Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Algorithm Selection via Ranking

Richard J. Oentaryo, Stephanus Daniel Handoko, and **Hoong Chuin Lau**
Living Analytics Research Centre, School of Information Systems
Singapore Management University, Singapore 178902
{roentaryo, dhandoko, hclau}@smu.edu.sg

Abstract

The abundance of algorithms developed to solve different problems has given rise to an important research question: How do we choose the best algorithm for a given problem? Known as algorithm selection, this issue has been prevailing in many domains, as no single algorithm can perform best on all problem instances. Traditional algorithm selection and portfolio construction methods typically treat the problem as a classification or regression task. In this paper, we present a new approach that provides a more natural treatment of algorithm selection and portfolio construction as a *ranking* task. Accordingly, we develop a *Ranking-Based Algorithm Selection* (RAS) method, which employs a simple polynomial model to capture the ranking of different solvers for different problem instances. We devise an efficient iterative algorithm that can gracefully optimize the polynomial coefficients by minimizing a ranking loss function, which is derived from a sound probabilistic formulation of the ranking problem. Experiments on the SAT 2012 competition dataset show that our approach yields competitive performance to that of more sophisticated algorithm selection methods.

Introduction

Over the years, a myriad of algorithms and heuristics have been developed to solve various problems and tasks. This brings about an important research question: How do we determine which algorithm is the best suited for a given problem? This issue has been pivotal in many problem domains, such as machine learning, constraint satisfaction, and combinatorial optimization. One of the most widely studied problems related to this issue is the propositional *satisfiability problem* (SAT) (Cook 1971), and a plethora of algorithms (i.e., *solvers*) have been devised to solve the SAT problem instances. Such solvers offer a wider range of practical use, as many problems in artifical intellgence and computer science can be naturally mapped into SAT tasks.

It is often the case, however, that one solver performs better at tackling some problem instances from a given class, but is substantially worse on other instances. Practitioners are thus faced with a challenging *algorithm selection* problem (Rice 1976): Which solver(s) should be executed to minimize some performance objective (e.g., expected runtime)? A popular answer is to evaluate the performance of each candidate solver on a representative set of problem instances, and then use only the solver that yields the best performance. We refer this as the *single-best* (SB) strategy. Unfortunately, this is not necessarily the best way, as different solvers are often complementary. That is, the SB strategy would ignore solvers that are not competitive on average but nonetheless give good performance on specific instances.

On the other hand, an ideal solution to the algorithm selection problems is to have an *oracle* that knows which solvers will perform the best on each problem instance. The oracle surely would give better results than the SB strategy, serving as the theoretical upper bound of the solvers' performance. Unfortunately, such perfect oracle is not available for SAT (or any other hard) problems, as it is hardly possible to know a solver's performance on a new instance exactly without actually executing it. This motivates the development of algorithm selection methods that serve as heuristic approximations of the oracle. A prime example is the empirical hardness model (Leyton-Brown, Nudelman, and Shoham 2002; Nudelman et al. 2004) adopted by SATZilla, an algorithm selection method that has won several annual SAT competitions (Xu et al. 2008; 2012).

Recently, it has been shown that we can exploit the complementarity of the solvers by combining them into an *algorithm portfolio* (Huberman, Lukose, and Hogg 1997; Gomes and Selman 2001). Such portfolio may include methods that pick a single solver for each problem instance (Guerri and Milano 2004; Xu et al. 2008), methods that makes online decisions to switch between solvers (Carchrae and Beck 2005; Samulowitz and Memisevic 2007), and methods that execute multiple solvers independently per instance, either in parallel, sequentially, or partly sequential/parallel (Gomes and Selman 2001; Gagliolo and Schmidhuber 2006; Streeter and Smith 2008; Kadioglu et al. 2011). A common trait among these methods is that they employ *regression* or *classification* methods to build an efficient predictor of a solver's performance for each problem instance, given the instance's features and solver's performance history (O'Mahony et al. 2008; Xu et al. 2012; Malitsky et al. 2013).

While showing successes to some extent, the contemporary regression- or classification-based algorithm selection methods are not designed to directly capture the notion of *preferability* among different solvers for a given problem instance. It is more natural to pose algorithm selection as a *ranking problem*. For instance, when constructing a sequential algorithm portfolio, we are usually interested in finding the *correct ordering* of the solvers so as to decide which solvers should be run first and which one later. Contemporary regression-based methods, such as (Xu et al. 2008)), typically use pointwise loss function (e.g., square loss), which is often biased toward problem instances with more data, that is solved cases in this context. Meanwhile, the classification-based methods, such as (Xu et al. 2012), does not warrant a unique ordering of the solvers, that is classification (voting) easily leads to ties between solvers, and it is not clear which one should be prioritized first.

