Journal of Artificial Intelligence Research 53 (2015) 745-778 Submitted 02/15; published 08/15

AutoFolio: An Automatically Configured Algorithm Selector

Marius Lindauer lindauer@cs.uni-freiburg.de
University of Freiburg

Holger H. Hoos hoos@cs.ubc.ca
University of British Columbia

Frank Hutter fh@cs.uni-freiburg.de
University of Freiburg

Torsten Schaub torsten@cs.uni-potsdam.de
University of Potsdam
INRIA Rennes

Abstract

Algorithm selection (AS) techniques – which involve choosing from a set of algorithms the one expected to solve a given problem instance most efficiently – have substantially improved the state of the art in solving many prominent AI problems, such as SAT, CSP, ASP, MAXSAT and QBF. Although several AS procedures have been introduced, not too surprisingly, none of them dominates all others across all AS scenarios. Furthermore, these procedures have parameters whose optimal values vary across AS scenarios. This holds specifically for the machine learning techniques that form the core of current AS procedures, and for their hyperparameters. Therefore, to successfully apply AS to new problems, algorithms and benchmark sets, two questions need to be answered: (i) how to select an AS approach and (ii) how to set its parameters effectively. We address both of these problems simultaneously by using automated algorithm configuration. Specifically, we demonstrate that we can automatically configure claspfolio 2, which implements a large variety of different AS approaches and their respective parameters in a single, highly-parameterized algorithm framework. Our approach, dubbed AutoFolio, allows researchers and practitioners across a broad range of applications to exploit the combined power of many different AS methods. We demonstrate AutoFolio can significantly improve the performance of claspfolio 2 on 8 out of the 13 scenarios from the Algorithm Selection Library, leads to new state-of-the-art algorithm selectors for 7 of these scenarios, and matches state-of-the-art performance (statistically) on all other scenarios. Compared to the best single algorithm for each AS scenario, AutoFolio achieves average speedup factors between 1.3 and 15.4.

1. Introduction

Over the last decade, tremendous progress in Boolean constraint solving technology has been achieved in several areas within AI, such as SAT (Biere, 2013), ASP (Gebser, Kaufmann, & Schaub, 2012), CSP (Tamura, Taga, Kitagawa, & Banbara, 2009), Max-SAT (Abramé & Habet, 2014) and QBF (Janota, Klieber, Marques-Silva, & Clarke, 2012). In all these areas, multiple algorithms with complementary solving strategies exist, and none dominates all others on all kinds of problem instances. This fact can be exploited by algorithm selection (AS) (Rice, 1976) methods, which use characteristics of individual prob-

lem instances (so-called *instance features*) to choose a promising algorithm for each instance. Algorithm selectors have empirically been shown to improve the state of the art for solving heterogeneous instance sets and, as a result, have won many prizes at competitions. For instance, SATzilla (Xu, Hutter, Hoos, & Leyton-Brown, 2008) won several categories in multiple SAT competitions, claspfolio 1 (Gebser, Kaminski, Kaufmann, Schaub, Schneider, & Ziller, 2011b) won the NP-track of the 2011 ASP Competition, CP-Hydra (O'Mahony, Hebrard, Holland, Nugent, & O'Sullivan, 2008) won the the 2008 CSP competition, ISAC^{++} (Ansótegui, Malitsky, & Sellmann, 2014) won the partial Max-SAT Crafted and Industrial track of the 2014 Max-SAT Competition, and AQME (Pulina & Tacchella, 2009) won the first stage of the main track of the 2010 QBF Competition.

Although many new AS approaches have been proposed over the years (cf. Smith-Miles, 2008; Kotthoff, 2014), there are only two flexible frameworks that allow for re-implementing and comparing existing approaches in a fair and uniform way: LLAMA (Kotthoff, 2013) and claspfolio 2 (Hoos, Lindauer, & Schaub, 2014). Of these, claspfolio 2 is more comprehensive, encompassing strategies from the algorithm selection systems 3S (Kadioglu, Malitsky, Sabharwal, Samulowitz, & Sellmann, 2011), aspeed (Hoos, Kaminski, Lindauer, & Schaub, 2015), claspfolio 1 (Gebser et al., 2011b), ISAC (Kadioglu, Malitsky, Sellmann, & Tierney, 2010), ME-ASP (Maratea, Pulina, & Ricca, 2014), SNNAP (Collautti, Malitsky, Mehta, & O'Sullivan, 2013) and SATzilla (Xu et al., 2008; Xu, Hutter, Hoos, & Leyton-Brown, 2011).

Figure 1 illustrates the performance benefits these existing selection strategies (as realized in claspfolio 2) yield across the wide range of AS benchmarks in the Algorithm Selection Library (Bischl et al., 2015b, 2015a). We observe that each approach has strengths and weaknesses on different scenarios. The SATzilla'11-like approach (the default of claspfolio 2) performs best overall, but only achieves better performance than the other approaches considered on 8 out of the 13 scenarios, with 3S, aspeed and ISAC yielding better performance in the remaining cases.

We further note that each of the selection approaches used a fixed default parameter configuration and might therefore fall short of its full performance potential. For example, imputation of missing instance features was not used at all in the approaches considered in Figure 1; while its use does not improve performance on some scenarios (e.g., ASP-POTASSCO), it yields improvements on others (e.g., SAT12-RAND, where the SATzilla'11-like approach plus mean imputation outperforms the single best algorithm by a factor of 1.2).

Generally, it is well known that the performance of many machine learning techniques depends on hyper-parameter settings (e.g., in the case of an SVM, the kernel, kernel hyper-parameter and soft margin; cf. Bergstra, Bardenet, Bengio, & Kégl, 2011; Snoek, Larochelle, & Adams, 2012; Thornton, Hutter, Hoos, & Leyton-Brown, 2013). However, the hyper-parameters of the machine learning models used in Figure 1 were fixed manually, based on limited experiments. Therefore, the performance of some of the algorithm selection systems we considered could likely be improved by using more carefully chosen hyper-parameter settings.

Facing a new algorithm selection problem, we thus have to answer three salient questions: (i) which selection approach to use; (ii) how to set the parameters of the selection approach (and its underlying machine learning model) effectively; and (iii) how to make

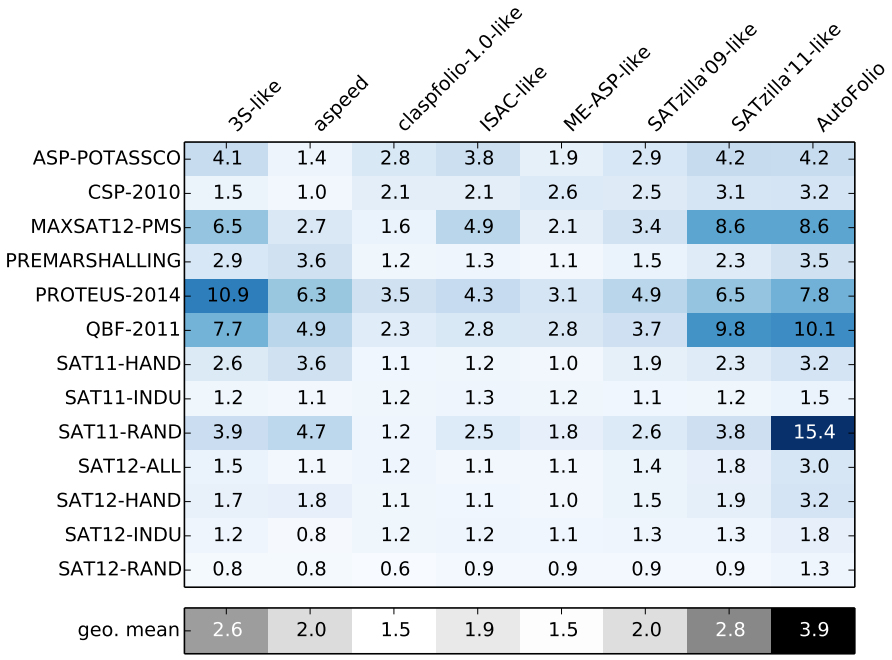


Figure 1: Factors by which the selection approach re-implemented in claspfolio 2 outperformed the single best algorithm in the 13 ASlib scenarios w.r.t. penalized average runtime (PAR10, which counts each timeout as 10 times the given runtime cutoff). These results are for 10-fold cross-validation, ignoring test instances that were not solved by any solver. The last row shows the geometric mean over all 13 scenarios.

best use of techniques augmenting pure AS, such as pre-solving schedules (Xu et al., 2008; Kadioglu et al., 2011). Instead of the common, manual trial-and-error approach, we propose to automatically answer these questions by using automated algorithm configuration methods (Hutter, Hoos, Leyton-Brown, & Stützle, 2009) to configure flexible AS frameworks. While the manual approach is error-prone, potentially biased and requires substantial human expert time and knowledge, the approach we introduce here is fully automatic, unbiased, and leverages the full power of a broad range of AS methods. It thus facilitates an easier and more effective use of algorithm selection and makes AS techniques accessible to a broader community.

Specifically, we present AutoFolio, a general approach for automatically determining a strong algorithm selection method for a particular dataset, by using algorithm configuration to search through a flexible design space of algorithm selection methods. We also provide an open-source implementation of AutoFolio (`www.ml4aad.org/autofolio/`) based on the algorithm configurator SMAC (Hutter, Hoos, & Leyton-Brown, 2011) and the algorithm selection framework claspfolio 2 (Hoos et al., 2014). The last column of Figure 1 previews the results obtained with AutoFolio and clearly shows significant improvements over claspfolio 2 on 10 of the 13 scenarios in ASlib.

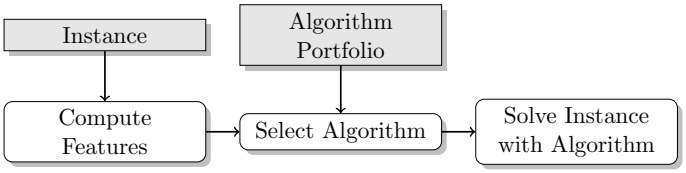


Figure 2: General outline of algorithm selection.

2. Background: Algorithm Configuration And Selection

In this section, we briefly introduce standard approaches to algorithm selection and algorithm configuration that form the basis of our AutoFolio approach.

2.1 Algorithm Selection

Figure 2 shows the general outline of algorithm selection (Rice, 1976). For a given problem instance, we first compute cheap instance features; these are numerical characteristics, including simple ones (such as the number of variables or clauses in a SAT instance) and more complex ones (such as statistics gathered from short probing runs of an actual SAT solver on the given instance). Based on these features, an appropriate algorithm from an algorithm portfolio (Huberman, Lukose, & Hogg, 1997; Gomes & Selman, 2001) is selected to solve the given instance. The overall workflow is subject to a runtime cutoff.

One major challenge in algorithm selection is to find a good mapping from instance features to algorithms. In the general offline algorithm selection approach we consider, this is done based on training data. Specifically, given a portfolio of algorithms A and a set of problem instances I, we use as training data a performance matrix of size $\#I \cdot \#A$ and a feature matrix containing a fixed-size feature vector for each $i \in I$. Based on this training data, we learn a mapping from instance features to algorithms using machine learning techniques, such as k-NN (Maratea et al., 2014), g-means (Kadioglu et al., 2010) or random forests (Xu et al., 2011).

2.1.1 Related Work On Algorithm Selection Systems

Recent successful algorithm selection systems include SATzilla (Xu et al., 2008; Xu, Hutter, Hoos, & Leyton-Brown, 2012a), 3S (Kadioglu et al., 2011; Malitsky, Sabharwal, Samulowitz, & Sellmann, 2012, 2013b), ISAC (Kadioglu et al., 2010; Ansótegui et al., 2014), CSHC (Malitsky, Sabharwal, Samulowitz, & Sellmann, 2013a) and claspfolio 1 (Gebser et al., 2011b). In recent years, these systems showed excellent performance in competitions for SAT, MAXSAT and ASP. We briefly review them in the following.

The original version of the pioneering algorithm selection system SATzilla (Xu et al., 2008) learned the mapping from instance features to algorithms by training ridge regression models. Each regression model predicts the performance of an algorithm for a given instance. Based on these predicted performances, SATzilla selects the algorithm with the best predicted performance. SATzilla's latest version (Xu et al., 2011) uses classification models that, for each pair of algorithms, predict the better-performing one, and selects the algorithm to be run using simple voting over the predictions thus obtained. These models are also cost-sensitive, that is, each training instance in the pairwise classification models is

weighted by the performance loss incurred when selecting the worse of the two algorithms. Furthermore, SATzilla introduced the concept of pre-solving schedules, that is, a short instance-independent schedule of algorithms running for a limited amount of time. If one algorithm of the pre-solving schedule solves the given instance, SATzilla can immediately terminate successfully, saving the time required to compute instance features. Furthermore, pre-solving schedules increase the robustness of algorithm selectors by not only relying on one selected algorithm but also on the pre-solvers to solve a given instance. One drawback of SATzilla is its use of grid search over all possible pre-solving schedules with up to three pre-solvers; for each schedule considered, SATzilla performs algorithm subset selection and trains the classification models, which can require substantial amounts of time (in our experiments, up to 4 CPU days).

