Evaluating random forest models for irace

Leslie Pérez Cáceres
IRIDIA, CoDE
Université libre de Bruxelles
leslie.perez.caceres@ulb.ac.be

Bernd Bischl
Department of Statistics
Ludwig-Maximilians-Universität
bernd.bischl@stat.uni-muenchen.de

Thomas Stützle
IRIDIA, CoDE
Université libre de Bruxelles
stuetzle@ulb.ac.be

ABSTRACT

Automatic algorithm configurators can greatly improve the performance of algorithms by effectively searching the parameter space. As algorithm configuration tasks can have large parameter spaces and the execution of candidate algorithm configurations is often very costly in terms of computation time, further improvements in the search techniques used by automatic configurators are important and increase the applicability of available configuration methods. One common technique to improve the behavior of search methods when evaluations are computationally expensive are surrogate model techniques. These models are able to exploit the scarce available data and help to direct the search towards evaluating the most promising candidate configurations. In this paper, we study the use of random forests models as surrogate models in irace, a flexible automatic configuration tool based on iterated racing that has been successfully applied in the literature. We evaluate the performance of the random forest model using different settings when trained with data obtained from the irace configuration process and we evaluate their performance under similar conditions as in the configuration process. This preliminary work aims at providing guidelines for the incorporation of random forest to the configuration process of irace.

CCS CONCEPTS

•**Computing methodologies** → **Search methodologies;**
•**Mathematics of computing** → *Probabilistic algorithms;*

KEYWORDS

automatic algorithm configuration, parameter tuning, irace, random forests

ACM Reference format:
Leslie Pérez Cáceres, Bernd Bischl, and Thomas Stützle. 2017. Evaluating random forest models for irace. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082057

1 INTRODUCTION

Algorithm configuration is a crucial step when developing or applying algorithms. Parameter settings have often a substantial effect on algorithm performance. Finding adequate parameter settings can be a complex task as parameter spaces may be large, the evaluation of a parameter setting is computationally expensive as it involves actually executing an algorithm, parameters may have strong (non-linear) inter-dependencies, and the evaluation of candidate algorithm configurations is inherently stochastic. The algorithm configuration problem is very relevant for the automatic design of algorithms, not only because of the effects of parameter settings on the performance of algorithms, but also because the design of algorithms itself can be modeled as an algorithm configuration problem. Design choices can be represented as the domain of categorical parameters that, under defined rules, can be combined to obtain more specialized and effective algorithms. Several general-purpose automatic algorithm configuration tools have been proposed in the literature such as Paramils [10], SMAC [9], GGA++ [1] and irace [15]. These tools provide methods to effectively configure algorithms by searching the algorithm parameter space trying to optimize performance over a provided set of training instances. Irace is an automatic configurator based on iterated racing, it has been successfully applied to configure different types of algorithms [15], it is freely available as an R package[1] and it does not require any specific knowledge of R or the inner workings of irace itself.

Surrogate models have been widely applied in optimization [4, 13, 14]. They have shown to be particularly useful when the problem at hand involves computationally expensive evaluations (e.g. simulations) or very limiting time constraints (e.g. real time systems). Sequential model-based optimization (SMBO) is an example of a model-based algorithm, in which a surrogate model is iteratively built using the available solution evaluations. This model is searched to select new solutions that optimize the prediction of the model and real evaluations are only performed on the most promising solutions. In this way, the surrogate model provides an inexpensive evaluation procedure that can be used to guide the search towards promising regions of the search space. As the evaluation of algorithm configurations is inherently costly, requiring the actual execution of candidate configurations on a number of training instances, it seems natural to exploit surrogate modeling techniques also in the context of automated algorithm configuration. Moreover, these models could be later used for the analysis of properties of algorithm components providing insights that could be used in automatic algorithm design. Some initial work in this direction has been done already more than a decade ago [5], and it has shown to be useful also in at least two high-performing automatic algorithm configuration packages, SMAC [9] and later also GGA++ [1]. SMAC is a state-of-the-art model-based automatic configurator based on SMBO. SMAC builds a random forest model

[1]The irace package: http://iridia.ulb.ac.be/irace/

to predict the expected improvement of unseen configurations in order to select the most promising to be evaluated. The GGA++ automatic configurator implements a model-based genetic algorithm that embeds a random forest model especially adapted to predict high-performing areas of the search space. This model is used to determine the most promising offspring to be evaluated. A study of different possible models to predict algorithm performance has been undertaken by Hutter et al. [11], where these models are called empirical performance models. They identified random forests as the most promising models to generate accurate predictions.

