Hybrid Ranking and Regression for Algorithm Selection

Jonas Hanselle(✉), Alexander Tornede, Marcel Wever, and Eyke Hüllermeier

Heinz Nixdorf Institute and Department of Computer Science, Paderborn University, Paderborn, Germany
{jonas.hanselle,alexander.tornede,marcel.wever,eyke}@upb.de

Abstract. Algorithm selection (AS) is defined as the task of automatically selecting the most suitable algorithm from a set of candidate algorithms for a specific instance of an algorithmic problem class. While suitability may refer to different criteria, runtime is of specific practical relevance. Leveraging empirical runtime information as training data, the AS problem is commonly tackled by fitting a regression function, which can then be used to estimate the candidate algorithms' runtimes for new problem instances. In this paper, we develop a new approach to algorithm selection that combines regression with ranking, also known as learning to rank, a problem that has recently been studied in the realm of preference learning. Since only the ranking of the algorithms is eventually needed for the purpose of selection, the precise numerical estimation of runtimes appears to be a dispensable and unnecessarily difficult problem. However, discarding the numerical runtime information completely seems to be a bad idea, as we hide potentially useful information about the algorithms' performance margins from the learner. Extensive experimental studies confirm the potential of our hybrid approach, showing that it often performs better than pure regression and pure ranking methods.

Keywords: Algorithm selection · Hybrid loss optimization · Combined ranking and regression

1 Introduction

Algorithm selection (AS) refers to the task of automatically selecting an algorithm from a set of candidate algorithms, which appears to be most suitable for a given instance of a problem class. A typical application of AS is the selection of solvers for computationally hard problems on a per-instance basis. Prominent examples of such problems include the Boolean satisfiability problem (SAT) [24] and the travelling salesman problem (TSP) [15]. Depending on the specific problem class, different criteria can be considered for assessing candidate algorithms. Especially important in this regard is an algorithm's efficiency measured in terms of its runtime.

On the basis of empirical runtime information, i.e., observations of runtimes on training instances, the AS problem is typically tackled by fitting regression functions, one per algorithm, to predict the runtime on new query instances [5,24]. Collecting the predictions for all algorithms, the presumably fastest one is then selected. Regression-based approaches proved to perform well in practice, often improving over the algorithm that performs best on average, also known as the single best solver (SBS), by orders of magnitude [24].

In spite of this practical success, one may wonder whether AS should indeed be tackled as a regression problem. First, since selection is eventually based on the *comparison* of the predicted runtimes, regression appears to be an unnecessarily difficult problem. Indeed, prediction errors could be tolerated as long as they do not change the *ranking* of the algorithms, or even less, the presumably best algorithm. From this point of view, one may also question symmetric loss functions like the squared error loss, as commonly used in regression. For example, if algorithms A and B have runtimes of, respectively, 10 and 13 min, the estimates 12 and 11 min are clearly better than 5 and 9 min in terms of the squared error. However, whereas the former switch the order of the two algorithms, the latter will still promote the faster algorithm, namely A.

These considerations may suggest to tackle AS as a ranking instead of a regression problem, and indeed, ranking methods from the field of preference learning have been used for constructing algorithm selectors [6,8,18,21,22]. Such models are learned from data comprised of problem instances together with respective rankings of the candidate algorithms. Data of that kind can often be collected more easily than precise numerical runtimes, which is another advantage of ranking methods. For example, if algorithm A finished before a given timeout is reached, while algorithm B did not, the preference $A \succ B$ can still be derived as training information, even if the concrete runtime of B is not known.

However, the ranking-based approach could be criticized as well, namely for ignoring potentially useful training information about the actual runtimes, if available, and the performance margins between algorithms. For example, a runtime of 2 min for algorithm A and 2.1 min for B leads to the same ranking $A \succ B$ as a runtime of 2 min for A and 200 min for B.

In this paper, we propose a hybrid approach to algorithm selection that combines both approaches, ranking and regression, hoping to benefit from the best of the two worlds: simplifying the learning task and solving the right problem while providing sufficiently detailed information such that concrete runtime information and margins between candidate algorithms are taken into account. To this end, we make use of hybrid loss functions [20]. Following a more formal description of the AS setting in the next section, our approach will be detailed in Sects. 3 and 4.

Our experimental evaluation in Sect. 5 confirms the potential of the proposed hybrid approach, which proves beneficial for several of the investigated scenarios. More specifically, optimizing our hybrid regression and ranking loss improves over optimizing the pure regression respectively ranking loss in terms of various metrics, eventually yielding a better performing algorithm selector.

