Machine Learning for Online Algorithm Selection under Censored Feedback

Alexander Tornede, [1] **Viktor Bengs,** [2] **Eyke Hüllermeier** [2]

[1]Department of Computer Science, Paderborn University
[2]Institute for Informatics, LMU Munich
alexander.tornede@upb.de, viktor.bengs@lmu.de, eyke@lmu.de

Abstract

In online algorithm selection (OAS), instances of an algorithmic problem class are presented to an agent one after another, and the agent has to quickly select a presumably best algorithm from a fixed set of candidate algorithms. For decision problems such as satisfiability (SAT), quality typically refers to the algorithm's runtime. As the latter is known to exhibit a heavy-tail distribution, an algorithm is normally stopped when exceeding a predefined upper time limit. As a consequence, machine learning methods used to optimize an algorithm selection strategy in a data-driven manner need to deal with right-censored samples, a problem that has received little attention in the literature so far. In this work, we revisit multi-armed bandit algorithms for OAS and discuss their capability of dealing with the problem. Moreover, we adapt them towards runtime-oriented losses, allowing for partially censored data while keeping a space- and time-complexity independent of the time horizon. In an extensive experimental evaluation on an adapted version of the ASlib benchmark, we demonstrate that theoretically well-founded methods based on Thompson sampling perform specifically strong and improve in comparison to existing methods.

1 Introduction

Algorithm selection (AS) considers the automatic selection of an algorithm most suitable for solving an instance of an algorithmic problem class, such as the boolean satisfiability problem (SAT) or the traveling salesperson problem (TSP). Suitability is often quantified in terms of a loss function (or performance measure), such as solution quality or the algorithm's runtime — we shall focus on the latter in the remainder of this work. AS is motivated by the phenomenon of performance complementarity, roughly stating that the best algorithm within a pool of candidate algorithms will typically vary from instance to instance (Wolpert and Macready 1997; Kerschke et al. 2019).

Considering AS in an online setting, i.e., *online* algorithm selection (OAS), the problem can be posed as an iterative decision making problem, where instances arrive and decisions have to be made in an online manner. This setting suggests modeling the task as a contextual multi-armed bandit problem (Chu et al. 2011). In contrast to standard AS, one usually does not assume an upfront training dataset, consisting of loss function measurements for some of the algorithms on some training instances. Instead, in the online setting, the algorithm is provided feedback in the form of an evaluation of the loss function for the selected algorithm on the current instance, after the selection has been made.

In practice, algorithm runtime distributions often exhibit a heavy-tail property (Gomes, Selman, and Crato 1997), meaning that some algorithms can take extremely long to terminate on some instances. To avoid an online AS system from stalling, it is common to run algorithms with a timeout after which the algorithm is forcefully terminated if it did not solve the instance before. Correspondingly, one may end up with unsolved instances and *right-censored* runtimes (Kleinbaum and Klein 2010): although the true runtime is not known precisely, it is known to exceed the timeout. Needless to say, learning algorithms should try to exploit such censored data, which nevertheless provide a kind of "weak supervision". Although different forms of the OAS problem have been studied quite intensively in the literature (cf. Section 3), censoring has hardly been considered so far.

To alleviate this situation, we revisit well-known linear contextual bandit algorithms for the OAS problem under censored data and and discuss their weaknesses. Thereupon, we present theoretically grounded adaptations of these algorithms, incorporating improvements toward learning from censored data and employing selection criteria tailored toward runtime-based loss functions from the field of AS. In an extensive experimental study, we show that our approaches improve in terms of performance upon comparable existing approaches while featuring a time- and space-complexity independent of the time horizon. All code including detailed documentation and the technical appendix can be found on Github (cf. Section 7).

