The Eleventh International Symposium on Combinatorial Search
(SoCS 2018)

A Regression-Based Methodology for Online Algorithm Selection

Hans Degroote,
Patrick De Causmaecker
KU Leuven
Department of Computer Science
CODeS & Imec-ITEC
Hans.Degroote@kuleuven.be

Bernd Bischl
Department of statistics
LMU Munich

Lars Kotthoff
Department of Computer Science
University of Wyoming

Abstract

Algorithm selection approaches have achieved impressive performance improvements in many areas of AI. Most of the literature considers the offline algorithm selection problem, where the initial selection model is never updated after training. However, new data from running algorithms on instances becomes available when algorithms are selected and run. We investigate how this online data can be used to improve the selection model over time. This is especially relevant when insufficient training instances were used, but potentially improves the performance of algorithm selection in all cases. We formally define the online algorithm selection problem and model it as a contextual multi-armed bandit problem, propose a methodology for solving it, and empirically demonstrate performance improvements. We also show that our online algorithm selection method can be used when no training data whatsoever is available, a setting where offline algorithm selection cannot be used. Our experiments indicate that a simple greedy approach achieves the best performance.

Introduction

Many AI problems are NP-complete. Nevertheless, in practice good solutions can be obtained by employing powerful heuristics. Such heuristics work well in some cases, but not in others. Due to the complex nature of the relationship between a heuristic and its performance on problem instances, it is difficult to identify under which conditions it performs well and, more generally, to identify the algorithm that is best suited for solving a given problem instance. Learning to identify in which cases an algorithm is best is known as the algorithm selection problem (Rice 1976).

A common approach for the algorithm selection problem uses algorithm portfolios (Huberman, Lukose, and Hogg 1997) – sets of algorithms that complement each other on an instance distribution – and a mechanism to select from among them. There has been much interest in algorithm selection and portfolios, fuelled by practical successes in for example SAT (Xu et al. 2008), CSP (O'Mahony et al. 2008) and AI planning (Helmert, Röger, and Karpas 2011) (see (Kotthoff 2014) for a survey).

Algorithm selection often uses machine learning to build models of how the algorithms in the portfolio behave. Most algorithm selection approaches in the literature work by building models based on performance data that was obtained offline. These models are then used to make predictions on new problem instances without ever changing them. If an insufficient amount of training data was supplied, or if it was not representative, performance will stay poor even after processing a large number of online instances, as the model is never updated.

In online algorithm selection, the new performance data that becomes available when running the selected algorithm on an instance is used to update the models after every selection – instead of remaining static, the models are continuously improved without the need to explicitly run additional experiments or generate more data.

This paper gives a definition of the online algorithm selection problem, models it as as a contextual multi-armed bandit problem, and proposes a general methodology. We empirically validate the methodology with experiments on a diverse set of standard algorithm selection benchmarks. We also show empirically that our online method is useful when no training data is available.

Related Work

While most of the literature on algorithm selection considers the offline setting, a number of researchers have investigated online algorithm selection over the years. (Gagliolo and Schmidhuber 2010) consider the problem of assigning resources to algorithms running in parallel. They present a methodology for learning online which of a portfolio of resource-allocating strategies is best and model the problem as a bandit problem where each arm corresponds to an allocator. Their methodology does not take context (which instance is being solved) into account, as we do in this paper, and neither do they consider the lower-level goal of how to use online data to learn within one such allocation strategy, which is the focus of this paper.

Instead of selecting one algorithm to solve an instance completely, an algorithm can also be selected to solve part of an instance, with a series of such selections taking place to solve the instance completely. This problem is studied in the field of operator selection or heuristic selection, where applications of online learning are also investigated. In such cases, learning and improving of models is limited to a single instance, and the features relate to the performance observed so far on the instance. To the best of our knowledge, there is no work on knowledge transfer between instances. Examples include (Veerapen, Maturana, and Saubion 2012) and (Phillips et al. 2015). Examples of other related methods are (Lagoudakis and Littman 2000), where reinforcement learning is used to decide whether and how to switch the algorithm while solving a problem instance, and (Cicirello and Smith 2005), where the problem of deciding with which algorithm to retry solving an instance is modelled as a max K-armed bandit.

Most closely related to this paper's setting is the work of (Malitsky, Mehta, and O'Sullivan 2013), but our methodology does not require additional experiments and has a lower overhead. We discuss this work in the methodology section.

The Online Algorithm Selection Problem

The goal of algorithm selection is to select for each instance an algorithm that solves it well. It can be seen as a cost-sensitive classification problem, where each instance is labelled with the respective best algorithm and the cost is the performance loss incurred due to an incorrect decision, which differs from instance to instance and from algorithm to algorithm (Bischl et al. 2012).

The core elements of algorithm selection are a set of instances I, a set of features F characterising the instances, a set of algorithms A (the portfolio), and a distribution $\mathcal{D}$ over the instances I, defining for each instance its probability of occurring. When an algorithm is run on an instance, its performance is observed. Without loss of generality, let the performance of any algorithm on any instance be an element of the real numbers $\mathbb{R}$. We assume that a higher performance value is better. The performance of each algorithm on each instance is defined by performance mapping $p\colon I \times A \to \mathbb{R}\colon p(i,a) = r$. This notation assumes deterministic performance. A stochastic algorithm's performance on an instance is defined as a distribution over $\mathbb{R}$.

