






# Comparing Surrogate Models for Tuning Optimization Algorithms

Gustavo Delazeri<sup>1</sup> , Marcus Ritt<sup>1</sup> , and Marcelo de Souza<sup>2</sup> 

<sup>1</sup> Instituto de Informática, Universidade Federal do Rio Grande so Sul,  
Porto Alegre, Brazil

`{gustavo.delazeri,marcus.ritt}@inf.ufrgs.br`

<sup>2</sup> Departamento de Engenharia de Software, Universidade do Estado de Santa  
Catarina, Ibirama, Brazil

`marcelo.desouza@udesc.br`

**Abstract.** Tuning an algorithm requires to evaluate it under different configurations on several problem instances. Such evaluations are costly. A way to reduce the configuration time when developing tuners is to use surrogate models, which map configuration-instance pairs to the approximate algorithm performance and thus allow to replace algorithm runs by fast calls to the model. Most applications of surrogate models found in the literature focus on predicting algorithm running time; much less effort has been devoted to predicting the quality of solutions of optimization algorithms. In this paper, we present a comparative study of surrogate models for predicting solution quality. We evaluate several surrogate models from the literature, including random forests, gradient boosting methods, and neural networks, and compare ways of handling different classes of parameters, data imputation strategies, and codification of instances. We demonstrate for two heuristic algorithms that the best models can accurately reproduce effects observed when tuning with the ground truth. Our code is available (<https://github.com/gutodelazeri/oracle>).

**Keywords:** Automatic algorithm configuration · Surrogate models · Optimization algorithms

## 1 Introduction

Selecting the best algorithm among a set of candidates for solving a problem is a common challenge, since often several algorithms of complementary strengths are available, or algorithms are parameterized. Following the seminal work of [36] algorithm selection can be described as a mapping from a problem space  $\mathcal{P}$  to an algorithm space  $\mathcal{A}$  that maximizes some performance measure  $m(p, a)$  for  $p \in \mathcal{P}, a \in \mathcal{A}$ . Here we are interested in selecting a single algorithm that has the best expected value with respect to some distribution over the problem space, a common approach in automatic algorithm configuration or tuning [24, 25, 32].

Often the algorithm space consists of a family of parameterized algorithms  $a_\theta$ , with parameter settings  $\theta$  from some parameter space  $\Theta = \times_{i \in [n]} \Theta_i$ , where  $\Theta_i$  is the domain of parameter  $i \in [n]$ . Thus the problem of automatic algorithm configuration is to find the best configuration

$$\theta^* = \operatorname{argmax}_{\theta \in \Theta} E_{\mathcal{P}}[m(p, a_\theta)] \quad (\text{AAC})$$

for some distribution over the problems  $\mathcal{P}$ . For stochastic algorithms the problem space  $\mathcal{P}$  can be understood as composed of instance-seed pairs together with an appropriate distribution. Parameters are often real or integer, but can also be ordinal, categorical, or conditional (i.e. only effective for specific settings of enabling parameters).

Solving problem (AAC) even approximately is hard, in particular because obtaining the measure  $m(p, a)$  requires running an algorithm on a problem instance, which can take considerable time, often minutes or hours. As a consequence, the (second-level) problem of selecting among optimizers for problem (AAC) is even harder. On a family of parameterized optimizers this is also known as the problem of *hyperparameter tuning*. It can be made more tractable by providing a surrogate function  $\hat{m}$  for measure  $m$  for some representative first-level configuration problems. Such surrogate functions are required to be:

1. fast to evaluate;
2. accurate in the sense that predicted values allow to correctly rank the performance of algorithms on instances;
3. handle all parameter classes, including numerical, categorical, and conditional parameters;
4. have a high fidelity in the sense that hyperparameter optimization on surrogates correctly predicts the behaviour under ground truth  $m$ .

Note that requirement 4 goes beyond simple accuracy, since we require the model to represent the ground truth on the level of observable effects, as opposed to just taking an accurate model as the ground truth for comparing hyperparameter optimizers. Clearly, as a function of the accuracy, this will be limited to effects of a minimal effect size. This requirement should allow to quickly assess the performance of new optimizers during design and test, and speed up benchmarking of optimizers.

In this paper we compare several representative models from the literature and evaluate their accuracy and ability to reproduce effects observable when comparing tuners. The paper is organized as follows. In the remainder of this section we discuss related work. We then introduce in Sect. 2 a selection of models for surrogate functions. Section 3 presents an experimental comparison of these surrogate models for tuning (hyper-)optimization algorithms. We conclude in Sect. 4.

