest, i.e., finding the combination of parameter values that results in the best expected performance of a given algorithm on instances belonging to a given family of problems. Here we present a formalisation of this problem, based on a description originally presented by Birattari [11, 3].

Assume that we have an algorithm containing k free parameters to be set by the user, and let θ_i denote a list of length k containing specific parameter values for that algorithm. We refer to θ_i as a candidate configuration for the algorithm under study, with $\Theta = \{\theta_1, \theta_2, \dots\}$ representing the set of all possible parameter configurations for that algorithm.[1] Similarly, let γ_j denote a given problem instance belonging to a problem class of interest, denoted by $\Gamma =$

[1] For the sake of simplicity, in the remainder of this work we refer to *the algorithm equipped with a given set of parameter values* as simply *a configuration*.

$\{\gamma_1, \gamma_2, \dots\}$. Also, let X_{ij} be a random variable representing the performance[2] of a candidate configuration $\theta_i \in \Theta$ on a given instance $\gamma_j \in \Gamma$, with φ_{ij} denoting a statistical parameter of X_{ij} that can be used to quantify the general quality of configuration θ_i as a solver of instance γ_j, e.g., the mean or median of X_{ij}.

Let $\Phi_{i:\Gamma} = \{\varphi_{ij} : \gamma_j \in \Gamma\}$ denote the set of quality values of a candidate configuration θ_i for all instances belonging to problem class Γ; and $\mu_{i:\Gamma}$ denote a statistical parameter of $\Phi_{i:\Gamma}$ which is of interest when comparing different configurations, e.g., the mean or the median performance across all instances belonging to Γ. Under these definitions, the parameter tuning problem tackled in this work can be defined as:

$$\text{Find } \theta^* = \arg\max_{\theta \in \Theta} \quad \mu_{i:\Gamma}, \tag{1}$$

that is, the problem of finding the configuration that maximises the performance of a given algorithm for a given class of instances. Automated approaches for addressing this problem generally try to obtain θ^* using information from a finite subset of problem instances.

An important point to be aware of is that the instances used for tuning are usually not the ones that are relevant in practice: an underlying assumption of methods that attempt to solve the problem defined above is that the instances used for tuning can be regarded as a representative sample of the problem class of interest, and can therefore be used for modelling and inference of the expected behaviour of the algorithm for that problem class.

It must also be highlighted that, under this definition of the parameter tuning problem, the *independent observations* to be used in any statistical modelling or inferential procedure refer to individual estimates of performance of a given configuration on a given instance, i.e., to individual values of φ_{ij}. Repeated runs of a given configuration on the same instance are useful for improving the accuracy of estimates of these performance values, but cannot count as

[2]Measured according to a given indicator of choice. In the remainder of this work, we assume the use of indicators for which larger values represent better performance.

independent degrees-of-freedom for the statistical procedures. Failure to account for this particular fact would result in pseudoreplication [12, 13], a violation of the assumption of independence underlying the statistical approaches used in most tuning procedures that leads to inflated type-I errors in inferential tests, and to artificially reduced standard errors in descriptive models.

In the next section we review some of the most common approaches used to tackle the parameter tuning problem. While in most cases the problem is not explicitly stated as above, the workings of these methods indicate that in most cases this is the problem (or at least one of the problems) they attempt to solve. After briefly discussing the existing approaches, we will present our proposed tuning framework in Section 4.

3. Overview of Parameter Tuning Methods

A variety of different tuning methods have been proposed over the years to determine the best configurations of algorithms when solving a given problem. Based on their working mechanisms and design principles, it is possible to group these methods in three major categories: racing methods, SMBO methods, and hyper-heuristics. In this section we review the most widely used methods from each category.

3.1. Racing Methods

The basic concepts of racing methods were initially proposed in the machine learning literature for solving the *model selection problem* [3]. The basic idea of these methods [14, 15] is that the search for the best model structure can be sped up by discarding inferior candidate models as soon as sufficient statistical evidence is gathered against them. A similar concept is used by racing methods for parameter tuning: discard candidate configurations as soon as they are detected as inferior according to some statistical criteria.

The most relevant methods in this class are all based on concepts originally introduced in the form of the F-Race [16]. The main concept behind F-Race is to

iteratively evaluate a given set of candidate configurations on a finite number of instances, gradually building statistical evidence until it is possible to conclude, at a predefined level of confidence, that one or more candidate configurations are significantly worse than the others. Once this is determined, those inferior configurations are eliminated and the process continues with the remaining ones. F-Race stops when a given termination criterion is observed, e.g., the maximum computational budget is used or the number of remaining configurations falls below a given threshold. At each iteration this method employs Friedman tests [17] as their main inferential procedure, followed (if statistically significant differences are detected) by post-hoc nonparametric pairwise comparisons between the estimated best configuration and all others. Configurations whose median performance is detected as significantly worse than that of the best one are discarded from the race. The F-Race method then proceeds by evaluating the remaining configurations on more instances, iteratively increasing the statistical power of the tests and enabling the detection of smaller differences in median performance. The method stops when only a single configuration remains, a given number of instances have been sampled, or a predefined computational budget has been exhausted.

Improvements to F-Race were proposed in the form of the Iterated F-Race (I/F-Race) method [18], later generalised as Iterated Racing (Irace) [8]. I/F-Race works by iteratively applying F-Race, generating new candidate configurations at each iteration by sampling from a multivariate random distribution of parameter values that is biased by the best configurations returned in the previous iterations [19]. This biased sampling drives the search process towards obtaining candidate configurations that are similar to the best ones observed up to a given iteration. *Iterated Racing* allows the use of different statistical tests in place of the Friedman test, prevents premature convergence of the tuning method by means of soft restart rules, and include elitist options to force the preservation of high-quality candidate configurations.

3.2. SMBO Methods

Tuning methods based on the sequential model-based optimisation (SMBO) approach are motivated by results from the literature on statistical modelling and black-box optimisation methods. From an initial set of observations of performance over the space of configurations, SMBO methods fit one or more response surfaces, which are then used to determine which new configurations should be sampled. These new results are then added to the existing sample, and used to update the response surfaces. As iterations progress, SMBO methods tend to generate models that are increasingly biased towards those regions of the parameter space which contain configurations with good performance. The three most widely known tuning methods based on SMBO are Sequential Parameter Optimisation (SPO) [20], BONESA [4], and Sequential Model-Based Algorithm Configuration (SMAC) [9], which are briefly discussed below.

Sequential Parameter Optimisation was proposed in 2005 [20, 6], and is based on a strategy of iteratively improving a prediction model to reveal the relationship between parameter values and algorithm performance. This model is then used to select the most promising values for the parameters. In the first iteration of SPO a few candidate configurations are generated using Latin Hypercube Sampling (LHS) [21, 22] over the space of algorithm parameters. These candidate configurations are evaluated on a problem instance, and this information is used to fit a statistical prediction model. The standard initial model used by SPO is a second-order linear regression model, but regression trees and Kriging have also been employed [6]. Based on the candidate configuration with the best observed performance and on the response surface, new candidate configurations are generated so as to maximise the probability, conditional on the available information, that they will present good performance values. These new points are evaluated and an updated model is fit, in a process that iterates until a predefined termination criterion is reached. At each cycle, the number of evaluations of each candidate configuration on the problem instance is increased, obtaining more accurate estimations of average performance. Besides searching for the best configuration, SPO also allows the user to analyse the

variation of algorithm behaviour with its parameter values using the statistical models generated, thereby enabling deeper experimental investigations and experiment-driven algorithm development.

BONESA [4] is a tuning method based on *learning* and *searching* loops. These two modules continuously exchange information as iterations progress: the learning loop uses a prediction model to compare candidate configurations, while the searching loop is responsible for sampling new candidate configurations based on the results of the learning module. The distinguishing feature of this method is its multi-objective approach: to select the best parameter values for a given problem class, BONESA uses a Pareto strength approach [4] and attempts to simultaneously maximise the performance of the algorithm for all problem instances used in the tuning effort.

In the first iteration, BONESA randomly samples a number of candidate configurations, evaluating them once for each available tuning instance. The learning loop uses this information to predict the utility values for new candidate configurations, using an approach based on the weighted average of the utilities of the nearest Neighbors of the proposed configurations. These predicted utilities are then used for comparing the candidate configurations using a criterion based on Pareto dominance and an adaptation of Welch's t test [23]. The results of the tests are then aggregated and used to calculate the Pareto strength of each candidate configuration [4] and to generate new configurations (based on the best ones), for which the Pareto strength is also calculated. Then, those with the highest Pareto strength values are selected to compose the new set of configurations to be evaluated on the tuning instances. The method iterates until a given stop criterion is reached.

Finally, the Sequential Model-Based Algorithm Configuration (SMAC) method [9, 24] was, similarly to the SPO, initially designed for tuning algorithm parameters on a single problem instance.[3] The method generates an initial set of candidate configurations and evaluates their performance on the instance. Based on

[3]Both methods can, however, be adapted for tuning algorithms for problem classes.

this information, it fits a predictive model of performance over the space of parameter values, and then performs a multi-start search for finding the candidate configuration that maximises an expected positive improvement function. This new candidate configuration is then evaluated and added to the pool of candidate configurations, and the process is repeated. SMAC has been used with different types of prediction model, including Gaussian Processes and Random Forests; and different search strategies, including DIRECT and CMA-ES.

3.3. Hyperheuristics

The term *hyperheuristics* [25, 26, 27, 28] is used here to classify those tuning methods which consist in the application of metaheuristics for obtaining the best parameter values of algorithms, trying to solve the parameter tuning problem by directly tackling its optimisation formulation, discussed in Section 2. While in principle any optimisation approach could be used to solve the parameter tuning problem, knowledge about the characteristics of this problem have motivated the development of specific strategies. Three of the most common ones are REVAC [29, 30], ParamILS [10] and CRS-Tuning [31], as presented below.

Nannen and Eiben proposed a parameter tuning method for Evolutionary Algorithms called *Relevance Estimation and Value Calibration* (REVAC) [29, 30], which aims to answer questions related to two aspects of algorithm design and configuration: (i) which of the free parameters of a given method are in fact relevant, i.e., effectively influence the performance of the algorithm; (ii) for those parameters that are in fact relevant, which values lead to the best performance of the algorithm.

REVAC is itself configured as an evolutionary strategy. The method begins with a population of randomly generated candidate configurations, which are evaluated according to a performance function, and new candidate configurations are obtained using usual recombination and mutation operators [30]. At each iteration the marginal probability density functions for each parameter of the algorithm are estimated from the population of candidate configurations. The Shannon entropy of these distributions is used to estimate the relevance of

each parameter. Parameters for which entropy decreases quickly as iterations progress need little information to be tuned, and are therefore considered more relevant to the performance of the EA. Conversely, those for which entropy does not decrease are considered less relevant, and may be discarded or receive arbitrary values. The method iterates until predefined stop criteria are reached.

ParamILS [10] is a framework of tuning methods, which is based on Iterated Local Search (ILS). Starting from a given initial candidate configuration, at each iteration the incumbent configuration is perturbed and undergoes a first improvement local search, to generate a new candidate configuration that replaces the incumbent one if it presents better performance. The neighbourhood of a given configuration is the set of all configurations that differ from it in a single parameter, and the determination of whether a candidate configuration is better than the incumbent one is performed using statistical tests, with problem instances as a blocking factor [10, 23]. Variants of this basic algorithm include [10] *FocusedILS*, which adaptively selects the number of training instances; and *Adaptive Capping of Algorithm Runs*, which controls the cutoff time for each run of the candidate configurations.