Algorithm selection maps each instance to an algorithm. To do so, first the vector of feature values of an instance is calculated, defined by feature-value mapping $\Phi\colon I \to \mathbb{R}^{|F|}\colon \Phi(i) = \varphi$. Then the selection mapping λ is queried. This mapping defines which algorithm should be used for each feature vector: $\lambda\colon \mathbb{R}^{|F|} \to A\colon \lambda(\varphi) = a$. So the algorithm selected for instance i is: $\lambda(\Phi(i))$. For non-deterministic selection mappings, each feature vector is mapped to a distribution over A.

The performance of a selection mapping on an instance distribution is defined as the expected performance of the algorithms it selects: $\mathcal{P}(\lambda, \mathcal{D}) = \mathbb{E}_{i\sim\mathcal{D}}[p(i, \lambda(\Phi(i)))]$. If the selection mapping always selects the same algorithm, this formula simplifies to the definition of the performance of a single algorithm on an instance distribution: $\mathcal{P}(a, \mathcal{D}) = \mathbb{E}_{i\sim\mathcal{D}}[p(i,a)]$.

The best possible selection mapping maps each instance to the algorithm with the best expected performance on it. Let λ^* be the best selection mapping, with $\forall i \in I\, \forall a \in A : p(i, \lambda^*(\Phi(i))) \geq p(i,a)$. The goal of algorithm selection is to find λ such that $min(\mathcal{P}(\lambda_*, \mathcal{D}) - \mathcal{P}(\lambda, \mathcal{D}))$.

A selection mapping is typically induced by training data H: a set of records $(i, \varphi, a, p(i,a))$, where i is an instance, φ its feature vector, a an algorithm and $p(i,a)$ the observed performance of the algorithm on the instance. Training data is usually obtained by sampling the instance distribution, computing the feature values of each sampled instance and running all algorithms on all sampled instances.

Let β be the strategy that maps the training data H to a selection mapping: $\beta\colon H \to \Lambda\colon \beta(H) = \lambda$, where Λ denotes the set of all possible selection mappings. Typically, supervised learning methods are used as such strategies.

In offline algorithm selection, the learning is one-shot: the selection mapping is learned once, based on the training data, and never changed. However, more data becomes available while the selection mapping is used: every time a selection is made for a new instance, the instance's features are calculated and the performance of the selected algorithm is observed. The idea of online algorithm selection is to add this data to the set of training data H and to allow the strategy β to update the selection mapping.

The high-level algorithm for online algorithm selection is given in algorithm 1. The algorithm for offline algorithm selection is very similar, but without lines 9 and 10: the performance data that becomes available is never used and the selection mapping is never updated.

Algorithm 1 Online algorithm selection

```
1:  Input: training data H
2:  Input: selection strategy β
3:  Input: feature-vector mapping Φ
4:  λ = β(H) # initialise selection mapping
5:  for instance i do
6:      φ = Φ(i) # calculate feature vector
7:      a = λ(φ) # select algorithm
8:      Solve i with a, observing performance p
9:      H = H ∪ {i, φ, a, p} # update training data
10:     λ = β(H) # update selection mapping
```

Assuming that strategy β and the algorithms A are deterministic, the performance of a strategy β over a series of T instances is the average performance of the individual selection mappings it generates over time, $P(\beta) = \sum_{t=1}^{T} \frac{1}{T} \mathcal{P}(\lambda^{(t)})$, where $\lambda^{(t)}$ is the selection mapping used at time t. If the strategy is non-deterministic, its expected performance is the expected value over all possible series of selection mappings of length T.

Methodology for Online Algorithm Selection

The training data used for algorithm selection is usually complete: for every training instance, the performance of all

algorithms is available. In contrast, the data generated during the online phase is very incomplete: for each instance, the performance of only one algorithm becomes available.

Many methods for offline algorithm selection cannot handle incomplete data. Specifically, all strategies that need to know for each training instance what the best algorithm is cannot be used in the online setting, because only the performance of a single algorithm is known. This includes all methods that model algorithm selection as a classification problem, directly predicting which algorithm is best, or approaches that use the performance of more than one algorithm, such as (O'Mahony et al. 2008). A possible solution is to impute the missing performance values, thereby obtaining complete data; however, how to impute based on the sparse online data is not obvious.

Alternatively, additional experiments can be run to determine the performance of each algorithm, but this comes at the cost of significant overhead. This approach can be improved by generating only data that is expected to be useful. This idea is explored in (Malitsky, Mehta, and O'Sullivan 2013), who cluster instances and learn the best algorithm for each cluster. A new online instance is assigned to a cluster and its best algorithm is estimated. If the estimated best is the same as the earlier best algorithm for the cluster, that algorithm is used. If they differ, all algorithms are run on the new instance, thereby generating complete data, and the best algorithm for the cluster is re-calculated.

In this paper, we consider a regression-based approach to algorithm selection, where each algorithm is modelled by a separate regression model that predicts its performance and the algorithm with the best predicted performance is chosen. See (Hutter et al. 2014) for an overview of regression techniques for performance prediction.