2 Algorithm Selection

In the (per-instance) algorithm selection problem, first introduced by Rice [17], one is concerned with automatically selecting the most suitable algorithm from a set of candidate algorithms $\mathcal{A} = \{A_1, \ldots, A_K\}$ for a specific instance $I \in \mathcal{I}$ of an algorithmic problem class such as the Boolean satisfiability problem (SAT). Formally, the goal is to find a mapping $s\colon \mathcal{I} \to \mathcal{A}$, also referred to as *algorithm selector*, from a problem instance space $\mathcal{I}$ to the set of candidate algorithms $\mathcal{A}$, which optimizes a costly-to-evaluate performance measure $m\colon \mathcal{I} \times \mathcal{A} \to \mathbb{R}$ of interest. The arguably most relevant example of such a measure, which is also considered in this paper, is runtime. The optimal algorithm selector (the oracle) is defined as

$$s^*(I) := \operatorname*{arg\,min}_{A \in \mathcal{A}} \mathbb{E}\left[m(I, A)\right], \tag{1}$$

for $I \in \mathcal{I}$, where the expectation accounts for the potential randomness of the algorithm (and any other random effects causing the performance of A on I to be non-deterministic).

2.1 Existing Approaches

To evaluate the performance measure m, an algorithm normally needs to be run on a given problem instance. This makes an exhaustive search over the algorithm space $\mathcal{A}$ computationally intractable or at least extremely costly. To circumvent this problem, a surrogate model $\widehat{m} : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$ can be used to estimate the performance. Such models, which should be cheap to evaluate, are trained on data collected from previous algorithm runs. A feature extraction function $f\colon \mathcal{I} \to \mathbb{R}^d$ is used to compute d-dimensional feature representations of problem instances, which then allow for modeling the algorithm performance as functions of instance features. To keep the notation simple, we will not distinguish between I and $f(I)$ in the remainder of this paper; instead, we denote both a problem instance and its feature representation by I. Using such a model, the canonical algorithm selector will suggest the algorithm A with the lowest predicted runtime on the instance I:

$$\hat{s}(I) := \operatorname*{arg\,min}_{A \in \mathcal{A}} \widehat{m}(I, A) \tag{2}$$

A natural choice for $\widehat{m}$ is an algorithm-specific regression model $\widehat{m}_k\colon \mathcal{I} \to \mathbb{R}$, directly estimating the runtime achieved by an algorithm $A_k \in \mathcal{A}$ on a problem instance of interest $I \in \mathcal{I}$ [7].

Early work on such surrogates can be found in [12], where the authors tackle the winner determination problem for the CPLEX solver. They demonstrate that, under certain conditions, the hardness of an instance represented by features, i.e., the expected performance of an algorithm on that instance, can be learned using machine learning approaches. Both linear and nonlinear models (multivariate adaptive regression splines [5]) were successfully applied for modeling the hardness of an instance (with respect to the root mean squared error).

In one of the earlier versions of the well-known algorithm selection approach Satzilla [24], the authors leverage such empirical hardness models on a per-algorithm basis. To this end, they learn one linear model per algorithm using ridge regression, which estimates its performance for unseen instances based on associated features.

Similarly, restart strategies are selected based on conditional runtime prediction models in [6]. These models are inferred through ridge linear regression conditioned on the satisfiability of an instance. Instead of directly selecting an algorithm based on the predicted runtime, the authors of [4] use regression techniques in a more indirect way: The runtimes predicted by random forests are used to map instances into another feature space, in which k-nearest neighbor methods are then applied to make the final selection.

As already explained in the introduction, an accurate prediction of runtimes is a sufficient but not necessary condition for selecting the best performing algorithm. Actually, such a selection rather corresponds to a *classification* instead of a regression problem, with the algorithms playing the role of the classes. Training a classifier, however, has a number of disadvantages. For example, by looking at the best algorithm only, large parts of the training data would be ignored. Likewise, recommendations are not very informative in this setting, as they do not differentiate between the (presumably) non-optimal algorithms. Alternatively, the AS problem could also be tackled as a *ranking* task, which can be seen as a compromise between classification and regression.