CRS-Tuning [31] is a tuning method for numerical and categorical parameters, which is composed of an evolutionary strategy combined with an approach called Chess Rating System (CRS) by its authors, which is used to rank the configurations. In this method initial configurations are randomly generated, and for each tuning instance each configuration is ran n times. Candidate configurations are compared pairwise, and the results of these comparisons (in terms of wins, losses and draws) are used to calculate a rating R for each configuration, which describes their relative qualities. Configurations that are considered as significantly worse than the best one are eliminated, and finally crossover and mutation operators are applied to the surviving configurations to create new ones, and the procedure iterates. The procedure is run until the maximum number of executions has been reached.

4. Proposed Tuning Framework

In this section we propose a modular structure for tackling the parameter tuning problem presented in Section 2. The proposed framework, which we will refer to as *MetaTuner*, can be used to instantiate distinct tuning approaches through the adoption of specific methods for each of its components, depending on the nature of the tuning process at hand. This modular approach results not only in a greater flexibility for the framework, but is also useful for faster development and testing of proposed improvements.

The proposed approach is based on a common assumption in the design of computer experiments [32], that if the number of instances and of candidate configurations is sufficient, enough information will be gathered so that the resulting response surfaces are somehow representative of the expected performance landscape of the algorithm for the problem class of interest. Under this assumption, optimising these surfaces will tend to drive the method towards regions of the parameter space containing good candidate configurations, allowing the method to iteratively concentrate its efforts on those regions of the parameter space with the highest average performance.

The general aspects of the proposed framework can be easily explained from the structure presented in Algorithm 1.[4]. The method starts by sampling a few configurations, which are evaluated on a randomly sampled initial set of tuning instances. The performance results obtained are then used for fitting a regression model of the expected performance of configurations on the problem class of interest. The regression model is then subject to perturbations (e.g., by perturbing the fitted parameters), resulting in a number of additional response surfaces. For each surface (including the unperturbed one) an optimisation process is executed, returning a new candidate configuration which maximises the value of the estimated average performance value for that model. These new

[4] An open-source implementation is available in the form of R package *MetaTuner* (`https://github.com/fcampelo/MetaTuner`). Details about the structure of the tool, as well as a full usage example, are available in the package documentation.

Algorithm 1 Proposed tuning framework

Require: Search space (Ω); Tuning instances (Γ_S); number of initial configs. (m_0); number of new configs/iter. ($m_\star$); number of initial instances (N_0); number of addit. instances/iter. ($N_\star$); size of archive ($n_\mathcal{E}$).

$t \leftarrow 0$

$\mathcal{A}^{(t)} \leftarrow$ GenerateInitialSample (Ω, m_0) ▷ Sample initial configurations

$\Gamma_\mathcal{A}^{(t)} \leftarrow$ SampleWithoutReplacement $(\Gamma_S, N_0 - N_\star)$ ▷ Sample initial instances

$\mathcal{P}_\mathcal{A}^{(t)} \leftarrow$ EvaluateConfigurations $\left(\mathcal{A}^{(t)}, \Gamma_A^{(t)}\right)$ ▷ Evaluate configs on instances

$\mathcal{E}^{(t)} \leftarrow \mathcal{A}^{(t)}$ ▷ Initialise elite archive

while Stop criteria not met **do**

  $t \leftarrow t + 1$

  if New instances available **then**

    $\Gamma' \leftarrow$ SampleWithoutReplacement $\left(\Gamma_S \backslash \Gamma_\mathcal{A}^{(t-1)}, N_\star\right)$ ▷ Sample new instances

    $\mathcal{P}' \leftarrow$ EvaluateConfigurations $\left(\mathcal{E}^{(t)}, \Gamma'\right)$ ▷ Eval. elite configs on new instances

    $\Gamma_\mathcal{A}^{(t)} \leftarrow \Gamma_\mathcal{A}^{(t-1)} \bigcup \Gamma'$ ▷ Update archive of instances visited

    $\mathcal{P}_\mathcal{A}^{(t)} \leftarrow$ Update $\left(\mathcal{P}_\mathcal{A}^{(t-1)}, \mathcal{P}'\right)$ ▷ Update archive of config. performances

  else

    $\Gamma_\mathcal{A}^{(t)} \leftarrow \Gamma_\mathcal{A}^{(t-1)}$

    $\mathcal{P}_\mathcal{A}^{(t)} \leftarrow \mathcal{P}_\mathcal{A}^{(t-1)}$

  end if

  $\mathcal{S}_1^{(t)} \leftarrow$ FitModel$\left(\mathcal{A}^{(t-1)}, \mathcal{P}_\mathcal{A}^{(t)}\right)$ ▷ Fit regression model

  $\theta_1'^{(t)} \leftarrow$ Optimise $\left(S_1^{(t)}\right)$ ▷ Find configuration that optimises $\mathcal{S}_1^{(t)}$

  for $j \in \{2, \ldots, m_\star\}$ **do**

    $\mathcal{S}_j^{(t)} \leftarrow$ PerturbModel $\left(S_1^{(t)}\right)$ ▷ Generate perturbed model

    $\theta_j'^{(t)} \leftarrow$ Optimise $\left(S_j^{(t)}\right)$ ▷ Find configuration that optimises $\mathcal{S}_j^{(t)}$

  end for

  $\mathcal{C}^{(t)} \leftarrow \left\{\theta_j'^{(t)} : j = 1, \ldots, m_\star\right\}$

  $\mathcal{P}_\mathcal{C}^{(t)} \leftarrow$ EvaluateConfigurations $\left(\mathcal{C}^{(t)}, \Gamma_\mathcal{A}^{(t)}\right)$ ▷ Evaluate candidate configs

  $\mathcal{A}^{(t)} \leftarrow \mathcal{A}^{(t-1)} \bigcup \mathcal{C}^{(t)}$ ▷ Add candidate configs to archive

  $\mathcal{P}_\mathcal{A}^{(t)} \leftarrow$ Update $\left(\mathcal{P}_\mathcal{A}^{(t)}, \mathcal{P}_\mathcal{C}^{(t)}\right)$ ▷ Update archive of config performances

  $\mathcal{E}^{(t)} \leftarrow$ SelectKBest $\left(\mathcal{A}^{(t)}, \mathcal{P}_\mathcal{A}^{(t)}, K = n_\mathcal{E}\right)$ ▷ Update elite archive

end while

$\mathcal{P}_\mathcal{E}^{(t)} \leftarrow$ RetrieveElitePerformances$\left(\mathcal{E}^{(t)}, P_\mathcal{A}^{(t)}\right)$

return Elite configurations ($\mathcal{E}^{(t)}$) and their estimated performance ($\mathcal{P}_\mathcal{E}^{(t)}$).

candidate configurations are then evaluated on all instances sampled so far, and added to an archive. Finally, the archive is truncated to a given size, maintaining only the candidate configurations with the best expected performance value for the problem class. The whole process then iterates by sampling a few more instances (if available), and proceeds until a predefined stopping condition is reached. If no new instances are available, the process simply continues generating new candidate configurations at each iteration, which are then evaluated on all instances. This process proceeds until a predefined stopping condition is reached.

In the remainder of this section we detail the implementation of an initial instantiation of the proposed framework. Although in this work we focus mainly on numeric parameters, notice that categorical / symbolic parameters can also be considered by (i) dummy encoding of categorical variables, or (ii) use of different regression models such as those used, e.g., for analysis of covariance [23, 33], which can be easily incorporated into the modular structure of the proposed framework.

4.1. Generation of Initial Candidate Configurations

Considering the importance of gathering enough information for generating a reasonable first set of regression models, a point of particular importance is to ensure that the sampling of initial candidates (line 2 of Algorithm 1) be well-distributed in the parameter space, so that the method will have the chance to investigate different regions of the space of parameters.

There are a few strategies that guarantee a well-spread initial sampling in continuous spaces. Some of the most widely known include Latin Hypercube Sampling (LHS) [21, 34], low-discrepancy sequences of points (LDSP) [35], and uniform designs (UD) [36]. Since LHS is possibly the one most widely used in computational experiments [37], the version of MetaTuner described here uses this particular sampling scheme for generating its initial set of candidate configurations.

Before proceeding, it is important to understand that performance degradation can occur if the parameters being tuned can assume values on possibly very different scales – e.g., in the case of polynomial mutation [38], the rate parameter exists in the $[0, 1]$ interval, while η can in principle assume any non-negative value. This is a well-known issue in the regression and machine learning literature [39], which can be avoided when tuning numerical parameters by simply rescaling all parameters to a common scale, e.g., $[0, 1]$:

$$\theta'_{i(l)} = \frac{\theta_{i(l)} - \theta_{(l,min)}}{\theta_{(l,max)} - \theta_{(l,min)}}, \quad i = 1, \ldots, m; \tag{2}$$

where $\theta_{i(l)}$ is the value of the l-th component of candidate configuration θ_i, and $\theta_{(l,min)}$ $\theta_{(l,max)}$ denote the lower and upper allowed values for the l-th parameter being tuned. Notice that this require all parameters to have upper and lower limits, which is generally not a problem – even for parameters that are theoretically unbounded, it is generally possible to define reasonable bounds based on theory or previous experience.

4.2. Evaluation of Candidate Configurations and Estimation of Quality Value

Given the possibly heterogeneous nature of the tuning instances and the expected variations of performance of different configurations, it is possible that the distributions of X_{ij}, i.e., of the performance of candidate configurations on the instances, exist on very different scales. While some regression models, particularly quantile regression [40], can deal with these differences of scale relatively well, most have their performance heavily degraded in the presence of such large scale differences and heterogeneity of variances. To alleviate these particular problems, the performance of candidate configurations on the tuning instances (lines 4, 10 and 24 of the algorithm) is calculated by running the configurations on the test instances and transforming the output (i.e., the value of the quality indicator used) to a common scale.

Let $x_{ij} \sim X_{ij}$ denote a single observation of the performance of configuration θ_i on instance γ_j. The observed performance in this case is calculated by linearly

scaling x_{ij} to the interval $[0, 1]$:

$$x'_{ij} = \frac{x_{ij} - x_{min,j}}{x_{max,j} - x_{min,j}}, \tag{3}$$

where $x_{min,j},\ x_{max,j}$ denote the smallest and largest values observed so far for instance j, across all configurations already evaluated. Once these values are calculated for all instances visited by a given configuration θ_i, the summary performance estimator p_{θ_i} is calculated as the sample average of the x'_{ij} values associated with that configuration. Notice that this average can be the simple mean, trimmed mean, median, or any other indicator of location. In the version used here, the median is employed due to its robustness to outliers and distributional asymmetries, an important characteristic when dealing with possibly heterogeneous tuning instances.