Regression-based methods are attractive for online algorithm selection because they naturally handle the incomplete online data: when a new instance is solved by a selected algorithm, the observed performance is used to update the regression model for that particular algorithm, while all others remain unchanged. Algorithm 2 describes the greedy online strategy that creates a selection mapping based on regression models and always greedily selects the algorithm with the best predicted performance. The greedy strategy can be used as input for the β of algorithm 1 (line 2) to create a fully specified online algorithm selection method. Note that algorithm 2 always completely retrains the regression models. If the regression models are updatable, they can be updated instead. Especially if the amount of training instances grows large, updating becomes preferable, because retraining time increases with the amount of training instances. For example, a Mondrian Forest (Lakshminarayanan, Roy, and Teh 2014) could be used as updatable random forest.

Algorithm 2 Greedy online algorithm selection strategy

1: Input: H
2: $\forall a \in A$: based on the relevant records in H, train regression model $m_a \colon \mathbb{R}^{|F|} \to \mathbb{R} \colon m_a(\varphi) = \hat{p}$, predicting algorithm a's performance on any feature vector φ.
3: Define $\lambda : \lambda(i) = \arg\max_{a \in A}(m_a(\Phi(i)))$
4: Return λ

Online Algorithm Selection as a Multi-armed Bandit

The online greedy strategy defined by combining algorithms 1 and 2 enables online learning: the regression models of the selected algorithms are expected to improve over time, resulting in more accurate selections. However, if during the training phase an algorithm was learned to be bad while in reality it performs well on some of the instances, this can be hard to correct, as the algorithm might never be selected. Methods that occasionally try algorithms not expected to be best can achieve better performance: the immediate loss in expected performance can be offset by later gains thanks to having learned a better selection model. An exploration vs. exploitation trade-off arises: one should not explore too much, but not too little either.

The exploration vs. exploitation trade-off is studied in reinforcement learning. In particular, online algorithm selection can be modelled as a contextual multi-armed bandit problem. In the standard, non-contextual, multi-armed bandit problem, one faces a stream of instances, and each instance must be assigned to one of the available 'arms'. Each arm has an unknown reward distribution, and whenever an instance is assigned to it, that distribution is sampled, resulting in an observed reward. The goal is to maximise the sum of rewards. The sum of rewards is maximised by efficiently trading off exploring (choosing arms not predicted to be best) and exploiting (focusing only on the arm predicted to be best). The contextual multi-armed bandit problem is a generalisation where every time an instance must be handled, a feature vector is presented, and where the underlying reward distribution of each arm depends on the values of those features. See (Li et al. 2010) for a formal definition of the contextual multi-armed bandit problem.

Online algorithm selection can be modelled as a contextual multi-armed bandit problem. Each algorithm corresponds to an arm, and solving an instance with an algorithm corresponds to assigning an instance to an arm. When an algorithm is selected, its underlying performance distribution for that instance is sampled and its performance is observed, which is equivalent to obtaining a reward. When selecting an algorithm to run on an instance, the instance's feature values are known, providing the context. Maximising the sum of rewards is equivalent to maximising the performance. This insight helps solve the exploration vs. exploitation trade-off for online algorithm selection, as proven methods from the field of multi-armed bandits can be used.

A simple strategy to assure continued exploration is ϵ-greedy, which selects the predicted best algorithm with probability $1 - \epsilon$ and with probability ϵ a random algorithm. It is parametrised by $\epsilon \in [0, 1]$. The online strategy corresponding to ϵ-greedy is similar to the one for greedy (Algorithm 2), except that the selection mappings it produces are stochastic: each algorithm has a probability $\frac{\epsilon}{|A|}$ of being selected. The remaining $1-\epsilon$ share is assigned to the predicted best algorithm.

ϵ-greedy does not distinguish between algorithms with high and low probability of being best, as it selects algorithms at random. However, if one is almost certain that an algorithm will not perform well on an instance, selecting it is unlikely to provide interesting information. More advanced techniques can make this distinction, by giving priority to algorithms that are more likely to be better. In this paper, we use a heuristic variant of the upper confidence bound approach (UCB). It calculates for each algorithm the predicted performance $\hat{p}$ and the standard deviation on this prediction sd. The algorithm with the highest value for $\hat{p} + sd \cdot \gamma$ is selected, where γ is a parameter that controls exploration. Higher γ values result in more exploration. An algorithm with predicted poor performance might be preferred over an algorithm with better predicted performance if the variance on its prediction is higher. This UCB variant is similar to the one in (Srinivas et al. 2009). The online strategy corresponding to UCB is equivalent to the greedy one of Algorithm 2, but with '$+sd * \gamma$' added to the prediction of the model on line 3.

Empirical Verification

In this section, we empirically demonstrate the potential of the proposed methodology. We also demonstrate that it is possible to learn useful algorithm selection models without any offline training data. The experiments reported here extend preliminary experiments on online algorithm selection published by us in (Degroote et al. 2016).

Experimental Setup

We use ASlib version 4.0 (Bischl et al. 2016a), a standard benchmark of algorithm selection scenarios. Each scenario consists of a set of instances, algorithms, and features. The performance of each algorithm on each instance is known. All features are numeric, and where feature values were missing we imputed them with the mean of all respective feature values in the scenario. The scenarios are diverse and stem from many problem domains, such as satisfiability, constraint programming and answer set programming. See the left side of Table 1 for an overview of the scenarios used and their characteristics. We did not consider for our experiments instances that were presolved, i.e. solved during the computation of the features, because for those instances no algorithm selection is necessary.