2 The Online Algorithm Selection Problem

The online algorithm selection problem is an iterative decision making problem, which comprises a problem instance space $\mathcal{I}$ featuring instances of an algorithmic problem class, and a set of candidate algorithms $\mathcal{A}$, which can solve such instances. The instances arrive iteratively over time, so that, at each time step t, an algorithm $s(h_t, i_t) = a_t \in \mathcal{A}$ has to be chosen by the *algorithm selector*

$$s : \mathcal{H} \times \mathcal{I} \to \mathcal{A} \tag{1}$$

which decides about the algorithm used to solve the instance $i_t \in \mathcal{I}$. Here, $h_t \in \mathcal{H}$ denotes the history of the selection process, consisting of triplets $\{(i_k, a_k, l_k)\}_{k=1}^{t-1}$ informing about the problem instances encountered so far, the algorithms that have been applied, and the corresponding evaluations in terms of losses $l_k = l(i_k, a_k)$ determined by a loss function $l : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$. This process either continues ad infinitum (unlimited horizon) or ends at some final time step T (final horizon). Consequently, the goal is to minimize the (average) loss achieved over the course of time, which, for the final horizon case, means to minimize

$$\mathcal{L}(s) = T^{-1} \sum\nolimits_{t=1}^{T} l(i_t, s(h_t, i_t)) \ . \tag{2}$$

In light of this, the optimal selector, called oracle or *virtual best solver* (VBS), is defined as

$$s^*(h_t, i_t) := \arg\min\nolimits_{a \in \mathcal{A}} \mathbb{E}[l(i_t, a) \mid h_t] \,, \tag{3}$$

where the expectation accounts for possible randomness in the application of algorithm a.

To approximate the VBS, any intelligent selection strategy has to leverage the historical data and perform some kind of *online learning*. This is a challenging task, especially as it involves the well-known exploration-exploitation dilemma of sequential decision making: The learner is constantly faced with the question of whether to acquire new information about the effectiveness of hitherto less explored algorithms, possibly improving but also taking the risk of large losses, or rather to exploit the current knowledge and choose algorithms that are likely to yield reasonably good results.

One way of constructing an OAS s is to learn, based on the history h, a (cheap-to-evaluate) surrogate loss function $\widehat{l}_h : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$ and invoke the principle of "estimated" loss minimization:

$$s(h, i) := \arg\min\nolimits_{a \in \mathcal{A}} \widehat{l}_h(i, a) \,. \tag{4}$$

For this purpose, we assume instances $i \in \mathcal{I}$ to be representable in terms of d-dimensional feature vectors $f(i) \in \mathbb{R}^d$, where $f : \mathcal{I} \to \mathbb{R}^d$ is a feature map. In the case of SAT, for example, features could be the number of clauses, the number of variables, etc.

Due to the online nature of the problem, it is desirable that OAS approaches have a time- and space-complexity independent of the time-horizon. In particular, memorizing all instances (i.e., entire histories h_t) and constantly retraining in batch mode is no option. Moreover, from a practical point of view, decisions should be taken quickly to avoid stalling an AS system.

2.1 Censored Runtimes

A loss function of specific practical relevance is the runtime of an algorithm, i.e., the time until a solution to a problem instance is found. However, in domains like combinatorial optimization, runtimes may exhibit a heavy-tail distribution, i.e., some algorithms may run unacceptably long on some instances (Gomes, Selman, and Crato 1997). This is why algorithms are usually executed under time constraints in the form of an upper bound on the runtime (a "cutoff") $C \in \mathbb{R}$. If an algorithm does not terminate until C, it is forcefully terminated, so as to avoid blocking the system. The instance is then treated as unsolved.

To account for unsolved instances, the loss is augmented by a penalty function $\mathcal{P} : \mathbb{R} \to \mathbb{R}$:

$$l(i, a) = m(i, a) 1_{[\![m(i,a) \leqslant C]\!]} + \mathcal{P}(C) 1_{[\![m(i,a) > C]\!]} \,, \tag{5}$$

where $1_{[\![\cdot]\!]}$ is the indicator function and $m : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$ returns the true (stochastic) runtime of an algorithm a on an instance i. Formally, when selecting algorithm a, either the runtime $m(i, a)$ is observed, if $m(i, a) \leqslant C$, or a penalty $\mathcal{P}(C)$ due to a *right-censored* sample (Kleinbaum and Klein 2010), i.e. $m(i, a) > C$, while the true runtime $m(i, a)$ remains unknown. With $\mathcal{P}(C) = 10C$, (5) yields the well-known *PAR10* score.