3S (Kadioglu et al., 2011; Malitsky et al., 2012, 2013b) uses a k-nearest neighbour approach to select an algorithm. For a given problem instance to be solved, it determines a set of similar training instances in the instance feature space and selects the algorithm with the best performance on this instance set. The performance of this k-NN approach is further improved by distance-based weighting (that is, weighting algorithm performance on an instance by the instance's distance to the new given instance) and using a clustering-based adaptive neighbourhood size (to adjust the size of the neighbourhood in different areas of the feature space). Furthermore, 3S uses mixed integer programming to compute pre-solving schedules more efficiently than SATzilla.

ISAC (Kadioglu et al., 2010) clusters instances in the instance feature space using the g-means algorithm and stores the cluster centre as well as the best-performing algorithm for each cluster. For each new problem instance, it then determines the nearest cluster centre (1-NN) and selects the algorithm associated with it.

The cost-sensitive hierarchical clustering system CSHC (Malitsky et al., 2013a) also partitions the feature space into clusters, but instead of ISAC's unsupervised clustering approach, it creates this partitioning in a supervised top-down fashion, much like a decision or regression tree algorithm. Starting with all instances (the entire feature space) at the root of a tree, it recursively splits the instances associated with a node into two child nodes, choosing each split along a single feature value, such that the performance of the best-performing algorithm in each child node is optimized. This cost-sensitive supervised approach based on trees closely resembles the cost-sensitive random forests in SATzilla, with the difference that, in contrast to SATzilla's pairwise voting approach, it only builds a single model.

Last but not least, claspfolio 1 (Gebser et al., 2011b) is the predecessor of claspfolio 2, which we use here (and describe in Section 2.1.2). In contrast to the flexible framework of claspfolio 2, claspfolio 1 was inspired by the earlier version of SATzilla and uses the same regression approach, but with a different machine learning method (support vector regression instead of ridge regression).

Further systems for algorithm selection combine and extend these techniques, for example, by combining regression and clustering approaches (Collautti et al., 2013), or by selecting algorithm portfolios (Yun & Epstein, 2012; Lindauer, Hoos, & Hutter, 2015a) or schedules (Amadini, Gabbrielli, & Mauro, 2014) instead of a single algorithm. For additional information, we refer the interested reader to two recent surveys on algorithm selection (Smith-Miles, 2008; Kotthoff, 2014).

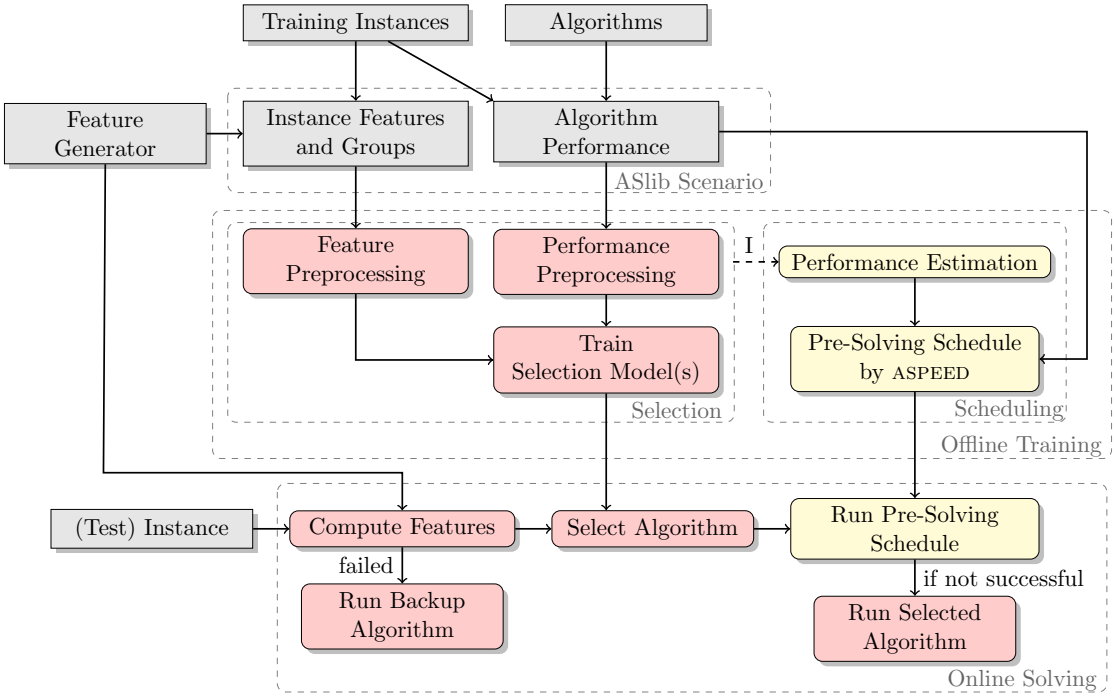


Figure 3: General workflow of claspfolio 2. Objects such as algorithms and instances are shown as rectangles, and activities are depicted as rectangles with rounded corners. Activities related to algorithm selection are shown in red and activities related to algorithm schedules in yellow.

2.1.2 The Algorithm Selection Framework claspfolio 2

We now explain the algorithm selection framework claspfolio 2 (Hoos et al., 2014; Lindauer, Hoos, & Schaub, 2015c) in some more detail, since it provides the basis for the concrete implementation of our general AutoFolio approach, as used in our experiments.

The claspfolio 2 framework implements the idea of algorithm selection in a flexible and general way. It provides a general view on the individual components of algorithm selectors, based on which it implements many different selection approaches and associated techniques. Therefore, claspfolio 2 is a natural candidate to serve as a basis for our AutoFolio approach.

Figure 3 shows the workflow of claspfolio 2, which is divided into an *ASlib Scenario* as input of claspfolio 2; *Offline Training* of *Selection* and *Scheduling*; and *Online Solving* a new instance:

ASlib scenario. As an input, claspfolio 2 reads an algorithm selection scenario, supporting the format of the Algorithm Selection library, ASlib. This consists of a performance matrix, instance features, groups of instance features[1] and some optional information, such as cross-validation splits or ground truth about the problem

1. We note that, according to the definition of ASlib, each feature group enables a list of instance features that are computed with a common block of feature computation code, and jointly incur the cost for running this code.

instances (for example, whether a SAT instance is satisfiable or unsatisfiable). For a full specification of the ASlib format, we refer the interested reader to `aslib.net`.

Offline training – selection. Based on the given scenario (training) data, claspfolio 2 pre-processes the instance features (for example, normalization or feature imputation) and performance data (for example, log-transformation). Using machine learning techniques, claspfolio 2 learns a selection model that maps instance features to algorithms.

Offline training – scheduling. To compute an efficient pre-solving schedule, claspfolio 2 first estimates the performance of the *Selection* module by using an internal cross-validation on the training data (Arrow I). Based on this performance estimation, claspfolio 2 computes a timeout-minimal pre-solving schedule using Answer Set Programming in aspeed (Hoos et al., 2015), assigning each algorithm a (potentially zero-length) time slice of the overall runtime budget. The estimation of the *Selection* module is necessary to compute the runtime budget for the pre-solving schedule. If the *Selection* module performs well, the pre-solving schedule may be empty, because the pre-solving schedule cannot perform better than a perfect predictor (that is, a predictor that always selects the best solver). In contrast, if the prediction performs very poorly (for example, as a result of non-informative instance features), the pre-solving schedule may be allocated the complete time budget, with the *Selection* module being ignored.

Online solving. The *Solving* workflow is as follows: a *feature generator* computes the instance features of a new problem instance to be solved; if this computation fails (for example, because of time or memory constraints) and no feature imputation strategy is selected, a *backup solver* – i.e., the single best performing solver in the offline training – is run on the instance; otherwise, the previously trained selection model uses the instance features to select an algorithm expected to perform well. If a pre-solving schedule is available, the schedule runs either before instance feature computation or after the selection of the algorithm, depending on a parameter setting of claspfolio 2 — this latter version being shown in Figure 3. The former has the advantage that the time to compute instance features can be saved if the instance is solved during pre-solving. The latter has the advantage that the algorithm chosen by the selector can be removed from the pre-solving schedule to prevent running it twice.

A list of all techniques we implemented for these modules is given in Section 3.2.

2.2 Algorithm Configuration

Figure 4 shows a general outline for algorithm configuration methods. Given a parameterized algorithm A with possible parameter settings $\mathbf{C}$, a set of training problem instances I, and a performance metric $m : \mathbf{C} \times I \to \mathbb{R}$, the objective in the algorithm configuration problem is to find a parameter configuration $\mathbf{c} \in \mathbf{C}$ that minimizes m across the instances in I. Prominent examples for the performance metric to be optimized are the runtime, solution quality, or misclassification cost the target algorithm achieves. The configuration

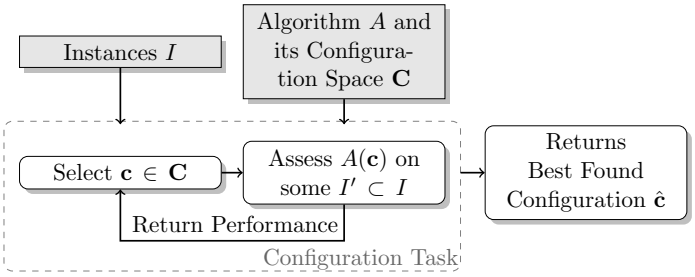


Figure 4: General outline of algorithm configuration.

procedure (or short *configurator*) iteratively evaluates the performance of parameter configurations $\mathbf{c} \in \mathbf{C}$ (by running A with them on one or more instances in I) and uses the result to decide about the next configurations to evaluate. After a given budget for the configuration process has been exhausted, the configurator returns the best known parameter configuration it found until then.

When A has n parameters $p_1, \ldots, p_n$, with respective domains $D_1, \ldots, D_n$, the *parameter configuration space* $\mathbf{C} = D_1 \times \cdots \times D_n$ is the cross-product of these domains, and each parameter configuration $\mathbf{c} \in \mathbf{C}$ assigns a value to each parameter. There are several types of parameters, including real-valued, integer-valued and categorical ones (which have a finite, unordered domain; for example, a choice between different machine learning algorithms). Furthermore, configuration spaces can be structured; specifically, a parameter p_i can be *conditional* on another parameter p_j, such that the value of p_i is only relevant if the parent parameter p_j is set to a specific value. For example, this is the case when p_j is a categorical choice between machine learning algorithms, and p_i is a sub-parameter of one of these algorithms; p_i will only be active if p_j chooses the algorithm it parameterizes further.

To date, there are four general configuration procedures: ParamILS (Hutter et al., 2009), GGA (Ansótegui, Sellmann, & Tierney, 2009), irace (López-Ibáñez, Dubois-Lacoste, Stützle, & Birattari, 2011), and SMAC (Hutter et al., 2011). In principle, we could use any of these as the configurator in our general AutoFolio approach. In practice, we have found SMAC to often yield better results than ParamILS and GGA (Hutter et al., 2011; Hutter, Lindauer, Balint, Bayless, Hoos, & Leyton-Brown, 2015; Lindauer, Hoos, Hutter, & Schaub, 2015b), and thus use it as the basis for the concrete implementation of AutoFolio discussed in the following. We now describe SMAC in more detail.

2.2.1 SMAC: Sequential Model-Based Algorithm Configuration

The sequential model-based algorithm configuration method SMAC (Hutter et al., 2011; Hutter, Hoos, & Leyton-Brown, 2015a) uses regression models that approximate the performance metric $m : \mathbf{C} \times I \to \mathbb{R}$ (Hutter, Xu, Hoos, & Leyton-Brown, 2014). It follows the general algorithm configuration workflow from above, alternating evaluations of m for some parameter configurations and instances with decision phases, in which the configurator uses the data gathered so far to select which configurations to evaluate next on which instances. SMAC's decision phases involve constructing a regression model $\hat{m} : \mathbf{C} \times I \to \mathbb{R}$ based on the data observed so far, and then using this model (as well as the model's uncertainty in its predictions) to select promising configurations to try next. This step automatically

trades off exploration (evaluating in regions of the configuration space where the model $\hat{m}$ is very uncertain) and exploitation (evaluating configurations predicted to perform well).