Ranking methods have been developed in the field of preference learning. Specifically relevant in the context of AS is so-called *label ranking* (LR) [23]. Here, instances are associated with rankings over a set of choice alternatives, in our case algorithms. Thus, training data is of the form

$$(I, A_1 \succ \cdots \succ A_z) \in \mathbb{R}^d \times \mathcal{R}(\mathcal{A}) \,, \tag{3}$$

where $\mathcal{R}(\mathcal{A})$ is the set of all total orders on $\mathcal{A}$, and $A_i \succ A_j$ suggests that algorithm A_i performs better than algorithm A_j. What is then sought is a model $h : \mathbb{R}^d \rightarrow \mathcal{R}(\mathcal{A})$, which, given an instance $I \in \mathcal{I}$ (resp. its feature representation $f(I)$), predicts a ranking over the set of candidate algorithms $\mathcal{A}$. A recommendation can then be derived from that ranking, for example in the form of the top-1 or more generally top-k candidates. An example of label ranking applied to AS can be found in [6], where the authors infer rankings of collaborative filtering algorithms for instances of recommendation problems. Similarly, the authors of [8] use neural network based LR techniques to select meta-heuristics for travelling salesman problem instances.

In [22], *dyadic* approaches to ranking and regression are presented, which do not only leverage instance but also algorithm features, allowing one to select from an extremely large set of algorithms. A ranking method based on the Plackett-Luce model is shown to perform very well in a setting with many algorithms and very few training data, called *extreme algorithm selection*. Similarly, [14] leverage a ranking approach motivated from a Bayesian perspective, where the joint utility score of a pair of algorithms for an instance is defined in terms of the difference of the individual utility scores.

For a comprehensive and up-to-date survey of methods for algorithm selection, we refer to [10].

3 Hybrid Ranking and Regression Losses

There are several motivations for casting AS as a (label) ranking instead of a regression problem. As already explained, ranking not only appears to be the simpler task, but actually also the "right" problem. Indeed, the goal of AS is better reflected by a (non-symmetric) ranking than by a (symmetric) regression loss. Besides, precise numerical performance degrees are not always observable, for example when an algorithm is timed out, leading to missing or censored data in the case of regression, while preferences can still be derived. On the other hand, if precise performances are available, then considering only the qualitative part of the training information, namely the order relations, comes with a certain loss of information. For example, information about the algorithms' actual performance degrees, and the differences between them, may provide useful information about the reliability of a (pairwise) comparison.

These considerations suggest that both aspects should be taken into account when training an algorithm selector: predicted runtimes should first of all match the order of algorithms, and if possible, even be close to the actually observed runtimes. This could be accomplished by training the predictor with a hybrid loss function that combines both aspects into a single criterion.

Therefore, we propose the use of hybrid ranking and regression approaches for the AS problem. To this end, we model the performance of each algorithm in the candidate set $A_k \in \mathcal{A}$ in terms of a scoring function $v_k \colon \mathcal{I} \rightarrow \mathbb{R}$. As will be seen, the scoring function is in direct correspondence to the performance measure m_k, though not necessarily the same. The overall scoring model v is then given by $v(I, A_k) := v_k(I)$. Similar to the original combined regression and ranking approach presented by Sculley [20], our hybrid loss functions are based on a convex combination of a ranking term L_{RANK} that imposes ordering constraints between the individual predictions $v_k(I)$, $k \in [K] := \{1, \dots, K\}$, and a regression term L_{REG} that relates v_k to the actual runtime $m(I, A_k)$ achieved by algorithm A_k on the respective instance I.

3.1 Training Data

As training data, we assume (possibly incomplete or partial) information about the performance of algorithms on a set of training instances $I_1, \dots, I_N \in \mathcal{I}$:

$$\mathcal{D} := \Big\{ \, (I_n, m'_1(I_n), \dots, m'_K(I_n)) \, \Big\}_{n=1}^{N} , \tag{4}$$

where $m'_k(I_n)$ is information about the performance (runtime) of algorithm A_k on the instance I_n. Usually, $m'_k(I_n)$ is the runtime itself, however, the performance is also allowed to be unknown ($m'_k(I_n) = \perp$), for example because the algorithm has not been executed. Moreover, $m'_k(I_n)$ might be censored information about the true performance. A practically motivated example of such information is a timeout ($m'_k(I_n) = TO$): algorithm A_k has been run on I_n, but not till the end, because it did not terminate within a given time frame.

From the information about each of the N instances I_n, we construct a set of training examples R_n for a regression learner and a set of training examples P_n for a preference learner. For regression, if $m'_k(I_n) \neq \perp$, we include an example $(I_n, y_{k,n})$ which is normalized by the timeout T_{max}, namely $y_{k,n} = 1$ if $m'_k(I_n) = TO$ and $y_{k,n} = m'_k(I_n)/T_{max}$. In the case where $m'_k(I_n) = \perp$, no information about A_k is included in R_n.

