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. In this extended abstract of our 2015 JAIR article of the same title, we summarize AUTOFOlio, which uses an algorithm configuration procedure to automatically select an AS approach and optimize its parameters for a given AS scenario. AUTOFOlio allows researchers and practitioners across a broad range of applications to exploit the combined power of many different AS methods and to automatically construct high-performance algorithm selectors. We demonstrate that AUTOFOlio was able to produce new state-of-the-art algorithm selectors for 7 well-studied AS scenarios and matches state-of-the-art performance statistically on all other scenarios. Compared to the best single algorithm for each AS scenario, AUTOFOlio achieved average speedup factors between 1.3 and 15.4.

1 Introduction

Over the last decade, tremendous progress in Boolean constraint solving technology has brought benefits to several areas within AI. 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 problem instances (so-called instance features) to choose a promising algorithm for each instance. Algorithm selectors have been shown empirically to improve the state of the art for solving heterogeneous instance sets and, as a result, have won many prizes at competitions.

Figure 1 illustrates the performance benefits these existing selection strategies (as realized in CLASPFOLIO 2 [Hoos et al., 2014]) yield across the wide range of AS benchmarks in the Algorithm Selection Library [Bischl et al., 2016]. We observe that each approach has strengths and weaknesses on different scenarios. The SATZILLA’11-like approach [Xu et al., 2011] performs best overall, but only achieves better performance than the other approaches considered on 8 out of the 13 scenarios, with 3S [Kadioglu et al., 2011], ASPEED [Hoos et al., 2015] or ISAC [Kadioglu et al., 2010] yielding better performance in the remaining cases.

All these selection methods make use of machine learning techniques, whose performance is known to depend on hyperparameter settings (e.g., in the case of an SVM, the kernel, kernel hyper-parameter and soft margin; cf. [Bergstra et al.,]
2 Preliminaries on AS and AC

Algorithm Selection. Figure 2 shows the general outline of algorithm selection [Rice, 1976; Smith-Miles, 2008; Kotthoff, 2014]. For a given problem instance, we first compute cheap instance features [Nudelman et al., 2004]; 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, a machine learning model is used to select an appropriate algorithm from a set of algorithms to solve the given instance [Huberman et al., 1997; Gomes and Selman, 2001].

Algorithm Configuration. Figure 3 shows a general outline for algorithm configuration methods [Hutter et al., 2009]. Given a parameterized algorithm $A$ with possible parameter settings $C$, a set of training problem instances $I$, and a performance metric $m : C \times I \rightarrow R$, the objective in the algorithm configuration problem is to find a parameter configuration $c \in C$ that optimize $m$ across the instances in $I$. In the following, we assume w.o.l.g. that performance metric $m$ is to be minimized, as is the case for running time or classification error. The configuration procedure (or short configurator) iteratively evaluates the performance of parameter configurations $c \in C$ (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.

3 Configuration of Algorithm Selectors

We now present our AUTOFOLIO approach of using algorithm configurators to automatically customize flexible 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

An AS scenario includes algorithms $A$, problem instances $I$, performance and feature data $D$, and a performance metric $m : A \times I \rightarrow R$ to be minimized, with the data split into disjoint sets $D_{train}$ and $D_{test}$. Let $S(D_{train}) : I \rightarrow A$ denote the selector learned by the AS system $S$ when trained on the data $D_{train}$. Then, the performance of $S$, $P(S)$ is the average performance of the algorithms it selects on the instances in the test data set $D_{test}$:

$$P(S) = \frac{1}{|D_{test}|} \sum_{i \in D_{test}} m(S(D_{train}), i).$$

Likewise, we can evaluate the performance of an AS system $S_e$ parameterized by a configuration $c$ as $P(S_e)$. In order to obtain an unbiased estimate of the configured selector’s performance, 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 following approach for each selection scenario: (i) we split the full set of instances into a training and a test set.
and (ii) we partition the training data further into $k$ folds (in our experiments, we use $k = 10$), which are used as follows.