In order to save time in evaluating new configurations $\mathbf{c}_{new} \in \mathbf{C}$, SMAC first evaluates them on a single instance $i \in I$; additional evaluations are only carried out (using a doubling schedule) if, based on the evaluations to date, $\mathbf{c}_{new}$ appears to outperform SMAC's best known configuration $\hat{\mathbf{c}}$. Once it has evaluated the same number of runs for $\mathbf{c}_{new}$ as for $\hat{\mathbf{c}}$, if $\mathbf{c}_{new}$ still performs better, SMAC updates its best known configuration $\hat{\mathbf{c}}$ to $\mathbf{c}_{new}$.

2.2.2 Previous Applications Of Algorithm Configuration

Algorithm configuration has been demonstrated to be very effective in optimizing algorithms for a wide range of problems, including SAT-based formal verification (Hutter, Babić, Hoos, & Hu, 2007), timetabling (Chiarandini, Fawcett, & Hoos, 2008), multi-objective optimization (López-Ibáñez & Stützle, 2010), mixed integer programming (Hutter, Hoos, & Leyton-Brown, 2010), AI planning (Vallati, Fawcett, Gerevini, Hoos, & Saetti, 2013), generation of heuristics (Mascia, López-Ibáñez, Dubois-Lacoste, & Stützle, 2014), occupancy scheduling (Lim, van den Briel, Thiébaux, Backhaus, & Bent, 2015) and kidney exchange matching (Dickerson & Sandholm, 2015). An important special case of algorithm configuration is hyperparameter optimization in machine learning (Bergstra et al., 2011; Snoek et al., 2012; Eggensperger et al., 2013).

The previous line of work most related to our application of configuration to algorithm selection is Auto-WEKA (Thornton et al., 2013). Auto-WEKA addresses the combined problem of selecting a machine learning algorithm from the WEKA framework (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten, 2009) and optimizing its hyperparameters. AutoFolio also needs to solve this combined algorithm selection and hyperparameter optimization problem, and in particular needs to do so for each of the problem formulations it considers: regression, classification and clustering. Further important design choices in AutoFolio are pre-solving and its parameters, as well as which instance features to use.

AutoFolio applies one meta-solving strategy (algorithm configuration) to another one (algorithm selection). A previous application of a meta-solving strategy to itself was the self-configuration of ParamILS (Hutter et al., 2009). However, in that case, self-configuration only yielded a modest improvement over the default configuration of ParamILS, whereas here, we achieve substantial improvements over the default configuration of claspfolio 2.

Algorithm configuration and algorithm selection have previously been combined in a different way, by using configuration to find good parameter settings of a highly parameterized algorithm, and then using selection to choose between these on a per-instance basis. Two systems implement this approach to date: ISAC (Kadioglu et al., 2010) and Hydra (Xu, Hoos, & Leyton-Brown, 2010). ISAC first clusters training problem instances into homogeneous subsets, uses a configurator to find a good solver parameterization for each cluster, and then uses a selector to choose between these parameterizations. Hydra iteratively adds new solver parameterizations to an initially empty portfolio-based selector, at each step tasking a configurator to find the solver parameterization that most improves the current portfolio.

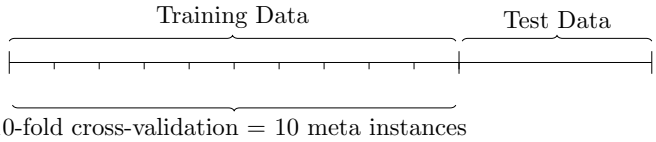


Figure 5: Split of instance sets in algorithm selection scenarios; cross-validation is performed inside the configuration process, the test set is withheld for evaluating the configured selector.

3. Configuration Of Algorithm Selectors

We now present our AutoFolio approach of using algorithm configurators to automatically customize flexible algorithm selection (AS) frameworks to specific AS scenarios. To apply algorithm configuration in this context, we need to specify a parameterized selector and its configuration space, as well as the performance metric by which we judge its performance.

3.1 Formal Problem Statement

To judge the performance of an algorithm selection (AS) system on an AS scenario, it is crucial to partition the given set of problem instances into a training and a test set, use the AS system only on the training set to train a selector s, and evaluate s only on the test set instances. (If the training set was instead used to evaluate performance, a perfect AS system could simply memorize the best solver for each instance without learning anything useful for new problem instances). This is the standard notion of a training-test split from machine learning.

An AS scenario includes algorithms A, problem instances I, performance and feature data $\mathbf{D}$, and a loss function $l : A \times I \to \mathbb{R}$ to be minimized (for example, the algorithms' runtime or solution cost), with the data split into disjoint sets $\mathbf{D}_{train}$ and $\mathbf{D}_{test}$. Let $S(\mathbf{D}_{train}) : I \to A$ denote the selector learned by the AS system S when trained on the data in $\mathbf{D}_{train}$. Then, the performance of S, $P(S)$ is the average performance of the algorithms it selects on the instances in the test data set $\mathbf{D}_{test}$:

$$P(S) = \frac{1}{|\mathbf{D}_{test}|} \cdot \sum_{i \in \mathbf{D}_{test}} l(S(\mathbf{D}_{train}), i). \tag{1}$$

Likewise, we can evaluate the performance of an AS system S_c parameterized by a configuration c as $P(S_c)$. However, we can *not* perform algorithm configuration by simply minimizing $P(S_c)$ with respect to $c \in \mathbf{C}$: this would amount to peeking at the test set many times, and even though it would yield a configuration c^* with low $P(S_{c^*})$, it could not be expected to perform well on instances not contained in D_{test}. Instead, in order to obtain an unbiased evaluation of the configured selector's performance in the end, we need to hold back a test set of instances that is not touched during the configuration process. In order to still be able to optimize parameters without access to that test set, the standard solution in machine learning is to partition the training set further, into k cross-validation folds. Overall, we use the instance set for each selection scenario as illustrated in Figure 5: (i) we split the full set of instances into a training and a test set and (ii) the training data

Algorithm 1: AutoFolio: Automated configuration of an algorithm selector

Input : algorithm configurator AC, algorithm selector S, configuration space $\mathbf{C}$ of S, training data of algorithm scenario $\mathbf{D}$ (with performance and feature matrix), number of folds k

1 randomly split $\mathbf{D}$ into $\mathbf{D}^{(1)}, \ldots, \mathbf{D}^{(k)}$
2 start AC with $\mathbf{D}^{(1)}, \ldots, \mathbf{D}^{(k)}$ as meta instances, using average loss across meta-instances as performance metric m, and using S as target algorithm with configuration space $\mathbf{C}$
3 **while** *configuration budget remaining* **do**
4 $\quad$ AC selects configuration $\mathbf{c} \in \mathbf{C}$ and meta instance $n \in \{1 \ldots k\}$
5 $\quad$ train S_c on $\mathbf{D} \backslash \mathbf{D}^{(n)}$, assess its loss on $\mathbf{D}^{(n)}$ and return that loss to AC
6 **return** *best configuration* $\mathbf{c}$ *of* S *found by* AC

is further partitioned into k folds (in our experiments, we use $k = 10$), which are used as follows.

Let $\mathbf{D}_{train}^{(1)}, \ldots, \mathbf{D}_{train}^{(k)}$ be a random partition of the training set $\mathbf{D}_{train}$. The cross-validation performance $CV(S_c)$ of S_c on the training set is then:

$$CV(S_c) = \frac{1}{k} \cdot \sum_{j=1}^{k} \left(\frac{1}{|\mathbf{D}_{train}^{(j)}|} \cdot \sum_{i \in \mathbf{D}_{train}^{(j)}} l(S_c(\mathbf{D}_{train} \backslash \mathbf{D}_{train}^{(j)}), i) \right) \tag{2}$$

In the end, we optimize the performance $CV(S_c)$ by determining a configuration $\hat{c} \in \mathbf{C}$ of the selector S with good cross-validation performance

$$\hat{c} \in \arg\min_{c \in \mathbf{C}} CV(S_c), \tag{3}$$

and evaluate $\hat{c}$ by training a selector $S_{\hat{c}}$ with it on the entire training data and evaluating $P(S_{\hat{c}})$ on $\mathbf{D}_{test}$, as defined in Equation 1.

Following Thornton et al. (2013), we use each of the k folds $\mathbf{D}_{train}^{(j)}$ as one instance within the configuration process. In order to avoid confusion between these AS instances and the base-level problem instances (e.g., SAT instances) to be solved inside the AS instance, we refer to the AS instance as a *meta-instance*. We note that many configurators, such as *FocusedILS* (Hutter et al., 2009), irace (López-Ibáñez et al., 2011) and SMAC (Hutter et al., 2011), can discard configurations when they perform poorly on a subset of meta-instances and therefore do not have to evaluate all k cross-validation folds for every configuration. This saves time and lets us evaluate more configurations within the same configuration budget. Based on these considerations, Algorithm 1 outlines the process to configure an algorithm selector with AutoFolio.

Since the instances in an AS scenario could be split into configuration and testing sets in many different ways, one such split does not necessarily yield a representative performance estimate. Therefore, to yield more confident results in our evaluation, we perform an additional *outer* cross-validation (as given by an ASlib scenario) instead of a single training-test

split. That is, we consider multiple training-test splits, configure the selector for each training set, assess its final configurations on the respective test data sets, and average results. We note, however, that in a practical application of AS, one would only have a single training set (which we would still split into k cross-validation splits internally) and a single test set.

3.2 Configuration Space Of Selectors

Most existing algorithm selectors implement one specific algorithm selection approach, using one specific machine learning technique. We note, however, that most selection approaches, at least implicitly, admit more flexibility, and in particular could be used with a range of machine learning techniques. For example, SATzilla'11 (Xu et al., 2011) uses voting on pairwise performance predictions obtained from cost-sensitive random forest classifiers, but, in principle, it could use other cost-sensitive binary classifiers instead of random forests.

Based on this observation, we consider a hierarchically structured configuration space with a top-level parameter that determines the overall algorithm selection approach — for example, a regression approach, as used in SATzilla'09 (Xu et al., 2008) or a k-NN approach, as used in ME-ASP (Maratea et al., 2014). For most selection approaches, we can then choose between different regression techniques, for example, ridge regression, lasso regression, support vector regression or random forest regression. Each of these machine learning techniques can be configured by its own (hyper-)parameters.

Besides the selection approach, further techniques are used for preprocessing the training data (for example, z-score feature normalization as a feature preprocessing step or log-transformation of runtime data as a performance preprocessing step). Preprocessing techniques can be configured independently from the selection approach, and are therefore also handled by top-level parameters.

We use a third group of parameters to control pre-solving schedules (Kadioglu et al., 2011; Xu et al., 2011), including parameters that determine the time budget for pre-solving and the number of pre-solvers considered. Pre-solving techniques can be freely combined with selection approaches; because they are not always needed, we added a top-level binary parameter that completely activates or deactivates the use of pre-solvers; all other pre-solving parameters are conditional on this switch.

We implemented these choices in the claspfolio 2 system described in Section 2.1.2. Figure 6 illustrates the complete configuration space thus obtained. Our current version, which we use for the concrete implementation of our AutoFolio approach, covers six different algorithm selection approaches:

(hierarchical) regression (inspired by SATzilla'09; Xu et al., 2008) learns a regression model for each algorithm; for a new instace, it then selects the algorithm with best predicted performance;

multiclass classification (inspired by LLAMA; Kotthoff, 2013) learns a classification model that directly selects an algorithm based on the features of a new instance;

pairwise classification (inspired by SATzilla'11; Xu et al., 2011) learns a (cost-sensitive) classification model for all pairs of algorithms; for a new instance, it evaluates all models and selects the algorithm with the most votes;

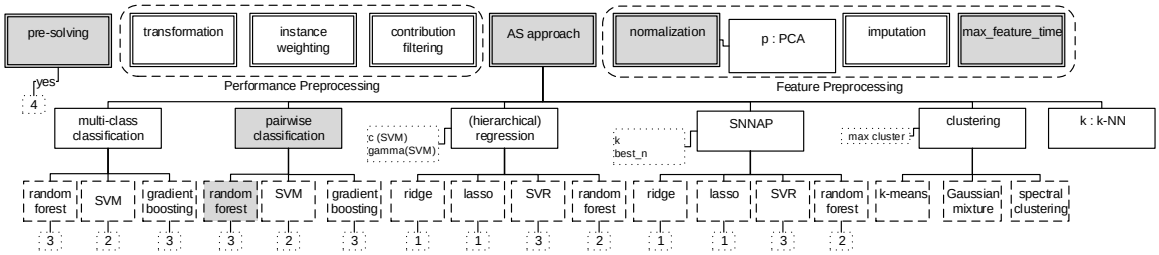


Figure 6: Configuration space of claspfolio 2, including 22 categorial parameters, 15 integer valued parameters and 17 continous parameters. Parameters in double boxes are top-level parameters; single boxes represent algorithm selection approaches based on classes of machine learning techniques, dashed boxes machine learning techniques and dotted boxes indicate the number of low-level parameters. Parameter boxes used in the default configuration are filled in grey.

clustering (inspired by ISAC; Kadioglu et al., 2010) determines subsets of similar training instances in the feature space and the best algorithm on these subsets; for a new instance, it determines the nearest cluster center and selects the associated algorithm;

k-NN (inspired by 3S; Kadioglu et al., 2011, and ME-ASP; Maratea et al., 2014) determines a set of similar training instances in the feature space for a given new instance and selects the algorithm with the best performance on this instance set;

SNNAP (inspired by Collautti et al., 2013) predicts the performance of each algorithm with regression models and uses this information for a k-NN approach in the predicted performance space.

