AutoFolio: Algorithm Configuration for Algorithm Selection

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

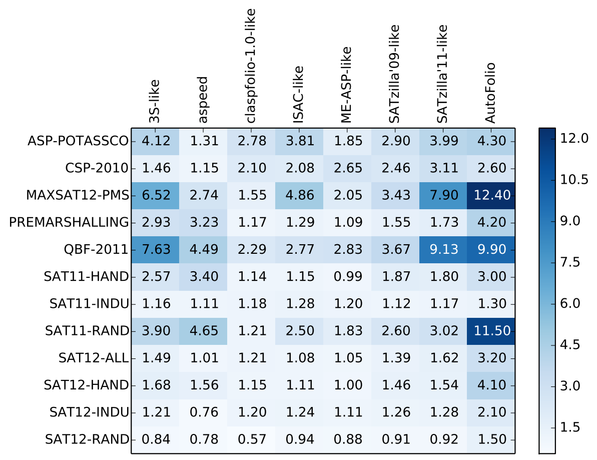
Abstract

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 use algorithm configurators to automatically configure *claspfolio 2*, which implements a large variety of different AS approaches and their respective parameters in a single highly parameterized algorithm framework. We demonstrate that this approach, dubbed *AutoFolio*, can significantly improve the performance of *claspfolio 2* on 11 out of the 12 scenarios from the Algorithm Selection Library and leads to new state-of-the-art algorithm selectors for 9 of these scenarios.

Introduction

Over the last decade, tremendous progress in Boolean constraint solving technology has been achieved in several areas within AI, notably SAT, ASP, CSP, Max-SAT and QBF. In all of 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 problem instances (so-called instance features) to choose a promising algorithm for each instance. Algorithm selectors have empirically proven 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 et al. 2008) won several categories in multiple SAT Competitions, and *claspfolio 1* (Gebser et al. 2011) won the NP-track of the 2011 ASP Competition.

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.



	3S-like	aspeed	claspfolio-1.0-like	ISAC-like	ME-ASP-like	SATzilla'09-like	SATzilla'11-like	AutoFolio
ASP-POTASSCO	4.12	1.31	2.78	3.81	1.85	2.90	3.99	4.30
CSP-2010	1.46	1.15	2.10	2.08	2.65	2.46	3.11	2.60
MAXSAT12-PMS	6.52	2.74	1.55	4.86	2.05	3.43	7.90	12.40
PREMARSHALLING	2.93	3.23	1.17	1.29	1.09	1.55	1.73	4.20
QBF-2011	7.63	4.49	2.29	2.77	2.83	3.67	9.13	9.90
SAT11-HAND	2.57	3.40	1.14	1.15	0.99	1.87	1.80	3.00
SAT11-INDU	1.16	1.11	1.18	1.28	1.20	1.12	1.17	1.30
SAT11-RAND	3.90	4.65	1.21	2.50	1.83	2.60	3.02	11.50
SAT12-ALL	1.49	1.01	1.21	1.08	1.05	1.39	1.62	3.20
SAT12-HAND	1.68	1.56	1.15	1.11	1.00	1.46	1.54	4.10
SAT12-INDU	1.21	0.76	1.20	1.24	1.11	1.26	1.28	2.10
SAT12-RAND	0.84	0.78	0.57	0.94	0.88	0.91	0.92	1.50

Figure 1: Factors by which the selection approach re-implemented in *claspfolio 2* outperformed the single best algorithm w.r.t. PAR10 (without unsolved test instances), which counts each timeout as 10 times the given runtime cutoff, based on 10-fold cross validation.

Although many new AS approaches have been proposed over the years (cf. (Kotthoff 2014)), there are only two flexible frameworks that allow for re-implementing and comparing these existing approaches in a fair and uniform way: *LLAMA* (Kotthoff 2013) and *claspfolio 2* (Hoos, Lindauer, and Schaub 2014). Of these, *claspfolio 2* is more comprehensive, encompassing strategies from *3S* (Kadioglu et al. 2011), *aspeed* (Hoos et al. 2014), *claspfolio 1* (Gebser et al. 2011), *ISAC* (Kadioglu et al. 2010), *ME-ASP* (Maratea, Pulina, and Ricca 2013) and *SATzilla* (Xu et al. 2008).