We excluded ASlib scenarios with performance that is not based on runtime, because online algorithm selection is most relevant when time is limited and runtime is optimised. We also excluded scenarios where the difference between the single best solver (always selecting the algorithm that is best on average) and the offline algorithm selection method's performance across all instances (using a 10-fold cross-validation) was very small ($< 5\%$ of the range of possible performance values), as in that case there is not much room for our online algorithm selection approach. These are scenarios CSP-2010, GRAPHS-2015, MAXSAT15-PMS-INDU, SAT11-INDU, SAT12-INDU, SAT12-RAND and SAT15-INDU. Complete results and all plots, including from the excluded scenarios, are available in the online appendix[1]. On excluded scenarios, online algorithm selection was observed to not be that useful, but neither did performance drop compared to the offline approach.

We evaluate performance with the PAR10 (penalised runtime 10) metric defined in ASlib, adding the cost to compute the features. With PAR10, performance is equal to the runtime when a solution is found within the scenario-dependent time limit, and for time-outs performance is set to ten times the time limit. Performance is normalised in relation to the single best solver (0) and virtual best solver (1) as $(\textit{observed} - \textit{singleBest})/(\textit{virtualBest} - \textit{singleBest})$. The single best solver is the (not necessarily unique) algorithm with best observed performance when averaged over all instances in the scenario. The virtual best solver is an oracle that assigns each instance to an algorithm with best observed performance for it. Note that normalised performance is negative when observed performance is worse than the single best solver.

The instances of each scenario were randomly split into three subsets: training, online and verification. The training set contains the complete performance data of all algorithms on the training instances and is used to initialise the regression models and the corresponding first selection mapping. Then the online instances are handled one by one, selecting an algorithm for each, and after every selection, the performance of (only) the selected algorithm is made available and the selection mapping can be retrained. The goal of the online strategy is to maximise the observed performance on the online instances, labelled "online performance".

The instances in the verification set are held out and do not provide any additional information to the online process. They are used to evaluate the quality of the selection models after processing all training and online instances and to evaluate how the models improve over time. All selections are greedy on the verification instances. This enables quantifying the quality of the selection model each strategy managed to learn. This is called "verification performance".

For a first set of experiments, we used 10% of the instances as training set, leaving 80% for the online set and 10% for the verification set. This simulates a setting with very little training data. Note that due to the differences in number of instances per scenario, the fixed 10% training data lets us validate the method for different numbers of training instances. For a second set of experiments, we did not use any training data at all, leaving 90% of the instances as online and 10% as verification. This simulates an extreme case of insufficient training data. All experiments were run ten times on different random training-online-verification partitions.

We considered three online strategies: greedy (algorithm 2), ϵ-greedy (eGreedy), and upper confidence bound (UCB) (both discussed in the previous section).

We compare the performance of our online methods to an offline approach trained only on the training instances, labelled 'offline'. This offers a baseline: if the online method works, it should outperform an offline method that starts from the same training data. As a second comparison, we

[1] https://bitbucket.org/HansDeg/socsmaterial

scenario	Algorithms	Instances	Features	Avg runtime (s)	random forest	2015/2017 winner
ASP-POTASSCO	11	1090	138	156	0.7213	0.7044
BNSL-2016	8	1179	86	2259	0.8328	*0.8444*
CPMP-2015	4	527	22	1165	0.2525	0.6777
CSP-MZN-2013	11	4642	155	1320	0.9113	n/a
MAXSAT12-PMS	6	876	37	921	0.7694	0.8007
MAXSAT-PMS-2016	19	601	37	781	0.4889	*0.5723*
MAXSAT-WPMS-2016	18	630	37	1090	0.7220	*0.9102*
PROTEUS-2014	22	2047	198	719	0.7860	0.8575
QBF-2011	5	1368	46	2064	0.8809	0.8423
QBF-2014	14	1254	46	530	0.7961	n/a
QBF-2016	24	825	46	667	0.7213	*0.5691*
SAT03-16_INDU	10	1967	483	1627	0.3763	*0.0084*
SAT11-HAND	15	296	115	3287	0.7820	0.6866
SAT11-RAND	9	600	115	2532	0.9385	0.9568
SAT12-ALL	31	1594	115	723	0.7793	0.7445
SAT12-HAND	31	762	115	855	0.7885	0.7377

Table 1: Left side: characterisation of the ASlib scenarios used in the experiments (number of algorithms, number of non-presolved instances, number of features and avg runtime per instance). Right side: comparison of the random forest regression method used in our experiments with the winners of the 2015 and 2017 ASlib competition, for scenarios used in those competitions (performance values as reported in the respective competition; best value if used in both. Random forest was evaluated in the same way as the competitors in the 2015 competition, but scenarios of the 2017 competition (italicised results) were evaluated using less training data). Performance is normalised to the single best solver (0) and virtual best solver (1); higher is better.

provide access to the complete data (the performance of all algorithms instead of only the performance of the selected algorithm) of any online instance after a selection for it has been made, and then retrain each algorithm's model based on that complete data. This method, labelled 'allData', is expected to be better than any online method because it has access to much more data than the online methods. This method offers an upper bound on the performance. However, the upper bound is non-strict, as it is technically possible for a model trained on less data to be better. Note that actually using this allData method would impose a big overhead in computation time, because of the additional algorithm runs required. The overhead can be quantified by multiplying the 'Algorithms' column with the 'Avg runtime' column of Table 1, subtracting 1 from the 'Algorithms' column (the selected algorithm is always run anyway).