The set P_n consists of pairwise preferences of the form $A_i \succ A_j$, suggesting that algorithm A_i performed better on I_n than algorithm A_j. We include such a preference, which we formally represent as (I_n, i, j), whenever one of the following conditions holds:

- $m'_i(I_n) \notin \{\perp, TO\}$, $m'_j(I_n) \notin \{\perp, TO\}$, $m'_i(I_n) < m'_j(I_n)$,
- $m'_i(I_n) \notin \{\perp, TO\}$, $m'_j(I_n) = TO$.

3.2 Loss Functions

As already said, the overall loss of a model v on a dataset $\mathcal{D}$ is a convex combination of a ranking and a regression loss:

$$L(\mathcal{D}, v) := \lambda L_{\text{RANK}}(\mathcal{D}, v) + (1 - \lambda) L_{\text{REG}}(\mathcal{D}, v) \ , \tag{5}$$

where the hyperparameter $\lambda \in [0, 1]$ can be tuned to balance the two objectives. Setting $\lambda = 0$ corresponds to a pure regression model, whereas $\lambda = 1$ results in a pure ranking model.

In general, any ranking loss L_{RANK} and any regression loss L_{REG} can be used to instantiate our generic framework. Here, we model the latter in terms of the mean squared error (MSE)

$$L_{\text{REG}}(R_n, v) := \frac{1}{|R_n|} \sum_{(I_n, y_{k,n}) \in R_n} \big(v_k(I_n) - y_{k,n}\big)^2 . \tag{6}$$

The overall loss $L_{\text{REG}}(\mathcal{D}, v)$ is then obtained by averaging (6) over all N training instances.

For ranking, we consider the squared hinge ranking loss given by

$$L_{\text{RANK}}(P_n, v) := \binom{|P_n|}{2}^{-1} \sum_{(I_n, i, j)} \ell\Big(\epsilon - v_i(I_n) + v_j(I_n)\Big) , \tag{7}$$

where $\epsilon \in \mathbb{R}^+$ is a margin and $\ell(x) = (\max\{0, x\})^2$. This loss function is a smooth convex approximation of the simple 0/1 loss and enforces a margin effect in the sense that, to have a loss of 0, the two predictions must be correctly ordered and have a distance of at least ϵ. Again, the loss on the entire data, $L_{\text{RANK}}(\mathcal{D}, v)$ is obtained by averaging over all N training instances. For computational reasons, since $L_{\text{RANK}}(P_n, v)$ contains a quadratic number of preferences, one may consider approximating this loss by sampling a subset of these preferences.

As an alternative to the squared hinge ranking loss (7), we also consider the following loss:

$$L_{\text{RANK}}(P_n, v) := \binom{|P_n|}{2}^{-1} \sum_{(I_n,i,j)} \ell\big(v_i(I_n), v_j(I_n)\big) , \tag{8}$$

with

$$\ell\big(x, y\big) = \log\big(\exp(-x) + \exp(-y)\big) + x . \tag{9}$$

This loss corresponds to the negative log-likelihood of observing a pairwise preference under the Plackett-Luce (PL) model for ranking data [13,16], which is commonly used in preference learning and label ranking [3].

4 Models and Optimization

For modeling the scoring functions $v_k\colon \mathcal{I} \to \mathbb{R}$, we consider three types of models, namely linear models, quadratic models, and feed-forward neural networks. Linear models define the score of an algorithm $A_k \in \mathcal{A}$ for a specific problem instance $I \in \mathcal{I}$ in terms of a linear combination of the instance features:

$$v_k(I) = \boldsymbol{w}_k^T I , \tag{10}$$

where $\boldsymbol{w}_k \in \mathbb{R}^d$ are the model parameters. To model quadratic relationships, a polynomial feature transformation $\phi\colon \mathbb{R}^d \to \mathbb{R}^{d(d+1)/2}$ is applied that maps the instance features to all monomials of degree 2. Consequently, the quadratic models are described by weight vectors $\boldsymbol{w}_k \in \mathbb{R}^{d(d+1)/2}$. We summarize all model parameters in a single parameter set $\boldsymbol{W} = \{\boldsymbol{w}_k \mid A_k \in \mathcal{A}\}$. Since all loss terms are convex, their convex combination (5) remains convex, and their minimization can be accomplished using gradient-based optimization methods. We apply the L-BFGS-B algorithm [2,25] for this task. To avoid overfitting, we employ weight decay by adding a regularization term $R(\boldsymbol{W}) = \gamma \sum_{k=1}^{K} \sum_{j=1}^{d} [\boldsymbol{w}_k]_j^2$, which can be adjusted by setting $\gamma \in \mathbb{R}$ to an appropriate value.