## 1.1 Related Work

Approaches to surrogate functions make different assumptions on the domain and co-domain of the function to be modeled, on access to additional information such as derivatives, or bounds, and use different surrogate models. [17], for

example, assumes a continuous, compact parameter space, a continuous function that is expensive to evaluate, with no additional information, and a model which is a weighted sum of radial basis functions centered at the samples plus a polynomial term. This is a form of *black-box* optimization, and many other approaches to black-box optimization assume real-valued parameters (e.g. [7, 18, 30]), some of which can be extended to handle integer parameters [8], or categorical and boolean parameters [1, 4, 41]. Approaches also differ in assumptions on the cost of evaluation. Expensive functions can take hours to evaluate, and thus the number of evaluations is typically limited to a few hundred, while for less expensive functions thousands of evaluations are common [38]. Other additional information that can be exploited in so-called *gray-box* optimization are objective functions of a known form (e.g. a sum of squares) the possibility of obtaining faster, less accurate samples in *multi-fidelity* objective functions [2], or assumptions on continuity (e.g. the knowledge of the Lipschitz constant [33]). Typical surrogate models include linear regression, kernel-based techniques (e.g. the radial basis functions mentioned above), Gaussian processes, (gradient-boosted) regression trees, random forests, and neural networks [9, 31, 39].

Most of the above approaches are concerned with optimization, and often follow some sequential model-based strategy, that successively acquires a promising sample (e.g. of best objective function value or highest information gain), updates the model, and returns the best sample when the computational budget is spent. In contrast, in this paper we are concerned with static surrogate models that represent typical tuning landscape well, as outlined in the introduction (requirements 1–4).

A collection of 110 hyperparameter optimization benchmarks, with 2 to 26 continuous, integer, categorical, or ordinal parameters has been introduced by [15]. It focuses on ease-of-use, reproducibility and benchmarks with multiple fidelity levels. All except six benchmarks which use random forests as a surrogate model are provided in a form of lookup tables. Closer to our work, [5] introduce a benchmark consisting of four expensive real-world problems (e.g. wind farm layout) that take from 2 to 60 seconds per evaluation. Two of these problems are continuous, two have mostly categorical variables. Besides comparing tuners, they compare mean errors of seven surrogate models on the best 1K of random samples, as well as samples collected during tuning on two problems. Experiments show that random forests [34] and gradient boosted regression trees (via XGBoost [6]) are among the best models, and these models tend to be more accurate when trained on random samples.

[11] have compared eight models to construct surrogate benchmarks for hyperparameter tuning, namely linear and ridge regression, two forms of support vector machines, Gaussian processes, random forests, and gradient boosted trees. Models have been built from 2K to 20K samples collected during the execution of four tuners on nine datasets. The datasets have 3 to 36 categorical and continuous parameters, with a dimension in [4, 82] after one-hot encoding of categorical variables. Results show gradient boosted trees and random forests to have best accuracy, as measured by root mean square error (RMSE) and Spearman’s rank

correlation, followed by Gaussian processes, with random forests giving the most “similar” tuned values. Even with similar values, however, a consistent ranking of tuners on ground truth and surrogate models is not guaranteed. [14] extends the former work to algorithm configuration by handling conditional variables by imputation of default or midpoint values, imputation of censored data, and explicitly handling of randomized algorithms. Quantile random forests serve as a surrogate model. There are 11 scenarios with about 10–300 parameters and 30–300 instance features, with nine scenarios predicting runtime. Experiments show a reasonable reproduction of the rank of tuners over the course of the tuning process.

Different from most of the above approaches, here we are interested in models for parameter spaces (requirement 3) when instances features are not available, and on approximating objective function values as opposed to running time, which is the focus of most of the literature on models for algorithm tuning.