We used random forests (Liaw and Wiener 2002) to create the regression models for all online and offline methods. Standard deviations were estimated with the Jackknife method (Sexton and Laake 2009). Random forests are a state-of-the-art method for algorithm selection (Hutter et al. 2014), used for example in the award-winning SATzilla system (Xu et al. 2008). The right side of Table 1 shows the performance of the random forest method we use compared to the performance of the winners of the algorithm selection competitions of 2015 (Kotthoff, Hurley, and O'Sullivan 2017) and 2017 (Lindauer, van Rijn, and Kotthoff 2017). The performance of the random forest models was determined with a 10-fold cross-validation. The systems used in the competitions were more sophisticated than our simple algorithm selection approach; they included feature selection and presolvers, which we do not consider. Still, as Table 1 shows, the results obtained by the random forest method we use are competitive, indicating that it is indeed a state-of-the-art method for the benchmark we consider.

The regression models were retrained every time 10 new data points were available. Retraining more frequently did not improve performance significantly, but did increase overhead. The time spent retraining regression models was negligible compared to algorithm runtimes.

We used R-packages llama (Kotthoff 2013), mlr (Bischl et al. 2016b), and batchtools (Lang, Bischl, and Surmann 2017) to run the experiments.

The ϵ of ϵ-greedy was set to 0.05 and the γ of UCB to 1. Parameter tuning experiments for the setting with 10% training data and 80% online data indicated that the best performance was observed for $\epsilon = 0$ and $\gamma = 0$. However, this would make both strategies equivalent to greedy, so we chose different values that ensure exploration.

Results

Table 2 shows the results for 10% training data. The online greedy approach consistently outperforms the offline approach in terms of online performance. It closes on average an additional 7.03% of the gap between the single best solver and the virtual best solver compared to the offline strategy. The greedy strategy always completely outperforms the exploring strategies (ϵ-greedy, UCB and random), which do not even outperform the offline strategy. Apparently they lose too much performance when exploring. The greedy strategy appears to be by far the best at solving the exploration vs. exploitation trade-off when a small amount of training data is available.

scenario	offline	greedy	ϵ-greedy	UCB	allData
ASP-POTASSCO	0.5120	0.4966	0.4500	0.3147	0.6987
BNSL-2016	0.6906	**0.7503**	0.6495	0.6737	0.7902
CPMP-2015	0.1758	**0.2145**	0.1522	**0.176**	0.2514
CSP-MZN-2013	0.8320	**0.8670**	0.7807	0.4407	0.9026
MAXSAT12-PMS	0.5247	**0.6127**	0.4456	0.3867	0.7048
MAXSAT-PMS-2016	0.1096	**0.2307**	-0.0797	-0.4016	0.4182
MAXSAT-WPMS-2016	0.0232	**0.2015**	-0.1382	-0.2227	0.4896
PROTEUS-2014	0.4183	**0.4268**	0.3632	0.3597	0.5380
QBF-2011	0.6676	**0.7438**	**0.6781**	0.6548	0.8179
QBF-2014	0.4388	**0.5427**	**0.4437**	0.3871	0.6865
QBF-2016	0.1182	**0.2168**	**0.1207**	-0.0570	0.5090
SAT03-16_INDU	-0.1087	**-0.0067**	**-0.0776**	-0.1237	0.1375
SAT11-HAND	0.2169	**0.2625**	0.2134	**0.2512**	0.5186
SAT11-RAND	0.8577	**0.8910**	0.8333	0.8369	0.9190
SAT12-ALL	0.4149	**0.4804**	0.3975	0.3341	0.6537
SAT12-HAND	0.2796	**0.3660**	**0.3377**	**0.3173**	0.6297
mean	0.3857	**0.4560**	0.3481	0.2705	0.6040

Table 2: Online performance (avg. performance on the online instances) with 10% training data. Performance is normalised in relation to the single best solver (0) and virtual best solver (1). Online strategies with results better than strategy 'offline' are bold.

Table 2 also shows that the offline strategy with 10% training data already consistently outperforms the single best solver (except on the SAT03-16_INDU scenario). This illustrates that a small number of training instances often suffices to perform useful offline algorithm selection.

Table 3 shows the verification performance, measuring how well the strategies managed to learn a good selection model. Recall that this verification performance is obtained by using each strategies' models at the end of the online phase to make greedy selections on the held-out verification instances (thus the exploring strategies no longer explore; only the quality of the models they managed to learn is evaluated). All online strategies learn better models than the offline strategy, which is as expected because they have access to more training data. Interestingly, greedy, which never explicitly explores, does a good job at learning. It closes on average an additional 13.66% of the gap between the single best solver and the virtual best solver compared to the offline strategy. UCB and ϵ-greedy do not perform better than greedy on average. This implies that in the setting with 10% training data, there is no benefit to exploring: the cost of exploration is high (as can be seen in Table 2) and no better models are learned.

The explanation for the good performance of greedy is likely that the initial training instances provided a sufficiently good baseline for the strategy to continue learning a good selection mapping. Recent theoretical results from the multi-armed bandit community show that this is indeed possible (Kannan et al. 2018), and a recent applied paper (Bietti, Agarwal, and Langford 2018), where contextual multi-armed bandit methods are evaluated on over 500 datasets, even states that 'across these experiments we show that minimizing the amount of exploration is a key design goal for practical performance. Remarkably, many problems can be solved purely via the implicit exploration imposed by the diversity of contexts'. Our results indicate that online algorithm selection is one of the problems that can be solved via only the implicit exploration imposed by the contexts (meaning the differences in instances encountered).