Previous work has shown that treating such right-censored data correctly is important in the context of standard (offline) AS (Xu et al. 2007; Tornede et al. 2020; Hanselle et al. 2020, 2021) and algorithm configuration (AC) (Hutter, Hoos, and Leyton-Brown 2011; Eggensperger et al. 2018, 2020). In the online setting, this might be even more critical, because the data does not only influence the learning but also the (active) sampling strategy of the AS.

The simplest approach for dealing with censored samples is to ignore them all together, which, however, causes an undesirable loss of information. Another simple strategy is imputation. For example, in the case of the *PAR10* score, censored samples are commonly replaced by the cutoff time C or its multiplicity $10\,C$. Obviously, such imputations of constant values may produce a strong bias. For example, imputation by C can lead to a systematic underestimation of the true runtime, and so does the ignorance of the censored (and hence long) runtimes (Tornede et al. 2020).

A more advanced technique for imputation of right-censored data developed by Schmee and Hahn (1979) leverages sampling from a truncated normal distribution, which has been frequently used in the context of AS and AC in the past (e.g. (Xu et al. 2007; Eggensperger et al. 2018)), but not necessarily improves upon the naïve imputation schemes previously discussed (Tornede et al. 2020).

Although recent work (Tornede et al. 2020) has shown that classical parameter-free survival analysis methods can perform very well in the offline AS setting, these methods cannot be easily transformed into online variants. For example, the well-known Cox proportional hazard model (Cox 1972) relies on estimating the baseline survival function through algorithms like the Breslow estimator (in its parameter-free version), which inherently requires the storage of all data in the form of so-called risk-sets (Breslow 1972). In principle, approximation techniques from the field of learning on data streams are conceivable (Shaker and Hüllermeier 2014). Yet, in this work, we will focus on veritable online algorithms that do not require any approximation.

2.2 OAS As A Bandit Problem

OAS can be cast as a contextual multi-armed bandit (MAB) problem comprising a set of arms/algorithms $\mathcal{A}$ to choose from. In each round t, the learner is presented a context, i.e., an instance $i_t \in \mathcal{I}$ and its features $f(i_t) \in \mathbb{R}^d$, and is requested to select one of the algorithms for this instance, i.e., pull one of the arms, which will be denoted by a_t. As a result, the learner will suffer a loss as defined in (5).

In the stochastic variant of the contextual MAB problem, the losses are generated at random according to underlying distributions, one for each arm, which are unknown to the learner. Thus, the expected loss $\mathbb{E}\left[l(i_t, a_t)|f(i_t)\right]$ suffered at time step t is taken with respect to the unknown distribution of the chosen algorithm's runtime $m(i_t, a_t)$ and depends on the instance (features) $f(i_t)$. Ideally, the learner should pick in each time step t an arm having the smallest expected loss for the current problem instance. Formally,

$$a_t^* \in \arg\min\nolimits_{a \in \mathcal{A}} \mathbb{E}\left[l(i_t, a)|f(i_t)\right], \tag{6}$$

suggesting the optimal strategy to be $s_t^*(h_t, i_t) = a_t^*$. Needless to say, this optimal strategy can only serve as a benchmark, since the runtime distributions are unknown. Nevertheless, having an appropriate model or estimate for the expected losses, one could mimic the choice mechanism in (6), giving rise to a suitable online algorithm selector (4).

For convenience, we shall write from now on $\boldsymbol{f}_i$, $l_{t,a}$ or $m_{i,a}$ instead of $f(i)$, $l(i_t, a)$ or $m(i, a)$ for any $i, i_t \in \mathcal{I}$, $a \in \mathcal{A}$. Moreover, we write $\|\boldsymbol{x}\|_A := \sqrt{\boldsymbol{x}^\intercal A^{-1}\boldsymbol{x}}$ for any $\boldsymbol{x} \in \mathbb{R}^d$ and semi-positive definite matrix $A \in \mathbb{R}^{d\times d}$, and $\|\boldsymbol{x}\| := \sqrt{\boldsymbol{x}^\intercal \boldsymbol{x}}$. In Section A of the appendix, we provide a list of frequently used symbols for quick reference.