Irace is an iterated racing algorithm where in each iteration first a set of candidate configuration is sampled according to a probabilistic sampling model, then the best configurations are determined using a racing procedure, and finally the probabilistic model is updated, taking into account a set of the best candidate configurations and biasing the probabilistic model towards these best candidate configurations. Irace can be seen therefore as an estimation of distribution algorithm. The sampling model helps irace to focus the search on good areas of the search space, while allowing it to explore the search space in the initial iterations when the sampling is still wide. The model used in irace has the advantage of being very simple and providing an easy way to control its search behavior. Regardless of this, the model is limited compared with more powerful ones such as the random forests models. For example, parameter interactions are only taken indirectly into account: irace is based on a kind of local modeling approach, but at each local model no parameter interactions are explicitly modeled. Given the amount of algorithm evaluations performed by irace during the configuration process, it would be interesting to use a more powerful model to improve the data usage and to generate information on parameter interactions or, more in general, the parameter landscape, which in turn could be used as feedback for the configuration process. For this aim, random forests seem to be a natural choice, given their ability to deal with different types of variables (discrete, numerical) and their previously reported good predictions when used as empirical performance models [11]. In this paper we study and evaluate the use of random forest models to support the configuration process performed by irace. We do so by examining the prediction performance of the random forest when trained on data stemming from an irace run. We consider the integration of random forests into irace by, after the sampling of candidate configurations, applying the predictions of the random forest models to filter the most promising candidate configurations before these are evaluated in a race. In contrast with SMAC, we do not search the model to select promising configurations and we perform the selection based on the predicted performance and not the expected improvement. In this sense, the usage made of random forest models we study in this work is more similar to the usage in GGA++.

The paper is structured as follows. Section 2 gives a short overview of irace and introduces the random forest models describing their use in the SMAC and GGA++ configurators. We describe the algorithm configuration scenarios, the data sets and the details of the random forest models used in this work in Section 3. Section 4 evaluates the performance of the random forest model using different characterizations over data sets obtained from the configuration scenarios. In Section 5, we give a preliminary comparison between irace and using the random forest to filter configurations in irace. Finally, Section 6 concludes and discusses future work.

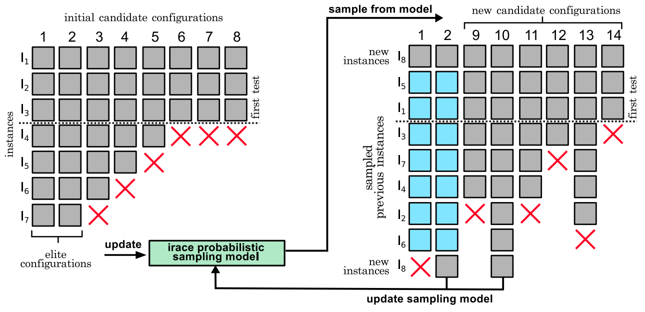

Figure 1: Configuration process in irace.

2 BACKGROUND

2.1 Algorithm configuration and irace

Algorithm configuration is the task of finding parameter settings (configurations) of a target algorithm, that exhibit good empirical performance on a given set of problem instances. Configuration tasks (scenarios) must define: a target algorithm, a performance measure to optimize, a parameter search space, a set of problem instances, and a configuration budget (evaluations or time). The most common performance measures are *solution quality* and *running time*. The first one is more common when configuring metaheuristics, while the second is more common when configuring exact algorithms. Normally, the instance set is assumed to be representative of the instances to be encountered when applying the target algorithm. A configuration scenario can be classified as *homogeneous* or *heterogeneous* depending on how homogeneous is the relative performance of configurations regarding the instance set. Heterogeneous scenarios have the additional difficulty of requiring evaluations on a large number of different instances to determine configurations that perform well across the instance set; homogeneous scenarios may evaluate less instances and explore more the parameter search space. Parameter spaces are defined by assigning a set of possible values to each parameter. The types of parameters available to be used depend of the configurator at hand. The parameter types supported by irace are: *categorical* parameters whose domain is a set of unordered values (e.g. the type of local search to be used in an algorithm), *ordered* parameters similar to the categorical ones but the values have an order relationship (e.g. perturbation strength: none, low, medium, high, very high), *integer*, and *real* parameters. Conditionality is also supported; a conditional parameter is a parameter that is active only if one or more parameters have certain values (e.g. the tabu list size will be only active when tabu search is selected as local search). When the conditions of a parameter are not met, the parameter is not active and ignored.

As already mentioned, irace [15] is an iterated racing-based algorithm [2, 6]. Figure 1 illustrates the configuration process performed by irace. The process starts by generating candidate configurations uniformly at random from the parameter space. Additionally, a set of user-defined initial configurations can be provided. A racing procedure is then applied to this set of configurations. It iteratively executes configurations on problem instances; configurations that perform statistically worse (assessed by means of a statistical test, commonly the Friedman test or t-test) are discarded and only the remaining configurations continue executing new instances until the termination criterion of the race is met. Next, the best surviving configurations (elite configurations) update a probabilistic model, from which new candidate configurations are sampled. Finally, the set of elite and new configurations undergo a new race. Elite configurations cannot be discarded from the race until all previously seen instances have been executed by the new configurations. For more details about irace we refer to [15].

2.2 Random forest

Random forests [7] is an ensemble model technique in which, several classification or regression trees are built in order to compose a high performing model. The aim of this method is to obtain a better predictive performance by overcoming the over-fitting that commonly affects single trees. This is done by training several trees with different subsets of the training data and aggregating their predictions. A data point is defined as $(\{x_i^1, \ldots, x_i^j\}, r_i)$, where $\{x_i^1, \ldots, x_i^j\}$ is a set of independent variables (predictors) and r_i is the dependent variable (response) to be predicted. The data subsets used to build each tree are sampled using the *bagging* technique, where m^{sample} data points are sampled with replacement from the data set. These points are used to build a tree by selecting at each node a random subset of m^{split} independent variables to be evaluated to perform a split. The split variable and splitting point are selected with a defined splitting criterion based on the dependent variable values of the data points present in the node. This splitting process is continued until at least m^{min} data points are in each terminal node. Once the full tree is built, new data points can be predicted by carrying them through the tree and aggregating the dependent variable values present at the terminal node reached.