Notice from Algorithm 1 that, at each iteration, configurations in the elite archive $\mathcal{E}^{(t)}$ are evaluated on the $N_\star$ new instances, while configurations that were not selected are not, even though all are used for modelling the average behaviour This is done to increase the accuracy of estimation of the average performance on the most promising configurations, and can be used, for instance, to attribute weights to each observation in the regression modelling. At each iteration, the new configurations generated by optimising the estimated response surfaces, are also evaluated on all instances visited so far, since they are expected to yield good average performances.

Finally, it must be highlighted that the values of p_{θ_i} need to be recalculated at each iteration for all configurations, since the normalising bounds can change across iterations.

4.3. Regression modelling

The role of regression modelling in the proposed tuning framework is to enable predictions of the expected performance of a given configuration, conditional on its parameter values. For this, modelling strategies need ideally to be i) reasonably accurate; ii) capable of working with relatively few data points; iii) computationally inexpensive (at least relatively to the cost of evaluating the

configurations); and iv) parsimonious in terms of the number of coefficients in the model. Another desirable trait is that the regression models scale reasonably well up to a reasonable number of parameters, e.g., 10 (which is a reasonable upper limit for free parameters that are expected to be adjusted by the user).

For continuous parameters, usual models include linear regression with ordinary [23] or weighted [41] least square estimators; quantile regression [42]; and ridge or lasso regression [43], among others. Even that all these kinds of regression modelling are used in this work, it is worth highlighting the shrinkage characteristics of Lasso and Ridge, which can contribute for providing regression models that prioritize the most important parameters. This characteristic is briefly introduced below.

4.3.1. Ridge and Lasso Regression

There are two reasons why ordinary least squares (OLS) regression is often inadequate [43], namely prediction accuracy and interpretation. Poor prediction accuracy can be caused by low bias and large variance of *OLS* regression. Interpretation is also often challenging, given the large number of coefficients commonly used when fitting models with several predictors.

As a remedy to both issues, some methods employ shrinkage techniques to remove coefficients that do not contribute to the explanatory power of a given model. Shrinking coefficients can be achieved, e.g., by including a penalty term in the problem of minimising the least squared errors. Considering the predictor of p_{θ_i} as a linear function of the form $\hat{p}_{\theta_i} = \beta_0 + \theta_i^T \beta : \ \beta_0 \in \mathbb{R}, \ \beta \in \mathbb{R}^p$, the problem becomes [44]:

$$\text{Find} \quad \beta^* = \underset{\beta \in \mathbb{R}^p}{\arg\min} \sum_{i=1}^{n} (p_{\theta_i} - \beta_0 - \theta_i^T \beta)^2 + \lambda \left\| \boldsymbol{\beta} \right\|_\alpha^2 \tag{4}$$

where $\lambda \in [0, \infty]$ is a regularisation parameter, and $\alpha \in \mathbb{Z}_{>0}$ regulates the order of the norm used for the penalisation term. Two special cases of this definition are the *lasso* ($\alpha = 1$) and *ridge* ($\alpha = 2$) regressions. The minimisation of (4) becomes more aggressive at shrinking coefficients towards zero (i.e., removing their associated terms from the model) as larger values of λ are used. This

regression approach can be useful in the presence of complex models with many terms, particularly when there is a large difference in the relevance of each term, as is often the case of algorithm parameters [29, 30]. In these cases, shrinkage will reduce all coefficient values, leading those least relevant to zero and amplifying the differences between them, simplifying the model and facilitating interpretation.

4.4. Generating Perturbed Models

The generation of response surfaces to be optimised at each iteration is performed in two steps: firstly, a regression model is fit using a modelling technique of choice. Secondly, the model obtained is perturbed several times, generating new response surfaces (line 20 of Algorithm 1). To generate the perturbed models, all (non-zero) coefficients of the model are subject to uniform noise. The range of this noise is defined by the standard errors of each coefficient, which can be obtained either analytically (e.g., in the case of linear regression models using OLS) or by resampling methods.

For ridge and lasso regression models, standard errors are obtained using a leave-one-out (LOO) strategy. After fitting a model using the approach described in the previous section, all coefficients that were shrunk down to zero are removed from the model. The resulting polynomial is then used as a basis for fitting k new models, each of which is fitted on a dataset obtained by ignoring the information regarding a single configuration. Based on the coefficients fit for each of these k leave-one-out models, the standard error of each coefficient is estimated as the sample standard deviation of the values obtained for that coefficient on all LOO models.

4.5. Model Optimisation

Considering that the response surfaces represent preliminary models of the average behaviour of the algorithm conditional on its parameter values, optimising them should yield a set of new candidate configurations with expected good performance. As the iterations progress and more candidate configurations are

evaluated on more instances, it is expected that the resulting models become increasingly accurate.

The main concept of the proposed tuning framework is to iteratively fit regression models of average behaviour as a function of parameter values, using increasing amounts of information, and optimising the resulting response surfaces (and perturbed versions of them, obtained by incorporating estimation uncertainties of the model coefficients - lines 18–22 of Algorithm 1) to search for more promising parameter values. The new candidate configurations returned by optimising these models are evaluated on all instances already visited by the method (line 24) and added to the archive (lines 25–26).

The optimisation approach to be used depends on the nature of the regression models, which may provide, e.g., analytical gradients or guarantees of unimodality. For more general or complex models, fast heuristics can be employed. In this work we opted for using Nelder-Mead Simplex [45, 46] to optimise the response surfaces.

4.6. MetaTuner as an SMBO method

It should be clear by now that the proposed method is situated within the scope of SMBO methods for parameter tuning. We finish this section by highlighting the similarities and differences between our work and other similar methods in the literature.

Three aspects in particular are considered here: the type of parameters that can be tuned; the ability to provide a model capable of informing the user about the algorithm behaviour and relevance of parameters; and the ability to tune algorithms for problem classes (i.e., using multiple instances) or for single instances.

Considering the types of parameters that can be tuned, the currently implemented version of MetaTuner can deal only with numeric parameters, although the proposed modular framework can be easily adapted to work with mixed parameter spaces, by modifying the regression model (e.g., using generalised linear models with mixed inputs [47]) and optimisation approaches (e.g., using

mixed-variables or bilevel optimisation methods). Currently, methods such as Irace, ParamILS, SMAC and CRS-Tuning have the ability to deal with both numerical and categorical parameters.

Another very important aspect of tuning methods is the possibility of using it not only to optimise the algorithm performance, but also to learn about the algorithm behaviour This is generally a feature of SMBO methods: besides our proposed method, SMAC, SPO and REVAC also return models of algorithm performance in terms of parameter values, which can be useful in assessing the relevance of specific parameters, as well as the sensitivity of an algorithm to their values.

In terms of the third aspect of interest, the tuning process can be focused on finding out the best parameter values for a single instance or to a problem class. Considering the single-instance scenario, only SPO was designed specially for it, whereas all other methods are multi-instances. BONESA is a somewhat hybrid case: it considers multiple instances, but instead of returning a configuration tuned for the problem class represented by those instances it provides a set of Pareto-nondominated candidate configurations, with the performance for each individual instance being considered as one individual objective.

The tuning framework proposed in this paper is intended to be applied with numeric or mixed parameters[5], return a configuration tuned for a problem class of interest, and provide a statistical model of parameters influence and relevance. Analysing it as an SMBO method, these characteristics set it apart from SPO (single-instance) and BONESA (which does not provide a model of parameters relevance). In relation to SMAC, the current implementation has the disadvantage of not yet dealing with categorical parameters. It does, however, present certain advantages that justify its proposal. First, it can easily incorporate robust regression models such as quantile regression, which works well under heteroscedasticity and in the presence of outliers (which can arise, e.g.,

[5]Even though the particular instantiation presented in this paper deals only with numeric parameters, the framework is designed to allow extensions to categorical ones as well.

from heavily heterogeneous problem classes). The use of models that incorporate implicit attribute selection, such as ridge and lasso regression, also allows the method to focus on finding out the most important parameters, as well as to return regression models even when the number of parameters is large. Also, the explicit consideration of modelling uncertainty - which motivates the use of perturbed models in the search phase of the method - allows a more comprehensive search, and can provide additional evidence at the end of the tuning process of the quality of the models fit and, consequently, their expected explanatory and predictive abilities for the performance of the tuned configurations when solving new instances from the same problem class.

Finally, even though the proposed framework does not necessarily employ evolutionary algorithms in the optimisation phase, it bears some similarities to other methods belonging to the wider class of Surrogate-Assisted Evolutionary Algorithms (SAEAs) [48, 24, 7, 49], to which several SMBO methods also belong. SAEAs are commonly employed in the solution of optimisation problems in which the computational cost of evaluating the objective or constraint functions is particularly high, which happens often in applied contexts [50, 24, 51, 52]. Actually, it is possible to express the parameter tuning problem as a noisy, expensive optimisation problem: the formulation presented in (1) already suggests tuning as an optimisation problem, where the objective is the maximisation of the expected performance of the method to a problem class of interest. The "expensive" part comes from the fact that the performance evaluation of a given candidate configuration requires the execution of not only one, but several runs of the algorithm equipped with that configuration on different instances of the problem class of interest. The "noisy" part comes from the uncertainties in the evaluation of the expected performance, which is estimated based on a finite number of runs executed on a finite subset of problem instances. This view of the tuning problem is used to motivate a simulation model employed in the experiments described in Section 5.1, where we investigate the performance of the proposed tuning method.

5. Experimental Results

To illustrate the use of the proposed approach we performed three experiments, in which we analyse the abilities and weakness of our framework, besides comparing them with some well known parameter tuning methods in the metaheuristics literature: Irace, ParamILS and SMAC.

In the first experiment, a simulation model was used to model a hypothetical average performance surface, over which random noise was added to simulate the across-instances and within-instance variance commonly experienced in real tuning scenarios. The objective of this first experiment is to evaluate the behaviour of the proposed tuning method under known, controllable conditions, which allows an exploration of the abilities and limitations of MetaTuner.

The second and third experiments contrast the performance of the proposed method against the three others already mentioned parameter tuning methods from the literature, in real-valued parameter tuning problems. In the second experiment three parameters of a well-known single objective algorithm are tuned, using two sets of challenging problems. Four different instantiations of MetaTuner are tested, in order to investigate the strengths and weaknesses of different variations of the proposed approach in relation to existing methods.

The third experiment is a comparison between four distinct variations of MetaTuner against four other tuning approaches: Irace, ParamILS and SMAC, as well as a simple random sampling method, in a scenario with a very limited number of available tuning instances. The analysis of relevance of the parameters returned by the regression models provided by the MetaTuner versions is performed in all experiments.