Figure 1 illustrates the performance benefits these selection strategies (as realized in *claspfolio 2*) yield across the wide range of AS benchmarks in the Algorithm Selection Library.[1] 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 it is only best on half the scenarios, with the approaches *3S*, *aspeed* or *ISAC* yielding better performance in the remaining

[1] `www.aslib.net`

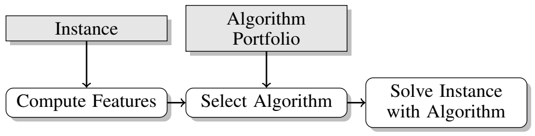


Figure 2: Workflow of algorithm selection

cases. Also, each of the approaches used a fixed parameter setting and might therefore fall short of its full potential. For example, imputation of missing instance features is not used at all in Figure 1; while it does not improve performance on some scenarios (e.g., *ASP-POTASSCO*), it yields improvements on others (e.g., *SAT12-RAND*, allowing *claspfolio 2* to outperform the single best algorithm by a factor of 1.2 with the *SATzilla'11*-like approach).

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 effectively; and (iii) how to set 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 algorithm configuration (Hutter et al. 2009) to configure flexible AS frameworks, such as *claspfolio 2*. While manual configuration is error-prone, biased by humans and requires a lot of human time and expert 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 applying algorithm configuration to algorithm selection and provide an open-source implementation[2] based on the algorithm configurators *SMAC* (Hutter, Hoos, and Leyton-Brown 2011) and *ParamILS* (Hutter et al. 2009) and the algorithm selector *claspfolio 2* (Hoos, Lindauer, and Schaub 2014). The last column of Figure 1 previews the result of *AutoFolio*, showing that the approach significantly improves the performance of *claspfolio 2* on all but one scenario.

Algorithm Selection and Configuration

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

Algorithm Selection. Figure 2 shows the workflow of algorithm selection (Rice 1976; Huberman, Lukose, and Hogg 1997). For a given problem instance, we first compute features; these are numerical characteristics, such as the number of variables or clauses in a SAT formula. Based on these features, an appropriate algorithm from an algorithm portfolio 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 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 training instances I, we use as training data a performance matrix of size $|I| \times |A|$ 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, Pulina, and Ricca 2013), g-means (Kadioglu et al. 2010) or Random Forests (Xu et al. 2011). We note that state-of-the-art portfolio-based approaches, such as *SATzilla* (Xu et al. 2012) and *3S* (Kadioglu et al. 2011), often use further techniques (such as pre-solving algorithm schedules) to increase their performance and thus do not solely rely on algorithm selection.

[2]`http://www.cs.uni-potsdam.de/wv/autofolio/`

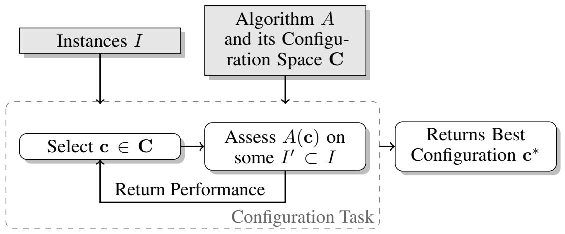

Instances I
Algorithm A and its Configuration Space $\mathbf{C}$
Select $\mathbf{c} \in \mathbf{C}$
Assess $A(\mathbf{c})$ on some $I' \subset I$
Return Performance
Returns Best Configuration $\mathbf{c}^*$
Configuration Task

Figure 3: Workflow of algorithm configuration

Algorithm Configuration. Figure 3 shows the basic workflow of algorithm configuration. The configuration task is carried out for a parameterized target algorithm with a given configuration space and a set of training problem instances; furthermore, the configurator is given a performance metric to optimize (e.g., runtime or solution quality the target algorithm achieves) and a configuration budget (e.g., the total runtime allowed for the configuration process). The configuration space is the cross-product of the parameters (which, for discrete parameters, is exponential in the number of target algorithm parameters). Furthermore, the configuration space can be structured, i.e., a parameter p_1 can be *conditional* on another parameter p_2 such that the value of p_1 is only relevant if p_2 is set to a specific value. Therefore, the configuration consists of top-level (non-conditional) parameters and conditional parameters. The configurators we consider in the following work as follows. In each iteration, a configuration is selected from the configuration space and the target algorithm is run with this configuration on one or several problem instances. Performance data collected from these runs is used by the configurator to select the next configuration to investigate. After the given configuration budget is exhausted, the configurator returns the best known parameter configuration it found until then.