Random forests are used in SMAC [9]. The configuration process in SMAC consists in first iteratively searching the random forest model to obtain a configuration that maximizes the expected improvement; then performing the evaluation of the configuration comparing it with the current best configuration on a set of instances; and, finally, improving the model to restart the search. For more details about the SMAC algorithm we refer to [9]. The use of random forests in SMAC has shown to be beneficial when configuring certain scenarios. The training data for the random forest is built by using the parameter values of a configuration (n^{param}) and the instance features (n^{feat}) as independent variables, and the observed performance as the dependent variable. Instance features are only added when available, otherwise an identifier of the instance is used. The random forest model is trained using these data points and the predicted performance is obtained by aggregating (using the mean or median) the predictions for all training instances. By default in SMAC, the random forest model consists of $m^{\text{trees}} = 10$ trees, each of them trained with a sample of $m^{\text{sample}} = 90\%$ of the available data points. To split a tree node, a minimum of $m^{\text{min}} = 10$ data points is required and $m^{\text{split}} = \left\lceil (n^{param} + n^{feat}) \cdot (5/6) \right\rceil$ of the independent variables are evaluated as candidates for the split.

GGA++ [1] employs a random forest model to assist in the cross-over of candidates configurations. GGA++ is a genetic-based configurator, that implements a tree-based representation of configurations which permits the definition of a cross-over operator. At each iteration, candidate configurations (belonging to a set of the population) are selected to mate and the resulting child is then mutated. The offspring of two parent configurations is obtained by searching the configuration space of all the possible combinations of the parents parameter values. The search is performed by sampling a number of configurations using a sampling method based on the parameter values of the parents and the structure of a random forest model trained with the available solution evaluations. The sampled configurations are evaluated with the random forest model and the configuration with the higher predicted performance is selected. The random forest model of GGA++ implements a splitting criterion designed to focus the model on the best 10% performing configurations. At each node, the possible split variables are evaluated by measuring how well they keep the 10% best performing configurations in one node, while maintaining the most low-performing ones in the other.

3 EXPERIMENTAL SETUP

In this section, we describe the experiments carried out to evaluate the random forest models to be used in the irace configuration process.

3.1 Configuration scenarios

We perform the experiments based on performance data obtained by configuring with irace four configuration scenarios (two for optimising solution quality and two for minimising running time):

ACOTSP: ACOTSP [16] as target algorithm for configuring solution quality with 11 parameters (3 categorical and 8 numerical). Executions have 20 seconds of execution time and the training and testing set are each composed of 50 and 250 instances respectively. These instances are random uniform Euclidean TSP instances varying from sizes 1000 to 3000. The total configuration budget is 5000 evaluations.

ACOTSP 2000: Same characteristics as the ACOTSP scenario, but the training and testing set is each composed of 200 random uniform Euclidean TSP instances of size 2000.

Regions 100: CPLEX [12] version 12.4 is used as target algorithm for minimising computation time with 74 categorical parameters. Evaluations have 5 seconds of maximum execution time and the training and testing set is composed of 1000 mixed integer programming (MIP) instances each. These instances are encodings of a combinatorial auction winner determination problem with 100 goods and 500 bids. The total configuration budget is 18000 seconds.

Spear: Spear [8] as target algorithm for minimising computation time with 26 categorical parameters. Evaluations have 300 seconds of maximum execution time and the training and testing set is composed of 302 SAT instances each. The total configuration budget is 172800 seconds.

The instance test sets mentioned above are only used when evaluating the performance of irace and are not used with the random forests models.

3.2 Data sets

To perform the training and evaluation of the random forests models, we use data sets of the performance of candidate configurations in the training instance set. A data point is defined as (θ_s, i, r_s^i), where θ_s are the parameters values selected in configuration s, i is an instance and r_s^i is the performance of configuration s on instance i. We choose not to use instance features given that these are not always available in real-world problems and the focus of this evaluation is on the more general case. Consequently, a different analysis should be performed if instance features are used to build the model. We define two types of data sets as follows:

uniform: data sets that were created by sampling uniformly 1000 configurations from each configuration scenario; these configurations are executed on the instance training set.

real: data sets that were created by obtaining the performance evaluated by irace at different stages of the configuration process. The **real-first** data set is composed by the data points obtained by irace on the first iteration of the configuration process and the **real-last** data set is composed of all data points obtained by a full irace execution.

In order to train the random forest model, we define categorical and numerical parameters as categorical and numerical independent variables respectively. Moreover, parameter search spaces commonly define conditional parameters that is, parameters that are only active when one or more other parameters have certain values. This conditionality can be handled as missing values in the data sets and therefore they must be imputed. In this work, we have assigned a constant string to impute missing categorical data points and use the double of the upper bound of the parameter domain $(2 * max(domain(d^{param})))$ for numerical ones. As already mentioned, the instances are also added to the data points as independent categorical variables.