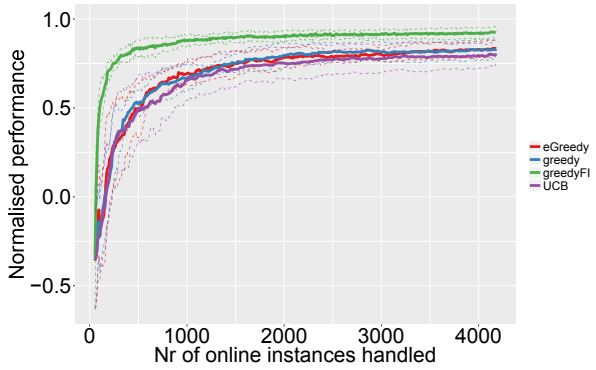

Figure 1: CSP-MZN-2013 (11 algorithms, 4642 instances, 155 features) without training data: verification performance over time (avg. performance on the verification instances, making greedy selection based on the regression models used at that time). Normalised to single best solver (0) and virtual best solver (1). Full lines show the means, dashed lines one standard deviation above and below.

We now discuss the results of the experiments without training data. The random forest implementation we used requires at least 5 training instances to initialise a regression model, so the online phase must start with round robin selec-

scenario	offline	greedy	ϵ-greedy	UCB	allData
ASP-POTASSCO	0.4794	0.4857	**0.4923**	**0.4896**	0.6446
BNSL-2016	0.7453	0.7685	0.7279	**0.8099**	0.8303
CPMP-2015	0.1569	0.2489	0.2082	0.1265	0.2082
CSP-MZN-2013	0.8247	0.8731	0.8714	0.8665	0.9114
MAXSAT12-PMS	0.5623	0.7789	0.7657	0.7064	0.7668
MAXSAT-PMS-2016	0.1194	0.3264	**0.3916**	-0.2765	0.2928
MAXSAT-WPMS-2016	0.2016	0.5350	0.5046	0.4925	0.7180
PROTEUS-2014	0.4000	0.4403	0.4340	0.4201	0.5216
QBF-2011	0.6902	0.8232	**0.8705**	**0.8402**	0.9105
QBF-2014	0.4373	0.6384	**0.6424**	0.5763	0.7915
QBF-2016	0.0594	0.2964	0.2211	0.1741	0.6653
SAT03-16_INDU	-0.0031	0.0894	0.0679	0.0678	0.2803
SAT11-HAND	0.2894	0.4636	**0.4674**	0.3835	0.6842
SAT11-RAND	0.8674	0.9429	0.9249	0.9092	0.9155
SAT12-ALL	0.3967	0.5329	0.5002	**0.5362**	0.7625
SAT12-HAND	0.3137	0.4811	**0.4982**	0.4514	0.7662
mean	0.4087	0.5453	0.5368	0.4734	0.6669

Table 3: Verification performance (avg. performance on the verification instances, making greedy selections based on the regression models learned during the online phase) with 10% training data. Performance is normalised in relation to the single best solver (0) and virtual best solver (1). Online strategies with results better than strategy 'greedy' are bold.

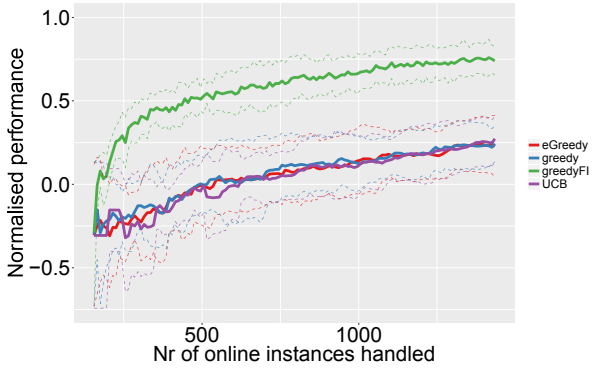

Figure 2: SAT12-ALL (31 algorithms, 1594 instances, 115 features) without training data: verification performance over time (avg. performance on the verification instances, making greedy selection based on the regression models used at that time). Normalised to single best solver (0) and virtual best solver (1). Full lines show the means, dashed lines one standard deviation above and below.

tion until each algorithm has solved 5 instances. After this, a first selection mapping can be initialised, which is unlikely to be good, but upon which an online strategy can improve. Figure 1 shows an example, plotting the verification performance over time for all online strategies and the allData strategy. As can be seen in the plot, the initial selection mappings, obtained after the round robin selections (based on 55 instances; 5 for each of the 11 algorithms), are much worse than the single best solver, but as more online instances are handled, they quickly start outperforming it. In this example the selection mappings learned by the online strategies even approach the performance of the selection mapping learned by the benchmark strategy that has access to all data (allData), despite the online strategies having access to only a fraction ($1/11$) of the data. Note also that the models learned by the online strategies all perform similarly. This illustrates that, on this scenario, the greedy strategy still does not learn models that are significantly worse than the strategies that explicitly explore, even without initial training data.