For each of these approaches, claspfolio 2 covers at least three different machine learning techniques (where appropriate). These are listed in Figure 6; for example, pairwise classification can be based on random forests, SVMs or gradient boosting (with 3, 2 and 3 hyper-parameters, respectively). For preprocessing strategies, it supports:

Performance preprocessing:

transformation applies log (Xu et al., 2008) or z-score normalization (Collautti et al., 2013) to the performance data;

instance weighting weights the instances by their impact on the performance of an algorithm selector, that is, instances get a low weight if all available algorithms perform equally, and high weight if the algorithms differ substantially in performance (Kotthoff, Gent, & Miguel, 2012);

contribution filtering removes algorithms that have less than a specified contribution to the performance of the oracle (also known as virtual best solver) (Xu et al., 2012a); this is a form of algorithm subset selection.

Feature preprocessing:

normalization transforms the instance features with min-max, z-score, decimal-point, log scheme or by application of PCA;

p:PCA applies principal component analysis on the features and selects the top p principal components, where p is a parameter (if PCA was activated);

imputation fills in missing feature values as the median, average or most frequent value of a feature – if imputation is deactivated and a feature vector is incomplete for a given instance, the single best solver is statically selected;

max_feature_time limits the amount of time spent to collect features – this ensures that not too much time is spent on feature computation; however, it can result in incomplete features with missing values (which get imputed if imputation is active).

We chose the default configuration of claspfolio 2 (used to initialize the algorithm configurator) to be a SATzilla'11-like configuration, since this was shown to be effective on SAT (Xu et al., 2012a) and ASP (Hoos et al., 2014), and since its overall high performance is evident from the results in Figure 1. This configuration uses pairwise cost-sensitive random forest classifiers, z-score feature normalization and a pre-solving schedule with at most three pre-solvers. Since we assume no prior knowledge about the algorithm selection scenarios, the default configuration uses the default instance features as defined by the scenario designers.

We chose claspfolio 2 as the basis for AutoFolio, because it has been designed to be flexible and is known to perform well.[2] We note that in principle, other selectors, such as SATzilla (Xu et al., 2008), ISAC (Kadioglu et al., 2010), SNNAP (Collautti et al., 2013) and LLAMA (Kotthoff, 2013), could be generalized in a similar way.

In addition to using claspfolio 2 as its algorithm selection framework, our current version of AutoFolio employs the algorithm configurator SMAC (described in Section 2.2.1). Like the selection framework, this configurator is also exchangeable: while we chose SMAC, because it performed best across the algorithm configuration problems we studied so far, in principle, other configurators could also be used, such as, GGA (Ansótegui et al., 2009) or irace (López-Ibáñez et al., 2011). Preliminary results (Lindauer et al., 2015b) showed that ParamILS can also optimize the performance of claspfolio 2, but was inferior to SMAC in all but one scenario, on which its performance advantage was small.

4. Empirical Performance Analysis

In this section, we empirically analyze the performance of our AutoFolio approach. In these experiments, AutoFolio employs claspfolio 2 using the well-known machine learning package *scikit-learn* (Pedregosa et al., 2011) (version 0.14.1) and the algorithm configurator SMAC (version 2.08.00). We ran AutoFolio on the thirteen algorithm selection scenarios that make up the Algorithm Selection Library 1.0 (Bischl et al., 2015b).[3]

2. Results on the performance of claspfolio 2 compared to other state-of-the-art algorithm selectors can be found at `aslib.net`.
3. We note that for experiments on ASlib scenarios, claspfolio 2 and other algorithm selectors do not need to perform actual runs of algorithms or feature generators, because the ASlib scenarios already

As shown in Table 1, these scenarios comprise a wide variety of hard combinatorial problems; each of them includes the performance data of a range of solvers (between 2 and 31) for a set of instances, and instance features organized in feature groups with associated costs. For all scenarios we consider here, the performance objective is runtime minimization. On a high level, these scenarios comprise the following data:

- ASP-POTASSCO: runtimes of different parameter configurations of the ASP solver clasp on a broad range of ASP instances collected by the Potassco group (Gebser, Kaminski, Kaufmann, Ostrowski, Schaub, & Schneider, 2011a);
- CSP-2010: runtimes of a single solver with two different configurations (with and without lazy learning; Gent, Jefferson, Kotthoff, Miguel, Moore, Nightingale, & Petrie, 2010) on a collection of CSP instances;
- MAXSAT12-PMS: runtime data from the 2012 MaxSAT Evaluation;
- PREMARSHALLING: runtimes of A*-based and IDA*-based solvers for the pre-marshalling problem, on real-world, time-sensitive pre-marshalling problem instances from the operations research literature;
- PROTEUS-2014: runtimes of different CSP and SAT solvers on a range of CSP instances, preprocessed with various CSP-to-SAT translation techniques;
- QBF-2011: runtime data for the QBF solvers from the AQME system (Pulina & Tacchella, 2009) on QBF instances from the 2010 QBF Solver Evaluation;
- SAT11-HAND, SAT11-INDU and SAT11-RAND: runtime data from the respective tracks of the 2011 SAT Competition;
- SAT12-ALL, SAT12-HAND, SAT12-INDU and SAT12-RAND: runtimes of various SAT solvers on a broad range of SAT instances used to train the algorithm selection system SATzilla (Xu, Hutter, Shen, Hoos, & Leyton-Brown, 2012b) for the respective tracks of the 2012 SAT Challenge.

We refer to Bischl et al. (2015b) for further details on these scenarios, including baseline experiments showing that algorithm selection can be applied effectively to all these scenarios. We point out that using this common library allows us to compare AutoFolio in a fair and uniform way against other algorithm selection methods. However, the price we pay for this uniform comparison is that we do not necessarily consider current state-of-the-art algorithms for solving the respective problems, since some of the ASlib data was collected several years ago. Furthermore, we note that the current version of ASlib only consists of deterministic performance data. We expect that future versions will also consider scenarios with stochastic performance data and multiple runs per algorithm and instance, using different pseudo-random number seeds. AutoFolio can be applied to such stochastic scenarios in a straightforward manner, by optimizing mean performance across all runs for the same instance

contain all necessary performance data and feature vectors (in order to allow for a fair comparison of algorithm selectors based on the same data, without confounding factor due to the hardware platform used to run experiments).

Scenario	$\#I$	$\#U$	$\#A$	$\#f$	$\#f_g$	t_c	Reference
ASP-POTASSCO	1294	82	11	138	4	600	(Hoos et al., 2014)
CSP-2010	2024	253	2	86	1	5000	(Gent et al., 2010)
MAXSAT12-PMS	876	129	6	37	1	2100	(Malitsky et al., 2013)
PREMARSHALLING	527	0	4	16	1	3600	(Tierney & Malitsky, 2015)
PROTEUS-2014	4021	428	22	198	4	3600	(Hurley et al., 2014)
QBF-2011	1368	314	5	46	1	3600	(Pulina & Tacchella, 2009)
SAT11-HAND	296	77	15	115	10	5000	(Xu et al., 2008)
SAT11-INDU	300	47	18	115	10	5000	(Xu et al., 2008)
SAT11-RAND	600	108	9	115	10	5000	(Xu et al., 2008)
SAT12-ALL	1614	20	31	115	10	1200	(Xu et al., 2012b)
SAT12-HAND	767	229	31	115	10	1200	(Xu et al., 2012b)
SAT12-INDU	1167	209	31	115	10	1200	(Xu et al., 2012b)
SAT12-RAND	1362	322	31	115	10	1200	(Xu et al., 2012b)

Table 1: Overview of algorithm selection scenarios in the Algorithm Selection Library, showing the number of instances $\#I$, number of unsolvable instances $\#U$ ($U \subset I$), number of algorithms $\#A$, number of features $\#f$, number of feature groups $\#f_g$, cutoff time t_c and literature reference.

4.1 Algorithm Configuration Setup

Following standard practice (Hutter et al., 2009), we performed multiple (in our case, 12) independent runs of the algorithm configurator SMAC for each scenario and then selected the configuration of claspfolio 2 with the best performance on training data. Each configurator run was allocated a total time budget of 2 CPU days. A single run of claspfolio 2 was limited to 1 CPU hour, using the *runsolver* tool (Roussel, 2011). As a performance metric, we used penalized average runtime with penalty factor 10 (PAR10), which counts each timeout as 10 times the given runtime cutoff (runtime cutoffs differ between the ASlib scenarios). We further study how the optimization of PAR10 influenced other metrics, such as the number of timeouts. The time required to evaluate a single configuration of claspfolio 2 varied between 10 CPU seconds and 1 CPU hour on our reference machine (see below), mostly depending on the difficulty of optimizing pre-solving schedules.

To obtain a robust estimate of AutoFolio's performance, we used 10-fold outer cross-validation as given in the specific ASlib scenarios, that is, we configured claspfolio 2 ten times for each scenario (with different training-test splits). Therefore, in total, we performed $12 \cdot 10 = 120$ configuration runs of 2 CPU days each for three different configuration spaces (see Section 4.2) and each of the thirteen ASlib benchmarks, requiring a total of 9 360 CPU days (25 CPU years). We note that although our thorough evaluation of AutoFolio required substantial amounts of computation, applying it to a single benchmark set with a given training-test split would only require 12 independent configuration runs of two days each and could thus be performed over the weekend on a modern desktop machine. Furthermore, applying AutoFolio to a new algorithm selection benchmark set is cheap in comparison to collecting the data for a new benchmark set. For instance, to collect

	categorical	integer	real	conditionals	configurations
AutoFolio$_{\text{VOTE}}$	$18 - 28$	7	3	14	$1 \cdot 10^6 - 6 \cdot 10^8$
AutoFolio	$28 - 38$	15	15	44	$3 \cdot 10^{11} - 2 \cdot 10^{14}$
AutoFolio$_{\text{EXT}}$	$47 - 247$	15	15	44	$2 \cdot 10^{17} - 2 \cdot 10^{77}$

Table 2: Overview of configuration spaces with the number of categorical, integer-valued and real-valued parameters, the number of conditionals, and an estimation of the number of configurations by ignoring the real-valued parameters. The number of categorical values varies between the scenarios depending on the number of algorithms, features and feature groups.

the algorithm performance data for the ASlib scenarios required between 25.7 CPU days (ASP-POTASSCO) and 596.7 CPU days (PROTEUS-2014), with an average of 212.3 CPU days (9 times as much as our configuration budget for AutoFolio).

We performed these experiments on the *bwUniCluster* in Karlsruhe, whose machines are equipped with two Octa-Core Intel Xeon E5-2670 (2.6 GHz, 20 MB cache) CPUs and 64 GB RAM each, running Hat Enterprise Linux 6.4. We note, however, that the runtimes of the selected algorithms and feature computations are part of the ASlib scenarios and do not depend on the hardware we used.

4.2 Different Configuration Spaces

As mentioned earlier, AutoFolio can be used to optimize the performance of single approach algorithm selectors, such as SATzilla, or multi-approach selectors, such as LLAMA or claspfolio 2, with much larger configuration spaces (see Figure 6). Therefore, we studied three different parameter spaces of AutoFolio based on claspfolio 2:

AutoFolio considers the configuration space described in Section 3.2 and additionally adds binary parameters that enable or disable feature groups[4] that are defined by the specific algorithm selection scenario. Algorithm subset selection is done using a heuristic based on the marginal contribution of each algorithm to the oracle performance;

AutoFolio$_{\text{VOTE}}$ considers only a subset of the configuration space of AutoFolio, that fixes the algorithm selection approach to pairwise classification with a voting scheme;

AutoFolio$_{\text{EXT}}$ considers the same configuration space as AutoFolio, but instead of parameters for each feature group, we added binary parameters for each instance feature and for each selectable algorithm. This increases the number of parameters substantially – for example, it adds 220 additional parameters for the PROTEUS-2014 scenario.