3.3 Random forest models

The random forest models are trained using the data sets described in the previous section. The performance prediction of an unseen configuration s is obtained as: $p_s = mean(p_s^1, \ldots, p_s^i)$, where p_s^i is the predicted performance of configuration s on instance i and $p_s^i = mean(p_s^{i,1}, \ldots, p_s^{i,j})$, where $p_s^{i,j}$ is the predicted performance of configuration s on instance i obtained by the tree j in the random forest model. In this work, we use the ranger [17] implementation of random forest, using the interface for the library provided as an R package[2]. We use the default split criterion that is, the reduction of node impurity, where impurity is measured as the Gini index or the estimated prediction variance for classification and regression trees, respectively.

4 RANDOM FOREST EVALUATION

In this section, we perform an analysis of the random forest models using algorithm performance data in order to evaluate their performance predicting the quality of unseen configurations and how they can support the configuration process effectively.

[2]ranger package version 0.7: https://cran.r-project.org/package=ranger

Table 1: Random forest model settings

	m^{trees}	m^{sample}	m^{split}	m^{min}
default settings	10	0.9	$\lfloor\sqrt{(n^{param}+1)}\rfloor$	10
tuned settings	500	1	$\lceil(n^{param}+1)\cdot 0.5\rceil$	10

Table 2: Correlation coefficient of the average ranking of the real and predicted performance of 1000 configurations using random forest models trained with real-first and real-last data sets, default random forest settings (Table 1) are used.

	ACOTSP	ACOTSP2000	Regions 100	Spear
real-first	0.7927	0.8836	0.2762	0.2155
real-last	0.8456	0.9031	0.38451	0.3069

4.1 Random forest models using irace data

In the following experiments we use the **real** data sets. Two models are built using the **real-first** and **real-last** data sets. The first one aims at studying how much it is possible to rely on the random forest model in early iterations, and the model trained with **real-last** aims at studying the performance of the models in the best possible case and their applicability after the execution of irace to predict high-performing configurations. The evaluation of the models is done using the data points of the **uniform** data sets and we evaluate their performance by calculating the correlation coefficient of the average ranking of each configuration over the training instances on the real and predicted performance. This evaluation is chosen given that we are not interested in obtaining an accurate prediction of the configuration performance, but more on predicting which configurations have more potential to be high-performing. For the computation time data sets, Regions 100 and Spear, we trained the models with the log-transformed run time as dependent variable. This is a common practice to improve performance of models trained to predict algorithm running times. Table 1 gives the random forests settings used in the experiments. The default settings were chosen based on the default settings of SMAC and the default settings of the ranger library. The tuned settings correspond to the settings used to perform experiments with the random forest and irace, and were defined based on the following experiments.

Table 2 gives the correlation coefficients of the ranking of real performance of the 1000 uniformly sampled configurations and the predicted performance obtained by the models trained with the **real-first** and **real-last** data sets. The initial tests show that the solution quality and running time data sets are very different. While both ACOTSP models obtain a high correlation coefficient on the two irace data sets, on the Spear and Regions 100 scenarios it is much lower. As can be expected, the performance of the models improves when the **real-last** data set is used for training, which is due to the fact that this data set contains more data points and therefore gives more information about the search space. We presume that the high correlation obtained by the models trained with **real-first** on the ACOTSP scenarios is due to the existence (based on our experience) of parameter values that strongly influence the performance of ACOTSP. The evaluations performed on the first iteration

Table 3: Correlation coefficient of the average ranking of the real and predicted performance of 1000 configurations using random forest models trained with the real-last data set, default settings and different values for the m^{trees}.

m^{trees}	2	10	50	100	500	1000
ACOTSP	0.800	0.846	0.889	0.908	0.916	0.915
ACOTSP2000	0.873	0.903	0.928	0.931	0.933	0.933
Regions 100	0.176	0.384	0.518	0.533	0.552	0.553
Spear	0.202	0.307	0.445	0.469	0.510	0.506

Table 4: Correlation coefficient of the average ranking of the real and predicted performance of 1000 configurations using random forest models trained with the real-last data set, default settings, $m^{\text{trees}} = 500$ and different values for m^{sp}.

m^{sp}	0.1	0.3	0.5	0.8	1.0
ACOTSP	0.890	0.926	0.931	0.928	0.924
ACOTSP2000	0.917	0.935	0.937	0.936	0.936
Regions 100	0.551	0.613	0.636	0.658	0.668
Spear	0.433	0.561	0.585	0.586	0.570

are enough for the random forest model to identify this feature and obtain a better prediction. On the other hand, the Regions 100 and Spear scenarios have a very low correlation for both models. This may be an effect of the size of the parameter space, that makes difficult for the random forest model to capture the features of the search space with the much larger number of parameters of SPEAR and CPLEX. The number of trees (m^{tree}) could play a big role in such poor results since we only use 10 trees in the forest.

We use the **real-last** data set to perform experiments exploring the number of trees built in the random forest. Table 3 gives the performance obtained by the models using different forest sizes. As expected, the performance of the models increases when the number of trees is increased. This is particularly clear for the biggest scenario (Regions 100), where the performance is strongly increased. For the ACOTSP scenarios the performance increases very slightly, indicating that for these data sets only a few trees are necessary. We set $m^{\text{trees}} = 500$ from now on for all the scenarios, given that the performance of the models stabilizes in all the cases for this number of trees. Note that increasing the number of trees, however, increases the computational effort required to build the forest.