## 2 Surrogate Models for Optimization Algorithms

In this section we present the selected procedures for data pre-processing and surrogate models. Concerning data pre-processing, all selected models are able to handle real and integer parameters, so we have to define only how to handle categorical and conditional variables to attend requirement 3. For categorical variables, if the underlying surrogate model is not already able to handle them directly, we test three encoding strategies: one-hot, binary, and by-index encoding. In all three a variable with  $n$  categories is first mapped to an integer  $i \in [n]$ . Then, for one-hot encoding  $i$  is mapped to  $e_i$ , where  $e_i = (0, \dots, 1, \dots, 0)$  is the  $i$ th unit vector, i.e. to  $n$  new binary variables; for binary encoding  $i$  is mapped to its base-2 representation  $(i)_2$ , leading to  $\lceil \log_2 n \rceil$  new binary variables; for index encoding,  $i$  is used directly. The categorical variables include the instances. Since the instances may have a considerable influence on the predicted variable, and the number of instances is limited, we study two further strategies for this case: instance-wise models, where we build an individual model for each instance, and a combined model for all instances where we treat the instance as a normal categorical variable. For conditional variables, we impute values, and test three strategies: imputing a fixed, chosen from the parameter’s domain uniformly at random, imputing the default value, and imputing random values. We have excluded out-of-domain, quantile, or mean imputation since previous work found no significant differences to default imputation, which is similar to our strategy.

From previous work it is also clear that tree-based models are often competitive [12, 27]. Thus we include random forests and gradient boosted trees. We have chosen two widely used implementations, random forest run [3] and ranger [44]. [16] have studied models for tabular data over real, integer and categorical features, and have found two deep neural network architectures, namely ResNet and FT-Transformer, as well as gradient boosted trees to be among the best models. Therefore we include these two neural network architectures, as well as two implementations of gradient boosted trees, namely CatBoost [35] and

XGBoost [6]. We did not include Gaussian process models in our study, since previous work indicate that random forest perform better with large parameter spaces with both numerical and categorical parameters [11].

The ResNet consists of a number of residual blocks, each block consisting of two layers: a linear layer followed by a rectified linear unit (ReLU), and a second linear layer. The residual blocks are followed by a final ReLU and a linear layer that maps to the regression variable [19]. The FT-Transformer consists of a number of transformer blocks, each consisting of the typical multi-head attention with query-key-value layers followed by a linear layer [42]). Categorical variables are handled as proposed by [16], who propose a specific embedding for each possible value. The number of layers of both architectures, as well as several other parameters are tuned following [16], as explained below in the experimental section.

Finally we include an interpolated tabular model, that uses Shepard’s method of inverse distance weighting [37] with a definable cutoff after considering the  $k$  closest samples. Closest samples are found by using the ball tree structure from scikit-learn. For distance weighting we use the heterogeneous Euclidean-overlap metric of [43]: for numerical parameters, the distance is defined as difference in parameter values, normalized to  $[0, 1]$ ; for categorical parameters the distance is 0 when parameter values are equal, and 1 otherwise. Undefined categorical values always have distance 1 to all other values.

All models are trained to learn to predict objective function values from the ground truth data, i.e. our approach can be classified as pointwise learning-to-rank (as opposed to learn to rank pairs or larger subsets of inputs directly).

### 3 Experimental Results

The main goal of the experiments is to compare the performance of the selected models for modeling optimization landscapes. We use two measures to judge performance. First, we measure the accuracy of the models in ranking pairs of parameter settings. This is done by comparing pairs of samples from the test set using the predicted and ground truth values. The accuracy is then defined as the number of correctly predicted orders over all pairs. Accuracy values  $a$  are in  $[0, 1]$  and are related to Kendall’s tau [29] by  $\tau = 2a - 1$ .

We first report on the effects of instance representation and pre-processing in Sect. 3.2. In Sect. 3.3 we analyze accuracy as measured by the relative root mean squared error and Kendal’s  $\tau$ . Finally, we measure the performance by the fidelity with which the different models are able to reproduce effects that can be observed on the ground truth data. For this experiment we have selected three parameters of irace and compare it on different settings. These experiments are explained in Sect. 3.4.

#### 3.1 Methodology

We have done the experiments using the tuner irace [32]. The irace configurator applies iterated racing, where in each iteration candidate parameter settings are run on a number of instances, statistically significantly worse configurations

are discarded, and new configurations are sampled according to a distribution of good parameter values, which evolves with each candidate parameter setting. These distributions are represented by their marginal distributions on each parameter and converge to the best settings. If not otherwise specified we run irace with default parameters, and run all tests with three fixed budgets. We have chosen a maximum budget of 3K evaluations, the smallest possible budget for irace of 780 evaluations, and an intermediate budget of 1890 evaluations. Therefore the results do not depend on the concrete computing platform or the running time. When running times are mentioned they have been obtained on a PC with a 12-core AMD Ryzen 9 3900X processor with 32 of main memory, running Ubuntu Linux 20.04. All models run in a server that responds queries on parameter settings by irace, generated by a wrapper that replaces the algorithm to be tuned. Our code is available at <https://github.com/gutodelazeri/oracle>.