Let $D_{\text{train}}^{(1)}, \ldots, D_{\text{train}}^{(k)}$ be a random partition of the training set $D_{\text{train}}$. The cross-validation performance $CV(S_c)$ of $S_c^j$ on the training set, which is optimized during configuration, is then:

$$CV(S_c) = \frac{1}{k} \sum_{j=1}^{k} \frac{1}{|D_{\text{train}}^{(j)}|} \sum_{i \in D_{\text{train}}^{(j)}} m(S_c(D_{\text{train}} \setminus D_{\text{train}}^{(j)}), i)$$

We minimize $CV(S_c)$ with an algorithm configuration procedure to yield

$$\hat{c} \in \arg \min_{c \in \mathcal{C}} CV(S_c).$$

We then evaluate the generalization performance of $\hat{c}$ by training a selector $S_\hat{c}$ with it on the entire training data and evaluating $P(S_\hat{c})$ on $D_{\text{test}}$, as defined in Equation 1.

Following Thornton et al. (2013), we use each of the $k$ folds $D_{\text{train}}^{(j)}$ as one meta-instance within the configuration process. We note that many configurators, such as FocusedILS [Hutter et al., 2009], irace [López-Ibáñez et al., 2016] 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, Figure 4 outlines the process to configure an algorithm selector with AUTOFOLIO.

### 3.2 Configuration Space of Selectors

Most existing algorithm selectors implement one specific AS approach, using one specific machine learning technique. However, most selection approaches, at least implicitly, admit more flexibility, and in particular could be used with a range of machine learning techniques. Based on this observation, we consider a hierarchically structured configuration space with a top-level parameter that determines the overall AS approach. For most selection approaches, we can then choose between different regression techniques. Each of these machine learning techniques can be configured by its own (hyper-)parameters. Additionally, further techniques can be used for preprocessing the training data and for preprocessing schedules, which can be configured independently from the selection approach, and are therefore also handled by top-level parameters.

We implemented these choices in the CLASPFOLIO 2 system. Figure 5 illustrates the complete configuration space thus obtained. Our current version, which we use for the concrete implementation of our AUTOFOLIO approach, covers six different AS approaches:

- **(hierarchical) regression** (inspired by SATZILLA’09; [Xu et al., 2008]) learns a regression model for each algorithm; for a new instance, 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;

- **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). For preprocessing strategies, it supports several feature and performance preprocessing strategies.

The current version of AUTOFOLIO employs the algorithm configurator SMAC [Hutter et al., 2011] to search for the best AS approach in this combined space. We chose SMAC, because it performed best across the algorithm configuration problems we studied so far.

### 4 Empirical Performance Analysis

We studied the performance of AUTOFOLIO across a wide variety of hard combinatorial problems from the algorithm selection library (ASlib [Bischl et al., 2016]). Figure 1 shows the performance of AUTOFOLIO (last column) compared to its components with fixed parameter settings. AUTOFOLIO found well-performing configurations of CLASPFOLIO 2 and performed better than all other approaches on 11 out of the

---

1. www.aslib.net
5 Conclusions and Future Work

In this extended abstract of our JAIR article [Lindauer et al., 2015b], we summarized 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 AS 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 AS 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 AS scenarios.

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) extend AUTOFOLIO’s configuration space by implementing more algorithm selection approaches (e.g., CSHC [Malitsky et al., 2013]); (ii) shrink large configuration spaces based on the analysis of parameter importance through fANOVA [Hutter et al., 2014] and ablation [Fawcett and Hoos, 2016], allowing the configurator to focus on the most important parameters; and (iii) automatically select between pre-configured algorithm selectors, based on features of a given AS scenario, and further improve performance by warmstarting automatic configuration from the configurations thus selected [Feurer et al., 2015b]. 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). Finally, we plan to investigate to which extent AUTOFOLIO can configure AS systems for selecting parallel portfolios [Lindauer et al., 2015a] to exploit the increasing availability of parallel computing resources.

Overall, our results for AUTOFOLIO strongly suggest that the automated configuration of AS systems can improve 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 AS system. Our open-source implementation of AUTOFOLIO is available at www.ml4aad.org/autofolio/.
References