Table 4 shows the online and verification results for all scenarios when no initial training data is available. Having access to a lot of online instances is important here, because the system needs time to make up for the bad initial performance during the round-robin phase and the phase when the selection mappings perform poorly. As there is no offline data, the offline benchmark is excluded, because it cannot learn a model. In 9 of 16 scenarios, the greedy strategy without training data outperforms the single best solver, but on average it still does worse than the single best solver. However, when only the scenarios with more than 1000 instances are considered, the single best solver is outperformed in 6 out of 8 scenarios, and on average 19.68% of the gap between the single best solver and the virtual best solver is closed. Furthermore, as can be seen from the verification performances in Table 4, in all but one of the scenarios, greedy's selection mapping after processing all online instances outperforms the single best solver, closing on average (over all scenarios) 37.49% of the gap between single best solver and virtual best solver. This implies that, had the greedy strategy had access to more online instances, it would have outperformed the single best solver on all but one scenario, even if it would stop processing further online data; it just needed more time (more online instances) to make up

	Online performance				Verification performance			
scenario	greedy	ϵ-greedy	UCB	allData	greedy	ϵ-greedy	UCB	allData
ASP-POTASSCO	**0.1656**	**0.1154**	-0.0289	0.5580	0.3030	0.2821	**0.4376**	0.6936
BNSL-2016	**0.5524**	**0.4389**	**0.3583**	0.7071	0.7328	**0.7559**	0.6919	0.8368
CSP-MZN-2013	**0.6827**	**0.5988**	**0.6133**	0.8472	0.8207	**0.8212**	0.7946	0.9117
PROTEUS-2014	**0.1099**	**0.0504**	**0.0945**	0.4296	0.3674	0.3045	0.3357	0.5364
QBF-2011	**0.4228**	**0.3815**	**0.4031**	0.7374	0.709	**0.7114**	0.6766	0.9031
QBF-2014	**0.0341**	**0.0050**	-0.0457	0.5197	0.3437	**0.3917**	**0.379**	0.8038
SAT12-ALL	-0.0970	-0.1668	-0.1604	0.3884	0.2284	0.2160	**0.2598**	0.7289
SAT03-16_INDU	-0.2964	-0.2734	-0.3254	0.0743	0.0210	-0.0181	-0.0821	0.2804
CPMP-2015	**0.0825**	**0.0565**	**0.0318**	0.1931	0.2324	**0.3353**	**0.2630**	0.2704
MAXSAT12-PMS	**0.3145**	**0.1717**	**0.1803**	0.5650	0.7064	0.6938	0.6766	0.8383
MAXSAT-PMS-2016	-1.1673	-1.3210	-1.5840	-0.6199	0.1500	**0.3824**	-0.2705	0.6391
MAXSAT-WPMS-2016	-1.2317	-1.5829	-1.5596	-0.4930	0.2645	0.0744	0.1289	0.6950
QBF-2016	-0.4711	-0.5233	-0.6815	0.1465	-0.0062	-0.0621	-0.1092	0.6643
SAT11-HAND	-0.1188	-0.1386	-0.1645	0.2094	0.1633	**0.2224**	**0.2677**	0.6416
SAT11-RAND	**0.5180**	**0.5036**	**0.4808**	0.7637	0.8298	0.7826	**0.8555**	0.9153
SAT12-HAND	-0.1861	-0.2214	-0.1804	0.2577	0.1326	**0.2341**	**0.1675**	0.7450
mean	-0.0429	-0.1191	-0.1605	0.3303	0.3749	**0.3830**	0.3420	0.6940

Table 4: Online performance (avg. performance on the online instances) and verification performance (avg. performance on the verification instances, making greedy selections based on the regression models learned during the online phase) when there is no training data. Performance is normalised in relation to the single best solver (0) and virtual best solver (1). Online results (left part) better than the single best solver (> 0) are bold. Verification results (right part) better than greedy are bold. Scenarios above the horizontal line contain more than 1000 instances, below less.

for the influence of the initial poor performance on the average. The implication is that even without initial training instances, the greedy strategy consistently performs well.

Figure 2 shows the verification performance over time for the SAT12-ALL scenario, when no training data is available. On this scenario, our online approach did not outperform the single best solver (left side of table 4). However, it did manage to learn a selection mapping that outperforms the single best solver (right side of table 4), and if more online instances had been available, it would have outperformed it. The plot shows that on the first 155 instances (5 for each of the 31 algorithms), performance is very bad because random selections are made. Afterwards, a first model is initialised, which performs worse than the single best solver. The models then improve as more instances are handled. Starting at about 500 instances, the models of the online strategies make better predictions than the single best solver. As more instances are handled, the quality of the models improves further. By the end of the simulation the online strategies have closed over 20% of the gap between the single best solver and the virtual best solver.

To conclude the experimental results: greedy appears to be a robust and well-performing strategy for online algorithm selection, especially when a small amount of training data is available, but also when starting without training data. This is surprising at first sight, but lies in line with recent insights from contextual multi-armed bandit research, which show that a pure greedy approach is often preferable (Kannan et al. 2018; Bietti, Agarwal, and Langford 2018).