**Configuration Scenarios.** We evaluate all surrogate models on two configuration scenarios, namely ACOTSP and LKH. Both are heuristic algorithms for the symmetric traveling salesperson problem (TSP). We use 10 Random Uniform Euclidean TSP instances with 2000 cities, generated using the *portgen* instance generator from the 8th DIMACS Implementation Challenge [28]. Both ACOTSP and LKH scenarios are part of the ACLib benchmark library for algorithm configuration [26]. The ACOTSP scenario implements ant colony optimization algorithms for TSP [10]. We use ACOTSP version 1.03, available at [40]. It has 4 real, 4 integer, and 3 categorical parameters; 5 of these parameters are conditional. The LKH scenario concerns the Lin-Kernighan-Helsgaun algorithm for TSP [20–22]. We use LKH version 2.0.9 [23]. It has 12 integer and 9 categorical parameters, all of them unconditional. In both scenarios we use no additional instance features, since we are only interested in building surrogate models for the selected instances. We test, however, the case of a separate model for each instance, see below, which arguably corresponds to the best possible set of features for the selected instances.

**Models.** We summarize the selected models in Table 3.1. All models use their default parameters, with the exception of the neural networks, which are tuned using the process proposed by [16], and PyRFR, which uses the configuration found by [13].

Model	Instances	Categoricals	Imputation
Interpolation	split, merged	model-defined	model-defined
CatBoost	split, merged	model-defined	model-defined
XGBoost	split, merged	binary, one-hot, index	fixed, default, random
skranger	split, merged	binary, one-hot, index	fixed, default, random
PyRFR	split, merged	binary, one-hot, index	fixed, default, random
FT-Transformer	merged	model-defined	model-defined
ResNet	merged	model-defined	model-defined

**Datasets.** For each of the above scenarios we created a dataset by sampling 5K configurations uniformly at random for each instance, for a total of 50K samples with their corresponding (ground truth) objective function value. In both scenarios objective function values were obtained by fixing the seed of the pseudo-random generator to 1, i.e. we limit ourselves here to studying a deterministic version of both algorithms. The cost metric for both scenarios is the objective function value obtained after a running time limit of 10 s. Samples for ACOTSP have been obtained with a single trial (`tries=1`), those for LKH with a single run (`RUNS=1`).

In the experiments, these datasets are split into a training set, a test set, and a validation set. We use three sizes of the training set, namely 300, 3K, and 30K samples. Training sets are generated to be laminar, i.e. smaller training sets are contained in larger ones. The remaining samples are evenly split among the test and the validation set. The test set is used for tuning the architecture of the neural networks (FT-Transformers and ResNets). We study all models with these three sizes, except for the models based on neural networks which require a higher number of samples and therefore are trained only with a training set of size 30K. To evaluate accuracy in Sects. 3.2 and 3.3 we select 5K samples from the test and validation set.

For handling categorical variables we study three ways of encoding them, as mentioned above: binary encoding, one-hot encoding, and encoding by the index. These encodings are applied to all models which cannot handle categorical data directly, namely gradient-boosted trees as implemented by XGBoost, and random forests as implemented in skranger and PyRFR. skranger has support for declaring variables to be categorical. However, these variables have to be numerical. For this reason, we test in skranger all encodings, but declare the encoding variables as categorical. We set handling of categorical values in skranger to consider all partitions (`respect_categorical_features="partition"`), since the number of values in the dataset is small, and testing all partitions will give the best possible splits.

For handling different instances we test two strategies: training a surrogate model for each instance individually, or training a single surrogate model for all instances. These strategies are called “split” and “merged”, respectively, in the experiments below. For “merged” models the instance is added as an additional categorical variable to the dataset, and is subject to the above transformations of categorical variables, when applicable.

In summary, the XGBoost model and the random forests (skranger, PyRFR) require imputation of missing values and encoding of conditionals; CatBoost, and the neural networks can handle all parameters as given. Furthermore, the interpolation baseline also does not need any pre-processing.

### 3.2 Effects of Instance Representation and Pre-processing

In a first experiment we analyze the influence of the representation of the instances, the parameter imputation for conditional variables, and handling of

categorical variables. Table 1 shows for the three different sample sizes the average relative root mean squared error (RRMSE)  $R$  and the average Kendall’s  $\tau$  for the different strategies. Averages apply only to the models for which different strategies have been applied (see Table 3.1). In the comparisons below we report  $p$ -values of sign tests comparing errors and  $\tau$  values over all models.