Instead of further pursuing regression or classification approach, we take on a new interpretation of algorithm selection and portfolio construction as a ranking task. To realize this, we propose in this paper a *Ranking-based Algorithm Selection* (RAS) methodology, which learns the appropriate (unique) ordering of solvers so as to identify the top K best solvers for a given problem instance. To our best knowledge, RAS is the first approach that is designed to directly optimize a ranking objective function suitable for algorithm selection (and in turn portfolio construction) task.

We summarize our main contributions as follows:

- We develop a ranking polynomial model that can capture the rich, nonlinear interactions between problem instance and solver features. We then extend its use to model the ordering of solvers for a specific problem instance.
- We devise an efficient iterative learning procedure for optimizing a ranking loss function, which is derived from a sound probabilistic formulation of preferability among different solvers for a specific problem instance.
- We evaluate the efficacy of our RAS approach through extensive experiments on the SAT 2012 competition data. The results show that RAS outperforms the single-best strategy and gives competitive performance to that of SATZilla and random forest-based selection method.

Related Work

Algorithm selection has been studied in different contexts (Gomes and Selman 1997) and focused on methods that generate or manage a portfolio of solvers. SATZilla is a successful algoritm selection strategy that utilizes an empirical hardness model (Xu et al. 2008), and more recently a cost-sensitive random forest classifier (Xu et al. 2012). The algorithm selection in SATZilla aims at building a computationally inexpensive predictor of a solver's runtime or class label on a given problem instance based on features of the instance and the solver's past performance. This ability serves as a basis for building an algorithm portfolio that optimizes some objective function (e.g., percentage of instances solved).

Gagliolo and Schmidhuber (2006) proposed another runtime prediction strategy called GambleTA. The idea was to allocate time to each algorithm online in the form of a bandit problem. While approaches like SATZilla need offline training, this method does training online and perform an online selection of algorithms. CPHydra (O'Mahony et al. 2008) accommodates case-based reasoning to perform algorithm selection for runtime prediction. More recently, Malitsky *et al.* (2013) proposed a new algorithm selection method based on cost-sensitive hierarchical clustering model.

On a different track, several works have been developed that treat algorithm selection as a recommendation problem. These methods typically use collaborative filtering (CF) methods that adopt a low-rank assumption. That is, the algorithm performance matrix can be well-approximated by a combination of low-rank matrices. In (Stern et al. 2010), a Bayesian CF model was developed for algorithm selection. As a base model, it employs Matchbox (Stern, Herbrich, and Graepel 2009), a probabilistic recommender system based on bilinear rating CF model. In a similar vein, Misir and Sebag (2013) proposed a CF model based on matrix factorization. However, a major shortcoming of these approaches is their reliance on the low-rank assumption, which may not hold for algorithm performance data. Moreover, for matrix factorization-based approach, an extra effort is needed to build a separate surrogate model for handling new problem instances (i.e., cold-start issue) (Misir and Sebag 2013).

In this work, we develop the RAS methodology that deviates from existing approaches by treating algorithm selection (and portfolio construction) as a ranking task. At the core of RAS is a probabilistic ranking model that is simpler than sophisticated algorithm selection methods in, e.g., SATZilla, and does not rely on low-rank assumption or a separate mechanism for handling novel problem instances.

Proposed Approach

Polynomial Model

Our RAS approach utilizes as its base a polynomial model that captures the rich interaction between problem instance features and solver features. For an instance p and a solver s, we denote their feature vectors as $\vec{p} = [p_1, \ldots, p_i, \ldots, p_I]$ and $\vec{s} = [s_1, \ldots, s_j, \ldots, s_J]$, where I and J are the total numbers of instance and solver features, respectively. Using this notation, we define our polynomial model as follows:

$$f_{p,s}(\Theta) = \sum_{i=1}^{I}\sum_{j=1}^{J} p_i s_j \left(\alpha_{i,j} + \sum_{i' \neq i} p_{i'} \beta_{i,i',j}\right) \quad (1)$$

where Θ is the set of all model parameters (i.e., polynomial coefficients) $\alpha_{i,j}$ and $\beta_{i,i'j}$ that we want to learn. The term $f_{p,s}(\Theta)$ thus refers to the *preference score* of a given instance-solver pair (p, s). We also note that $p_i s_j$ and $p_i p_{i'} s_j$ can be regarded as the order-2 and order-3 *interaction terms* between the features of instance p and solver s respectively, and the polynomial coefficients $\alpha_{i,j}$ and $\beta_{i,i',j}$ are the corresponding *interaction weights*.