The number of variables evaluated when splitting a node has a big impact on the computational effort required to train the random forest model [17]. To study the effect of the number of variables evaluated for a split (m^{split}) in the performance of the model, we define a new parameter m^{sp} so that $m^{\text{split}} = \lceil (n^{param} + 1) \cdot m^{\text{sp}} \rceil$. We train the models with different values of m^{sp}. The correlation coefficients of the predicted and real performance are given in Table 4. This parameter has no big impact in the models of the two ACOTSP scenarios. For the Regions 100 scenario, the best performance is obtained by evaluating the most possible variables to find good splits for them. We set $m^{\text{sp}} = 0.5$, given that it obtains a reasonable performance for all data sets.

Table 5: Correlation coefficient of the average ranking of the real and predicted performance of 1000 configurations using random forest models trained with the real-last data set, using default settings, $m^{\text{trees}} = 500$, $m^{\text{split}} = 0.5$ and different values for the m^{sample} setting.

m^{sample}	0.05	0.2	0.4	0.6	0.8	1.0
ACOTSP	0.817	0.895	0.916	0.927	0.928	0.931
ACOTSP2000	0.915	0.933	0.936	0.937	0.937	0.937
Regions 100	0.575	0.603	0.621	0.634	0.634	0.637
Spear	0.521	0.546	0.556	0.582	0.574	0.584

Finally, we perform experiments with the number of data points sampled to build the trees (m^{sample}). Table 5 gives the correlation coefficients obtained by using different values of m^{sample}. In all cases the performance increases when the number of sampled points is increased. The ACOTSP2000 model reaches its better performance when sampling 0.6% of the data points, which is evidence of homogeneity of the scenario. The number of points sampled can have an effect on the ability of the model of generalize the predictions; smaller values of this parameter allows trees to be trained on different areas of the search space. The results suggest that having access to all the data points leads to better performance than fragmenting the data set, thus we set $m^{\text{sample}} = 1$.

The final performance of the models using the selected settings is given in Figure 2. The performance is improved compared to the initial one (Table 2). These settings are still preliminary and further configuration is possible. As future work we plan to use irace to set these settings for a wide range of configuration problems. Even more, since these results suggest that the best settings for the models depend on the scenario features, methods to adapt these parameters according to the characteristics of the scenario and the stage of the irace search process could be required.

4.2 Model and irace convergence

The previous models were evaluated by predicting the mean performance of uniformly sampled configurations. In irace, the sampling of configurations is centered around the best solutions and thus it is not really evident if the random forest model will be good at predicting the performance of the configurations that are in the (current) best part of the parameter space. We performed experiments using **real-first** and **real-last** as training data and 1000 configurations sampled from the model of the best configuration obtained by irace in the first and last iteration respectively. Figure 3 gives the results obtained by the models. Surprisingly, the ACOTSP models obtain a very bad performance when predicting the configurations sampled from the area of the search space on which irace has converged (models trained with **real-last**). On the other hand, it seems that the Regions 100 and Spear models are able to predict well the configurations sampled in the best area, while they poorly predict the ones sampled at the start of the search process. These results indicate that the characteristics of the scenario are very important when using the random forest models to predict the performance of configurations in the configuration process. Poor predictive performance in late stages of the configuration

Table 6: Wilcoxon paired test p-values (significance 0.05) of the mean performance of 20 executions of irace with and without random forests models.

irace vs.	best	proportional	uniform
ACOTSP 2000	1.0	0.841	**0.003**
Regions 100	0.985	0.841	**1.907e-06**
Spear	0.927	0.097	0.189

process, as observed for the ACOTSP scenarios, might indicate the convergence of the search and the need to (1) evaluate more configurations in the particular area of the parameter landscape in order to intensify, or (2) perform a restart of the search to explore other areas of the landscape.

5 CONFIGURATION SELECTION WITH RANDOM FORESTS IN IRACE

In this section, we evaluate the use of random forest models to support the selection of newly sampled configurations in irace. Each iteration, we train a model using all the available data points in irace and this model is used to predict the performance of $10 * n^{\text{conf}}$ newly sampled configurations, where n^{conf} is the number of configurations required for the next race. We evaluate 3 sampling and selection strategies:

proportional: $10 * n^{\text{conf}}$ configurations are generated from the irace model and n^{conf} of them are selected randomly proportionally to the performance predicted by the random forest model.

best: $10 * n^{\text{conf}}$ configurations are generated from the irace model and the n^{conf} configurations are selected that have the best predicted performance according to the random forest model.

uniform: $10 * n^{\text{conf}}$ configurations are generated uniformly at random (that is, not following the irace model) and n^{conf} of them are selected randomly proportionally to the performance predicted by the random forest model.

We use for these tests only the ACOTSP2000, Regions 100 and Spear scenarios and perform 20 repetitions of the configuration process. The results obtained by irace are given in Figure 4 and in Table 6 are given the p-values of the Wilcoxon paired test comparing the variants to irace without random forest model.