**Table 1.** Relative root mean squared error  $R$  and Kendall’s  $\tau$  for different sample sizes  $N$ , and different representation, imputation, and encoding strategies.

$N$	Instance				Imputation						Encoding					
	Merged		Split		Default		Fixed		Random		One hot		Binary		Index	
	R	$\tau$	R	$\tau$	R	$\tau$	R	$\tau$	R	$\tau$	R	$\tau$	R	$\tau$	R	$\tau$
300	0.26	0.48	0.27	0.46	0.14	0.50	0.14	0.47	0.14	0.51	0.26	0.47	0.27	0.45	0.26	0.48
3000	0.15	0.63	0.16	0.61	0.08	0.67	0.09	0.67	0.09	0.63	0.15	0.64	0.16	0.61	0.15	0.62
30000	0.07	0.78	0.08	0.76	0.04	0.78	0.04	0.78	0.05	0.75	0.07	0.79	0.08	0.75	0.07	0.78

We can observe that errors decrease and Kendall’s  $\tau$  consistently increases with increasing sample sizes. For 30K samples all RRMSE are below 0.1, and the value of Kendall’s  $\tau$  above 0.75. Turning to representation of instances, we find that using a single model (strategy “merged”) is better than strategy “split” which models each instance individually (all  $p < 0.001$ ). Note that since we have 10 instances, each individual model is built with only 10% of the samples. We next look at the imputation strategy. In this case averages are only over the ACOTSP scenario, since scenario LKH does not have conditional variables. Here differences are only significant for larger sample sizes, and indicate that imputing fixed or default values tends to be better than imputing random values for 3K samples (all  $p < 0.004$ ) and for 30K samples (all  $p < 0.001$ ), but not significantly different from each other. Finally we turn to the encoding. Here we find that binary and index encoding are not significantly different (all  $p > 0.02$ ), but for larger sample size one-hot encoding performs better than the other strategies (all  $p < 0.001$  for 3K and 30K samples).

Based on these results, we focus in the remaining experiments on a single model, and fix imputation and encoding strategies, where they apply, to “fixed” and “one-hot”, and compare the resulting seven models.

### 3.3 Accuracy of the Surrogate Models

Table 2 shows RRMSE and Kendall’s  $\tau$  for the seven selected models, both scenarios and all three samples sizes, except for the neural networks, which have been trained only with 30K samples. Again we can see that RRMSE decreases and Kendall’s  $\tau$  in agreement increases with the sample size. We also can see that the error for scenario LKH is considerably higher than for ACOTSP. A sample size of 300 seems inadequate for a good prediction, and increasing the number of samples to 3K and then 30K quality improves considerably, in particular for scenario LKH, although it remains harder to predict, with RRMSE values of about



0.1. The baseline model using interpolation shows worse performance, although it is not far off from the other models. Overall both two gradient-boosted trees and the random forest pyRFR work well, and the Transformer-based neural network has the best performance.

**Table 2.** Relative root mean squared error  $R$  and Kendall’s  $\tau$  for different sample sizes  $N$ , and both scenarios for all seven selected models.

Scen.	Model	300		3 K		30 K	
		R	$\tau$	R	$\tau$	R	$\tau$
ACOTSP	Interpolation	0.14	0.57	0.10	0.67	0.07	0.78
ACOTSP	skranger	0.14	0.49	0.09	0.68	0.05	0.79
ACOTSP	PyRFR	0.13	0.50	0.08	0.71	0.04	0.81
ACOTSP	XGBoost	0.13	0.54	0.06	0.72	0.03	0.79
ACOTSP	CatBoost	0.12	0.66	0.07	0.74	0.03	0.82
ACOTSP	ResNet	–	–	–	–	0.02	0.80
ACOTSP	FT-Transformer	–	–	–	–	0.02	0.84
LKH	Interpolation	0.45	0.15	0.36	0.30	0.19	0.71
LKH	skranger	0.45	0.36	0.29	0.52	0.12	0.77
LKH	PyRFR	0.44	0.45	0.23	0.59	0.09	0.83
LKH	XGBoost	0.43	0.48	0.24	0.66	0.09	0.80
LKH	CatBoost	0.45	0.44	0.20	0.65	0.10	0.78
LKH	ResNet	–	–	–	–	0.10	0.74
LKH	FT-Transformer	–	–	–	–	0.09	0.80

### 3.4 Agreement in Reproduction of Effects

In this section we assess how well the surrogate models can replicate the observable effects caused by changing the default configuration of irace. We have selected the three parameters shown in Table 3 and set them to the two listed levels, with all remaining parameters kept at their defaults. Then we compare the ground truth (i.e. irace executing the algorithms) to the results obtained using the surrogate models trained with 30 K samples. For the comparison we ran irace with the three budget values 780, 1890, and 3000 as explained above, and replicate each run 10 times with different seeds for irace. The best obtained configuration is then evaluated on all 10 instances.