Without loss of generality, we consider a setting whereby we are given a set of numeric features to represent a problem instance, but there is no explicit feature provided about a given solver. For this, we construct *real-valued* feature vector $\vec{p} \in \mathbb{R}^I$ to describe an instance, and *binary* feature vector $\vec{s} \in \{0, 1\}^J$ for a solver. We use one-hot encoding scheme to construct the binary vector, i.e., $\vec{s} = [0, \ldots, 1, \ldots, 0]$, where the position of the "1" uniquely identifies a solver.

Ranking Desiderata

In this work, we propose a new take on algorithm selection as a ranking task. Let P and S be the sets of all problem instances and all solvers respectively. The algorithm selection task is to provide an instance $p \in P$ with a total ranking $>_p$ of all solvers $s \in S$. We note that a sound total ranking $>_p$ needs to fulfill several criteria (Rendle et al. 2009):

$$\forall s, s' \in S \colon s \neq s' \Rightarrow s >_p s' \vee s' >_p s \tag{2}$$

$$\forall s, s' \in S \colon s >_p s' \wedge s' >_p s \Rightarrow s = s' \tag{3}$$

$$\forall s, s', s'' \in S \colon s >_p s' \wedge s' >_p s'' \Rightarrow s >_p s'' \tag{4}$$

The formulae (2)–(4) refer to the *totality* (i.e., s and s' should be comparable), *anti-symmetry* (i.e., unless $s = s'$, s and s' should have different ranks), and *transitivity* properties (i.e., if s ranks higher than s' and s' ranks higher than s'', then s should rank higher than s'') properties, respectively (Davey and Priestley 2002; Rendle et al. 2009).

Under this notation, the RAS model essentially learns to rank solvers based on the following training set $\mathcal{D}$:

$$\mathcal{D} = \{(p, s, s') | s, s' \in S \wedge s >_p s' \wedge s \neq s'\} \tag{5}$$

where each triplet $(p, s, s') \in \mathcal{D}$ means that problem instance p prefers solver s over solver s'. Notably, our goal is to find the ordering of only the *top K solvers*, i.e., the K solvers with the lowest runtime on a given instance. Thus, our training data only include the triplets (p, s, s') such that s performs better than s' and s is among the top K solvers. This seemingly biased approach turns out to be beneficial in practice, as in the end we hardly care about the other solvers that are not in the top K list. Empirically, we also find that this approach gives faster training and better results compared to including *all* solvers s that are better than s'.

Probabilistic Foundation

We now present our probabilistic formulation of algorithm selection as ranking task. For a problem instance p, we define the *likelihood* $P(s >_p s'|\Theta)$ for ranking $>_p$ of solvers, and *prior* for the polynomial coefficients $P(\Theta)$. Under the Bayesian framework, finding the correct ranking of solvers s is equivalent to maximizing the posterior probability:

$$P(\Theta| >_p) = \frac{P(>_p |\Theta) P(\Theta)}{P(>_p)} \propto P(>_p |\Theta) P(\Theta) \tag{6}$$

where the denominator $P(>_p)$ is independent from Θ.

In this work, we shall assume that: 1) all problem instances p are independent from one another; and 2) the ordering of each solver pair (s, s') for an instance p is independent from that of every other pair. Using these assumptions, we can express the likelihood $P(>_p |\Theta)$ as:

$$\begin{aligned} P(>_p |\Theta) = \prod_{(p,s,s') \in P \times S \times S} & P(s >_p s'|\Theta)^{I[(p,s,s') \in \mathcal{D}]} \\ & \times (1 - P(s >_p s'|\Theta))^{I[(p,s,s') \notin \mathcal{D}]} \end{aligned} \tag{7}$$

where $I[x]$ is the indicator function (i.e., 1 if condition x is true, and 0 otherwise). In turn, due to the totality (2) and anti-symmetry (3) properties of a sound ranking, we can simplify the likelihood $P(>_p |\Theta)$ into:

$$P(>_p |\Theta) = \prod_{(p,s,s') \in \mathcal{D}} P(s >_p s'|\Theta) \tag{8}$$

Next, we define the probability that an instance p prefers solver s over solver s' as:

$$P(s >_p s'|\Theta) = \sigma(f_{p,s,s'}(\Theta)) \tag{9}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the logistic (sigmoid) function, and $f_{p,s,s'}(\Theta)$ refers to the *preferability* of solver s over s' for instance p. In order to satisfy all the three properties (2)–(4), we choose to decompose $f_{p,s,s'}(\Theta)$ into:

$$f_{p,s,s'}(\Theta) = f_{p,s}(\Theta) - f_{p,s'}(\Theta) \tag{10}$$

where $f_{p,s}(\Theta)$ is the preference score given by (1). That is, we define the preferability $f_{p,s,s'}(\Theta)$ as the difference between the preference scores of two solvers s and s'.