Conclusions and Future Work

In this paper, we considered the online algorithm selection problem. We modelled it as a contextual multi-armed bandit problem, and proposed a methodology based on constructing a regression model for each algorithm. Our methodology can process the kind of incomplete online data generated during the algorithm selection process, consisting of the performance of only the selected algorithm for each online instance. Our empirical validation on ASlib demonstrated that processing online data with the proposed methodology results in improved performance. A simple greedy online method that does not explicitly explore had the best overall performance; explicitly exploring does not seem worthwhile for online algorithm selection. The experiments also showed that even when no training data is available – a setting where offline algorithm selection does not work – online algorithm selection eventually learns to outperform the single best solver. In this setting as well, the simple greedy method performed best. Overall, we showed that the greedy strategy is a good choice for the online algorithm selection problem.

Future work includes investigating whether more advanced solutions to the contextual multi-armed bandit problem, such as LinUCB (Li et al. 2010) and ILoveToConBandits (Agarwal et al. 2014), can outperform the greedy method, and to investigate how the characteristics of an algorithm selection scenario influence the performance gained by applying online algorithm selection.

Acknowledgments

Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX (http://comex.ulb.ac.be).

References

Agarwal, A.; Hsu, D.; Kale, S.; Langford, J.; Li, L.; and Schapire, R. 2014. Taming the monster: A fast and simple algorithm for contextual bandits. In *International Conference on Machine Learning*, 1638–1646.

Bietti, A.; Agarwal, A.; and Langford, J. 2018. Practical evaluation and optimization of contextual bandit algorithms. *arXiv preprint arXiv:1802.04064*.

Bischl, B.; Mersmann, O.; Trautmann, H.; and Preuß, M. 2012. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, 313–320. ACM.

Bischl, B.; Kerschke, P.; Kotthoff, L.; Lindauer, M.; Malitsky, Y.; Fréchette, A.; Hoos, H.; Hutter, F.; Leyton-Brown, K.; Tierney, K.; et al. 2016a. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence* 237:41–58.

Bischl, B.; Lang, M.; Kotthoff, L.; Schiffner, J.; Richter, J.; Studerus, E.; Casalicchio, G.; and Jones, Z. M. 2016b. mlr: Machine Learning in R. *Journal of Machine Learning Research* 17(170):1–5.

Cicirello, V. A., and Smith, S. F. 2005. The max k-armed bandit: A new model of exploration applied to search heuristic selection. In *AAAI*, 1355–1361.

Degroote, H.; Bischl, B.; Kotthoff, L.; and De Causmaecker, P. 2016. Reinforcement learning for automatic online algorithm selection-an empirical study. In *ITAT 2016 Proceedings*, volume 1649, 93–101.

Gagliolo, M., and Schmidhuber, J. 2010. Algorithm selection as a bandit problem with unbounded losses. In *International Conference on Learning and Intelligent Optimization*, 82–96. Springer.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A Baseline for Building Planner Portfolios. In *ICAPS-2011 Workshop on Planning and Learning (PAL)*, 28–35.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An Economics Approach to Hard Computational Problems. *Science* 275(5296):51–54.

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

Kannan, S.; Morgenstern, J.; Roth, A.; Waggoner, B.; and Wu, Z. S. 2018. A smoothed analysis of the greedy algorithm for the linear contextual bandit problem. *arXiv preprint arXiv:1801.03423*.

Kotthoff, L.; Hurley, B.; and O'Sullivan, B. 2017. The ICON Challenge on Algorithm Selection. *AI Magazine* 38(2):91–93.

Kotthoff, L. 2013. Llama: leveraging learning to automatically manage algorithms. *arXiv preprint arXiv:1306.1031*.

Kotthoff, L. 2014. Algorithm Selection for Combinatorial Search Problems: A Survey. *AI Magazine* 35(3):48–60.

Lagoudakis, M. G., and Littman, M. L. 2000. Algorithm selection using reinforcement learning. In *ICML*, 511–518. Citeseer.

Lakshminarayanan, B.; Roy, D. M.; and Teh, Y. W. 2014. Mondrian forests: Efficient online random forests. In *Advances in neural information processing systems*, 3140–3148.

Lang, M.; Bischl, B.; and Surmann, D. 2017. batchtools: Tools for r to work on batch systems. *The Journal of Open Source Software* 2(10).

Li, L.; Chu, W.; Langford, J.; and Schapire, R. E. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, 661–670. ACM.

Liaw, A., and Wiener, M. 2002. Classification and regression by randomforest. *R news* 2(3):18–22.

Lindauer, M.; van Rijn, J. N.; and Kotthoff, L. 2017. Open algorithm selection challenge 2017: Setup and scenarios. In *Proceedings of the Open Algorithm Selection Challenge*, 1–7.

Malitsky, Y.; Mehta, D.; and O'Sullivan, B. 2013. Evolving Instance Specific Algorithm Configuration. In *Symposium on Combinatorial Search*.

O'Mahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and O'Sullivan, B. 2008. Using Case-based Reasoning in an Algorithm Portfolio for Constraint Solving. In *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science*.

Phillips, M.; Narayanan, V.; Aine, S.; and Likhachev, M. 2015. Efficient search with an ensemble of heuristics. In *IJCAI*, 784–791.

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

Sexton, J., and Laake, P. 2009. Standard errors for bagged and random forest estimators. *Computational Statistics & Data Analysis* 53(3):801–811.

Srinivas, N.; Krause, A.; Kakade, S. M.; and Seeger, M. 2009. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*.

Veerapen, N.; Maturana, J.; and Saubion, F. 2012. An exploration-exploitation compromise-based adaptive operator selection for local search. In *GECCO 14*, 1277–1284. ACM.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. Satzilla: portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research* 32:565–606.