Configuration of Algorithm Selectors

We now present our *AutoFolio* approach, using algorithm configurators to automatically configure algorithm selectors. To apply algorithm configuration in this context, we need to specify a parameterized selector to be configured, a configuration space for this selector, and the problem instances used for evaluating the performance of the selector. We note that

Algorithm 1: Configuration Procedure of an Algorithm Selector

```
Input : algorithm configurator $AC$, algorithm selector $AS$, configuration space $\mathbf{C}$ of $AS$, data of
        algorithm scenario $\mathbf{D}$ (with performance and feature matrix), number of folds $k$
1 randomly split $\mathbf{D}$ into $\mathbf{D}_1 \dots \mathbf{D}_k$
2 start $AC$ with $\mathbf{D}_1 \dots \mathbf{D}_k$ as meta instances, use average as meta performance metric and $AS$ with $\mathbf{C}$ as target algorithm
3 while configuration budget remaining do
4     $AC$ selects configuration $\mathbf{c} \in \mathbf{C}$ and meta instance $i \in \{1 \dots k\}$
5     $AS(\mathbf{c})$ trains on $\mathbf{D} \backslash \mathbf{D}_i$, assesses its performance on $\mathbf{D}_i$ and returns that performance to $AC$
6 return Best configuration c of AS found by AC
```

these latter instances are specific algorithm selection scenarios; we call these *meta-instances* to distinguish them from the problem instances (e.g., SAT instances) that are part of each algorithm selection scenario.

Meta-instances. For the configuration process, we need a performance estimation of the algorithm selector on some data $\mathbf{D}$. We can gain such an estimate by training the selector on a subset of the data and evaluating its performance on another subset of the data disjoint from that used for training. However, the particular subsets chosen can affect the performance of this approach quite strongly (in particular, on heterogeneous data). Therefore, a better approach is to use k-fold cross validation: we randomly split $\mathbf{D}$ into k equally-sized parts $\mathbf{D}_1, \dots, \mathbf{D}_k$, iteratively assess how well the selector performs on $\mathbf{D}_i$ when trained on $\mathbf{D} \backslash \mathbf{D}_i$, and then average the results. Following Thornton et al. (2013), we use each of these splits as one meta-instance within the configuration process, see Algorithm 1. We note that modern configurators, such as FocusedILS (Hutter et al. 2009), *irace* (López-Ibáñez et al. 2011) and *SMAC* (Hutter, Hoos, and Leyton-Brown 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.

For each algorithm selection scenario, we hold back a test set of instances that is not touched during the configuration process in order to obtain an unbiased evaluation of the configured selector's performance. Since we could split our instances into configuration and testing set in many different ways, one such split does not necessarily yield a representative performance estimate. Therefore, to yield more confident results, we perform an additional *outer* cross-validation: we consider multiple configuration/testing set splits, configure the selector for each configuration set, assess its final configurations on the respective test data sets, and average results.

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 in principle 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, other cost-sensitive binary classifiers could be used instead of random forests.

Based on this observation, we consider a hierarchically structured configuration space with a top-level parameter that decides the overall algorithm selection approach — e.g., a regression approach, as used in *SATzilla'09* (Xu et al. 2008) or a k-NN approach, as used in *ME-ASP* (Maratea, Pulina, and Ricca 2013). For most selection approaches, we can then choose between different machine learning techniques, e.g., ridge regression, lasso regression or support vector regression for a regression approach. Each of these machine learning techniques can again have its own (hyper-)parameters.

Besides the selection approach, further techniques are used for preprocessing the training data (e.g., z-score feature normalization as a feature preprocessing step or log-transformation of runtime data as performance preprocessing step). Preprocessing techniques can be configured independently from the selection approach, and are therefore also handled by top-level parameters. These include binary parameters that enable or disable feature groups that are defined by the specific algorithm selection scenarios. We note that, according to the definition of the Algorithm Selection Library[1], 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.

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 used. Pre-solving is known to be effective in selection scenarios, where (a) some instances can be solved very quickly by some solvers or (b) the algorithm selector poorly selects algorithms, e.g., because of poor instance features. As Figure 1 shows, the algorithm schedules of *aspeed* are effective on some scenarios but not on all. 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.