5.1. Simulation Model Experiment

In this experiment,[6] the expected performance of a (hypothetical) algorithm on a (also hypothetical) set of instances was represented by a simulation model

[6]This experiment was performed in a Intel Core 2 Quad Machine, with 4 2.83GHz cores, 3.7 Gib of memory running Ubuntu 18.04.

with the following structure:

$$p^k_{\theta_i;\gamma_j} = p_{\theta_i;\Gamma} + \tau_{\theta_i;\gamma_j} + \epsilon^k_{\theta_i;\gamma_j} \tag{5}$$

where:

- $p^k_{\theta_i;\gamma_j}$ represents the performance value obtained by a configuration θ_i on an instance γ_j at the k-th run;
- $p_{\theta_i;\Gamma}$ is the expected value for the performance of the configuration θ_i on the whole instance class Γ, that is, the *grand mean* of the performance of that particular configuration for the problem class of interest. This value is defined using a function over the space of parameters Θ.
- $\tau_{\theta_i;\gamma_j}$ is the deviation between the expected performance value of θ_i on γ_j and the *grand mean* of θ_i on the instance class Γ. In this model the variance of the $\tau_{\theta_i;\gamma_j}$ values is used to simulate the *across-instance variance*;
- $\epsilon^k_{\theta_i;\gamma_j}$ is the deviation between the observed performance value of θ_i on γ_j at the k-th run and the expected performance of θ_i on γ_j. The variance of the $\epsilon^k_{\theta_i;\gamma_j}$ values is used in this model to simulate the *within-instance variance*;

Equation 5 models the performance $p^k_{\theta_i;\gamma_j}$ of a candidate configuration θ_i on an instance γ_j at the k-th run as an additive effects model composed of the grand mean $p_{\theta_i;\Gamma}$, the effect $\tau_{\theta_i;\gamma_j}$of instance γ_j on that grand mean, and the variability between runs of the configuration on that particular instance, $\epsilon^k_{\theta_i;\gamma_j}$.

For this experiment the effects $\tau_{\theta_i;\gamma_j}$ and $\epsilon^k_{\theta_i;\gamma_j}$ were generated using shifted exponential distributions with zero mean and variances σ^2_{AI} (for $\tau_{\theta_i;\gamma_j}$) and σ^2_{WI} (for $\epsilon^k_{\theta_i;\gamma_j}$). The grand mean $p_{\theta_i;\Gamma}$ was represented using analytic functions with known optima, to allow the assessment of MetaTuner as a tuning approach using a hypothetical algorithm dealing with known performance landscapes.

Two functions were used to model hypothetical performance landscapes:

- **Quadratic function**:

$$f(\boldsymbol{\theta}) = 2 + 100\theta_1^2 + 5\sum_{i=2}^{n}\theta_i \ , \text{with } \begin{cases} \theta_1 \in [-10, 0] \\ \theta_i \in [0, 1], i = 2, \ldots, n \end{cases} \quad (6)$$

- **Ackley function**:

$$f(\boldsymbol{\theta}) = -a\ exp\left(-b\sqrt{\frac{1}{3}\sum_{i=1}^{3}\theta_i}\right) - exp\left(\frac{1}{3}\sum_{i=1}^{3}cos(c\theta_i)\right) + a + exp(1) \quad (7)$$

with $\theta_i \in [-32.8, 32.8],\ i = 1, \ldots, n; \quad a = 20;\ b = 0.2;\ c = 2\pi$

The dimensions of the functions were varied between 2 and 8. The first function represents a smooth, "well behaved" response landscape, and all parameter tuning methods were expected to converge to the vicinity of the global optimum value ($\boldsymbol{\theta}^* = \mathbf{0}$). Moreover, the first parameter (θ_1) is much more influential than the others, a fact that should be captured by the regression models used in MetaTuner.

The second function is used to investigate the ability of the parameter tuning methods to explore performance landscapes with multiple local optima. This function is commonly used in simulated experiments with metaheuristics, and was chosen here to represent a much more challenging average performance landscape. Actual average performance surfaces are probably located in between the two extremes represented by these test models, which are used here to investigate general performance trends.

For each function and each dimension, all parameter tuning methods were executed 30 times, under a budget of $300n$ configurations to be evaluated.[7] The variances in the model were arbitrarily chosen as $\sigma^2_{AI} = 4$ and $\sigma^2 WI = 1$. MetaTuner was set up with an initial sampling of $m_0 = 20$ configurations generated by Latin Hypercube Sampling, $n_0 = 5$ initial instances randomly drawn from the tuning set, $m_i = 5$ new configurations generated at each iteration, and $n_i = 1$ new tuning instances added to the pool at each iteration. The median

[7] This budget was determined by exploratory tests with similar functions, using Irace

was used during the tuning process as the summary function for calculating the expected performance of each configuration in all methods, except SMAC. [8] Other initial parameters of SMAC, Irace and ParamILS were set as their standard configurations. Four versions of MetaTuner were used: using Linear, Quantile, Lasso or Ridge regression, all with a polynomial model of order 3 and using Nelder-Mead for optimizing the regression models.

For ParamILS the numerical parameters must be informed as a sequence of discrete values. For the Quadratic function, the possible values for θ_1 were $\{-10.0, -9.5, \ldots, -0.5, 0.0\}$, with initial value of -5.5; for all other parameters the possible values were $\{0.00, 0.05, \ldots, 0.95, 1.00\}$, with initial value 0.50. In the case of Ackley function, for all parameters the possible states were set as 21 equally-spaced values in the range ± 32.80, with an initial value of 0.5.

To compare the output of MetaTuner versions and other parameter tuning methods, the gap between the function values associated with the best candidate configurations achieved by each approach and the optimum value were analysed Figure 1 illustrates the mean performances regarding the optimality gap.

Considering the quadratic function, Irace had clearly the worst performance among all methods, while Metatuner using Linear and Quantile models presented the best results. This was expected for Metatuner, as this simulated performance landscape can be easily represented by the polynomial form of the regression methods tested. The differences were statistically significant at the 95% confidence level (Friedman test, $p = 9.7 \times 10^{-7}$), with the Quantile version of Metatuner being significantly better (in terms of across-dimensions mean performance) to all other methods except Linear version.

As for the Ackley landscape, ParamILS exhibited a better overall performance, followed by Irace and SMAC. The Metatuner versions tested were not capable of adequately learning this more complex landscape, as it cannot be properly described by the regression models used. The differences were also

[8] The version of SMAC used in the experiments, available from `https://www.cs.ubc.ca/labs/beta/Projects/SMAC/v2.10.02/quickstart.html` , does not use the median.

detected as statistically significant ($p = 2.8 \times 10^{-9}$), with ParamILS performing significantly better than all other methods.

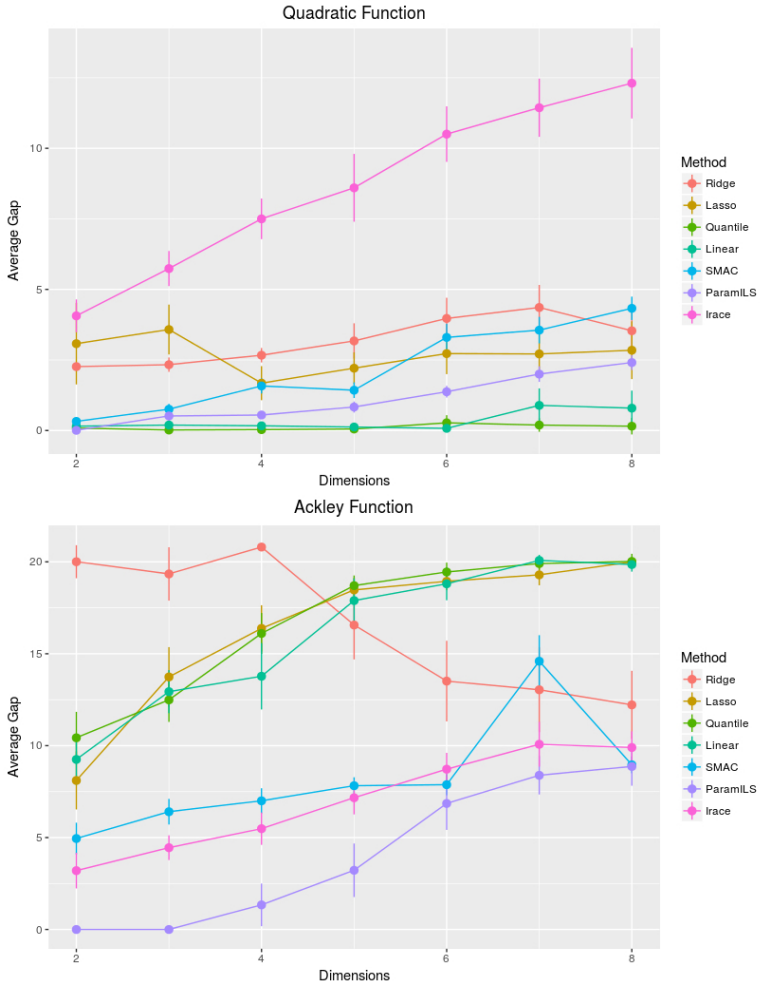

Figure 1: Point estimates and 95% confidence intervals of the mean optimality gaps, for the quadratic (top) and Ackley (bottom) simulated performance landscapes. MetaTuner versions were labelled based on the model employed.

In terms of runtime, the methods that rely on explicit modelling of the performance landscape (i.e., Metatuner and SMAC) resulted in runtimes that were not only substantially higher than those which do not (i.e., Irace and ParamILS), a problem that is amplified as the dimensions are increased, as illustrated in Fig-

ure 2.[9] This is an important point when tuning algorithms repeatedly, or for computationally "cheap" problem classes, but it can be argued that it is not a major obstacle against the use of model-based approaches for two main reasons: first, tuning is most often a one-time task, so the time required for this activity is generally not as important as the expected performance improvement it generates. Second, when tuning algorithm parameters for computationally expensive problems, even the considerably higher computational burden due to model-building can often be disregarded.

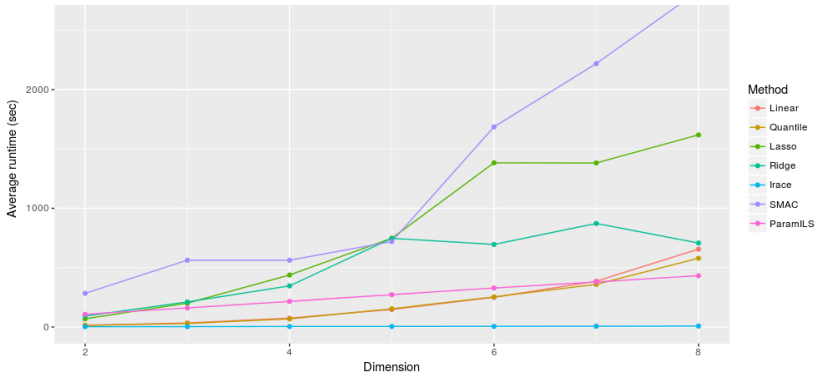

Figure 2: Average runtime of different tuning approaches for the Ackley performance landscape.

Finally, the use of simulated performance landscapes in this experiment allows us to evaluate the ability of Metatuner to detect the parameters that are the main drivers of performance - which, since Metatuner works on normalised parameter spaces, is simply a matter of examining which model components present the largest magnitudes (surprisingly, the version of SMAC used does not provide to the user a parameters model in its output). For the quadratic case, all versions of Metatuner were able to detect θ_1 as the most relevant parameter with fairly high consistency, i.e., in near all runs (tables detailing these results are provided in Appendix 1). It is worth highlighting that the main objective in this issue is to identify which parameter is more relevant (in this

[9] The results for the quadratic performance landscape were similar.

case, θ_1), and no necessarily its actual form of occurrence (in this case, θ_1^2).

Unlike the quadratic scenarios, the models output by Metatuner for the Ackley problems were more heterogeneous in terms of which parameters had the largest coefficients. This was expected, as the Ackley function does not have any parameter that stands out in terms of its influence on the function value - in a sense, the symmetry of this function means that all parameters have essentially the same relevance.