The overall results evidence no improvement in the performance obtained by irace when using the random forest model to select new configurations. Sampling configurations uniformly and predicting them is not a good strategy for the ACOTSP2000 and Regions 100 scenarios. This is probably because disabling the sampling model of irace does not allow the search to converge properly towards generating high-performing candidate configurations. Interestingly, there is no significant difference for the Spear scenario in any of the tests performed. We presume that this is due to the high heterogeneity of the scenario that makes good configurations difficult to find in early stages of the search. The configuration process is therefore very dependent of the instance sampling and evaluating a large number of instances is more beneficial than exploring a large number of configurations. The random forest seems not to contribute to face this difficulty. In this sense, the use of instance features in the random forest training data set could lead to improved performance given that it could help to deal with the heterogeneity of the scenario. The results suggest that, to get an improved irace performance, the configuration process must be adapted in order to profit from the capabilities of the random forest models. Given the good performance obtained in Section 4 by the random forest model for the ACOTSP2000 scenario, the intuition is that the use of this model to select configurations will increase the intensification in irace from the first iteration. The plots in Figure 5 show the convergence of the mean performance over the test set of the best configurations in each iteration of the 20 irace executions. As expected, the irace version using the random forest model converges quicker in the ACOTSP2000 and Regions 100 scenarios, while for the Spear scenario the convergence is not greatly affected. The random forest has the opportunity to contribute more in the early stages of the search, when the sampling model of irace has not yet converged to a specific area of the parameter search space. These preliminary results are an indication that adjustments in the search behavior of irace are needed in order to make better use of the random forest model. For example, the convergence of the sampling model can be delayed, in order to allow more exploration guided by the random forest model. Additionally, the evaluation of the model every iteration might be useful to detect stages of the search process and even more, the models can be used to predict the importance of parameters. In turn, this prediction of the parameter importance and possible interactions between parameters may lead to an improved sampling model for irace.

6 CONCLUSIONS AND FUTURE WORK

We have presented a preliminary analysis of the use of random forest models in irace. Experiments with irace data show that the random forests can obtain good prediction performance using data obtained by irace. However, the quality of these predictions seems to be dependent on the characteristics of the scenario and also on the convergence state of the sampling model from which the predicted configurations are sampled. Despite the fact that these preliminary tests with irace and random forests do not indicate a major leap in performance, the good results obtained by predicting configurations based on irace data, indicates that these models can contribute to the configuration process, particularly if the configuration budget is small. Uniformly sampling new configurations and selecting them based on the predictions of the random forest model reduced the performance of irace in most of the studied scenarios. This is a clear indication of the importance of the sampling model used in irace. Nevertheless, the benefits of using random forests could be more evident in scenarios where parameters have strong interactions that the irace model cannot easily detect.

In future work, we will extend the experiments using irace and random forest to other scenarios that have different characteristics to the ones used in this work. The use of weights for the training data, based on the number of configurations executed in each problem instance, will be investigated. The selection of data points to train the random forests might be also of interest, especially when convergence is achieved and a specialized model on a particular area of the search space is required. It is also part of the future work, the study of the use of surrogate models to classify configurations in high and poor performing ones, instead of predicting their performance. Additionally, the use of the random forest structure to analyse the parameter landscape is a promising feature of these models as variable importance can be calculated from the model, and this information can be exploited for the sampling process.

REFERENCES

[1] Carlos Ansótegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. 2015. Model-Based Genetic Algorithms for Algorithm Configuration. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15)*, Qiang Yang and Michael Wooldridge (Eds.). IJCAI/AAAI Press, Menlo Park, CA, 733–739.

[2] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. 2007. Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement. In *Hybrid Metaheuristics*, Thomas Bartz-Beielstein, María J. Blesa, Christian Blum, Boris Naujoks, Andrea Roli, Günther Rudolph, and M. Sampels (Eds.). Lecture Notes in Computer Science, Vol. 4771. Springer, Heidelberg, Germany, 108–122.

[3] Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss (Eds.). 2010. *Experimental Methods for the Analysis of Optimization Algorithms.* Springer, Berlin, Germany.

[4] Thomas Bartz-Beielstein, C. Lasarczyk, and Mike Preuss. 2010. The Sequential Parameter Optimization Toolbox. See [3], 337–360.

[5] Thomas Bartz-Beielstein and Sandor Markon. 2004. Tuning search algorithms for real-world applications: A regression tree based approach. In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC 2004).* IEEE Press, Piscataway, NJ, 1111–1118.

[6] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. 2010. F-Race and Iterated F-Race: An Overview. See [3], 311–336.

[7] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32. DOI: http://dx.doi.org/10.1023/A:1010933404324

[8] Frank Hutter, Domagoj Babić, Holger H. Hoos, and Alan J. Hu. 2007. Boosting Verification by Automatic Tuning of Decision Procedures. In *FMCAD'07: Proceedings of the 7th International Conference Formal Methods in Computer Aided Design.* IEEE Computer Society, Washington, DC, USA, Austin, Texas, USA, 27–34.

[9] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization, 5th International Conference, LION 5*, Carlos A. Coello Coello (Ed.). Lecture Notes in Computer Science, Vol. 6683. Springer, Heidelberg, Germany, 507–523.

[10] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research* 36 (Oct. 2009), 267–306.

[11] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206 (2014), 79–111.

[12] IBM. 2017. ILOG CPLEX Optimizer. http://www.ibm.com/software/integration/optimization/cplex-optimizer/. (2017).

[13] D. R. Jones, M. Schonlau, and W. J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* 13, 4 (1998), 455–492.