Figure 4 shows the complete configuration space of *claspfolio 2* (Hoos, Lindauer, and Schaub 2014)[3], which we used as the basis for *AutoFolio*. It covers five different algorithm selection approaches and for each of them, three different machine learning techniques (where appropriate). Furthermore, it supports several preprocessing strategies and pre-solving schedules computed by *aspeed*.

We chose the default configuration of *claspfolio 2* (used to initialize the algorithm configurator) to be a *SATzilla'11*-like configuration since it was shown to be effective on ASP (Hoos, Lindauer, and Schaub 2014), *SATzilla'11* is also strong on SAT (Xu et al. 2012) and based of the results of Figure 1. This configuration uses pairwise cost-sensitive random forest classifiers, at most three pre-solvers and z-score feature

[3] http://www.cs.uni-potsdam.de/claspfolio/

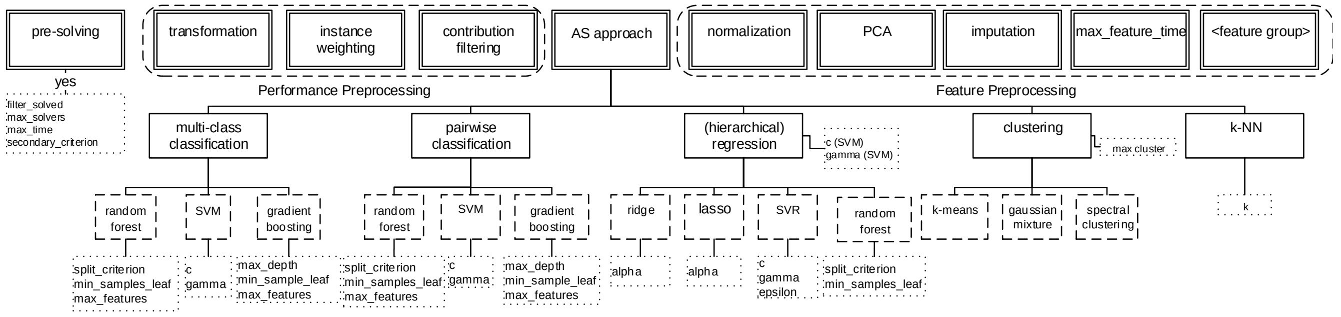


Figure 4: Configuration space of *claspfolio 2*, including 19 categorial parameters, 12 integer valued parameters and 11 continous parameters. Additionally, there is a binary parameter for each feature group of the algorithm scenario. Parameters in double boxes are top-level parameters; single boxes represent algorithm selection approaches based on classes of machine learning techniques, dashed boxes machine learning techniques and dotted boxes lists of low-level parameters.

normalization. Since we assume no prior knowledge about the algorithm selection scenarios, the default configuration uses all available instance features. We note that these instance features introduce substantial computational overhead on algorithm selection scenarios with large feature computation times (such as industrial SAT instances).

We chose *claspfolio 2* as the basis for *AutoFolio*, because it has been designed to be flexible and is known to perform well (Hoos, Lindauer, and Schaub 2014). We note, that in principle, other selectors, such as *SATzilla* (Xu et al. 2008), *ISAC* (Kadioglu et al. 2010) and *SNNAP* (Collautti et al. 2013) could be generalized in a similar way.

Next to using *claspfolio 2* as its algorithm selection framework, our current (first) instance of *AutoFolio* employs two complementary state-of-the-art algorithm configurators, *SMAC* (Hutter, Hoos, and Leyton-Brown 2011)[4]. and *ParamILS* (Hutter et al. 2009)[5]. Like the choice of selectors, this choice of configurators can also be changed in the future.

Empirical Performance Analysis

In this section, we empirically analyze the performance of our *AutoFolio* approach (in these experiments based on *claspfolio 2* using *sklearn* 0.15.0 (a widely used machine learning package for Python, see Pedregosa et al. 2011), *SMAC* 2.06.01, and *ParamILS* 2.3.7 as described in the previous section). We ran *AutoFolio* on the twelve algorithm selection scenarios that make up the Algorithm Selection Library. 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. We refer to the library's website[1] for the details on all scenarios but point out that using this common library allows us to compare *AutoFolio* in a fair and uniform way against other algorithm selection methods.