Lastly, we complete the Bayesian formulation by computing the prior $P(\Theta)$. In this work, we choose a Gaussian prior with zero mean and diagonal covariance matrix:

$$P(\Theta) \propto \exp\left[-\frac{\lambda}{2} \|\Theta\|^2\right] \tag{11}$$

where $\|\Theta\|^2 = \sum_i \sum_j \alpha_{i,j}^2 + \sum_i \sum_{i'} \sum_j \beta_{i,i',j}^2$ and λ is the inverse variance of the Gaussian prior distribution.

Combining the likelihood (8) and prior (11), we can then compute the posterior distribution $P(\Theta| >_p)$ as:

$$P(\Theta| >_p) \propto \prod_{(p,s,s')} \sigma(f_{p,s,s'}(\Theta)) \exp\left[-\frac{\lambda}{2} \|\Theta\|^2\right] \tag{12}$$

By taking the negative logarithm of the posterior, we can finally derive the optimization criterion $\mathcal{L}$ for our RAS method, hereafter called the *ranking loss*:

$$\mathcal{L} = -\sum_{(p,s,s')} \ln(\sigma(f_{p,s,s'}(\Theta))) + \frac{\lambda}{2} \|\Theta\|^2 \tag{13}$$

The optimal ranking $>_p$ can in turn be attained by finding the polynomial coefficients Θ that minimize $\mathcal{L}$. It is worth noting that $\mathcal{L}$ is a convex function and a global optima exists for such function. Also, the regularization term $\|\Theta\|^2$ serves to penalize large (magnitude) values of the coefficients $\alpha_{i,j}$ and $\beta_{i,i',j}$, thereby reducing the risk of data overfitting.

Learning Procedure

To minimize $\mathcal{L}$, we adopt an efficient *stochastic gradient descent* (SGD) procedure, which provides stochastic approximation of the batch (full) gradient descent method. The batch method leads to a "correct" optimization direction, but its convergence is often slow. That is, we have $O(|S| \times |S|)$ triplets in the training data $\mathcal{D}$, and so calculating the full gradient for each polynomial coefficient will be expensive.

In the SGD approach, instead of computing the full gradient of the overall loss $\mathcal{L}$ for all triplets, we update the polynomial coefficients based only on *sample-wise loss* $\mathcal{L}_{p,s,s'}$:

$$\mathcal{L}_{p,s,s'} = -\ln(\sigma(f_{p,s,s'}(\Theta))) + \frac{\lambda}{2} \|\Theta\|^2 \tag{14}$$

Algorithm 1 SGD Procedure for Ranking Optimization

```
Input: Training data $\mathcal{D}$, regularization parameter $\lambda$, learning rate $\eta$, maximum iterations $T_{max}$
Output: Polynomial coefficients $\Theta = \{\alpha_{i,j}\} \cup \{\beta_{i,i',j}\}$
1:  Initialize all $\alpha_{i,j}$ and $\beta_{i,i',j}$ to small random values
2:  repeat
3:      Shuffle the order of all triplets in $\mathcal{D}$
4:      for each triplet $(p, s, s')$ from the shuffled $\mathcal{D}$ do
5:          Compute $\sigma_{p,s,s'}(\Theta) \leftarrow \sigma\left(f_{p,s'}(\Theta) - f_{p,s}(\Theta)\right)$
6:          Compute residue $\delta \leftarrow \sigma_{p,s,s'} - 1$
7:          for each feature pair $(s_j, s'_j)$ of solvers $s, s'$ do
8:              Compute $\Delta s_j \leftarrow s_j - s'_j$
9:              for each feature $p_i$ of instance $p$ do
10:                 $\alpha_{i,j} \leftarrow \alpha_{i,j} - \eta\left[\delta p_i \Delta s_j + \lambda \alpha_{i,j}\right]$
11:                 for each feature $p_{i'}$ $(i' \neq i)$ of instance $p$ do
12:                     $\beta_{i,i',j} \leftarrow \beta_{i,i',j} - \eta\left[\delta p_i p_{i'} \Delta s_j + \lambda \beta_{i,i',j}\right]$
13:                 end for
14:             end for
15:         end for
16:     end for
17: until maximum iterations $T_{max}$
```