The neural network is given by a simple feed-forward architecture as illustrated in Fig. 1.

We adapt the training procedure from [19] for our setting of hybrid ranking and regression. For adjusting the model's weights $\boldsymbol{W}$, backpropagation is applied. The *Adam* optimizer [11] was selected as a gradient-based optimization method for minimizing the loss function. Regularization is implemented in terms of *early stopping*. Before the training procedure starts, a fraction of the original training dataset is selected as a validation set and removed from the training data. During the training, the model's loss on this validation set is computed periodically. A rising validation loss is an indicator of overfitting, thus the training procedure is stopped if an increase in the validation loss is observed for several consecutive checks. Afterwards, the model parameters are fixed to the set of weights that achieved the best validation loss during training.

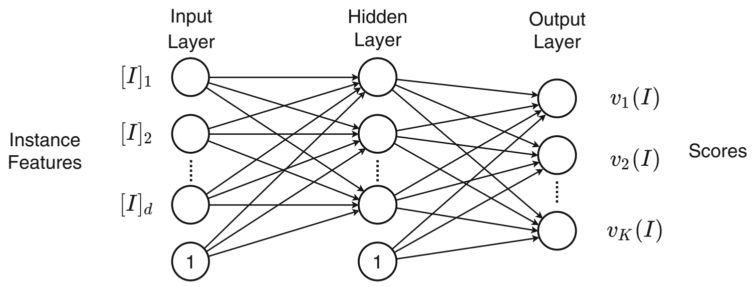

Fig. 1. Architecture of the neural network. Problem instance feature descriptions are fed into the input layer. The nodes of the fully connected hidden layer use a sigmoidal activation function in order to learn non-linear relationships. The nodes in the output layer use the identity as an activation function. Here, $[I]_j$ denotes the j-th entry of the instance feature vector.

5 Evaluation

In order to evaluate the performance of the proposed hybrid ranking and regression approach to the algorithm selection problem, we make use of the *ASlib* benchmark [1]. This benchmark contains several AS *scenarios*, which are collections of performance data of algorithms achieved on several problem instances. As we consider runtime as a selection criterion in the scope of this paper, we evaluated our approach using scenarios from the algorithmic problem domains of Boolean satisfiability (SAT), mixed integer programming (MIP), constraint satisfaction (CSP), and container pre-marshalling (CPMP).

5.1 Performance Metrics

For assessing the performance achieved by the proposed approaches, we consider both ranking measures as well as specific algorithm selection measures. Ranking measures quantify how well the ranking over algorithms according to their predicted performance corresponds to the ranking implied by their true performance. We represent a ranking of the algorithms $\{A_1, \ldots, A_K\}$ in terms of a mapping $\pi \colon [K] \to [K]$, such that $\pi(k)$ is the position of the algorithm A_k in the ranking—allowing for ties,[1] we may have $\pi(i) = \pi(j)$ for $i \neq j$. One prominent measure is the rank correlation coefficient Kendall's tau [9]. Given a ground truth ranking π and a predicted ranking $\hat{\pi}$, Kendall's τ is defined as

$$\tau(\pi, \hat{\pi}) = \frac{C - D}{\sqrt{(C + D + T_{\pi}) \cdot (C + D + T_{\hat{\pi}})}}\,, \tag{11}$$

where C is the number of correctly ordered pairs $((\pi(i) - \pi(j))(\hat{\pi}(i) - \hat{\pi}(j)) > 0)$, D is the number of incorrectly ordered pairs $((\pi(i) - \pi(j))(\hat{\pi}(i) - \hat{\pi}(j)) < 0)$,

[1] Ties are mainly caused by timeouts in the "ground truth" data but rarely occur in the predicted performances.

and T_π and $T_{\hat{\pi}}$ are the number of ties in ranking π and $\hat{\pi}$, respectively. Kendall's τ takes values in $[-1, 1]$, where $\tau(\pi, \hat{\pi}) = 1$ means that the rankings $\hat{\pi}$ and π are in perfect agreement and $\tau(\pi, \hat{\pi}) = -1$ the exact opposite (one of them is the reversal of the other one).

A widespread performance measure in the field of AS with respect to runtime is the penalized average runtime with a penalty factor of 10 (PAR10). Typically, the algorithms for the problem domains considered in this paper are not run for an indefinite amount of time until they eventually terminate, but are rather aborted after a predefined timeout is exceeded. The PAR10 score simply averages the runtime achieved by the selected algorithms for all problem instances of a scenario and accounts for timed out runs with 10 times the timeout as their runtime. We ignore feature costs, i.e., the runtime of the feature extraction function f, when computing PAR10 scores, as not all of the considered scenarios provide this information.