While preliminary, the results of this experiment using the simulation model can yield some interesting insights: first, they suggest that MetaTuner may be able to discover the underlying structure of the response surface in cases where the structural form of the regression models used is adequate, despite the presence of noise due to within-instances and between-instances variances – e.g., in the case of the quadratic model, where the general shape of the performance surface can be described by the polynomial form used in MetaTuner. In cases where the performance surface is expected to exhibit strong multimodality or sensitivity to parameter values, more flexible approximation models (e.g., neural networks) can be used, albeit at an increased computational cost for the tuning effort.

5.2. Tuning DE Parameters

In this experiment, the parameter tuning methods were used for tuning parameters of a "standard" Differential Evolution, DE/rand/1/bin[10]. Three parameters were selected for tuning: the mutation factor $F \in [0.1, 5]$, the crossover rate CR $\in [0, 1]$, and the multiplier $K \in [10, 20]$, used to calculate the population size as $N_{pop} = K \times dim$, with dim the dimension of the problem being solved. The stop criterion of the DE was the use of $10000 \times dim$ objective function evaluations.

The tuning process was analysed for two optimisation scenarios: a first one consisting of similar problem instances, and a second with more heterogeneous

[10]The implementation available in the *R package ExpDE* [53] was used

problems. In the homogeneous scenario the DE was used to solve 20 optimisation problems sampled from functions 15 and 21 of the BBOB benchmark set [54] with dimensions $2, 4, 6, \ldots, 40$. Four versions of MetaTuner used in the prior experiment, as well as SMAC, Irace and ParamILS, were each run 30 times, and at each run a budget of 1500 algorithm runs was used.[11]. Each of the 30 best configurations returned by each method was then used to solve 19 *validation instances*, sampled from the same BBOB functions but with dimensions $3, 5, 7, \ldots, 39$. The mean performance of each configuration on this validation set were recorded. The overall performance of each parameter tuning method was represented by the mean performance values on the validation set, of the best configurations returned on the 30 replicates.

For the heterogeneous scenario the training set was formed by 30 functions sampled from functions BBOB 1 to 24 with dimensions between 8 and 11. The validation set was composed of 20 other functions sampled from the same set. The same budget of the former scenario was used for each tuning run. With the exception of the tuning budget, all parameters of MetaTuner, SMAC, Irace and ParamILS were set as in the prior experiment.[12] Considering ParamILS, the possible values of each parameter were the following: $F \in [0.1, 0.35, 0.60, \ldots, 5]$ with initial value equal to 2.35; $CR \in [0, 0.05, 0.10, \ldots, 1]$ with initial value equal to 0.5; and $F \in [10, 11, 12, \ldots, 20]$ with initial value of 15.

Figure 3 presents the distribution of the overall mean performance of the 30 best candidates returned by all methods for the homogeneous scenario. This figure suggests that Irace has the worst results, ParamILS and SMAC the best, and the MetaTuner versions were between these bounds but the distribution of observations presents a substantial overlap. To objectively evaluate these differences, an inferential approach was employed. Preliminary analyses suggested

[11] This budget was based on the work of Nannen and Eiben [30], which used a budget of 500 runs per parameter

[12] This experiment was run in a Intel Xeon Silver machine, with 2.10GHz x 32 cores, 62.9 Gib RAM, running Ubuntu 19.04.

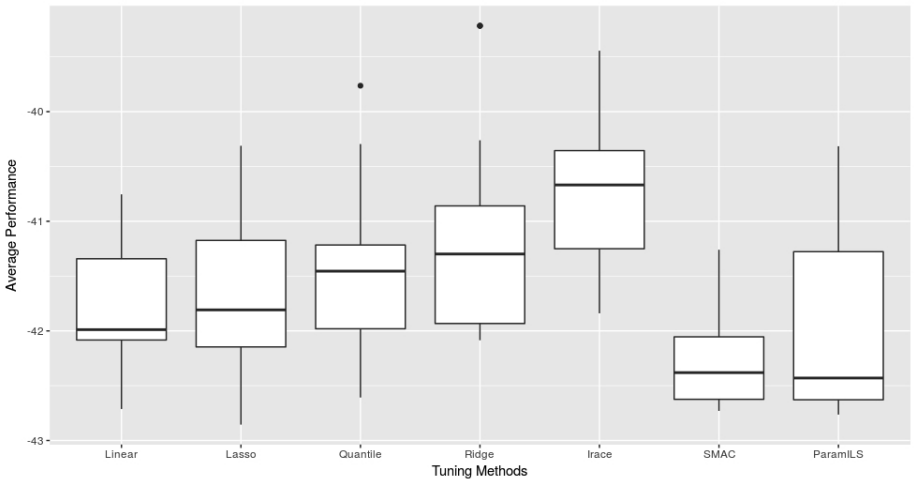

Figure 3: Overall mean performance - Homogeneous scenario. MetaTuner versions are indicated by the regression modelling.

that the normality assumption could not be assumed, so a Kruskal-Wallis test [23] was performed to detect differences in this scenario, suggesting statistically significant differences at the 95% confidence level ($p < 3.65 \times 10^{-14}$). Pairwise Wilcoxon-Mann-Whitney tests were then performed to pinpoint the differences, indicating SMAC and ParamILS tied in first place; Linear and Lasso similar to each other in the second place; Quantile worse than Linear and similar to Lasso and Ridge; and Ridge better than Irace (significantly the worst).

Figure 4 presents the results observed for the heterogeneous scenario. This figure suggests that several methods present somewhat similar performances, with ParamILS presenting a somewhat less stable behaviour. The Kruskall-Wallis test indicated that at least some of the differences observed were statistically significant ($p = 1.53 \times 10^{-10}$), and subsequent Wilcoxon-Mann-Whithney tests detected SMAC and Irace in the first place, followed by ParamILS and Ridge (similar to each other); Lasso (similar to ParamILS and worse than Ridge). The versions Linear and Quantile of Metatuner presented the worst performance, with large outliers.

In terms of runtime, Irace was marginally faster than the Metatuner versions

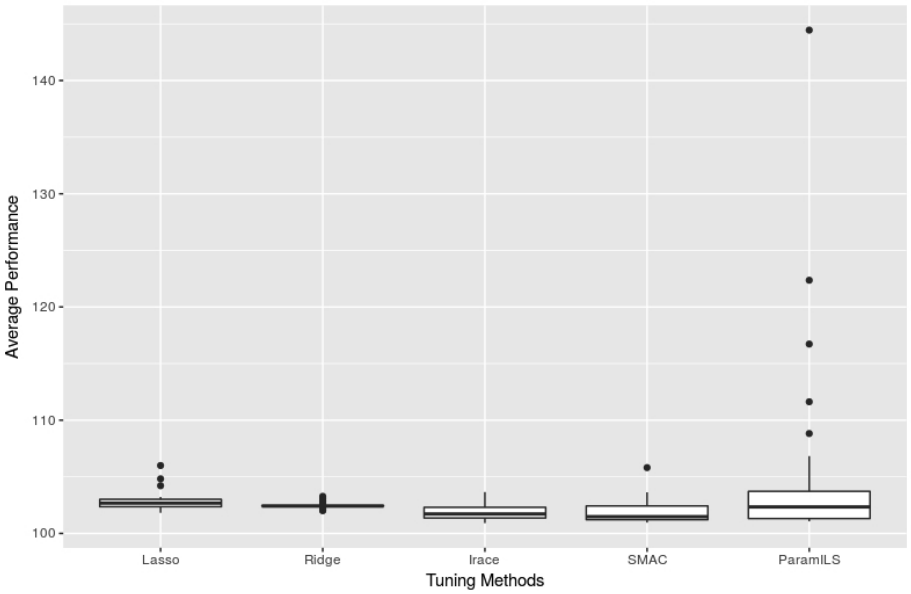

Figure 4: Overall mean performance - Heterogeneous scenario. Linear and Quantile versions of MetaTuner were omitted due to the presence of extreme outliers, which suggest that these versions can sometimes fail strongly, and may therefore not be interesting for general use.

(what can be observed from the table 1). This reinforces a point mentioned in the discussion of Experiment 1, namely that as the computational cost of evaluating the algorithm being tuned increases, the additional burden of building regression models tends represent a smaller portion of the total computational effort, and consequently the tuning times start becoming less dissimilar.

ParamILS and SMAC had considerably worse median runtimes for this experiment, but as with all time considerations this is much more an effect of implementation details than of specific computational efficiency: unlike MetaTuner and Irace (which are both native to R language), the implementations of these two methods had to, at every evaluation, call an external R script to load and run the optimiser (DE algorithm), which added considerable computational overhead.

Besides analysing the average performance and runtime of parameter tuning methods, another interesting aspect to investigate is the general distribution of

Table 1: Median runtimes for the DE tuning experiment.

Method	**Median runtime (seconds)**	
	Homogeneous	**Heterogeneous**
Irace	728	255
Ridge	767	688
Lasso	767	340
Linear	729	264
Quantile	725	265
ParamILS	25027	15989
SMAC	24963	17928

parameter values obtained by the methods. Figures 5 and 6 illustrate the best parameter values found by the tuning methods (the parameter values are in their original scales). These results suggest the use of reasonably low values of F for both scenarios, but also indicate a large spread of values returned by CR and K.

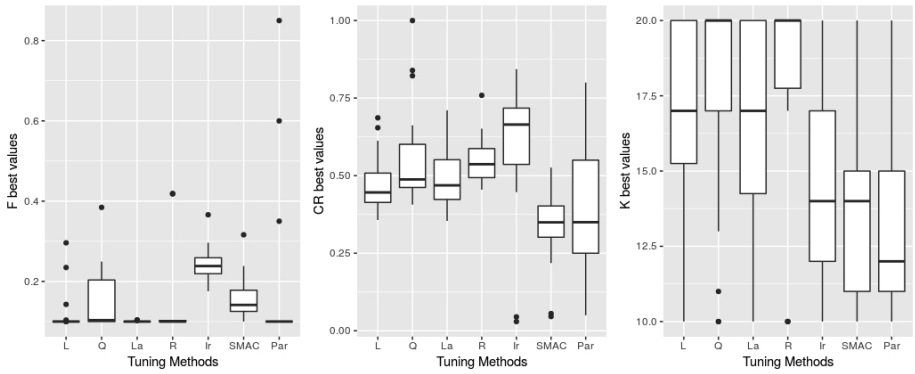

Figure 5: Distribution of the best parameter values - Homogeneous scenario. The versions of MetaTuner are labelled as: L - Linear; Q - Quantile; La - Lasso; R - Ridge. Irace is labelled as "Ir", and ParamILS as "Par"

Another interesting aspect that Metatuner enables researchers to investigate are the most important contributors to a given algorithm's performance on a problem class, which is illustrated in the table 2. The columns of this table

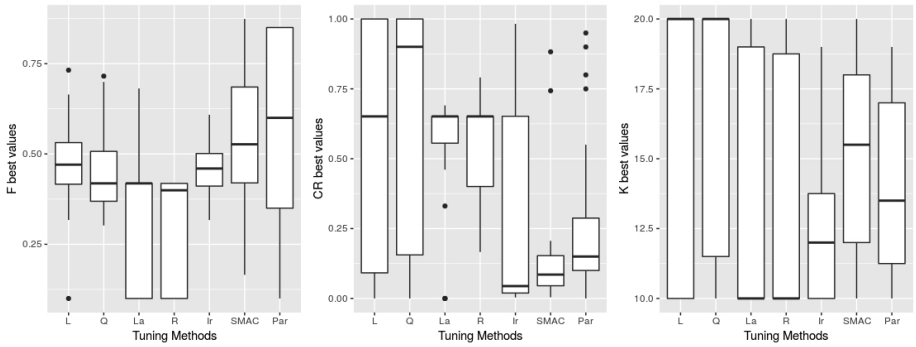

Figure 6: Distribution of the best parameter values. The parameter tuning methods are labelled as in the prior figure.

indicate which model parameters were interpreted by MetaTuner as "the most relevant" most often (**Freq#1**), second most-often (**Freq#1**) etc.. The two numbers in brackets represent how many times the parameter was the most relevant, followed by how many times it was among the 3 most relevant ones.