4. If the selected feature groups result in an empty feature set, claspfolio 2 will statically select the single best algorithm on training data.

Scenario	Default		AutoFolio$_{\text{vote}}$		AutoFolio		AutoFolio$_{\text{ext}}$	
	PAR10	#TOs	PAR10	#TOs	PAR10	#TOs	PAR10	#TOs
ASP-POTASSCO	124.8^{Δ}	19	$\mathbf{119.8}^{\Delta}$	**18**	125.0^{Δ}	19	152.7^{Δ}	25
CSP-2010	384.7^{Δ}	10	$\mathbf{329.7}^{\Delta}$	**8**	355.1^{Δ}	9	358.1^{Δ}	9
MAXSAT12-PMS	264.0	7	$\mathbf{135.7}^{*\dagger\Delta}$	**3**	246.3	7	268.2^{Δ}	8
PREMARSHALLING	2513.8^{Δ}	33	$1953.6^{\dagger\Delta}$	**24**	$2005.1^{\dagger\Delta}$	25	$\mathbf{1922.5}^{\dagger\Delta}$	**24**
PROTEUS-2014	3274.2	321	$\mathbf{1274.0}^{*\dagger\Delta}$	**110**	$1379.2^{*\dagger}$	117	3102.7	280
QBF-2011	1068.4^{Δ}	26	$\mathbf{866.6}^{\dagger\Delta}$	**19**	910.2^{Δ}	21	946.9^{Δ}	22
SAT11-HAND	7093.2^{Δ}	29	5781.4^{Δ}	23	$\mathbf{5552.8}^{\dagger\Delta}$	**22**	8085.8	33
SAT11-INDU	7851.2	37	$6616.5^{\dagger\Delta}$	31	$\mathbf{5932.3}^{*\dagger\Delta}$	**27**	7671.3	36
SAT11-RAND	3684.0	34	$1441.9^{*\dagger}$	12	$\mathbf{967.4}^{*\dagger\Delta}$	**7**	$1301.7^{*\dagger}$	10
SAT12-ALL	2087.0	261	$\mathbf{890.4}^{*\dagger\Delta}$	**102**	$979.1^{*\dagger\Delta}$	115	$1077.0^{*\dagger}$	126
SAT12-HAND	2081.2	86	$\mathbf{1079.5}^{*\dagger\Delta}$	**43**	$1212.3^{*\dagger\Delta}$	49	$1285.5^{*\dagger}$	52
SAT12-INDU	1019.8	69	$\mathbf{682.9}^{*\dagger\Delta}$	**44**	$774.6^{*\dagger\Delta}$	52	990.7	67
SAT12-RAND	708.2	52	$\mathbf{391.6}^{*\dagger\Delta}$	**29**	$440.8^{*\dagger}$	33	$543.1^{*\dagger}$	41

Table 3: Comparing different configuration spaces of AutoFolio based on test performance. The best performance is shown in bold face; $*$ and † indicate performance significantly better than that of the default configuration of claspfolio 2 at significance levels $\alpha = 0.05$ and $\alpha = 0.1$, respectively, according to a one-sided permutation test with 100 000 permutations. Performances values that, according to the permutation test, are not significantly worse (at $\alpha = 0.05$) than the best performance for a given scenario are marked with Δ.

We fixed the selection approach in AutoFolio$_{\text{vote}}$ to pairwise classification with a voting scheme, since SATzilla'11-like was the most promising single approach in our experiments (see, e.g., Figure 1). On the other hand, the extended configuration space, AutoFolio$_{\text{ext}}$, was obtained by adding algorithm subset selection and feature selection to the configuration task. Feature selection is well known to improve many machine learning models, and often only a small subset of instance features is necessary to predict the runtime of algorithms (Hutter, Hoos, & Leyton-Brown, 2013).

We note that each configuration in AutoFolio$_{\text{vote}}$ can also be found in AutoFolio, and each configuration of AutoFolio is also part of AutoFolio$_{\text{ext}}$, that is, AutoFolio$_{\text{vote}}$ $\subset$ AutoFolio $\subset$ AutoFolio$_{\text{ext}}$. Table 2 gives an overview of the configuration space sizes.

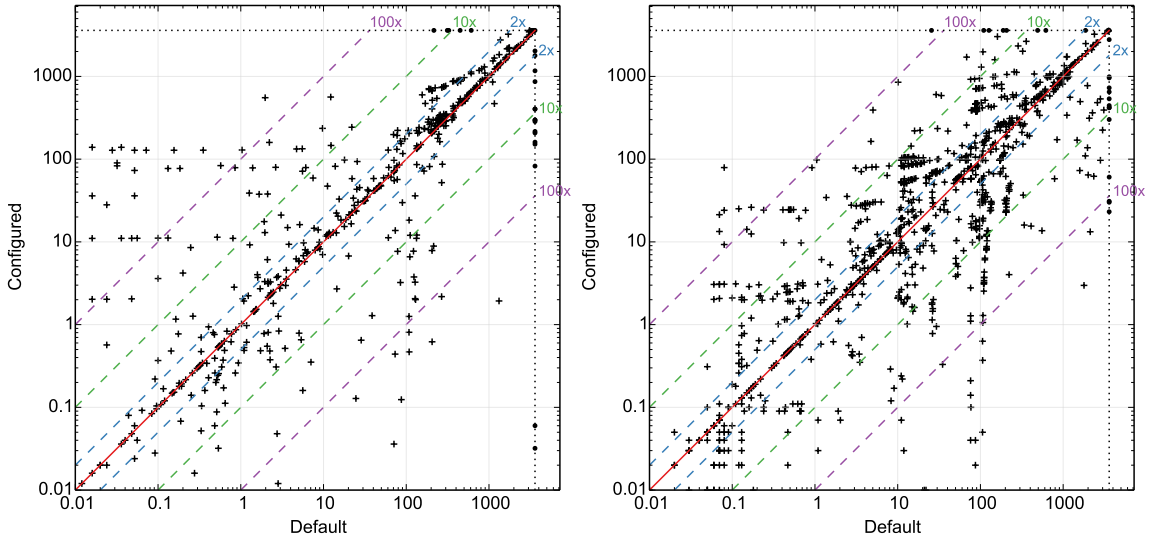
4.3 Analysis Of Configuration Process

In Table 3, we compare the performance of the default configuration of claspfolio 2 (namely, SATzilla'11-like) with that of the configurations optimized by AutoFolio$_{\text{vote}}$, AutoFolio and AutoFolio$_{\text{ext}}$. For all selection scenarios, AutoFolio$_{\text{vote}}$ improved performance on test data in comparison to the default configuration of claspfolio 2. AutoFolio improved on all but one scenario and AutoFolio$_{\text{ext}}$ on all but three scenarios. Performance improvements on test data were statistically significant at $\alpha = 0.1$ and $\alpha = 0.05$ for ten and seven scenarios for AutoFolio$_{\text{vote}}$, for nine and seven for AutoFolio, and five and four for AutoFolio$_{\text{ext}}$, respectively, according to a one-sided permutation test with 100 000 permutations.

On 11 of the 13 ASlib scenarios, configuration in at least one of the configuration spaces we considered led to statistically significant improvements ($\alpha = 0.1$); we now discuss the remaining two scenarios, ASP-POTASSCO and CSP-2010. On ASP-POTASSCO, performance improved substantially on the training data (AutoFolio reduced the PAR10 score by $\approx 30\%$), but this did not transfer to test data (with none of the differences between test performances being statistically significant). We note that the default configuration of claspfolio 2 was manually optimized on this scenario (Hoos et al., 2014), and that AutoFolio found very similar configurations with very similar performance. On CSP-2010, all AutoFolio variants improved over the default, but only insignificantly so. We note that it is hard to improve performance substantially on this benchmark, which only contains two algorithms.

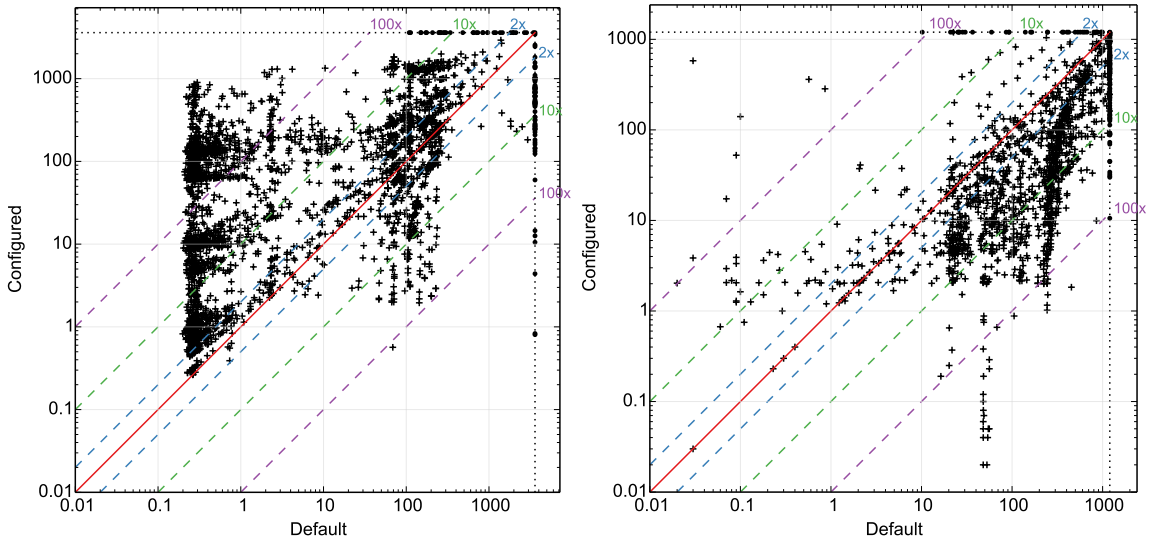
On PREMARSHALLING, AutoFolio solved 8 additional problem instances and reduced PAR10 by more than 25%; nevertheless, this performance difference was only weakly significant (at $\alpha = 0.1$). This is due to the strong constraints on the pre-solving schedule in the default configuration of claspfolio 2 (at most 3 solvers for at most 256 seconds). While more extensive pre-solving schedules decreased the number of timeouts on PREMARSHALLING, they also introduced overhead on many of the other instances in this scenario, making it harder for AutoFolio to achieve more significant performance improvements. The scatter plot of Figure 7a shows that AutoFolio produced fewer timeouts than default claspfolio 2, but AutoFolio required higher runtime on some other instances (points above the diagonal). Similarly, AutoFolio solved a lot more instances on PROTEUS-2014 and some more on QBF-2011, but AutoFolio had a higher runtime on some other instances (see Figure 7c and 7b). However, the number of timeouts improved so much on PROTEUS-2014 (from 321 to 117) that the performance improvement was statistically significant here. Finally, SAT12-ALL is an example of a more clear-cut case: AutoFolio improved the performance of claspfolio 2 on most instances and also substantially reduced the number of timeouts (see Figure 7d).

Overall, AutoFolio$_{\text{VOTE}}$ performed best in these experiments, followed by AutoFolio, and with some distance, AutoFolio$_{\text{EXT}}$. With respect to statistical significance, AutoFolio$_{\text{VOTE}}$ and AutoFolio performed quite similarly, the former being better three times and the latter being better once. Based on the results, we suspect that the added flexibility in AutoFolio as compared to AutoFolio$_{\text{VOTE}}$ pays off when the configuration budget is large enough to evaluate enough configurations to effectively search its larger space. This was the case for the three *SAT11* scenarios, for which AutoFolio reached the best performance: these scenarios only contain relatively few problem instances, making each evaluation of claspfolio 2 quite fast and allowing SMAC to evaluate about 40 000 configurations within 2 days. In contrast, an evaluation of a configuration on the largest ASlib scenario, PROTEUS-2014, can cost up to an hour, and SMAC evaluated only about 600 configurations, which was not enough to explore the design space of AutoFolio; accordingly, the performance of AutoFolio$_{\text{EXT}}$ on PROTEUS-2014 improved only slightly in comparison to the default configuration, while AutoFolio$_{\text{VOTE}}$ made progress faster and performed statistically significantly better than AutoFolio. Therefore, we believe that AutoFolio is a good choice when we can evaluate many configurations, be it because the scenario is small or because a large configuration budget is available. On the other hand,



(a) PREMARSHALLING. Number of timeouts reduced from 33 (default) to 25 (configured).