5.2 Evaluation Setup

The experimental results were obtained by conducting a 10-fold cross validation. In each fold, a fraction of 90% of a scenario's problem instances and the corresponding algorithm performances was used for training the algorithm selector, and the remaining 10% were used as a test set. For each scenario, we used the full set of features provided by ASLib [1]. Missing feature values were imputed with the feature's mean. Afterwards, feature values were standardized before training the models. Algorithm runtimes are given in terms of the PAR10 format, i.e., timed out runs are accounted for with 10-times the timeout. As the set of pairwise preferences P_n grows quadratically in the number of candidate algorithms, we approximate it by a sample $\hat{P}_n$ containing at most 5 pairwise algorithm comparisons for each instance. Should this number of comparisons not be available, we sample the maximum number of possible comparisons.

To evaluate the influence of the hyperparameter λ on the predictive performance, we conducted the experiments for $\lambda \in \{0.0, 0.1, \ldots, 1.0\}$. For training the linear and quadratic models, we set the regularization parameter $\gamma = 10^{-3}$ and ran the L-BFGS-B [2,25] algorithm for at most 100 iterations in order to minimize the loss functions. For the neural network-based approaches, we used the *Adam* [11] optimizer with a learning rate of $\eta = 10^{-3}$ for minimizing the loss functions and a batch size of 128. The architecture consists of a single hidden layer with 32 nodes, each using the sigmoid activation function $t \mapsto \frac{1}{1+e^{-t}}$. For early stopping, a fraction of 0.3 of the original training data is used as a validation set. We compute the loss on this validation set every 8 epochs and stop the training procedure if it increases for 8 consecutive checks. After the training, the model weights are set to the values for which the best validation loss was observed. If early stopping does not apply, the training procedure is stopped after a maximum number of 1,000 epochs. We evaluated the performance metrics for six independent runs on different random seeds and aggregated them by averaging the results.

The implementation of the proposed approaches including a documentation is provided on GitHub.[2]

5.3 Results

In the following, we discuss the results obtained by the experimental evaluation for all considered approaches, i.e., the two ranking loss functions in combination with the mean squared error as regression loss: the linear models (PL-LM, Hinge-LM), the quadratic models (PL-QM, Hinge-QM), and the neural networks (PL-NN, Hinge-NN). Figure 2 shows the average Kendall's τ rank correlation achieved by each of the proposed approaches for several values of λ. Recall that lower values of λ correspond to emphasizing the regression objective while higher values correspond to emphasizing the ranking objective. At first glance, we observe a tendency that larger values for λ lead to better rankings. Notably, however, in various cases the peak performance is not achieved for $\lambda = 1$, but rather for a proper compromise between ranking and regression. Consider for example the MIP-2016, SAT11-RAND or SAT11-HAND scenario, for which several of the proposed approaches achieve their peak performance for intermediate λ values.

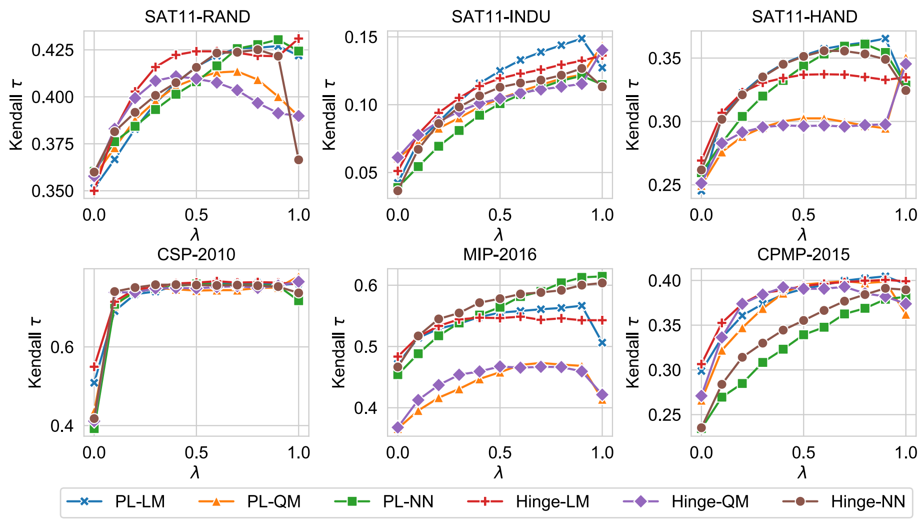

Fig. 2. Average Kendall's τ rank correlation coefficient achieved by the proposed approaches for different values of λ on a variety of AS scenarios. In multiple cases, an intermediate setting of $\lambda \in (0, 1)$ achieves a better rank correlation than pure regression ($\lambda = 0$) or pure ranking ($\lambda = 1$).