To minimize $\mathcal{L}_{p,s,s'}$, we can compute the gradient of the loss function with respect to each coefficient $\alpha_{i,j}$ and $\beta_{i,i',j}$:

$$\begin{aligned}\frac{\partial \mathcal{L}_{p,s,s'}}{\partial \alpha_{i,j}} &= \frac{\partial}{\partial \alpha_{i,j}}\left(-\ln(\sigma(f_{p,s,s'}(\Theta)))\right) + \lambda \alpha_{i,j} \\ &= \left(\sigma(f_{p,s,s'}(\Theta)) - 1\right) p_i \Delta s_j + \lambda \alpha_{i,j} \qquad (15)\\ \frac{\partial \mathcal{L}_{p,s,s'}}{\partial \beta_{i,i',j}} &= \frac{\partial}{\partial \beta_{i,i',j}}\left(-\ln(\sigma(f_{p,s,s'}(\Theta)))\right) + \lambda \beta_{i,i',j} \\ &= \left(\sigma(f_{p,s,s'}(\Theta)) - 1\right) p_i p_{i'} \Delta s_j + \lambda \beta_{i,i',j} \qquad (16)\end{aligned}$$

where $\Delta s_j = s_j - s'_j$ refers to the j^{th} feature difference between two solvers s and s'.

This leads to the following update formulae for the polynomial coefficients $\alpha_{i,j}$ and $\beta_{i,i',j}$:

$$\alpha_{i,j} \leftarrow \alpha_{i,j} - \eta\left[\delta p_i \Delta s_j + \lambda \alpha_{i,j}\right] \qquad (17)$$

$$\beta_{i,i',j} \leftarrow \beta_{i,i',j} - \eta\left[\delta p_i p_{i'} \Delta s_j + \lambda \beta_{i,i',j}\right] \qquad (18)$$

where $\eta \in [0, 1]$ is a (user-specified) learning rate, and $\delta = \sigma(f_{p,s,s'}(\Theta)) - 1$ is the (common) residue term.

Algorithm 1 summarizes our SGD learning procedure for minimizing the ranking loss $\mathcal{L}$ in the RAS framework. The parameter updates take place in lines 7–15, and can be done efficiently by "caching" the residue term δ before entering the update loop (see line 6). Further speed-up can be obtained by exploiting the *sparsity* of the (binary) solver feature vector $\vec{s}$, owing to the one-hot encoding scheme. Accordingly, the (worst) time complexity of the coefficient updates in each iteration is $O(|\mathcal{D}| \times I^2)$, where $|\mathcal{D}|$ is the total number of data samples/triplets (p, s, s') and I is the number of problem instance features. The memory requirement of our approach is also quite modest; we only need to store the model parameters Θ with a complexity of $O(I^2 \times J)$, where J is the number of solver features. We repeat the SGD procedure for a maximum of T_{max} iterations.

Prediction Phase

Upon completion of the SGD learning process, we would have obtained the polynomial coefficients Θ that minimize the ranking loss $\mathcal{L}$. Using the learned polynomial model, we can then predict the ranking of different solvers s for a new problem instance p. This can be done by computing the preference score $f_{p,s}(\Theta)$ for a given (p, s) pair. Accordingly, for different solvers s applied to problem instance p, we can rank them in a descending order of $f_{p,s}(\Theta)$, and pick the top K solvers as our recommendation.

Experiments

Dataset

For our experiments, we use the SAT 2012 datasets supplied by the UBC group[1], after SATZilla won the SAT 2012 Challenge. Table 1 summarizes the datasets. In total, there are six datasets, which are divided into three instance categories and two phase groups. The instance categories include handcrafted (HAND), industrial application (INDU), and random (RAND). The two phase groups differ mainly in the solver sets and the execution time limits. Solvers from the Phase 1 datasets were tested with a time limit of 1200 seconds (12S), while those from Phase 2 were run up to 5000 seconds (50S). Note that the solvers in Phase 2 are a subset of the solvers in Phase 1, i.e., solvers in Phase 2 are those that performed well and passed Phase 1 of the competition.

Each dataset is represented as an instance-solver matrix. An element in the matrix contains the runtime of a solver on a given problem instance, if the solver is able to solve the instance within the time limit. Otherwise, the element is treated as unsolved, and encoded as "1201" for Phase 1 and "5001" for Phase 2. Following Malitsky's setup[2], we removed all *unsolvable* problem instances, i.e., instances that could not be solved by any solver within the time limit. Last but not least, problem instances are characterized by the 125 features the UBC team proposed (option -base), while for solvers we used a binary feature representation obtained via one-hot encoding (as described in "Proposed Approach").

Evaluation

As our evaluation metrics, we use *Hit @ Top K* (denoted as Hit@K) and *Mean Average Precision @ Top K* (denoted as MAP@K), the well-known ranking metrics in information retrieval community (Baeza-Yates and Ribeiro-Neto 1999). Both metrics measure the prediction quality for each ranked list of solvers (sorted in descending order) that is returned for a problem instance p. The Hit@K metric refers to the number of problem instances successfully solved, assuming that at least one solver that can solve a given problem instance exists at the top K of the ranked list.