Table 2 suggests that (considering the different combinations of exponents), the MetaTuner versions detected mainly the main effects of CR (29 times as the most relevant and 69 times among the 3 most relevant), F (25 times as the most relevant and 37 times among the 3 most relevant), and the interaction of $F \times CR$ (38 times as the most relevant and 94 times among the 3 most relevant) as possibly the main contributors to the performance of DE/rand/1/bin on the homogeneous scenario. For heterogeneous scenario, the results suggest the main effect of CR (54 times as the most relevant and 113 times among the 3 most relevant), and the interaction of $F \times CR$ (51 times as the most relevant and 85 times among the 3 most relevant) as the main contributors to the performance of DE/rand/1/bin.

In fact, the influence of F, CR and of the interaction $F \times CR$ showed by MetaTuner echoes in the literature, in several works, when using several versions of DE for solving challenging problems. In [55] is presented visual "maps" showing the quality of DE solutions as a function of choice of F and CR values, and this choice can vary with the DE version and the problem class. A parameter

combination framework for DE is proposed in [56] which employ a strategy of combining different regions of the parameters space of F and CR in order to improve results of DE. In [57] is presented the best 63 combinations of values of F and CR determined by a self-adaptive DE for solving three well known benchmark problem sets. Another self-adaptive variation of DE is presented in [58], and in this work was showed experimentally the influence of different regions of the parameters space F and CR for the quality of DE solutions, for several challenging problems.

Another interesting aspect is that the population size multiplier K does not seem to appear prominently as the most relevant factor, which suggests that as long as the computational budget is maintained the population size can be regarded as secondary in comparison to a good selection of F and CR. According to [59], several strategies of choosing the population size, whether or not related to the size of problem have been proposed, but it is not clear the impact of each of them in general.

Table 2: Most relevant terms of DE/rand/1/bin.

Scenario	**Version**	**Freq #1**	**Freq #2**	**Freq #3**
Homogeneous	Linear	$F \times CR$ [14;27]	CR^2 [13;22]	$F \times CR^2$ [1;19]
	Quantile	$F \times CR$ [13;22]	CR^2 [8;14]	F^2 [7;12]
	Lasso	F [18;25]	CR^3 [6;23]	CR [2;10]
	Ridge	$F \times CR \times K$ [14;20]	$F \times CR$ [8;18]	$F \times CR^2$ [2;8]
Heterogeneous	Linear	CR^2 [18;25]	$F \times CR$ [6,15]	$F \times CR^2$ [2;14]
	Quantile	CR^2 [21;25]	F^3 [7;16]	$F \times CR^2$ [6;13]
	Lasso	$F \times CR^2$ [12;16]	CR^3 [11;24]	CR [3;20]
	Ridge	$F \times CR^2$ [25;27]	$F \times CR \times K$ [3;21]	CR [1;19]

In general, the results of this experiment shows MetaTuner as a competitive approach in relation to other parameter tuning methods both on Homogeneous or Heterogeneous scenarios. Furthermore, MetaTuner's explicit modelling of performance as a function of the tunable parameters enables the identification of the most relevant contributors to the success of a given algorithm, providing researchers with interesting analyses that can be used to guide algorithm devel-

opment and adaptation. This latter abillity is a clear advantage of MetaTuner in relation to the other parameter tuning methods used here.

5.3. Tuning SAPS for the SAT problem

This experiment is based on guidelines by Montero *et al.* [60], and its main objective is a comparison of MetaTuner with well known methods based on the literature in cases where few tuning instances are available.

The same four instantiations of MetaTuner used in the prior experiments were used and compared with Irace, SMAC and ParamILS, as well as against a random sampling approach used to provide a performance baseline. The methods were used for tuning the parameters of the Scaling and Probabilistic Smoothing (SAPS) algorithm [61, 62] for solving the SAT problem. Four parameters were tuned: $\alpha \in [1.01, 1.4]$, $wp \in [0, 0.06]$, $\rho \in [0, 1]$, and $ps \in [0, 0.2]$. The ranges were discretised to seven equally-spaced values for ParamILS.[13] Only 10 instances were available[14], and all of them were used for both training and validation. Although this may result in some overfitting of the resulting configurations to the instances, this is an unfortunate consequence of the very limited number of available instances for tuning.

The performance measurement used for each SAPS configuration on each instance was the time-to-convergence, with a timeout of 15 seconds. Cases in which the algorithm failed to converge within that time received a performance value calculated as $(15.00001 + min(max(0, sol/100000), 0.001))$, where sol is the lowest number of false clauses found. The computational budget used for the tuning process was 1250 evaluations, for all tuning methods (this budget was chosen after preliminary tests using ParamILS and SMAC), and the other initial parameters of all parameter tuning methods were set as in the prior experiment. The random sampling method consisted of randomly generating 125 configurations and running each one once on each instance, returning the

[13] Following recommendation from `http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/index.html`, from where the instances were obtained.

[14] All instances are satisfiable examples of the SAT problem.

configuration with the best median performance. The seeds used for running the SAPS algorithm on instances were the same, for all tuning methods. This experiment was performed in the same machine that was used for the first experiment.

Ten independent replicates were run for each tuning method. The best configurations returned by each method were then ran 30 times on each instance (using seeds for the random number generator different from those used during the tuning process), and the overall performance of each configuration was calculated as the grand mean of the observed performances on the instances.

Figure 7 presents the distribution of the mean overall performance of the tuning methods, considering the 10 best configurations returned by each one. It is clear from the figure that all tuning methods actually provide better configurations than would be obtained by simply performing a random search over the space of parameters. However, two MetaTuner approaches (Linear and Quantile, like in the Heterogeneous scenario of the prior experiment) presented outlying replicates in which their performance were worse than that of random search. Even if rare, this is enough to generate scepticism about these two alternatives.

A Kruskal-Wallis test was able to detect the differences as statistically significant at the 95% confidence level ($p = 7.7 \times 10^{-6}$), and the subsequent pairwise Wilcoxon-Mann-Whitney tests indicated ParamILS and Irace as the best, followed by: Lasso, Ridge and SMAC; Quantile and Linear; and Random approach as the worst. Table 3 displays the median runtime of all approaches. As in the previous experiment, as the computational cost of evaluating the algorithm becomes larger, the differences in added computational burden of the methods become less important.

Figure 8 illustrates the distribution of the parameter values obtained at the end of the 10 replicates for all methods. All of them were able to generate solutions with a large spread of values, suggesting that high-quality configurations for this particular problem possibly emerge from the interaction of parameter values, instead of being dominated by a few main effects of the parameters. Table 4 shows the most relevant parameters returned by the versions of MetaTuner

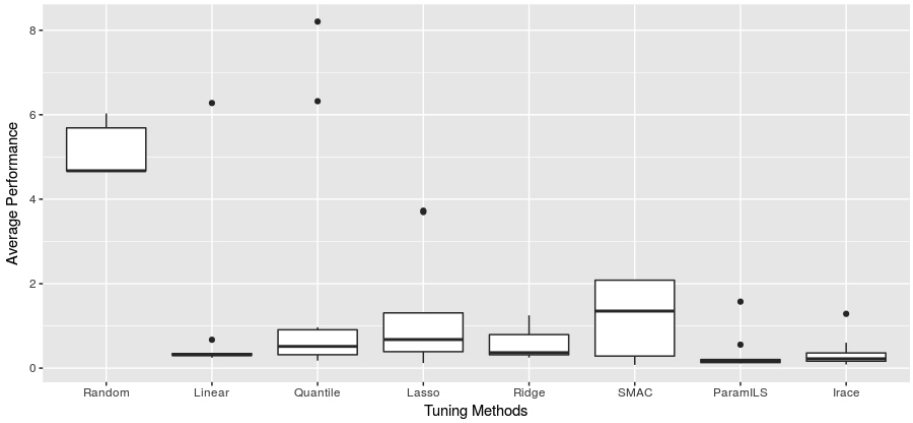

Figure 7: Overall mean performance of each tuning method for the SAPS algorithm.

Table 3: Median runtimes for the SAPS tuning experiment.

Method	**Median run time (seconds)**
Ridge	6005
Lasso	4965
Linear	3772
Quantile	3850
Irace	3376
ParamILS	4895
SMAC	3122

for this scenario. Parameters ps and ρ seem to appear as the most relevant more often, but there is generally a more heterogeneous distribution of parameters identified as the most relevant ones, which can indicate that all parameters share a similar importance in terms of determining the performance of the algorithm.

From the literature, in [61] is presented an approach called "RSAPS" (Reactive Saps), which is a self-adaptive SAPS. The results of this work suggest that the search intensification of SAPS can be dynamically improved by adapting the values of ρ and ps while fixing values for wp and α, reaching better results than the original SAPS. Although a more intensive research is necessary about this issue, it suggest that ρ and ps can have important influence on the quality

of solutions provided by SAPS, as indicated by MetaTuner. As in the prior experiment, MetaTuner is the only one parameter tuning method used here that is able to provide this sort of insight about the relation between algorithm parameters values and problem class.