(b) QBF-2011. Number of timeouts reduced from 26 (default) to 21 (configured).



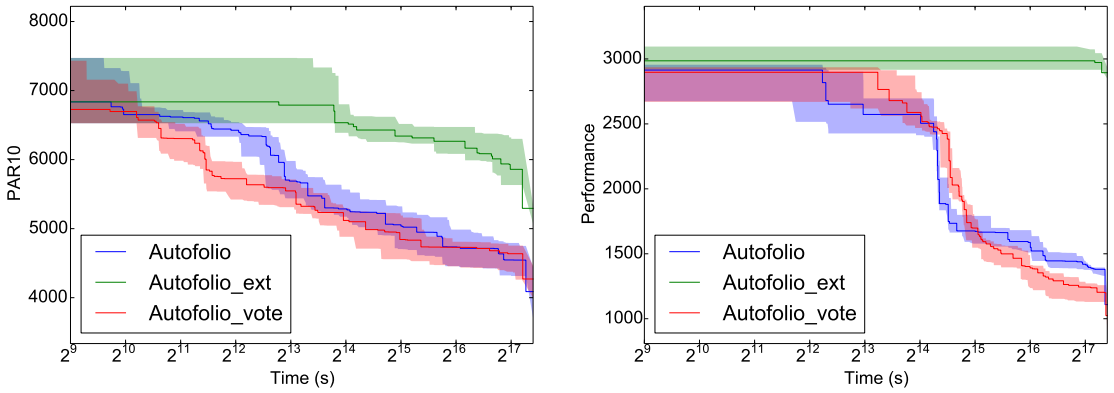
(c) PROTEUS-2014. Number of timeouts reduced from 321 (default) to 117 (configured).

(d) SAT12-ALL. Number of timeouts reduced from 261 (default) to 115 (configured).

Figure 7: Scatter plots comparing the per-instance performance of default CLASPFOLIO 2 (SATzilla'11-like) and AutoFolio. Left: On PREMARSHALLING, AutoFolio improved penalized average runtime (PAR10) by reducing the number of timeouts, at the cost of increased runtimes on many other instances. Right: on SAT12-ALL, AutoFolio improved performance on most instances and also reduced the number of timeouts.

AutoFolio$_{\text{VOTE}}$ should be used on larger scenarios or when the configuration budget is quite small.

Figure 8 shows the progress of the configuration process in terms of training performance as a function of time for SAT11-HAND and PROTEUS-2014, as the scenarios with the



(a) SAT11-HAND (b) PROTEUS-2014

Figure 8: The training PAR10 performance of the best configuration over time. The line shows the median over the 10 folds of the outer cross-validation and the filled area indicates performance between the 25 and 75-quantile.

most and the fewest configuration evaluations performed in the fixed configuration budget. For both scenarios, the very large configuration space of AutoFolio$_{\text{EXT}}$ resulted in a period of stagnation before performance improved. On PROTEUS-2014, the performance started to improve only near the end of the configuration budget. In contrast, AutoFolio and AutoFolio$_{\text{VOTE}}$ performed quite similarly on both scenarios, with AutoFolio$_{\text{VOTE}}$ being somewhat faster to make progress (note the logarithmic time axis). Surprisingly to us, very different selection approaches were chosen for AutoFolio and AutoFolio$_{\text{VOTE}}$. Because of its restricted configuration space, AutoFolio$_{\text{VOTE}}$ had to choose pairwise classification with a voting scheme, but AutoFolio also used other approaches for some outer folds of these scenarios: regression (2 times in each of the two scenarios), clustering (1 and 3 times, resp.) and SNNAP (3 and 4 times, resp.).

From Figure 8, we can also estimate the influence of the configuration budget on the performance of our final algorithm selector. For example, if we halve the configuration time budget to 1 day, the penalized average runtime on the training set only increases by about 8%.

4.4 Which Choices Lead To Good Performance?

To analyze which choices were most important in AutoFolio, we applied two complementary methods for assessing parameter importance in algorithm configuration spaces: functional ANOVA (Hutter, Hoos, & Leyton-Brown, 2014, 2015b) for a global measure of parameter importance and ablation analysis (Fawcett & Hoos, 2015b, 2015a) for a local measure. For a high-level overview of the parameters in AutoFolio, we refer back to Section 3.2; full details, including the default values and ranges of all parameters, are given in an online appendix available at `www.ml4aad.org/autofolio`.

4.4.1 Functional ANOVA (fANOVA)

Functional ANOVA (fANOVA, see, e.g., Sobol, 1993) is a general method for partitioning the variance of a function into components corresponding to subsets of its arguments. Hutter et al. (2014) demonstrated that this technique can be applied to efficiently quantify the importance of an algorithm's parameters. Their approach can re-use the performance data collected during the configuration process for this purpose (without requiring new algorithm executions) and is therefore computationally very efficient (in our experiments, it required minutes). The overall approach is to fit an empirical performance model (Hutter, Xu, Hoos, & Leyton-Brown, 2014) $\hat{m} : \mathbf{C} \times I \to \mathbb{R}$ to the measured performance data, which can be used to predict performance for arbitrary configurations and instances, and to then study parameter importance in that model. After fitting that model, fANOVA marginalizes it across problem instances:

$$\hat{f}(c) = \frac{1}{|I|} \cdot \sum_{i \in I} \hat{m}(c, i). \tag{4}$$

It then computes the variance of the function $\hat{f}$ across the entire configuration space $\mathbf{C}$ and partitions this variance into additive components due to each subset of the algorithm's parameters. Of particular interest are unary subsets, which often explain a substantial part of the variance and tend to be easiest to interpret. It is important to note that fANOVA partitions the variance of $\hat{f}$ over the *entire* configuration space. While this provides a global measure of parameter importance, it takes into account many poorly-performing configurations.

To use fANOVA in the context of our study, for each ASlib scenario, we merged the performance data from 12 independent SMAC runs and removed all data points that reported a timeout[5] or that resulted in an empty feature set. We did the latter, because in this case claspfolio 2 statically selects the single best solver, causing most parameters to become unimportant for the performance of claspfolio 2.

For brevity, we only report results for scenario SAT12-ALL. Table 4 shows the ten most important parameters of AutoFolio and AutoFolio$_{\text{ext}}$ for this scenario. In both configuration spaces, the maximal time spent to compute the instance features (*max-feature-time*) turned out to be the most important parameter. This parameter is so important, because setting it too small will result in too few features (or even none, disabling the selection mechanism) and setting it too large will lead to an increased overhead in feature computation (see Figure 9).

The second most important parameter of AutoFolio was the marginal contribution filtering as a heuristic for algorithm subset selection. Algorithm subset selection is especially important for the AS scenarios based on SAT solving, because they include many SAT solvers and because the performance of these solvers is often highly correlated (Xu et al., 2012a). For AutoFolio$_{\text{ext}}$, the contribution filtering heuristic is less important, because the configuration space includes binary parameters for each individual algorithm, allowing the configurator (here SMAC) to directly perform subset selection. In this context, including mphaseSATm and marchrw is of special importance. The solver mphaseSATm is the single best algorithm on SAT12-ALL and has one of the highest marginal contributions to

5. We only observed timeouts for a particular configuration on the larger data sets: the clustering approach with spectral clustering.

Parameter	Main Effect
max-feature-time	$23.43\% \pm 2.05$
contr-filter	$6.82\% \pm 2.30$
approach	$6.39\% \pm 0.63$
feature-step:CG	$0.76\% \pm 0.09$
pre-solving	$0.69\% \pm 0.09$
impute	$0.29\% \pm 0.06$
perf:transformation	$0.26\% \pm 0.05$
time-pre-solving	$0.22\% \pm 0.06$
feature:normalization	$0.06\% \pm 0.01$
pre-solving:max-solver	$0.05\% \pm 0.01$

(a) AutoFolio

Parameter	Main Effect
max-feature-time	$11.07\% \pm 5.32$
approach	$5.90\% \pm 4.40$
pre-solving	$1.29\% \pm 1.61$
contr-filter	$0.80\% \pm 0.92$
algorithms:mphaseSATm	$0.72\% \pm 0.22$
imputation	$0.69\% \pm 0.27$
F:algorithms:marchrw	$0.30\% \pm 0.18$
time-pre-solving	$0.23\% \pm 0.41$
pre-solving:sec_mode	$0.11\% \pm 0.24$
perf:transformation	$0.11\% \pm 0.04$

(b) AutoFolio$_{\text{EXT}}$

Table 4: Average main effects ($\pm$ stdev) over outer cross-validation splits of the ten most important claspfolio 2 parameters on SAT12-ALL according to fANOVA.

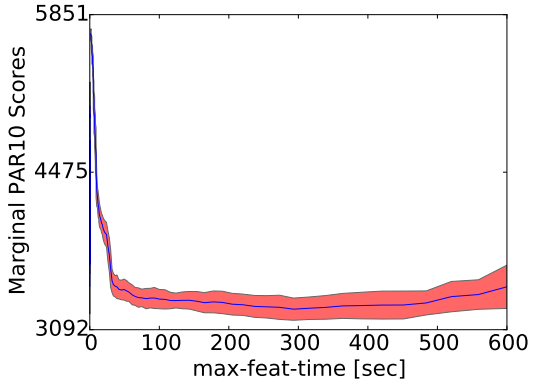
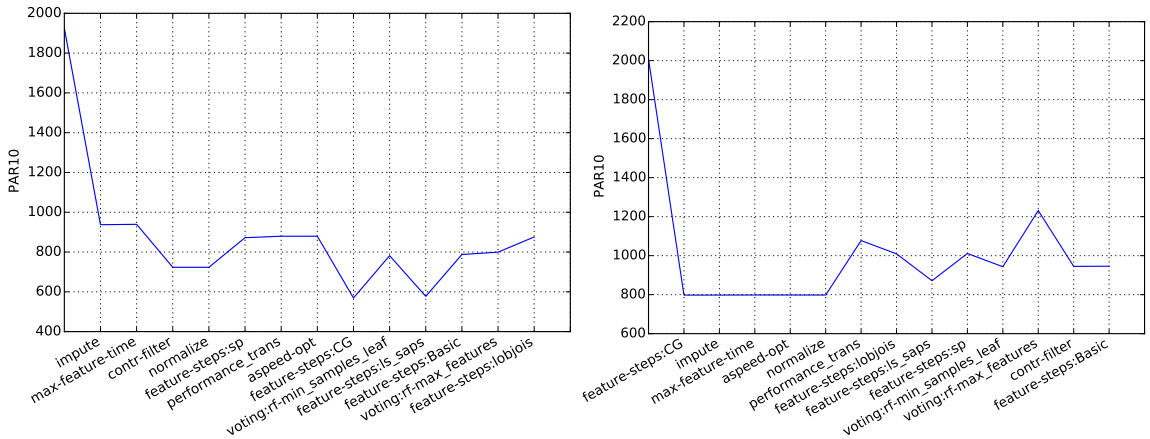


Figure 9: Marginal performance predictions for parameter *max-feature-time* on the data of one outer fold in the configuration space of AutoFolio. The blue line indicates the mean of the predicted marginal performance and the red area its standard deviation.

the oracle. Similarly, marchrw has a high marginal contribution and is the only algorithm whose performance is not highly correlated with that of another solver (see the exploratory data analysis by Bischl et al., 2015b).

We note once again that this analysis determines global parameter importance with respect to the entire parameter space. For example, the importance of the maximal feature computation time is mostly so high, not because it is crucial to change it to improve the performance of claspfolio 2, but because the configuration space contains settings that will drastically *worsen* its performance. To gain complementary insights about which parameters should be changed to improve performance, we next performed ablation analysis.[6]

6. We note that fANOVA can also be used to yield a more local analysis of parameter importance by partitioning the variance of performance in high-performance regions of a given configuration space (Hutter



(a) 2nd outer fold (b) 7th outer fold

Figure 10: Ablation paths on two outer-folds of SAT12-ALL. In (a), the most important parameter is *impute* and *feature-step:CG* has a smaller effect. In (b), *feature-step:CG* is the most important parameter and *impute* has no effect on the performance.