[14] Slawomir Koziel, David Echeverría Ciaurri, and Leifur Leifsson. 2011. Surrogate-Based Methods. In *Computational Optimization, Methods and Algorithms*, Slawomir Koziel and Xin-She Yang (Eds.). Studies in Computational Intelligence, Vol. 356. Springer, Berlin/Heidelberg, 33–59.

[15] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. 2016. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 3 (2016), 43–58.

[16] Thomas Stützle. 2002. 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/

[17] Marvin N. Wright and Andreas Ziegler. 2015. ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Arxiv preprint arXiv:1508.04409 [stat.ML]* (2015). https://arxiv.org/abs/1508.04409

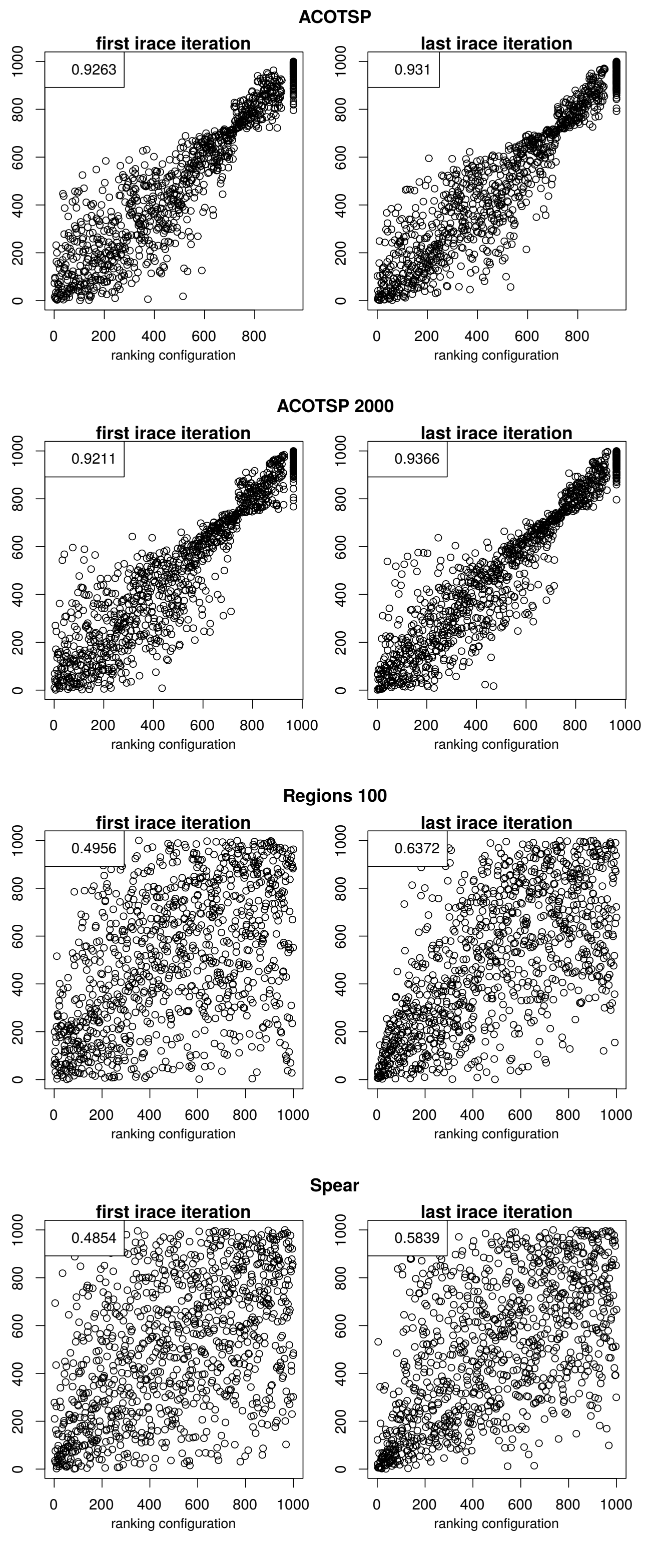

Figure 2: Average ranking of the real (x axis) and predicted (y axis) performance of 1000 configurations using random forest models trained with real-first and real-last data sets using tuned settings (see Table 1).