Figure 3 shows the PAR10 scores achieved by the proposed approaches. Again, we observe that neither pure regression nor pure ranking achieve the

[2] https://github.com/JonasHanselle/CoRRAS.

best performance consistently. Instead, a combination of the two appears to be favorable for most of the AS scenarios. Especially in the CSP-2010 and the MIP-2016 scenarios, the best performances, i.e. lowest PAR10 scores, are achieved for most of the proposed models when considering a hybrid ranking and regression loss.

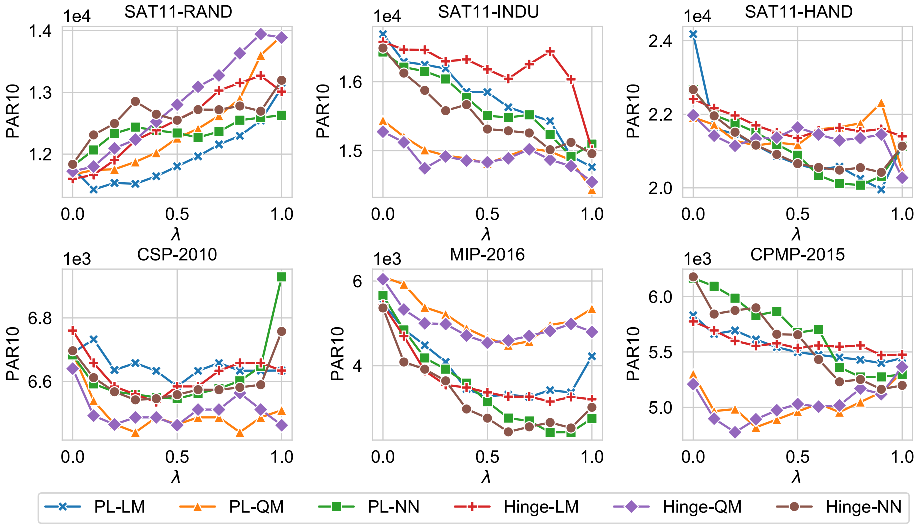

Fig. 3. Penalized average runtime achieved by selecting the top ranked algorithm predicted by the proposed models for each problem instance of the considered AS scenario.

Table 1 shows the number of scenarios for which a pure regression approach ($\lambda = 0$), a pure ranking approach ($\lambda = 1$), or a hybrid ranking and regression approach ($\lambda \in \{0.1, \ldots, 0.9\}$) achieves the best performances according to Kendall's τ and the PAR10 score. Regarding the rank correlation, unsurprisingly none of the proposed models achieved the best performance with the pure regression setting. The hybrid ranking and regression results are either on par with pure label ranking results or ahead of them. With respect to the PAR10 scores, hybrid regression and ranking performs the best for all model-loss combinations. Overall, for the majority of model-scenario combinations, a hybrid regression and ranking approach performs the best. While setting the hyperparameter λ to an intermediate value yields promising results, we could not reliably identify an optimal default value for this parameter. Instead, as can be seen in the plots in Figs. 2 and 3, the value for which the best performance is achieved depends both on the model and the scenario at hand.

Table 1. Number of scenarios for which each configuration achieved the best (average) performance according to Kendall's τ coefficient resp. PAR10 score. Recall that $\lambda = 0$ means pure regression, $\lambda \in (0, 1)$ a hybrid approach, and $\lambda = 1$ pure ranking.

Model	τ			PAR10		
	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
PL-LM	0	**6**	0	0	**5**	1
PL-QM	0	**3**	**3**	1	**3**	2
PL-NN	0	**4**	2	1	**5**	0
Hinge-LM	0	**4**	2	1	**4**	1
Hinge-QM	0	**3**	**3**	1	**3**	2
Hinge-NN	0	**5**	1	1	**4**	1

6 Conclusion

In this paper, we advocated the use of hybrid ranking and regression for the algorithm selection problem, mainly with the objective to tackle the "right" problem — which is selection, or, more generally, ranking — while not losing potentially useful numerical information about observed performances (runtimes). The proposed framework is built upon optimizing combined loss functions that take both regression and ranking criteria into account. We investigated three classes of models for estimating algorithm performances, namely linear models, quadratic models, and non-linear models in the form of neural networks. The results obtained by our experimental evaluation confirm that considering both ranking and regression objectives often leads to better algorithm choices than solely relying on one of the two objectives.