To obtain MAP@K, we first need to compute the Average Precision @ top K, i.e., AP$(p, K) = \frac{\sum_{i=1}^{K} P(i,p) \times rel(i)}{\sum_{i=1}^{K} rel(i,p)}$, where $rel(i, p)$ is a binary term indicating whether the i^{th} retrieved solver has solved problem instance p or not, and

[1]http://www.cs.ubc.ca/labs/beta/Projects/ SATzilla

[2]http://4c.ucc.ie/ ymalitsky/APBS.html

$P(i, p)$ is the precision (i.e., percentage of correctly predicted solved cases) at position i for instance p. We can then compute MAP@K by averaging all AP(p, K) over all problem instances, i.e., MAP@$K = \frac{1}{|P|} \sum_{p=1}^{|P|} \text{AP}(p, K)$, where $|P|$ is the total number fo problem instances. Chiefly, the MAP@K metric penalizes the incorrect (or tied) ordering of the solvers for each problem instance.

It is worth mentioning that, when $K = 1$, the Hit@K and MAP@K metrics are equal to the *count* and *percentage of solved instances* used in SATZilla's evaluation, respectively (Xu et al. 2012). In this work, we consider several baselines: 1) the oracle, i.e., the best solver for each problem instance, 2) the single-best solver, i.e., the algorithm that solves the largest number of problem instances in a given dataset, 3) the SATZilla, based on the results published in (Malitsky et al. 2013), and 4) the random forest (RF), which is an approximation of SATZilla's algorithm selection method. The RF approximation is meant to provide a reference for more general evaluation settings, particularly for $K > 1$. Notably, we can only extract MAP@1 from the published results in (Malitsky et al. 2013), and the result for $K > 1$ is not available. Thus, for $K > 1$, we shall use RF as our main baseline.

As our evaluation procedure, we adopt 10-fold cross validation. Specifically, we partition the problem instances (i.e., rows of the matrix) into 10 equal parts, and generate 10 pairs of training and testing data. For each fold, we enforce that 10% of the instances contained in the testing data do not appear in the training data. We then report the (micro-)average of MAP@K and the total sum of Hit@K over 10 folds. We set the parameters of our RAS method as follows: the learning rate $\eta = 10^{-2}$, regularization parameter $\lambda = 10^{-4}$, and maximum iterations $T_{max} = 25$. For the RF method, we set the number of trees to 99 as per (Xu et al. 2012), and configured it to be as close as possible to SATZilla.

Results

We first evaluate our approach based on the MAP@K metric. For this study, we varied K from 1 to 3. Table 2 compares the MAP@K scores of our RAS method with those of the RF and SATZilla (SZ) methods. It must be noted, though, that currently our goal is not to surpass SATZilla, but demonstrate that our alternative, simpler ranking-based method could attain sufficiently satisfactory results. Further improvements are certainly possible by extending the RAS method, which is beyond the scope of this paper.

From Table 2, we can make several observations:

- The MAP@K results obtained by RAS are generally comparable to—if not better than—those of the RF method, except for the RAND50S dataset. We can attribute this to the more irregular, nonlinear structure of the RAND50S dataset, which involves problem instances that were randomly generated. Similarly, RAS produces comparable or better results than SATZilla, except for the INDU50S and RAND50S datasets. Nevertheless, the overall results still show that our simpler RAS method can compete well with the more sophisticated RF and SATZilla methods.
- As K increases from 1 to 3, the MAP@K of our RAS method improves more steadily than that of the RF method (e.g., HAND12S and INDU12S). This suggests that RAS is able to produce a more consistent and unique ordering of the solvers. In contrast, non-ranking methods such as RF are more susceptible to producing tied rankings. As such, our RAS approach is more appropriate for algorithm portfolios that require a (proper) scheduling of the solvers, e.g., a sequential algorithm portfolio.

Table 1: Statistics of the SAT 2012 competition data

Phase	Dataset	#Instances	#Solvers	#Solved	#Unsolved
1	HAND12S	535	31	7,923	8,662
	INDU12S	952	31	17,860	11,652
	RAND12S	1,040	31	11,202	21,038
2	HAND50S	219	15	1,760	1,525
	INDU50S	241	18	3,564	774
	RAND50S	492	9	2,856	1,572

Table 2: MAP@K results for SAT 2012 competition data

	MAP@1			MAP@2		MAP@3	
Dataset	RAS	SZ	RF	RAS	RF	RAS	RF
HAND12S	0.811	0.810	0.813	0.856	0.844	0.860	0.832
INDU12S	0.926	0.872	0.894	0.945	0.892	0.943	0.874
RAND12S	0.973	0.929	0.969	0.976	0.976	0.975	0.976
HAND50S	0.890	0.895	0.868	0.911	0.890	0.917	0.903
INDU50S	0.905	0.921	0.904	0.934	0.929	0.930	0.928
RAND50S	0.923	0.982	0.976	0.940	0.980	0.938	0.978