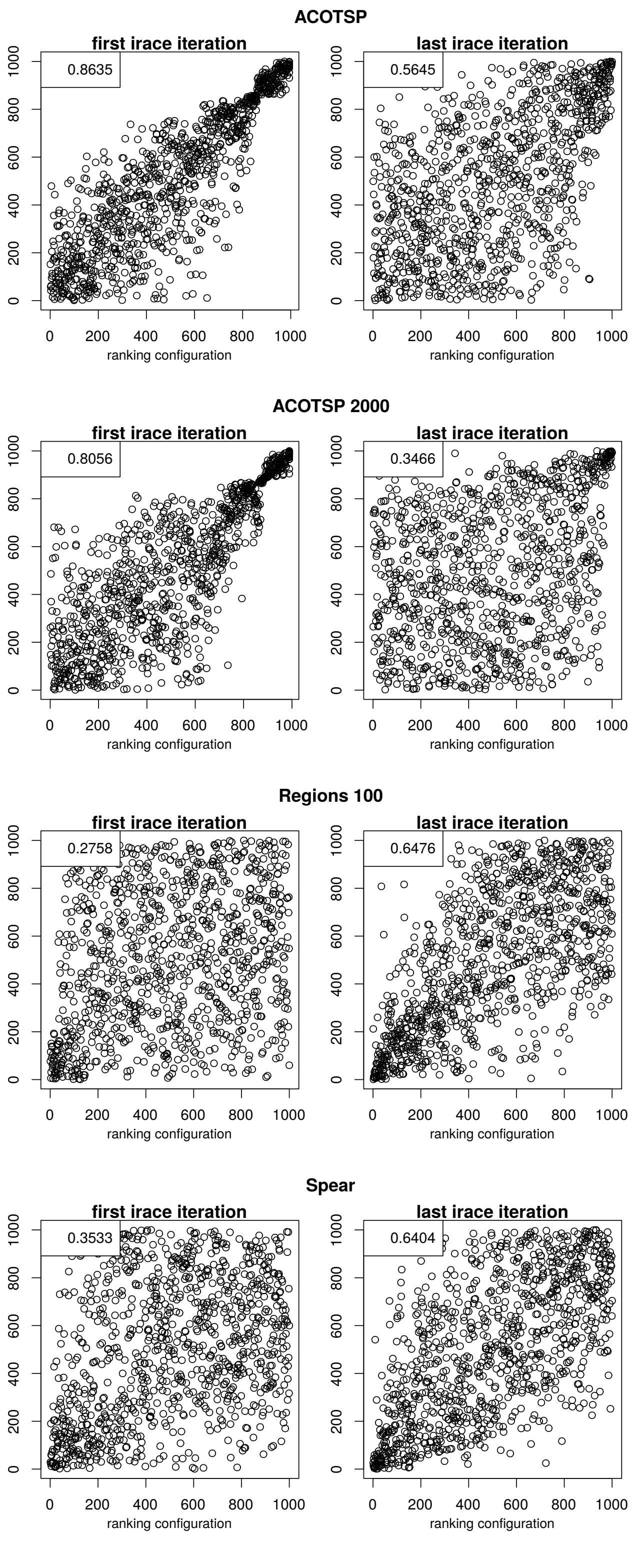

Figure 3: Average ranking of the real (x axis) and predicted (y axis) performance of 1000 configurations using random forest models trained with real-first and real-last data sets, tuned settings (see Table 1) and evaluated on configurations sampled from the best configuration irace model.

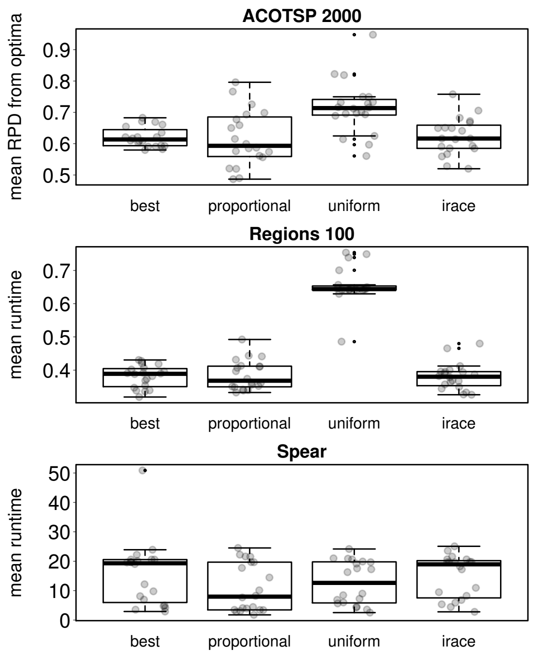

Figure 4: Mean relative percentage deviation from optima and running time over the test set of 20 configurations obtained by irace and the irace versions using random forests.

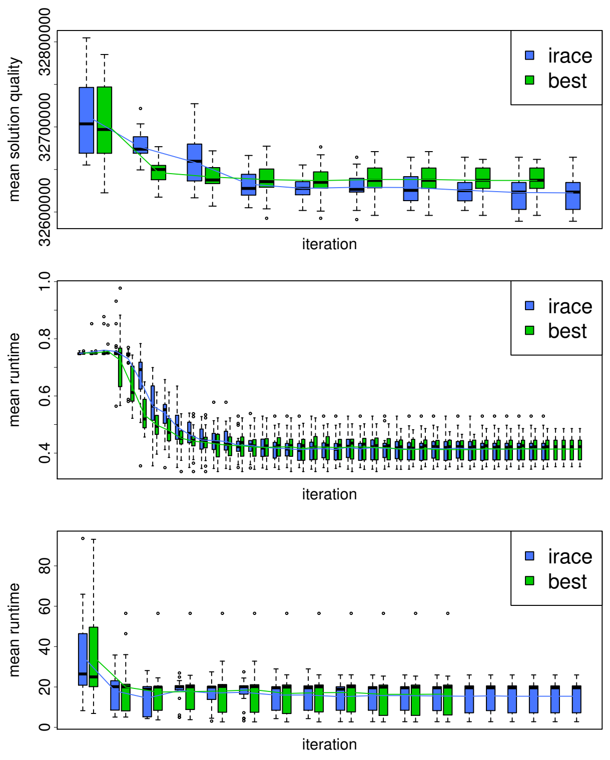

Figure 5: Mean performance over the test set of the iteration best configurations of irace best using random forests and irace.