Algorithm Configuration Setup. Following standard practice, we performed multiple (in our case, 16) independent runs for each of our two configurators 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. As a performance metric, we used penalized average runtime with factor 10 (PAR10), which counts each timeout as 10 times the given runtime cutoff (runtime cutoffs differ between the ASlib scenarios). We further show how the optimization of PAR10 influenced other metrics, such as number of timeouts and PAR1. The time required to evaluate a single configuration of *claspfolio 2* varied between 2 CPU seconds and 1 CPU hour, 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, i.e., we configured *claspfolio 2* ten times for each scenario (with different configuration set/test set splits). Therefore, in total, we performed a total of $16 \cdot 2 \cdot 10 = 320$ configurations runs of 2 CPU days for each of the twelve ASlib benchmarks, requiring a total of 7 680 CPU days. We performed these experiments on a cluster 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 that although our thorough evaluation of *AutoFolio* required this substantial compute power, applying it to a new benchmark set with a given training-test split would only require 32 independent configuration runs of two days each.

Analysis of Configuration Process. In Table 1, we compare the performance of *claspfolio 2*'s default configuration (*SATzilla'11*-like, as discussed in the previous section) with that of the configuration optimized by *AutoFolio*. For all selection scenarios, *AutoFolio* achieved better performance on training data, and improved performance on test data was obtained on all but one scenario. Performance improvements on test data were statistically significantly at $\alpha = 0.1$ and $\alpha = 0.05$ for eight and five scenarios, respectively, according to a permutation test with 100 000 permutations.

The performance improvement on the training data of *CSP-2010* did not transfer to test data, due to over-tuning to the special characteristics of the training data. We note, however, that in each of its folds, only 1 to 2 timeouts were encountered, causing the observed performance differences to be statistically insignificant. There were also some (smaller)

[4]http://www.aclib.net/smac/

[5]http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/

Scenario	Training ∅ PAR10 Default	Training ∅ PAR10 Config.	Training ∅#TOs Default	Training ∅#TOs Config.	Training ∅ PAR1 Default	Training ∅ PAR1 Config.	Test ∅ PAR10 Default	Test ∅ PAR10 Config.	Test $\sum$ #TOs Default	Test $\sum$ #TOs Config.	Test ∅ PAR1 Default	Test ∅ PAR1 Config.
ASP-POTASSCO	120.9	**88.9****	16.1	10.6	41.2	36.4	132.5	**123.0**	20	19	43.1	38.3
CSP-2010	376.8	**167.3****	8.8	1.9	128.2	113.7	**356.9**	413.3	9	11	123.9	129.8
MAXSAT12-PMS	296.3	**119.3****	7.0	2.2	99.7	57.5	285.3	**169.9***	7	4	103.5	62.7
PREMARSHALLING	2552.1	**1360.6****	30.0	13.8	502.7	417.9	2185.1	**1663.7***	28	20	462.8	434.0
QBF-2011	1001.8	**760.9****	21.4	14.9	270.9	251.8	937.1	**924.9**	22	21	250.2	268.7
SAT11-HAND	6918.0	**4370.2****	25.2	14.6	1168.7	1035.1	7561.1	**5935.8***	31	24	1194.8	1135.0
SAT11-INDU	7722.3	**4749.5****	32.7	18.9	1260.8	1016.2	7981.5	**7536.8**	38	36	1280.0	1196.6
SAT11-RAND	3264.5	**745.7****	26.9	4.3	530.9	308.4	3593.5	**1294.8****	32	10	572.9	363.4
SAT12-ALL	1723.2	**909.5****	182.4	93.6	350.1	204.9	1615.0	**925.8****	187	106	348.4	207.9
SAT12-HAND	1896.3	**912.4****	69.7	31.5	341.7	209.8	1859.9	**968.2****	75	38	342.6	208.4
SAT12-INDU	1192.1	**581.9****	72.3	32.2	286.5	178.5	1182.8	**638.3****	80	41	281.6	177.4
SAT12-RAND	678.5	**358.8****	44.8	23.3	161.5	89.9	695.8	**382.9****	51	28	161.3	91.6

Table 1: Results for *AutoFolio* with outer 10-fold cross validation (∅ denotes arithmetic mean). To avoid artificial inflation of PAR10 scores, problem instances that were not be solved by any solver were removed from the test sets. The best PAR10 values for each scenario are shown in bold face; cases where the performance difference to the next best entry is statistically significant (according to a permutation test with 100 000 permutations) at $\alpha = 0.1$ and $\alpha = 0.05$, are marked with * and **, respectively.

over-tuning effects for *QBF-2011* and *SAT11-INDU*. For *ASP-POTASSCO*, we note that the default configuration of *claspfolio 2* was manually optimized on this scenario (Hoos, Lindauer, and Schaub 2014), and *AutoFolio* found very similar configurations with nearly identical performance.