RAS: Ranking-based algorithm selection; SZ: SATZilla; RF: Random forest

For our next set of experiments, we applied the predictions of both the RF and RAS methods to select the best K solvers for various SAT instances. For the RF method, we select solvers with the K lowest predicted runtimes, where ties are resolved arbitrarily. We show the number of solved instances using the top 1 solver (i.e., Hit@1) in Figure 1. Based on the results, we can conclude the following:

- Consistent with the MAP results, our RAS approach attains comparable, if not better, Hit@1 scores than those of the RF method, except again for the RAND category. The Hit@1 scores of the RAS method are also competitive to those of SATZilla's algorithm selection method.
- Our RAS approach is shown to be substantially better and closer to the oracle solution than the single-best baseline (and hence any other individual solvers), especially for the HAND12S, HAND50S, and RAND50S categories.

Finally, we analyzed the runtime "cummulative distribution function" (CDF) of the solvers selected by different algorithm selection methods (Xu et al. 2012). This is obtained by measuring the CPU time taken to solve each problem instance. Figure 2 presents the runtime CDFs of the solvers by plotting the number of solvable instances against the CPU time. As shown, our RAS approach compares favorably to the RF and single-best (SB) selection strategies in general[3]. That is, the results signify that RAS is able to select more efficient solvers—and thus solve more instances—within smaller CPU time as compared to the RF and SB methods.

[3]We are not able to include the runtime CDF of SATZilla, as we do not know which solvers are selected by it for each CPU time.

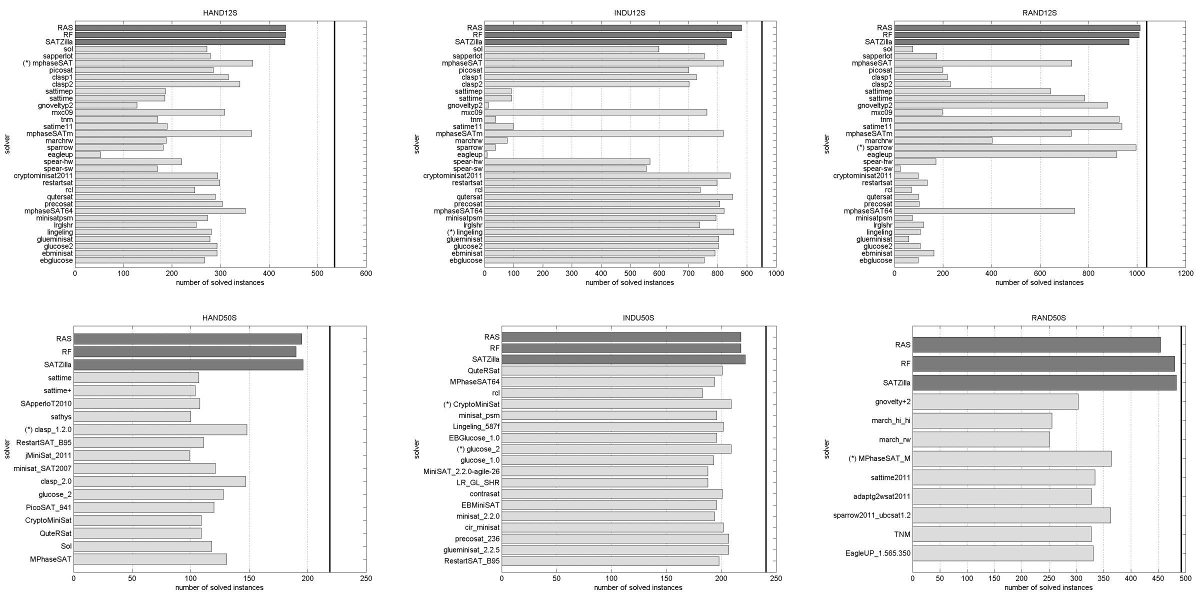

Figure 1: Hit@1 of various solvers on SAT 2012 data. (*) indicates the single-best solver, and the vertical line is the oracle.