4.4.2 Ablation Analysis

Ablation analysis provides an answer to the question "Which changes in parameter values from one configuration to another caused the biggest improvement in performance?". It does so by iteratively changing the parameter value with the largest impact on performance on a path between two given configurations, e.g., the default configuration of an algorithm and an optimized configuration. Unlike fANOVA, ablation analysis does not attempt to summarize parameter importance across an entire configuration space, but focusses locally on paths between configurations of interest. The results obtained from ablation analysis are therefore complementary to those from fANOVA. Unfortunately, ablation is costly, since it requires new algorithm runs to assess the performance of configurations on the path between the two given configurations. For our AutoFolio experiments on SAT12-ALL, we allocated a time budget of 6 days – the maximum wall-clock time permitted for jobs on our cluster – for ablation on each of our 10 outer cross-validation folds, and within that budget, obtained results for 6 of those.

Our ablation results indicate that *feature-step:CG* – a Boolean parameter that enables or disables the computation of clause graph features – is the single most important parameter to change from claspfolio 2's default. By default, *feature-step:CG* was activated, but it turns out that the clause graph features are often too expensive to compute within the time we allow for feature computation. Therefore, it was indeed a good decision by the configuration procedure to deactivate this optional feature computation step. According to our ablation results, this was done in 5 out of 6 outer cross-validation folds and, on average, on these 5 folds, it was responsible for 99% of the performance improvements achieved

et al., 2014); here, we did not do this, since we used ablation analysis to study parameter importance locally.

by configuration (standard deviation 37%[7]). In contrast, as seen in our fANOVA results, *feature-step:CG* is quite unimportant globally, with a main effect of only 0.76%. The second most important parameter to change was the activation of feature imputation (*impute*); on average, this was responsible for 39% of the overall performance improvement (standard deviation 56%) and was made in all 6 outer cross-validation folds we analyzed.[8] However, *impute* only had an effect on the performance if *feature-step:CG* was not deactivated before *impute* was changed in the ablation path. This was only the case in 2 out of the 6 ablation paths (e.g., see Figure 10a) and hence, *impute* had no impact on performance for the other 4 paths (e.g., see Figure 10b). These two parameters have dependent effects, since imputation is particularly important if clause graph features are computed: these features time out for many large instances and thus require imputation.

The globally most important parameter, according to fANOVA, *max-feature-time*, was found to be rather unimportant to change from its default value. The parameter was changed between the default and optimized configuration in all outer folds of SAT12-ALL, but – since the default value already was very good – on average only 2% of the overall performance improvement could be attributed to this change. We note that along the Ablation path, *max-feature-time* was never flipped to a value that resulted in worse performance than the default configuration, while many such such poorly-performing values exist and explain the globally high importance of this parameter.

4.5 Comparison Against Other Algorithm Selectors

In Table 5, we compare AutoFolio with SATzilla'15[9] (Xu et al., 2011), SNNAP (version 1.4; Collautti et al., 2013) and ISAC (implementation in SNNAP 1.4; Kadioglu et al., 2010).[10] We note that ISAC and SNNAP are pure algorithm selectors, whereas SATzilla'15 and claspfolio 2 can additionally use pre-solver schedules. Furthermore, we added a naïve approach, RandSel, by simulating an uninformed user who selects uniformly at random between SNNAP, ISAC and SATzilla'15. Overall, AutoFolio performed best for 7 out of the 13 scenarios and was statistically indistinguishable from the best system for all other scenarios, according to a one-sided permutation test with 100 000 permutations and significance level $\alpha = 0.05$. Therefore, AutoFolio is the only system that achieves state-of-the-art performance for all scenarios.

SATzilla'15 performed second best, but yielded statistically significantly worse performance than AutoFolio on 5 of the 13 scenarios. Even though not statistically significant, SATzilla'15 performed slightly better than AutoFolio on 5 scenarios. The reason for

7. This large standard deviation arises from the fact that in some folds, the parameter change was actually responsible for *more* than 100% of the performance difference: in those folds, this change alone would have sufficed to achieve better performance than the optimized configuration.
8. The sum of the relative performance of a subset of parameter improvements is not limited to 100%, since it was computed relative to the difference between the default and the optimized configuration. In 5 out of the 6 ablation paths, some parameter changes lead to a better performance than the final optimized configuration, and some parameter changed worsened the performance again.
9. Alexandre Fréchette, the current main developer of SATzilla, provided an internal new implementation of SATzilla (version 0.9.1b-count-cheap-feat-12) that is no longer limited to SAT.
10. Other state-of-the-art selectors, such as 3S (Kadioglu et al., 2011) and CSHC (Malitsky et al., 2013a), are not publicly available with their training procedures, and we were therefore unable to train them on our scenarios.

	Oracle	SB	SNNAP	ISAC	SATzilla'15	RandSel	AutoFolio
ASP-POTASSCO	21.3	534.1	203.8	291.9	170	221.6	**125***
CSP-2010	107.7	1087.4	1087.5	1027	**276***	796.8	355*
MAXSAT12-PMS	40.7	2111.6	895	786.4	**166.8***	615.6	246.3*
PREMARSHALLING	227.6	7002.9	9042.1	5880.8	3179.1	6034	**2005.1***
PROTEUS-2014	26.3	10756.3	4058.7	3328	2050.3	3145.6	**1379.2***
QBF-2011	95.9	9172.3	7386.2	3813.5	1245.2	4148.3	**910***
SAT11-HAND	478.3	17815.8	9209.3	13946.2	6211.5*	9789	**5552.7***
SAT11-INDU	419.9	8985.6	6632.6*	8461.2	8048.8	7714.2	**5932.3***
SAT11-RAND	227.3	14938.6	4859	3140.4	**877.5***	2958.9	967*
SAT12-ALL	93.7	2967.9	1427.5	2989.3	**876.9***	1764.5	979*
SAT12-HAND	113.2	3944.2	2180.5	4110.8	**1031.5***	2440.9	1212*
SAT12-INDU	88.1	1360.6	789*	1409.5	839.7*	1012.7	**774.6***
SAT12-RAND	46.9	568.5	593.1	**434.5***	485.3*	504.3	440*

Table 5: Performance comparison between AutoFolio, SNNAP, ISAC, and SATzilla'15, as well as the single best solver (SB, selected based on PAR10 on the training set) as a baseline, and the oracle (also known as virtual best solver) as a bound on the optimal performance of an algorithm selector. We show PAR10 scores averaged over 10 outer cross-validation folds, with instances not solved by any solver removed from the test set to avoid artificially inflating the PAR10-scores. The RandSel column shows the expected performance by picking uniformly at random one of SNNAP, ISAC and SATzilla'15. The best performance is shown in bold face. All performance values that are not statistically significantly better than the best-performing system for a given scenario, according to a one-sided permutation test with 100 000 permutations and a significance level $\alpha = 0.05$, are marked with $*$.

this might be that SATzilla'15 performs an extensive search to determine the best combination of pre-solving schedule (grid search), algorithm subset (iterated local search) and trained selection model.

We note that, in order to compensate for the 24 CPU days spent to find a well-performing configuration of AutoFolio, compared to simply using the single best solver, on average across all scenarios AutoFolio would have to consecutively solve instances for 42 CPU days (standard deviation 23), less than two times the configuration budget.

Although AutoFolio improved substantially over the single best solver (SB) on all scenarios (up to a speedup factor of 15.4 on SAT11-RAND), there is still a gap to the Oracle performance (also known as virtual best solver in the SAT community). This gap could be closed further in at least two ways: (i) using a larger configuration budget for AutoFolio, or (ii) by developing better instance features, which are the basis for all algorithm selection methods.

5. Conclusions

We presented AutoFolio – to the best of our knowledge, the first approach to automatically configuring algorithm selectors. Using a concrete realization of this approach based on the highly parameterized algorithm selection framework claspfolio 2, we showed that by using state-of-the-art algorithm configurators, algorithm selectors can be customized to

robustly achieve peak performance across a range of algorithm selection scenarios. The resulting approach performs significantly (and sometimes substantially) better than manually configured selectors and can be applied out-of-the-box to previously unseen algorithm selection scenarios.

In comprehensive experiments with the 13 algorithm selection scenarios from different domains (SAT, Max-SAT, CSP, ASP, QBF, and container pre-marshalling) that make up the algorithm selection library ASlib, our concrete realization AutoFolio outperformed the best single solver for each selection benchmark by factors between 1.3 and 15.4 (geometric average: 3.9) in terms of PAR10 scores. Overall, AutoFolio established improved state-of-the-art performance on 7 out of 13 scenarios and performed on par with the previous state-of-the-art approaches on all other scenarios; overall, it clearly yielded the most robust performance across our diverse set of benchmarks.

We also studied the effect of different configuration spaces. Here, we showed that the medium-size configuration space of AutoFolio can lead to state-of-the-art performance if the configuration budget allows the evaluation of sufficiently many configurations. In contrast, if the selection scenario is large (in terms of number of algorithms and problem instances), or if the configuration budget is limited, configuration in a more constrained space, as used in $\text{AutoFolio}_{\text{VOTE}}$, typically leads to better performance.

The performance of $\text{AutoFolio}_{\text{VOTE}}$ was independently verified in the ICON Challenge on Algorithm Selection (Kotthoff, 2015), which evaluated 8 different AS systems with a small configuration budget of 12 CPU hours with respect to three metrics: PAR10 score, number of instances solved and misclassification penalty. As throughout this paper, the metric we optimized in AutoFolio was PAR10 score, and AutoFolio ranked first with respect to this metric. It also ranked first with respect to number of instances solved and second with respect to misclassification penalty (leading to an overall second place).

In future work, we plan to investigate how the potential gains of larger configuration spaces (including feature and algorithm subset selection) can be used more effectively. To this end, we would like to (i) study performance with larger configuration budgets that allow the configurator to assess more configurations; (ii) evaluate other algorithm configurators, such as irace (López-Ibáñez et al., 2011) and GGA (Ansótegui et al., 2009); (iii) extend the configuration space of AutoFolio by implementing more algorithm selection approaches (e.g., CSHC; Malitsky et al., 2013a); (iv) shrink the larger configuration space based on the analysis of parameter importance through fANOVA (Hutter et al., 2014) and Ablation (Fawcett & Hoos, 2015b), allowing the configurator to focus on the most important parameters; and (v) automatically select between pre-configured algorithm selectors, based on features of a given algorithm selection scenario, and further improve performance by starting automatic configuration from the configurations thus selected (Feurer, Springenberg, & Hutter, 2015). Another promising avenue for reducing the computational cost of our approach would be to pre-select algorithms, features, and problem instances based on the techniques proposed by Hoos et al. (2013) or based on the collaborative filtering approach by Misir and Sebag (2013). Finally, we plan to investigate to which extent AutoFolio can configure algorithm selection systems for selecting parallel portfolios (Lindauer et al., 2015a) to exploit the increasing availability of parallel computing resources.

Overall, we believe that the automated configuration of algorithm selection systems improves the performance and versatility of those systems across a broad range of application

domains. Our AutoFolio approach also facilitates future improvements, by making it easier to realize and assess the performance potential inherent in new design choices for the various components of an algorithm selection system. Our open-source implementation of AutoFolio is available at `www.ml4aad.org/autofolio/`.

Acknowledgements

M. Lindauer was supported by the DFG (German Research Foundation) under Emmy Noether grant HU 1900/2-1 and project SCHA 550/8-3, H. Hoos by an NSERC Discovery Grant, F. Hutter by the DFG under Emmy Noether grant HU 1900/2-1 and T. Schaub by the DFG under project SCHA 550/8-3, respectively. This work was performed on the computational resource bwUniCluster funded by the Ministry of Science, Research and Arts and the universities of the State of Baden-Württemberg, Germany, within the framework program bwHPC.

References

Abramé, A., & Habet, D. (2014). On the extension of learning for Max-SAT. In Endriss, U., & Leite, J. (Eds.), *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS'14)*, Vol. 264 of *Frontiers in Artificial Intelligence and Applications*, pp. 1–10. IOS Press.

Amadini, R., Gabbrielli, M., & Mauro, J. (2014). SUNNY: a lazy portfolio approach for constraint solving. *Theory and Practice of Logic Programming*, *14*(4-5), 509–524.

Ansótegui, C., Malitsky, Y., & Sellmann, M. (2014). Maxsat by improved instance-specific algorithm configuration. In Brodley, C., & Stone, P. (Eds.), *Proceedings of the Twenty-eighth National Conference on Artificial Intelligence (AAAI'14)*, pp. 2594–2600. AAAI Press.

Ansótegui, C., Sellmann, M., & Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In Gent, I. (Ed.), *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP'09)*, Vol. 5732 of *Lecture Notes in Computer Science*, pp. 142–157. Springer-Verlag.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., & Weinberger, K. (Eds.), *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NIPS'11)*, pp. 2546–2554.