To compare the models, we follow [13] and introduce scores that reflect the concordance of effects in surrogate models with the ground truth. For each of the three budgets and two scenarios, we compare the objective values found by the best configuration for a high and low level of one of the three parameters. The comparison is based on a Mack-Skillings test with a confidence level  $\alpha = 0.05$  and Bonferroni correction for multiple tests. In each case we attribute a score

**Table 3.** Selected parameters of irace with a brief description, and the tested levels.

Parameter	Description	Levels
firstTest	No. of instances evaluated before first statistical test	2, 10
elitistLimit	Max. no. of statistical tests without elimination	1, 5
confidence	Confidence level of statistical test	0.2, 0.95

**Table 4.** Overall scores for all seven models, three effects, and two scenarios.

Model	Confidence		ElitistLimit		firstTest	
	ACOTSP	LKH	ACOTSP	LKH	ACOTSP	LKH
CatBoost	0.17	0.00	0.00	0.00	0.00	0.33
FT-Transformer	0.17	0.00	0.00	0.00	0.00	0.33
Interpolation	0.17	0.00	0.00	0.00	0.33	0.33
PyRFR	0.17	0.00	0.00	0.00	0.00	0.33
XGBoost	0.33	0.33	0.00	0.00	0.00	0.33
resnet	0.33	0.17	0.00	0.00	0.00	0.33
skranger	0.33	0.17	0.00	0.00	0.00	0.33

that is 0, if the low level is significantly higher, 0.5 if both levels are statistically not different, and 1 if the high level is significantly higher. To compare a model to the ground truth, we report the average absolute distance of the model’s scores to the scores of the ground truth. In this model, a score of 0 corresponds to complete agreement with the ground truth, and a score of 1 to complete disagreement.

Table 4 shows the results. We can see that there is overall a very good agreement of models and ground truth, with scores never higher than 0.33, with one exception. Models of best accuracy, as reported in the previous section, also have best scores, with two exceptions: the interpolation baseline has also a comparable score, while XGBoost is slightly worse on parameter “confidence”. A closer look at the ground truth shows that for parameter “elitistLimit” the effects are not statistically significant. This explains the good scores for all models, which also find no effect, but limits the scope of the conclusions. In contrast, setting parameter “firstTest” to the high level is in four of six cases statistically different, and the models are able to reproduce this effect mostly.

We finally have a look at evaluation times, to see to what extent requirement 1 is satisfied. Average evaluation times per call to the models ranges from 1 ms to about 160 ms with two neural networks being the slowest to evaluate (without using a GPU). Therefore, in our experiments speedups range from 50 to 10000. Since the evaluation times are independent from algorithm execution times and grow only slowly with the number of samples, clearly speedups will grow with algorithm execution times.

## 4 Conclusions

In this paper we have compared surrogate models for the tuning of optimization algorithms, including several strategies for handling categorical and conditional variables, if the underlying model cannot represent them directly, and two ways for handling instances, namely by a single model or by per-instance models. We find that a one-hot encoding with a fixed value imputation and a combined model work best. Among the models one based on random forests (pyRFR) and two gradient-boosted trees (XGBoost, CatBoost) work well. A neural network (FT-Transformer) has the overall best performance. This also holds when evaluating the agreement of the surrogate models to the ground truth with the effects of changing tuner parameters, although XGBoost performs worse in this setting. Overall we can confirm that models that have been found to work well in the literature for surrogate models for hyperparameter optimization and execution time, also work well for objective function values, and that neural networks maybe an interesting alternative, which confirms findings of [16] with regard to tabular data. In future work, we plan to extend the scope of this study to more scenarios and a broader selection of tuners and models.

**Acknowledgments.** This research has been supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. M. de Souza acknowledges the support of the Santa Catarina State University, Brasil. M. Ritt acknowledges the support of CNPq, Brasil (grant 437859/2018-5) and Google Research Latin America (grant 25111).