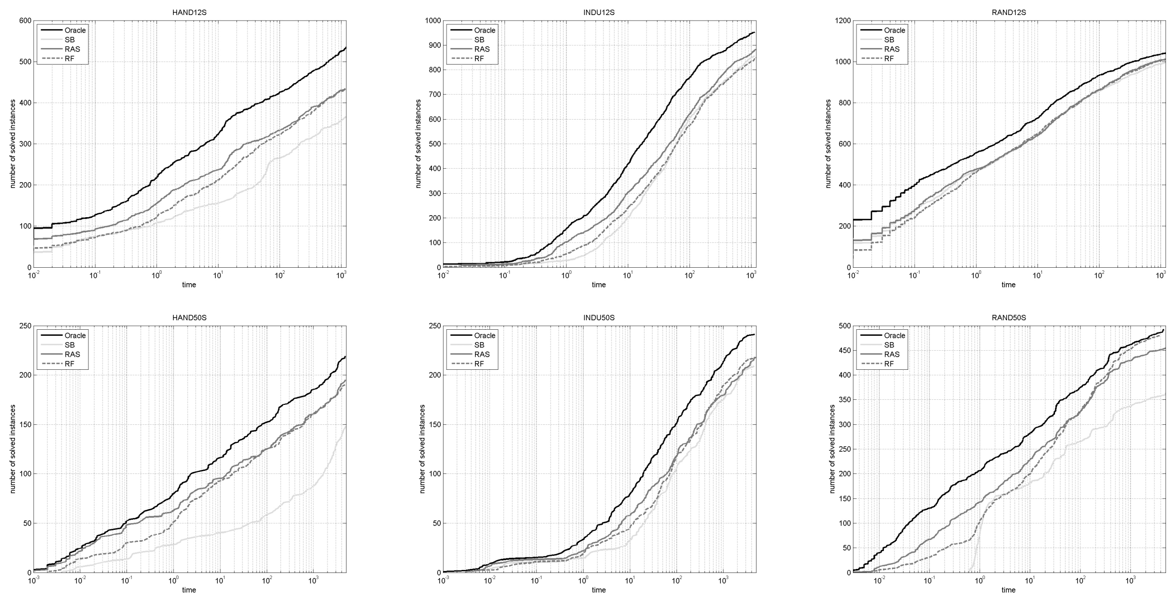

Figure 2: Runtime CDF of different algorithm selection methods on SAT 2012 data.

Conclusion

In this paper, we put forward a new perspective on algorithm selection and portfolio construction as a ranking task. Our prposed solution consists of a ranking-based algorithm selection (RAS) method that employs a polynomial model to capture the interactions between problem instance and solver features. To find the polynomial coefficents that reflect the proper ordering of different solvers for each problem instance, we devise an efficient iterative procedure that stems from a solid probabilistic formulation of the ranking task. The efficacy of our approach has been demonstrated via experiments on the SAT 2012 dataset. Moving forward, we wish to further improve the current RAS method, and extend its applications to other problem domains.

Acknowledgments

This work is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, Media Development Authority (MDA).

References

Baeza-Yates, R. A., and Ribeiro-Neto, B. 1999. *Modern information retrieval*. Boston, MA: Addison-Wesley.

Carchrae, T., and Beck, J. C. 2005. Applying machine learning to low-knowledge control of optimization algorithms. *Computational Intelligence* 21(4):372–387.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 151–158.

Davey, B. A., and Priestley, H. A. 2002. *Introduction to lattices and order*. Cambridge University Press.

Gagliolo, M., and Schmidhuber, J. 2006. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence* 47(3-4):295–328.

Gomes, C., and Selman, B. 1997. Algorithm portfolio design: Theory vs. practice. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 190–197.

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

Guerri, A., and Milano, M. 2004. Learning techniques for automatic algorithm portfolio selection. In *Proceedings of the European Conference on Artificial Intelligence*, 475–479.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An economics approach to hard computational problems. *Science* 275(5296):51–54.

Kadioglu, S.; Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Algorithm selection and scheduling. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 454–469.

Leyton-Brown, K.; Nudelman, E.; and Shoham, Y. 2002. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 556–572.

Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2013. Algorithm portfolios based on cost-sensitive hierarchical clustering. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 608–614.

Misir, M., and Sebag, M. 2013. Algorithm selection as a collaborative filtering problem. Technical report, INRIA.

Nudelman, E.; Devkar, A.; Shoham, Y.; and Leyton-Brown, K. 2004. Understanding random sat: Beyond the clauses-to-variables ratio. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 438–452.

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 Irish Conference on Artificial Intelligence and Cognitive Science*.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 452–461.

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

Samulowitz, H., and Memisevic, R. 2007. Learning to solve QBF. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 255–260.

Stern, D.; Herbrich, R.; Graepel, T.; Samulowitz, H.; Pulina, L.; and Tacchella, A. 2010. Collaborative expert portfolio management. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 179–184.

Stern, D. H.; Herbrich, R.; and Graepel, T. 2009. Matchbox: large scale online bayesian recommendations. In *Proceedings of the International Conference on World Wide Web*, 111–120.

Streeter, M., and Smith, S. F. 2008. New techniques for algorithm portfolio design. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 519–527.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32(1):565–606.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2012. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, 228–241.