On *PREMARSHALLING*, *AutoFolio* solved 8 additional problem instances and reduced PAR10 by more than 20%; 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 decrease the number of timeouts on *PREMARSHALLING*, they also introduce overhead on most of the other instances in this scenario, making it harder for *AutoFolio* to achieve more significant performance improvements. Similar considerations apply to the *MAXSAT12-PMS* and *SAT11-HAND* scenarios.

Which Choices Lead to Good Performance?. We examined the 120 configurations optimized by *AutoFolio* (12 scenarios $\times$ 10-fold outer cross validation) to obtain some insights into which choices are made. First of all, as we expected, *AutoFolio* used only a subset of feature groups for the SAT scenarios, since the computational cost associated with some features is not recouped by improvements in algorithm runtime. We found that feature imputation was used in 83% of the configurations; especially on the SAT scenarios, which include many missing features. However, it did not matter which of the imputation strategies were used (we implemented most frequent value, mean or median as imputation strategies).

The pre-solving schedule turned out to be an important component that was used in 73% of the configurations. The parameters controlling the details of the schedule, i.e., number of pre-solvers and maximal time of pre-solving, varied across validation folds and selection scenarios.

As we initially assumed, no algorithm selection strategy dominated all others, see Figure 5. Overall, the pairwise classification approach was chosen most frequently (55%; 33% in combination with random forests, as in *SATzilla'11*).

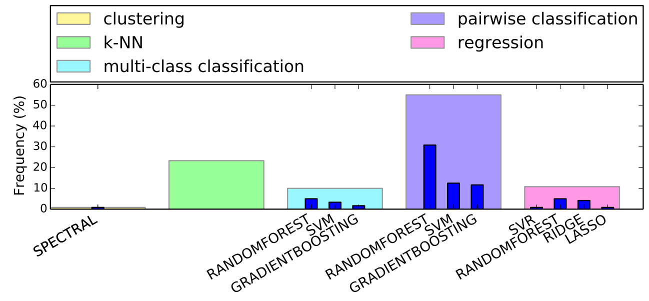


Figure 5: Frequency of algorithm selection approaches.

Comparison Against Other Selectors. In Table 2, we compare *AutoFolio* with the random forest regression approach used in *ASlib* (Bischl et al. 2014), *SATzilla'11* (Xu et al. 2011), *SNNAP* (1.5.0; Collautti et al. (2013)) and *ISAC* (implementation in *SNNAP* 1.5.0; Kadioglu et al. (2010)).[6] *SATzilla'11* does not support the *ASlib* format and we can only compare to published results (with different cross validation splits) for the three SAT 2011 scenarios (Xu et al. 2012). We note that *ASlib*(RF), *ISAC* and *SNNAP* are pure algorithm selectors, whereas *SATzilla* and *claspfolio 2* additionally use pre-solver schedules. Since *SNNAP* and *ISAC* do not support feature groups and their associated costs, we measured their performance using the default feature groups defined in the ASlib scenarios. Overall, *AutoFolio* performed best on eight out of twelve scenarios.

We note that the SAT11 scenarios comprise a relative small number of problem instances, so that the training set for *clasp-*

[6]Other state-of-the-art selectors, such as *3S* (Kadioglu et al. 2011) and *CSHC* (Malitsky et al. 2013), are not publicly available with their training procedures, and we were therefore unable to train them on our scenarios.