Biere, A. (2013). Lingeling, plingeling and treengeling entering the sat competition 2013. In Balint, A., Belov, A., Heule, M., & Järvisalo, M. (Eds.), *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, Vol. B-2013-1 of *Department of Computer Science Series of Publications B*, pp. 51–52. University of Helsinki.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Frechétte, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., & Vanschoren, J. (2015a). `www.aslib.net`.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Frechétte, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., & Vanschoren, J. (2015b). Aslib: A benchmark library for algorithm selection. *Computing Research Repository (CoRR)*, *abs/1506.02465*.

Chiarandini, M., Fawcett, C., & Hoos, H. (2008). A modular multiphase heuristic solver for post enrolment course timetabling. In *Proceedings of the Seventh International Conference on the Practice and Theorysy of Automated Timetabling (PATAT'08*, pp. 1–8.

Collautti, M., Malitsky, Y., Mehta, D., & O'Sullivan, B. (2013). SNNAP: Solver-based nearest neighbor for algorithm portfolios. In Blockeel, H., Kersting, K., Nijssen, S., & Zelezný, F. (Eds.), *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'13)*, Vol. 8190 of *Lecture Notes in Computer Science*, pp. 435–450. Springer-Verlag.

Dickerson, J., & Sandholm, T. (2015). Futurematch: Combining human value judgments and machine learning to match in dynamic environments. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15)*, pp. 622–628. AAAI Press.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., & Leyton-Brown, K. (2013). Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*.

Fawcett, C., & Hoos, H. (2015a). `www.cs.ubc.ca/labs/beta/Projects/Ablation/`.

Fawcett, C., & Hoos, H. (2015b). Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 1–28.

Feurer, M., Springenberg, T., & Hutter, F. (2015). Initializing Bayesian hyperparameter optimization via meta-learning. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15)*, pp. 1128–1135. AAAI Press.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., & Schneider, M. (2011a). Potassco: The Potsdam answer set solving collection. *AI Communications*, *24*(2), 107–124.

Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M., & Ziller, S. (2011b). A portfolio solver for answer set programming: Preliminary report. In Delgrande, J., & Faber, W. (Eds.), *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, Vol. 6645 of *Lecture Notes in Computer Science*, pp. 352–357. Springer-Verlag.

Gebser, M., Kaufmann, B., & Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, *187-188*, 52–89.

Gent, I., Jefferson, C., Kotthoff, L., Miguel, I., Moore, N., Nightingale, P., & Petrie, K. (2010). Learning when to use lazy learning in constraint solving. In Coelho, H., Studer, R., & Wooldridge, M. (Eds.), *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI'10)*, pp. 873–878. IOS Press.

Gomes, C., & Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, *126*(1-2), 43–62.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. (2009). The WEKA data mining software: An update. *SIGKDD Explorations*, *11*(1), 10–18.

Hoos, H., Kaminski, R., Lindauer, M., & Schaub, T. (2015). aspeed: Solver scheduling via answer set programming. *Theory and Practice of Logic Programming*, *15*, 117–142.

Hoos, H., Kaufmann, B., Schaub, T., & Schneider, M. (2013). Robust benchmark set selection for boolean constraint solvers. In Pardalos, P., & Nicosia, G. (Eds.), *Proceedings of the Seventh International Conference on Learning and Intelligent Optimization (LION'13)*, Vol. 7997 of *Lecture Notes in Computer Science*, pp. 138–152. Springer-Verlag.

Hoos, H., Lindauer, M., & Schaub, T. (2014). claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming*, *14*, 569–585.

Huberman, B., Lukose, R., & Hogg, T. (1997). An economic approach to hard computational problems. *Science*, *275*, 51–54.

Hurley, B., Kotthoff, L., Malitsky, Y., & O'Sullivan, B. (2014). Proteus: A hierarchical portfolio of solvers and transformations. In Simonis, H. (Ed.), *Proceedings of the Eleventh International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR'14)*, Vol. 8451 of *Lecture Notes in Computer Science*, pp. 301–317. Springer-Verlag.

Hutter, F., Babić, D., Hoos, H., & Hu, A. (2007). Boosting verification by automatic tuning of decision procedures. In O'Conner, L. (Ed.), *Formal Methods in Computer Aided Design (FMCAD'07)*, pp. 27–34. IEEE Computer Society Press.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. In Lodi, A., Milano, M., & Toth, P. (Eds.), *Proceedings of the Seventh International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR'10)*, Vol. 6140 of *Lecture Notes in Computer Science*, pp. 186–202. Springer-Verlag.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In Coello, C. (Ed.), *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, Vol. 6683 of *Lecture Notes in Computer Science*, pp. 507–523. Springer-Verlag.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In Xing, E., & Jebara, T. (Eds.), *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, Vol. 32, pp. 754–762. Omnipress.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2015a). `www.ml4aad.org/smac`.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2015b). `www.ml4aad.org/fanova`.

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

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Identifying key algorithm parameters and instance features using forward selection. In Pardalos, P., & Nicosia, G. (Eds.), *Proceedings of the Seventh International Conference on Learning and Intelligent Optimization (LION'13)*, Vol. 7997 of *Lecture Notes in Computer Science*, pp. 364–381. Springer-Verlag.

Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H., & Leyton-Brown, K. (2015). The Configurable SAT Solver Challenge (CSSC). *Artificial Intelligence*. under review.

Hutter, F., Xu, L., Hoos, H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods and evaluation. *Artificial Intelligence*, *206*, 79–111.

Janota, M., Klieber, W., Marques-Silva, J., & Clarke, E. (2012). Solving QBF with counterexample guided refinement. In Cimatti, A., & Sebastiani, R. (Eds.), *Proceedings of the Fifteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, Vol. 7317 of *Lecture Notes in Computer Science*, pp. 114–128. Springer-Verlag.

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2011). Algorithm selection and scheduling. In Lee, J. (Ed.), *Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP'11)*, Vol. 6876 of *Lecture Notes in Computer Science*, pp. 454–469. Springer-Verlag.

Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). ISAC - instance-specific algorithm configuration. In Coelho, H., Studer, R., & Wooldridge, M. (Eds.), *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI'10)*, pp. 751–756. IOS Press.

Kotthoff, L. (2013). LLAMA: leveraging learning to automatically manage algorithms. *Computing Research Repository (CoRR)*, *abs/1306.1031*.

Kotthoff, L. (2014). Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 48–60.

Kotthoff, L. (2015). ICON Challenge on Algorithm Selection.. `http://challenge.icon-fet.eu/challengeas`.

Kotthoff, L., Gent, I., & Miguel, I. (2012). An evaluation of machine learning in algorithm selection for search problems. *AI Communications*, *25*(3), 257–270.

Lim, B., van den Briel, M., Thiébaux, S., Backhaus, S., & Bent, R. (2015). HVAC-Aware Occupancy Scheduling. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15)*, pp. 679–686. AAAI Press.

Lindauer, M., Hoos, H., & Hutter, F. (2015a). From sequential algorithm selection to parallel portfolio selection. In Dhaenens, C., Jourdan, L., & Marmion, M. (Eds.), *Proceedings of the Nineth International Conference on Learning and Intelligent Optimization (LION'15)*, Lecture Notes in Computer Science, pp. 1–16. Springer-Verlag.

Lindauer, M., Hoos, H., Hutter, F., & Schaub, T. (2015b). Autofolio: Algorithm configuration for algorithm selection. In *Proceedings of the Workshops at Twenty-nineth National Conference on Artificial Intelligence (AAAI'15)*.

Lindauer, M., Hoos, H., & Schaub, T. (2015c). `www.cs.uni-potsdam.de/claspfolio/`.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Tech. rep., IRIDIA, Université Libre de Bruxelles, Belgium.

López-Ibáñez, M., & Stützle, T. (2010). Automatic configuration of multi-objective ACO algorithms. In Dorigo, M., M-Birattari, Caro, G. D., Doursat, R., Engelbrecht, A. P., Floreano, D., Gambardella, L., Groß, R., Sahin, E., Sayama, H., & Stützle, T. (Eds.), *Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS'10)*, Lecture Notes in Computer Science, pp. 95–106. Springer-Verlag.

Malitsky, Y., Mehta, D., & O'Sullivan, B. (2013). Evolving instance specific algorithm configuration. In Helmert, M., & Röger, G. (Eds.), *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SOCS'14)*. AAAI Press.

Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2012). Parallel SAT solver selection and scheduling. In Milano, M. (Ed.), *Proceedings of the Eighteenth International Conference on Principles and Practice of Constraint Programming (CP'12)*, Vol. 7514 of *Lecture Notes in Computer Science*, pp. 512–526. Springer-Verlag.

Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2013a). Algorithm portfolios based on cost-sensitive hierarchical clustering. In Rossi, F. (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, pp. 608–614.

Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2013b). Boosting sequential solver portfolios: Knowledge sharing and accuracy prediction. In Pardalos, P., & Nicosia, G. (Eds.), *Proceedings of the Seventh International Conference on Learning and Intelligent Optimization (LION'13)*, Vol. 7997 of *Lecture Notes in Computer Science*, pp. 153–167. Springer-Verlag.

Maratea, M., Pulina, L., & Ricca, F. (2014). A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming*, *14*, 841–868.

Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., & Stützle, T. (2014). Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & OR*, *51*, 190–199.

Misir, M., & Sebag, M. (2013). Algorithm selection as a collaborative filtering problem. Tech. rep., INRIA & LRI, Universite Paris Sud XI.

O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., & O'Sullivan, B. (2008). Using case-based reasoning in an algorithm portfolio for constraint solving. In Bridge, D., Brown, K., O'Sullivan, B., & Sorensen, H. (Eds.), *Proceedings of the Nineteenth Irish Conference on Artificial Intelligence and Cognitive Science (AICS'08)*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pulina, L., & Tacchella, A. (2009). A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, *14*(1), 80–116.

Rice, J. (1976). The algorithm selection problem. *Advances in Computers*, *15*, 65–118.

Roussel, O. (2011). Controlling a solver execution with the runsolver tool. *Journal on Satisfiability, Boolean Modeling and Computation*, *7*, 139–144.

Smith-Miles, K. (2008). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, *41*(1).

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In Bartlett, P., Pereira, F., Burges, C., Bottou, L., & Weinberger, K. (Eds.), *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NIPS'12)*, pp. 2960–2968.

Sobol, I. (1993). Sensitivity estimates for nonlinear mathematical models. *Mathematical Modeling and Computational Experiment*, *1*(4), 407–414.

Tamura, N., Taga, A., Kitagawa, S., & Banbara, M. (2009). Compiling finite linear CSP into SAT. *Constraints*, *14*(2), 254–272.

Thornton, C., Hutter, F., Hoos, H., & Leyton-Brown, K. (2013). Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In I.Dhillon, Koren, Y., Ghani, R., Senator, T., Bradley, P., Parekh, R., He, J., Grossman, R., & Uthurusamy, R. (Eds.), *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pp. 847–855. ACM Press.

Tierney, K., & Malitsky, Y. (2015). An algorithm selection benchmark of the container premarshalling problem. In Dhaenens, C., Jourdan, L., & Marmion, M. (Eds.), *Proceedings of the Nineth International Conference on Learning and Intelligent Optimization (LION'15)*, Lecture Notes in Computer Science, pp. 17–22. Springer-Verlag.

Vallati, M., Fawcett, C., Gerevini, A., Hoos, H., & Saetti, A. (2013). Automatic generation of efficient domain-optimized planners from generic parametrized planners. In Helmert, M., & Röger, G. (Eds.), *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SOCS'14)*. AAAI Press.

Xu, L., Hoos, H., & Leyton-Brown, K. (2010). Hydra: Automatically configuring algorithms for portfolio-based selection. In Fox, M., & Poole, D. (Eds.), *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, pp. 210–216. AAAI Press.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2008). SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, *32*, 565–606.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2011). Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI)*.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2012a). Evaluating component solver contributions to portfolio-based algorithm selectors. In Cimatti, A., & Sebastiani, R. (Eds.), *Proceedings of the Fifteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, Vol. 7317 of *Lecture Notes in Computer Science*, pp. 228–241. Springer-Verlag.

Xu, L., Hutter, F., Shen, J., Hoos, H., & Leyton-Brown, K. (2012b). SATzilla2012: improved algorithm selection based on cost-sensitive classification models. In Balint, A., Belov, A., Diepold, D., Gerber, S., Järvisalo, M., & Sinz, C. (Eds.), *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, Vol. B-2012-2 of *Department of Computer Science Series of Publications B*, pp. 57–58. University of Helsinki.

Yun, X., & Epstein, S. (2012). Learning algorithm portfolios for parallel execution. In Hamadi, Y., & Schoenauer, M. (Eds.), *Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION'12)*, Vol. 7219 of *Lecture Notes in Computer Science*, pp. 323–338. Springer-Verlag.