Table 4: Most relevant parameters - SAPS

Version	**Freq #1**	**Freq #2**	**Freq #3**
Linear	ps^2[6;10]	ρ[2;3]	$wp \times \rho$[1;3]
Quantile	ps^2[6;7]	α^2[1;3]	wp^2[1;2]
Lasso	ρ^3[4;7]	ρ[4;5]	$\alpha \times \rho^2$[2;6]
Ridge	$wp^2 \times \rho$[5;9]	$wp \times \rho \times ps$[3;5]	ρ^3[1;4]

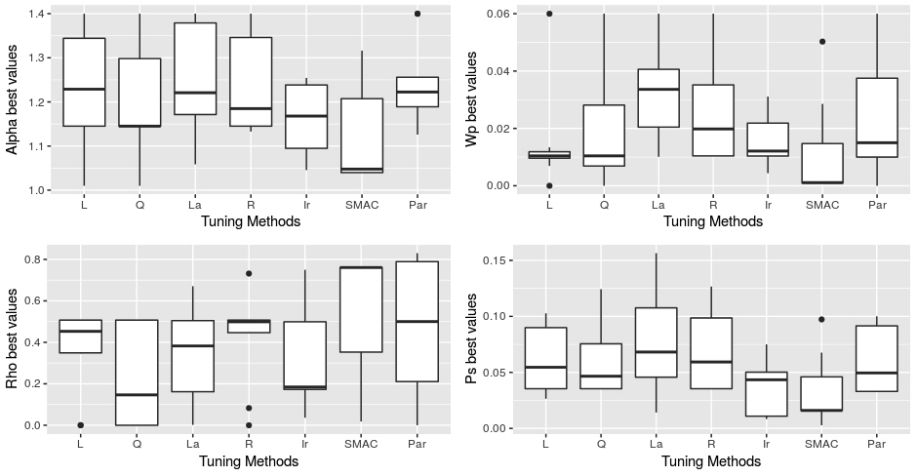

Figure 8: Distribution of parameter values obtained for the SAPS problem. The versions of MetaTuner are labelled as: L - Linear; Q - Quantile; La - Lasso; and R - Ridge. Irace is labelled as "Ir", and ParamILS as "Par"

6. Conclusions

In this work we present a new parameter tuning framework based on concepts from Sequential Model Based Optimisation (SMBO) methods. The proposed

framework is centred on the sequential optimisation of perturbed regression models of expected algorithm performance conditional on parameter values, and on the sequential evaluation of new problem instances on the most promising candidate configurations. It is proposed to be, at the same time, a tool for: **a)** reaching good parameter values and; **b)** provide to the user regression models which can identify the most relevant parameters, in terms of the main or interaction effects.

The proposed method was tested in three different experiments, and the main conclusions we can draw from these are that: (i) in general, Metatuner is able to yield competitive parameter values when compared with those obtained by other well known parameter methods, in a variety of problem scenarios; and (ii) the proposed method can strongly suggest the most relevant parameters when dealing with a tuning scenario for which the relation between the algorithm performance and the parameter values is dominated by few main and interacting effects of the parameters.

Future works included: further testing and development, needed to effectively establish its power and limitations, like the limitations of MetaTuner in terms of the number of parameters that can be tuned, and the adaptation of the principles described here to the explicit tuning of categorical or hierarchical parameters (without resorting e.g. to dummy-variable encoding). The effects of using alternative algorithms in the optimizating regression modellings phase, in terms of performance and regression models returned by MetaTuner; and lastly, an investigation of the convergence problems with the versions Quantile and Linear is mandatory.

6.1. A brief discussion on convergence

While a formal proof of convergence of the proposed approach to optimal parameter values is not provided in this paper, we can offer some qualitative discussion on the expected asymptotic properties of MetaTuner.

Given a problem class of interest, the ability of the proposed framework to reach the optimal parameter values is dependent on three aspects: (i) the ability

to generate candidate configurations arbitrarily close to the optimal parameter set $\theta^{\star}$; (ii) the quality of the regression, i.e., the predictive ability of the model in terms of estimating the performance of new candidate configurations for the problem class of interest; (iii) the quality of the optimiser used, i.e., given the response surface provided by the regression model, its ability to converge to the estimated optimum of that surface.

First, the ability of the proposed method to generate candidate configurations arbitrarily close to the optimum can be guaranteed (albeit only asymptotically) by arbitrarily increasing the initial sampling - e.g., using Latin Hypercube Sampling [21, 34] or even a simple grid design - within the space of valid configurations Θ. Moreover, even a sparse initial sampling (which is the common case) can result in configurations arbitrarily close to the optimum, based on the combination of iterative model building and optimisation approach used.

The quality of the regression models can be split in two parts: structure and fit. In terms of structure, it should be obvious that a poor choice of model structure (e.g., fitting a plane to observations that follow a highly nonlinear relationship) may prevent the method from approaching the optimal configuration. However, we expect that even low-order models (e.g., capturing quadratic or cubic terms) can generate iteratively better approximations, at least in the neighbourhood of high-quality candidate configurations, where lower order truncations can adequately approximate the underlying performance surface. Moreover, it is possible (albeit at an elevated computational cost) to increase the order of the regression models used, or to employ machine learning methods capable of adequately approximating or interpolating high-order surfaces. In terms of fit the convergence to an arbitrarily good approximation of the optimal configuration can be guaranteed (again, only asymptotically, and at a possibly prohibitive computational cost), e.g., by using interpolation models instead of regression [63] and increasing the number of configurations and instances visited (either initially or iteratively), so that approximation errors would tend to zero on visited configurations. We expect, however, that the proposed method would be able to converge to the vicinity of optimal configurations, or at least of

high-quality local optima, given the design principles employed in the development of MetaTuner (e.g., model perturbations based on quantifiable modelling uncertainties, etc.) and assuming that the regression models used do not have their underlying assumptions severely violated.

Finally, the convergence of MetaTuner is conditional on the adequacy of the optimiser used to generate new candidate configurations based on the perturbed models. If non-revisiting global optimisers are used, e.g., DIRECT [64], then convergence to the optima of each perturbed model can be asymptotically guaranteed. This, coupled with the considerations presented earlier on the ability of the models to converge iteratively to good approximations around optimal configurations suggests that the method should be able to return those solutions. In general, however, computationally cheaper heuristics such as Simulated Annealing are recommended for the model optimisation, so as not to result in prohibitively long tuning times, which sacrifices the convergence guarantees.

In summary, even though we cannot at this time provide adequate proofs or bounds on the convergence of MetaTuner to the optimal configuration for a given problem class, we have reason to assume that its structure allows, at least in principle, the discovery of these optimal points. Further studies, both theoretical and practical, are necessary to better explore the abilities and limitations of the proposed method.

References

[1] M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, Springer, 2010.

[2] A. E. Eiben, J. E. Smith, Introduction to Evolutionary Computing, Springer, 2003.

[3] M. Birattari, Tuning Metaheuristics - A Machine Learning Perspective, 1st Edition, Springer-Verlag Berlin Heidelberg, 2005.

[4] A. E. Eiben, S. K. Smit, Parameter tuning for configuring and analyzing evolutionary algorithms, Swarm and Evolutionary Computation 1 (2011) 19–31.

[5] H. H. Hoos, Automated algorithm configuration and parameter tuning, in: Autonomous Search, Springer, 2012, pp. 37–71.

[6] T. Bartz-Beielstein, Sequential parameter optimization, IEEE Congress on Evolutionary Computation (2009) 773–780.

[7] D. R. Jones, M. Schonlau, W. J. Welch, Efficient global optimization of expensive black-box functions, Journal of global Optimization 13 (1) (1998) 455–492.

[8] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43 – 58.

[9] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proceedings of the 5th International Conference on Learning and Intelligent Optimization, LION'05, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 507–523.

[10] F. Hutter, H. H. Hoos, K. Leyton-Brown, T. Stutzle, Paramils: an automatic algorithm configuration framework, Journal of Artificial Intelligence Research 36 (2009) 267–306.

[11] M. Birattari, On the estimation of the expected performance of a metaheuristic on a class of instances. how many instances, how many runs?, Tech. Rep. TR/IRIDIA/2004-001., IRIDIA, Université Libre de Bruxelles, Belgium (2004).

[12] S. H. Hurlbert, Pseudoreplication and the design of ecological field experiments, Ecological Monographs 54 (2) (1984) 187–211.

[13] S. E. Lazic, The problem of pseudoreplication in neuroscientific studies: is it affecting your analysis?, Lazic BMC Neuroscience 5 (11) (2010) 1–17.

[14] O. Maron, A. W. Moore, Hoeffding races: Accelerating model selection search for classification and function approximation, in: Advances in neural information processing systems, 1994, pp. 59–66.

[15] A. Moore, M. S. Lee, Efficient algorithms for minimizing cross validation error, in: Proceedings of the 11th International Conference on Machine Learning, 1994, pp. 190–198.

[16] M. Birattari, T. Stutzle, L. Paquete, K. Varrentrapp, A racing algorithm for configuring metaheuristics, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002, Morgan Kaufmann Publishers, San Francisco, CA, 2002, pp. 11–18.

[17] D. J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, 5th Edition, Chapman & Hall/CRC, 2011.

[18] P. Balaprakash, M. Birattari, T. Stützle, Improvement strategies for the f-race algorithm: Sampling design and iterative refinement, in: International workshop on hybrid metaheuristics, Springer, 2007, pp. 108–122.

[19] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated f-race: An overview, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, Springer, 2010, pp. 311–336.

[20] T. Bartz-Beielstein, C. W. G. Lasarczyk, M. Preuss, Sequential parameter optimization, in: Proceedings Congress on Evolutionary Computation 2005 (CEC'05), Edinburgh, Scotland, 2005, pp. 773–780.
URL `http://www.spotseven.de/wp-content/papercite-data/pdf/blp05.pdf`

[21] M. D. McKay, R. J. Beckman, W. J. Conover, A comparison of three methods for selecting value of input variables in the analysis of output from a computer code, Technometrics 21 (2) (1979) 239–245.

[22] G. D. Wyss, K. H. Jorgensen, A User's Guide to LHS: Sandia's Latin Hypercube Sampling Software, Risk Assessment and Systems Modeling Department - Sandia National Laboratories (February 1998).

[23] D. C. Montgomery, Design and Analysis of Experiments, 5th Edition, John Wiley & Sons, New York, NY, 2012.

[24] F. Hutter, H. H. Hoos, K. Leyton-Brown, An evaluation of sequential model-based optimization for expensive blackbox functions, in: Proc. Genetic and Evolutionary Computation Conference, 2013, pp. 1209–1216.

[25] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg, Hyper-heuristics: An emerging direction in modern search technology., in: B. M. Springer (Ed.), Handbook of metaheuristics, 2003, pp. 457–474.

[26] F. Caraffini, F. Neri, M. Epitropakis, Hyperspam: A study on hyper-heuristic coordination strategies in the continuous domain., Information Sciences (477) (2019) 186–202.

[27] S. S. Choong, L.-P. Wong, C. P. Lim, Automatic design of hyper-heuristic based on reinforcement learning., Information Sciences (436) (2018) 89–107.

[28] P. B. C. Miranda, R. B. C. Prudêncio, G. L. Pappa, H3ad: A hybrid hyper-heuristic for algorithm design., Information Sciences 414 (2017) 340–354.

[29] V. Nannen, A. E. Eiben, A method for parameter calibration and relevance estimation in evolutionary algorithms, GECCO'06 - Genetic and Evolutionary Computation Conference (2006) 183–190.

[30] V. Nannen, A. E. Eiben, Relevance estimation and value calibration of evolutionary algorithm parameters, International Joint Conference on Artificial Intelligence (2007) 975–980.

[31] N. Večěk, M. Mernik, B. Filipič, M. Črepinšek, Parameter tuning with chess rating system (crs-tuning) for meta-heuristic algorithms, Information Sciences 372 (2016) 446–469.

[32] J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn, Design and analysis of computer experiments, Statistical Science 4 (4) (1989) 409–423.

[33] M. J. Crawley, The R Book, Wiley, second edition, 2012.

[34] K. Q. Ye, Orthogonal column latin hypercubes and their application in computer experiments, Journal of the American Statistical Association 93 (444) (1998) 1430–1439.

[35] L. Kuipers, H. Niederreiter, Uniform distribution of sequences, John Wiley & Sons, New York, 2005.