	Oracle	*SB*	*ASlib* (RF)	*SATzilla'11*	*SNNAP*	*ISAC*	*AutoFolio*	*SB*/*af*
ASP-POTASSCO	21.3	534.2	124.8	NA	203.8	291.9	**123**	4.3
CSP-2010	107.7	1087.5	**378**	NA	1087.5	1027	413.3	2.6
MAXSAT12-PMS	40.8	2111.6	294.5	NA	895	786.4	**169.9**	12.4
PREMARSHALLING	227.6	7002.9	3921.9	NA	9042.1	5880.8	**1663.7**	4.2
QBF-2011	96	9172.4	1038.9	NA	7386.2	3813.5	**924.9**	9.9
SAT11-HAND	709.4	17966	9637.1	6138.1	9940.9	14592.9	**5935.8**	3.0
SAT11-INDU	420	8985.7	7465.8	**5889.3**	6632.6	8461.2	7536.8	1.3
SAT11-RAND	227.4	14938.6	4856.9	**990.2**	4859	3140.4	1294.8	11.5
SAT12-ALL	93.8	2968	1843.3	NA	1427.5	2989.3	**925.8**	3.2
SAT12-HAND	113.3	3929.2	2556.1	NA	2180.5	4110.8	**968.2**	4.1
SAT12-INDU	88.2	1366.8	1058.3	NA	789.0	1409.5	**638.3**	2.1
SAT12-RAND	46.9	568.6	618.3	NA	593.1	434.5	**382.9**	1.5

Table 2: Performance comparison between *AutoFolio* (*af*), single best solver (*SB*) – selected based on PAR10 on the training set – as a baseline, the oracle (also known as VBS) as a bound on the optimal performance of an algorithm selector and other algorithm selectors, using 10-fold cross validation and PAR10 scores, with unsolved instances removed from the test set to avoid artificially inflating PAR10 scores. The best performance value for each scenario is shown in bold face. Typo in original: We misinterpreted the output of *SNNAP* and *ISAC* - we corrected their performance values.

folio 2 was small (only 243 instances, due the two levels of 10-fold cross-validation used in our experiments). Thus, the potential for over-tuning was higher on these scenarios. Somewhat as a surprise to us, *AutoFolio* used a k-NN approach on *SAT11-INDU* which, in our experience, yields more overtuning than pairwise classification. Therefore, we re-ran the SAT11 experiments with a reduced *AutoFolio* configuration space that fixed the AS approach to pairwise classification. This yielded better results than *SATzilla'11* on all three scenarios, with PAR10 scores of 4918.6 on *SAT11-HAND*, 5778.2 on *SAT11-INDU* and 888.2 on *SAT11-RAND*.

Related Work

As far as we know, this is the first time that algorithm configuration is used to optimize algorithm selection. A related approach for general supervised machine learning is Auto-WEKA (Thornton et al. 2013), a system that addresses the combined problem of selecting a machine learning algorithm from the WEKA framework (Hall et al. 2009) and optimizing its hyperparameters. Auto-WEKA and AutoFolio use the same cross-validation mechanism to define meta instances for the configuration process. Even though machine learning is also part of algorithm selectors, *AutoFolio* has to consider many other aspects in its configuration space, such as presolving, cost-sensitive approaches, and feature steps.

The *AutoFolio* approach is not limited to using a particular algorithm configurator or algorithm selection framework. In principle, other configurators, such as *irace* (López-Ibáñez et al. 2011) or *gga* (Ansótegui, Sellmann, and Tierney 2009), or other selectors, such as *LLAMA* (Kotthoff 2013), *SNNAP* (Collautti et al. 2013) or *ISAC* (Kadioglu et al. 2010) could be used. However, the selector should be highly parameterized to cover a wide range of approaches, which, to our best knowledge, is so far only the case for *claspfolio 2*.

Algorithm configuration and algorithm selection have previously been combined in a different way, by using algorithm configuration to find good parameter settings of a highly parameterized algorithm and then using algorithm 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, and 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 will most improve the portfolio.

A previous application of a meta-solving strategy to another meta-solving strategy was the self-configuration of *ParamILS* (Hutter et al. 2009). However, in contrast to the substantial improvements we achieve for *claspfolio 2*, that self-configuration only yielded a small improvement over *ParamILS*'s default.