The proposed approaches rely on minimizing a convex combination of a ranking and a regression loss function. We investigated the squared hinge ranking loss and a ranking loss based on the Plackett-Luce model in combination with the mean squared error as a regression loss. In future work, we plan to further elaborate on suitable hybrid losses and to investigate the performance of other combinations. Of particular interest are regression methods for censored data, as these allow for modeling timeouts in a theoretically sound way. Another important question concerns the influence of the hyperparameter λ, which balances the regression and the ranking objectives. As we did not observe a suitable default value, it would be interesting to identify properties of algorithm selection scenarios that seem to influence the optimal choice of this parameter, i.e., which allow for deciding which of the two objectives, regression or ranking, should be emphasized more.

Acknowledgements. This work was supported by the German Federal Ministry of Economic Affairs and Energy (BMWi) within the "Innovationswettbewerb Künstliche Intelligenz" and the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901/3 project no. 160364472).

The authors also gratefully acknowledge support of this project through computing time provided by the Paderborn Center for Parallel Computing (PC^2).

References

1. Bischl, B., et al.: Aslib: A benchmark library for algorithm selection. Artif. Intell. **237**, 41–58 (2016)
2. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. **16**(5), 1190–1208 (1995)
3. Cheng, W., Dembczynski, K., Hüllermeier, E.: Label ranking methods based on the Plackett-Luce model. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21–24, 2010, Haifa, Israel, pp. 215–222. Omnipress (2010)
4. Collautti, M., Malitsky, Y., Mehta, D., O'Sullivan, B.: SNNAP: Solver-based nearest neighbor for algorithm portfolios. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) ECML PKDD 2013. LNCS (LNAI), vol. 8190, pp. 435–450. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40994-3_28
5. Friedman, J.H.: Multivariate adaptive regression splines. Ann. Stat., 1–67 (1991)
6. Haim, S., Walsh, T.: Restart strategy selection using machine learning techniques. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 312–325. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_30
7. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artif. Intell. **206**, 79–111 (2014)
8. Kanda, J., Soares, C., Hruschka, E., de Carvalho, A.: A meta-learning approach to select meta-heuristics for the traveling salesman problem using MLP-based label ranking. In: Huang, T., Zeng, Z., Li, C., Leung, C.S. (eds.) ICONIP 2012. LNCS, vol. 7665, pp. 488–495. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34487-9_59
9. Kendall, M.G.: The treatment of ties in ranking problems. Biometrika **33**(3), 239–251 (1945)
10. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithm selection: Survey and perspectives. Evol. Comput. **27**(1), 3–45 (2019)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, Conference Track Proceedings (2015)
12. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 556–572. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46135-3_37
13. Luce, R.D.: Individual choice behavior. John Wiley, Oxford, England (1959)
14. Oentaryo, R.J., Handoko, S.D., Lau, H.C.: Algorithm selection via ranking. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25–30, 2015, Austin, Texas, USA, pp. 1826–1832 (2015)
15. Pihera, J., Musliu, N.: Application of machine learning to algorithm selection for TSP. In: 26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10–12, pp. 47–54. IEEE Computer Society (2014)
16. Plackett, R.L.: The analysis of permutations. J. Roy. Stat. Soc. Ser. C (Appl. Stat.) **24**(2), 193–202 (1975)

17. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
18. de Sá, C.R., Soares, C., Knobbe, A.J., Cortez, P.: Label ranking forests. Exp. Syst. **34**(1) (2017)
19. Schäfer, D., Hüllermeier, E.: Dyad ranking using Plackett-Luce models based on joint feature representations. Mach. Learn. **107**(5), 903–941 (2018). https://doi.org/10.1007/s10994-017-5694-9
20. Sculley, D.: Combined regression and ranking. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25–28, pp. 979–988 (2010)
21. Tornede, A., Wever, M., Hüllermeier, E.: Algorithm selection as recommendation: From collaborative filtering to dyad ranking. In: CI Workshop, Dortmund (2019)
22. Tornede, A., Wever, M., Hüllermeier, E.: Extreme algorithm selection with dyadic feature representation. CoRR abs/2001.10741 (2020)
23. Vembu, S., Gärtner, T.: Label ranking algorithms: A survey. In: In: Fürnkranz J., Hüllermeier E. (eds.) Preference Learning, pp. 45–64. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14125-6_3
24. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. **32**, 565–606 (2008)
25. Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. ACM Trans. Math. Softw. **23**(4), 550–560 (1997)