[36] J.-H. Ning, K.-T. Fang, Y.-D. Zhou, Uniform design for experiments with mixtures, Communications in Statistics - Theory and Methods 40 (10) (2011) 1734–1742.

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

[38] K. Deb, S. Agrawal, A niched-penalty approach for constraint handling in genetic algorithms, in: Artificial Neural Nets and Genetic Algorithms, Springer-Verlag Science + Business Media, 1999, pp. 235–243.

[39] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, Data Mining: Practical machine learning tools and techniques, Morgan Kaufmann, 2016.

[40] R. Koenker, K. F. Hallock, Quantile regression, Journal of Economic Perspectives 15 (4) (2001) 143–156.

[41] T. Strutz, Data fitting and uncertainty: A practical introduction to weighted least squares and beyond, Vieweg and Teubner, 2010.

[42] R. Koenker, Quantile Regression, Cambridge University Press, 2005.

[43] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society. Series B (Methodological) 58 (1) (1996) 267–288.

[44] A. B. Owen, A robust hybrid of lasso and ridge regression, Tech. rep., Stanford University (October 2006).

[45] J. A. Nelder, R. Mead, A simplex method for function minimization, The Computer Journal 7 (4) (1965) 308–313.

[46] J. C. Nash, R. Varadhan, G. Brothedieck, General-Purpose Optimization, r Documentation (2017).
URL `http://stat.ethz.ch/R-manual/R-devel/library/stats/html/optim.html`

[47] A. J. Dobson, A. G. Barnett, An Introduction to Generalized Linear Models, 4th Edition, 2018.

[48] A. Díaz-Manríquez, G. Toscano, J. H. Barron-Zambrano, E. Tello-Leal, A review of surrogate assisted multiobjective evolutionary algorithms, Computational Intelligence and Neuroscience (4) (2016) 1–14.

[49] L. Shi, K. Rasheed, A survey of fitness approximation methods applied in evolutionary algorithms., in: B. H. Springer (Ed.), Computational Intelligence in Expensive Optimization Problems, 2010, pp. 3–28.

[50] F. Goulart, S. T. Borges, F. C. Takahashi, F. Campelo, Robust multiobjective optimization using regression models and linear subproblems., in: Proc. Genetic and Evolutionary Computation Conference, 2017, pp. 569–576.

[51] R. Jiao, S. Zeng, C. Li, Y. Jiang, Y. Jin, A complete expected improvement criterion for gaussian process assisted highly constrained expensive optimization, Information Sciences (471) (2019) 80–96.

[52] S. M. Mousavi, J. Sadeghi, S. T. A. Niaki, N. Alikar, A. Bahreininejad, H. S. C. Metselaar, Two parameter-tuned meta-heuristics for a discounted inventory control problem in a fuzzy environment, Information Sciences 276 (2014) 42–62.

[53] F. Campelo, M. Botelho, Experimental investigation of recombination operators for differential evolution, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO '16, ACM, 2016, pp. 221–228.

[54] N. Hansen, A. Auger, S. Finck, R. Ros, Real-parameter black-box optimization benchmarking bbob-2010: Experimental setup, Tech. Rep. Research report RR-7215, INRIA (2010).

[55] M. Cervenka, H. Boudná, Visual guide of f and cr parameters influence on differential evolution solution quality, in: 24th International Conference on Engineering Mechanics, 2018, pp. 141–144. doi:10.21495/91-8-141.

[56] Z. D. J. Zhang, Parameter combination framework for the differential evolution algorithm, Algorithms 12 (4) (2019) 1–22. doi:doi:10.3390/a12040071.

[57] T. R. R. A. Sarker, S. M. Elsayed, Differential evolution with dynamic parameters selection for optimization problems, IEEE Transactions on Evolutionary Computation 18 (5) (2014) 689–707.

[58] B. B. M. M. J. Brest, S. Greiner, V. Zume, Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems, IEEE Transactions on Evolutionary Computation 10 (6) (2007) 646–657.

[59] A. P. Piotrowski, Review of differential evolution population size, Swarm and Evolutionary Computation 32 (2017) (2016) 1–24.

[60] E. Montero, M.-C. Riff, B. Neveu, A beginner's guide to tuning methods, Applied Soft Computing 17 (2014) 39–51.

[61] D. A. D. Tompkins, H. H. Hoos, Scaling and probabilistic smoothing: Dynamic local search for unweighted max-sat, in: Conference of the Canadian Society for Computational Studies of Intelligence, Springer, 2003, pp. 145–159.

[62] D. A. D. Tompkins, H. H. Hoos, Ubcsat: An implementation and experimentation environment for sls algorithms for sat and max-sat, in: Theory and Applications of Satisfiability Testing: 7th International Conference, SAT 2004., 2004, pp. 306–320.

[63] R. L. Hardy, Theory and applications of the multiquadric-biharmonic method: 20 years of discovery., Computers & Mathematics with Applications 19 (1990) 163–208.

[64] D. R. Jones, C. D. Perttunen, B. E. Stuckman, Lipschitzian optimization with the lipschitz constant, Journal of Optimization Theory and Applications 79 (1993) 157–181.

Appendix 1: Parameter relevance tables, Experiment 1

Tables 5 and 6 show a summary of the most relevant parameters for the scenarios in Experiment 1 (section 5.1), considering each version of MetaTuner and each dimension of Quadratic and Ackley Functions. Having both the algorithm performance and the parameter values scaled in the interval [0,1] when building the regression models, the most relevant parameters for each run were those with highest absolute values in the regression models.

The columns of these tables indicate which model parameters were interpreted by MetaTuner as "the most relevant" most often (**Freq#1**), second most-often (**Freq#1**) etc.. The two numbers in brackets represent how many times the parameter was the most relevant, followed by how many times it was among the 3 most relevant ones.

Table 5: More relevant parameters - Quadratic Function

Dimension	Version	Freq #1	Freq #2	Freq #3
2	Linear	$\theta_1[21;22]$	$\theta_1^2[5;24]$	$\theta_1\theta_2^2[3;7]$
	Quantile	$\theta_1[30;30]$	-	-
	Lasso	$\theta_1[29;30]$	$\theta_1^3[1;29]$	-
	Ridge	$\theta_1[30;30]$	-	-
3	Linear	$\theta_1[26;29]$	$\theta_1\theta_2[2;10]$	$\theta_1^2[1;25]$
	Quantile	$\theta_1[17;23]$	$\theta_1^2[4;20]$	$\theta_1\theta_2[4;10]$
	Lasso	$\theta_1[30;30]$	-	-
	Ridge	$\theta_1[30;30]$	-	-
4	Linear	$\theta_1[27;27]$	$\theta_1^2[2;26]$	$\theta_1\theta_3[1;7]$
	Quantile	$\theta_1[29;30]$	$\theta_1^2[1;30]$	-
	Lasso	$\theta_1[30;30]$	-	-
	Ridge	$\theta_1[30;30]$	-	-
5	Linear	$\theta_1[30;30]$	-	-
	Quantile	$\theta_1[30;30]$	-	-
	Lasso	$\theta_1[29;30]$	$\theta_3^2\theta_4[1;1]$	-
	Ridge	$\theta_1[30;30]$	-	-
6	Linear	$\theta_1[30;30]$	-	-
	Quantile	$\theta_1[30;30]$	-	-
	Lasso	$\theta_1[30;30]$	-	-
	Ridge	$\theta_1[30;30]$	-	-
7	Linear	$\theta_1[17;22]$	$\theta_1\theta_5[2;4]$	$\theta_1\theta_7[2;4]$
	Quantile	$\theta_1[9;14]$	$\theta_1^2[3;15]$	$\theta_1\theta_5[3;8]$
	Lasso	$\theta_1[30;30]$	-	-
	Ridge	$\theta_1[30;30]$	-	-
8	Linear	$\theta_1[29;29]$	$\theta_7[1;5]$	-
	Quantile	$\theta_1[30;30]$	-	-
	Lasso	$\theta_1[30;30]$	-	-
	Ridge	$\theta_1[30;30]$	-	-

Table 6: More relevant parameters - Ackley Function

Dimension	**Version**	**Freq #1**	**Freq #2**	**Freq #3**
2	Linear	$\theta_1\theta_2$[11;16]	θ_2^2[7;16]	θ_1^2[6;12]
	Quantile	$\theta_1\theta_2$[12;17]	θ_2^2[7;18]	θ_1^2[7;12]
	Lasso	θ_1^3[12;21]	θ_2^3[8;20]	$\theta_1\theta_2$[4;9]
	Ridge	θ_2[30;30]	-	-
3	Linear	$\theta_1\theta_3$[6;11]	$\theta_2\theta_3$[5;9]	θ_1^2[5;6]
	Quantile	$\theta_1\theta_2$[10;13]	θ_3^2[6;11]	θ_2^2[5;10]
	Lasso	θ_2^3[3;11]	θ_2[3;8]	θ_3[2;7]
	Ridge	$\theta_2\theta_3$[23;25]	$\theta_2^2\theta_3$[3;3]	θ_3[2;24]
4	Linear	θ_1^2[6;12]	θ_3^2[4;9]	θ_4^2[4;9]
	Quantile	θ_2^2[8;10]	θ_1^2[6;11]	θ_3^2[4;10]
	Lasso	θ_3^3[3;6]	$\theta_1\theta_2\theta_3$[3;4]	$\theta_1\theta_3^2$[3;3]
	Ridge	$\theta_2\theta_3$[11;18]	$\theta_3\theta_4$[6;13]	θ_4[5;19]
5	Linear	θ_3^2[6;11]	θ_4^2[5;8]	θ_1^2[2;8]
	Quantile	θ_4^2[6;11]	θ_3^2[4;8]	θ_2^2[3;14]
	Lasso	θ_3^3[3;3]	θ_4^3[2;2]	$\theta_1\theta_3\theta_5$[1;2]
	Ridge	θ_5[14;24]	θ_3[3;16]	$\theta_3\theta_4\theta_5$[3;3]
6	Linear	θ_6^2[5;9]	$\theta_1\theta_2$[4;8]	θ_2^2[4;7]
	Quantile	θ_5^2[6;8]	θ_4^2[5;9]	θ_6^2[4;6]
	Lasso	θ_5[2;2]	$\theta_2\theta_4\theta_6$[1;3]	θ_2[1;2]
	Ridge	$\theta_2\theta_3\theta_4$[8;11]	θ_2^3[4;8]	θ_3[3;6]
7	Linear	θ_6^2[2;6]	θ_4^2[2;6]	θ_7^2[2;4]
	Quantile	θ_6^2[5;7]	θ_5^2[5;7]	θ_2^2[4;5]
	Lasso	$\theta_1\theta_2\theta_7$[1;2]	$\theta_4\theta_6^2$[1;1]	$\theta_4\theta_5\theta_7$[1;1]
	Ridge	θ_2^2[3,5]	θ_5^2[3;5]	θ_3[2;5]
8	Linear	θ_4[6;10]	θ_2[5;8]	θ_5[4;10]
	Quantile	θ_5[5;9]	θ_7[4;7]	θ_4[4;6]
	Lasso	$\theta_3\theta_6$[2;3]	$\theta_5\theta_8$[2;2]	$\theta_3\theta_5$[1;4]
	Ridge	θ_5^2[25;30]	θ_2^2[3;19]	θ_3^2[1;15]