Conclusions

We presented *AutoFolio*, to the best of our knowledge the first approach to automatically configure algorithm selectors. Using a concrete realization of this approach based on the highly parameterized algorithm selection framework *claspfolio 2*, we showed that state-of-the-art algorithm configurators can automatically find optimized configurations of algorithm selectors that perform significantly (and sometimes substantially) better than manually configured selectors. The automatically configured *claspfolio 2* system showed performance improvements of a factor between 1.3 and 12.4 in terms of PAR10 scores in comparison to the best single solver for the given algorithm portfolios

Next, we will investigate why *AutoFolio* showed overtuning in some scenarios and how to prevent this. We will assess restricting the configuration space to methods less prone to over-tuning, using other methods for defining meta instances (e.g., bootstrapping), and integrate *AutoFolio* into the Algorithm Configuration Library (Hutter et al. 2014) to assess using other configurators (e.g., *irace* (López-Ibáñez et al. 2011) and *gga* (Ansótegui, Sellmann, and Tierney 2009)).

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.

References

Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In Gent, I., ed., *Proceedings of CP'09*, volume 5732 of *Lecture Notes in Computer Science*, 142–157. Springer-Verlag.

Bischl, B.; Kerschke, P.; Kotthoff, L.; Lindauer, M.; Malitsky, Y.; Frechétte, A.; Hoos, H.; Hutter, F.; Leyton-Brown, K.; Tierney, K.; and Vanschoren, J. 2014. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*. under review.

Collautti, M.; Malitsky, Y.; Mehta, D.; and O'Sullivan, B. 2013. SNNAP: Solver-based nearest neighbor for algorithm portfolios. In Blockeel, H.; Kersting, K.; Nijssen, S.; and Zelezný, F., eds., *Proceedings of ECML/PKDD'13*, volume 8190 of *Lecture Notes in Computer Science*, 435–450. Springer-Verlag.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Schaub, T.; Schneider, M.; and Ziller, S. 2011. A portfolio solver for answer set programming: Preliminary report. In Delgrande, J., and Faber, W., eds., *Proceedings of LPNMR'11*, volume 6645 of *Lecture Notes in Computer Science*, 352–357. Springer-Verlag.

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

Hoos, H.; Kaminski, R.; Lindauer, M.; and Schaub", T. 2014. aspeed: Solver scheduling via answer set programming. *Theory and Practice of Logic Programming* First View:1–26. Available at `http://arxiv.org/abs/1401.1024`".

Hoos, H.; Lindauer, M.; and Schaub, T. 2014. claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming*. to appear; available at `http://arxiv.org/abs/1405.1520`.

Huberman, B.; Lukose, R.; and Hogg, T. 1997. An economic approach to hard computational problems. *Science* 275:51–54.

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

Hutter, F.; López-Ibánez, M.; Fawcett, C.; Lindauer, M.; Hoos, H.; Leyton-Brown, K.; and Stützle, T. 2014. Aclib: a benchmark library for algorithm configuration. In Pardalos, P., and Resende, M., eds., *Proceedings of LION'14*, Lecture Notes in Computer Science. Springer-Verlag. to appear.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION'11*, volume 6683 of *Lecture Notes in Computer Science*, 507–523. Springer-Verlag.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC - instance-specific algorithm configuration. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of ECAI'10*, 751–756. IOS Press.

Kadioglu, S.; Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Algorithm selection and scheduling. In Lee, J., ed., *Proceedings of CP'11*, volume 6876 of *Lecture Notes in Computer Science*, 454–469. Springer-Verlag.

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.

López-Ibáñez, M.; Dubois-Lacoste, J.; Stützle, T.; and Birattari, M. 2011. The irace package, iterated race for automatic algorithm configuration. Technical report, IRIDIA, Université Libre de Bruxelles, Belgium.

Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2013. Algorithm portfolios based on cost-sensitive hierarchical clustering. In Rossi, F., ed., *Proceedings of IJCAI'13*, 608–614.

Maratea, M.; Pulina, L.; and Ricca, F. 2013. A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming* First View:1–28.

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.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

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

Thornton, C.; Hutter, F.; Hoos, H.; and 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.; and Uthurusamy, R., eds., *Proceedings of KDD'13*, 847–855. ACM Press.

Xu, L.; Hutter, F.; Hoos, H.; and 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.; and 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.; and Leyton-Brown, K. 2012. Evaluating component solver contributions to portfolio-based algorithm selectors. In Cimatti, A., and Sebastiani, R., eds., *Proceedings of SAT'12*, volume 7317 of *Lecture Notes in Computer Science*, 228–241. Springer-Verlag.

Xu, L.; Hoos, H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In Fox, M., and Poole, D., eds., *Proceedings of AAAI'10*, 210–216. AAAI Press.