## References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019)
2. Astudillo, R., Frazier, P.I.: Thinking inside the box: a tutorial on grey-box bayesian optimization. In: Proceedings of the 2021 Winter Simulation Conference, December 2021
3. AutoML. RFR: A extensible C++ library for random forests with Python bindings, December 2022. [https://github.com/automl/random\\_forest\\_run](https://github.com/automl/random_forest_run)
4. Bergstra, J., Yamins, D., Cox, D.: Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning, vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, June 2013, pp. 115–123. <https://proceedings.mlr.press/v28/bergstra13.html>
5. Bliet, L., Guijt, A., Karlsson, R., Verwer, S., de Weerdt, M.: EXPObench: benchmarking surrogate-based optimisation algorithms on expensive black-box functions. In: CoRR abs/2106.04618 (2021). [arXiv: 2106.04618](https://arxiv.org/abs/2106.04618)
6. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2016, pp. 785–794. ACM, San Francisco (2016). ISBN: 978-1-4503-4232-2. <https://doi.org/10.1145/2939672.2939785>

7. Claesen, M., Simm, J., Popovic, D., Moreau, Y., Moor, B.D.: Easy hyperparameter search using optunity. In: CoRR abs/1412.1114 (2014). [arXiv: 1412.1114](https://arxiv.org/abs/1412.1114)
8. Costa, A., Nannicini, G.: RBFOpt: an open-source library for black-box optimization with costly function evaluations. *Math. Program. Comput.* **10**(4), 597–629 (2018). <https://doi.org/10.1007/s12532-018-0144-7>
9. Cowen-Rivers, A., Lyu, W., Wang, Z., Tutunov, R., Jianye, H., Wang, J., Ammar, H.: HEBO: heteroscedastic evolutionary bayesian optimisation, December 2020. <https://valohaichirpprod.blob.core.windows.net/papers/huawei.pdf>
10. Dorigo, M., Stützle, T.: *Ant Colony Optimization*. MIT Press, Cambridge (2004)
11. Eggenesperger, K., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Efficient benchmarking of hyperparameter optimizers via surrogates. In: Bonet, B., Koenig, S. (eds.) *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 25–30 January 2015, pp. 1114–1120. AAAI Press, Austin (2015). <http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9993>
12. Eggenesperger, K., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Efficient benchmarking of hyperparameter optimizers via surrogates. In: Bonet, B., Koenig, S. (eds.) *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 25–30 January, 2015, pp. 1114–1120. AAAI Press, Austin (2015). <http://ceur-ws.org/Vol-1201/paper-06.pdf>
13. Eggenesperger, K., Lindauer, M., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Efficient benchmarking of algorithm configuration procedures via model-based surrogates. In: CoRR abs/1703.10342 (2017). [arXiv: 1703.10342](https://arxiv.org/abs/1703.10342)
14. Eggenesperger, K., Lindauer, M., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Efficient benchmarking of algorithm configurators via model-based surrogates. In: *Mach. Learn.* **107**(1), 15–41 (2018). <https://doi.org/10.1007/s10994-017-5683-z>
15. Eggenesperger, K., et al.: HPOBench: a collection of reproducible multi-fidelity benchmark problems for HPO. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)* (2021)
16. Gorishniy, Y., Rubachev, I., Khrulkov, V., Babenko, A.: Revisiting deep learning models for tabular data. In: CoRR abs/2106.11959 (2021). [arXiv: 2106.11959](https://arxiv.org/abs/2106.11959)
17. Gutmann, H.-M.: A radial basis function method for global optimization. In: *J. Global Optim.* **19**(3), 201–227 (2001). issn: 0925–5001. <https://doi.org/10.1023/A:1011255519438>
18. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. *Evolutionary Comput.* **9**(2), 159–195 (2001). <https://doi.org/10.1162/106365601750190398>
19. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016. <https://doi.org/10.1109/cvpr.2016.90>
20. Helsgaun, K.: An effective implementation of the lin-kernighan traveling salesman heuristic. *Eur. J. Oper. Res.* **126**, 106–130 (2000)
21. Tinós, R., Helsgaun, K., Whitley, D.: Efficient recombination in the Lin-Kernighan-Helsgaun traveling salesman heuristic. In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D. (eds.) *PPSN 2018. LNCS*, vol. 11101, pp. 95–107. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-99253-2\\_8](https://doi.org/10.1007/978-3-319-99253-2_8)
22. Helsgaun, K.: General k-opt Submoves for the Lin-Kernighan TSP Heuristic. *Math. Programm. Comput.* **1**(2-3), 119–163 (2009)
23. Helsgaun, K.: Source Code of the Lin-Kernighan-Helsgaun Traveling Salesman Heuristic (2018). <http://webhotel4.ruc.dk/~keld/research/LKH>

24. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-25566-3\\_40](https://doi.org/10.1007/978-3-642-25566-3_40)
25. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. *J. Artif. Intell. Res.* **36**, 267–306 (2009). <https://doi.org/10.1613/jair.2861>
26. Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ACLib: a benchmark library for algorithm configuration. In: Pardalos, P.M., Resende, M.G.C., Vogiatzis, C., Walteros, J.L. (eds.) LION 2014. LNCS, vol. 8426, pp. 36–40. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-09584-4\\_4](https://doi.org/10.1007/978-3-319-09584-4_4)
27. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.* **206**, 79–111 (2014). <https://doi.org/10.1016%2Fj.artint.2013.10.003>. <https://doi.org/10.1016/j.artint.2013.10.003>
28. Johnson, D.S., McGeoch, L.A., Rego, C., Glover, F.: 8th DIMACS Implementation Challenge: The Traveling Salesman Problem (2001). <http://dimacs.rutgers.edu/archive/Challenges/TSP>
29. Kendall, M.G.: A new measure of rank correlation. *Biometrika* **30**(1-2), 81–93 (1938). <https://doi.org/10.1093/biomet/30.1-2.81>
30. Klein, A., Dai, Z., Hutter, F., Lawrence, N., González, J.: Meta-surrogate benchmarking for hyperparameter optimization. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook (2019)
31. Lindauer, M., et al.: SMAC3: a versatile bayesian optimization package for hyperparameter optimization. In: CoRR (2021). [arXiv: 2109.09831](https://arxiv.org/abs/2109.09831) [cs.LG]
32. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: iterated racing for automatic algorithm configuration. *Oper. Res. Perspect.* **3**, 43–58 (2016). <https://doi.org/10.1016/j.orp.2016.09.002>
33. Malherbe, C., Vayatis, N.: Global optimization of lipschitz functions. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML 2017. Sydney, NSW, Australia: JMLR.org, pp. 2314–2323 (2017)
34. Pedregosa, F., et al.: Scikit-learn: machine learning in python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)
35. Prokhorenkova, L., Gusev, G., Vorobev, A., Drogush, A.V., Gulin, A.: CatBoost: unbiased boosting with categorical features. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. NIPS 2018, pp. 6639–6649. Curran Associates Inc., Montréal (2018)
36. Rice, J.R.: The algorithm selection problem. *Adv. Comput.* **15**, 65–118 (1976). [https://doi.org/10.1016/S0065-2458\(08\)60520-3](https://doi.org/10.1016/S0065-2458(08)60520-3)
37. Shepard, D.: A two-dimensional interpolation function for irregularlyspaced data. In: Proceedings of the 1968 23rd ACM National Conference. ACM Press (1968). <https://doi.org/10.1145/800186.810616>
38. Škvorc, U., Eftimov, T., Korošec, P.: GECCO black-box optimization competitions. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM, July 2019. <https://doi.org/10.1145/3319619.3321996>
39. Springenberg, J.T., Klein, A., Falkner, S., Hutter, F.: Bayesian optimization with robust bayesian neural networks. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 12. Curran Associates Inc. (2016). <https://proceedings.neurips.cc/paper/2016/file/a96d3afec184766bfeca7a9f989fc7e7-Paper.pdf>

40. Stützle, T.: ACOTSP: a software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem (2002). <http://www.aco-metaheuristic.org/aco-code>
41. Turner, R., et al.: Black-Box Optimization for Machine Learning (2020). [https://github.com/rdturnermtl/bbo\\_challenge\\_starter\\_kit](https://github.com/rdturnermtl/bbo_challenge_starter_kit)
42. Vaswani, A., et al.: Attention is all you need. In: Guyon, I., et al. (eds.) *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates Inc. (2017). <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>
43. Wilson, D.R., Martinez, T.R.: Improved heterogeneous distance functions. *J. Artif. Intell. Res.* **6**, January 1997. <https://doi.org/10.1613/jair.346>
44. Wright, M.N., Ziegler, A.: Ranger: a fast implementation of random forests for high dimensional data in C++ and R. *J. Stat. Softw.* **77**(1 ) (2017). <https://doi.org/10.18637/jss.v077.i01>