3 Related Work

Most related from a problem perspective is the work by Degroote et al.. In a series of papers (Degroote et al. 2016; Degroote 2017; Degroote et al. 2018), they define the OAS problem in a similar form as we do and present different context-based bandit algorithms. In contrast to their setting, the one presented in Section 2 does not feature a prior offline training phase, as our goal is to investigate a true online setting where learning has to be performed from scratch. In addition, their approaches essentially rely on batch learning algorithms, making their time- and space-complexity dependent on the time horizon[1]. Moreover, they do not explicitly consider the problem of censoring, but apply a PAR10 imputation (as standard in ASlib). Lastly, compared to our work, their approaches lack a theoretical foundation, for instance, their models on the runtimes would in principle even allow negative runtime predictions.

The majority of other work related to OAS is situated in the fields of (online) algorithm scheduling (Lindauer, Bergdoll, and Hutter 2016) and dynamic algorithm configuration (Biedenkapp et al. 2020) (aka. algorithm control (Biedenkapp et al. 2019)), where the goal is to predict a schedule of algorithms or dynamically control the algorithm during the solution process of an instance instead of predicting a single algorithm as in our case. Gagliolo and Schmidhuber (2006), Gagliolo and Legrand (2010), Gagliolo and Schmidhuber (2010), Pimpalkhare et al. (2021), and Cicirello and Smith (2005) essentially consider an online algorithm scheduling problem, where both an ordering of algorithms and their corresponding resource allocation (or simply the allocation) has to be computed. Thus, the prediction target is not a single algorithm as in our problem, but rather a very specific composition of algorithms, which can be updated during the solution process. Different bandit algorithms are used to solve this problem variant. Lagoudakis and Littman (2000), Armstrong et al. (2006), van Rijn, Doerr, and Bäck (2018), Laroche and Féraud (2017) and Lissovoi, Oliveto, and Warwicker (2020) in one way or another consider the problem of switching (a component of) an algorithm during the solution process of an instance by means of reinforcement learning or bandit algorithms. They can be considered to be in the field of algorithm control and dynamic algorithm configuration.

Another large corpus of related work can be found in the field of learning from data streams, where the goal is to select an algorithm for the next instance assuming that the data generating process might show a distributional shift (Gama 2012). To achieve this, Rossi, de Carvalho, and Soares (2012) and van Rijn et al. (2014) apply windowing techniques and apply offline AS approaches, which are trained on the last window of instances and used to predict for the next instance. Similarly, van Rijn et al. (2015) dynamically adjust the composition and weights of an ensemble of streaming algorithms. In a way, the methods presented by Degroote et al. (2018) can be seen as windowing techniques where the window size is set to $t-1$, if t is the current time step.

Another related branch of the literature is realtime algorithm configuration (Fitzgerald et al. 2014; Fitzgerald, Malitsky, and O'Sullivan 2015; El Mesaoudi-Paul et al. 2020), where in contrast to our setting, one seeks to find a suitable configuration of one single target algorithm (instead of the algorithm itself) for incoming problem instances.

Finally, Gupta and Roughgarden (2017) analyze several versions of the AS problem on a more theoretical level, including our version of OAS, and show for some problem classes the existence of an OAS approach with low regret under specific assumptions.

4 Modeling Runtimes

As hinted at earlier, online algorithm selectors based on a bandit approach can be reasonably designed through the estimation of the expected losses occurring in (6). To this end, we make the following assumption regarding the runtime of an algorithm $a \in \mathcal{A}$ on problem instance $i \in \mathcal{I}$ with features $\boldsymbol{f}_i \in \mathbb{R}^d$:

$$m_{i,a} = \exp(\boldsymbol{f}_i^\intercal \boldsymbol{\theta}_a^*)\exp(\epsilon_{i,a}), \tag{7}$$

where $\boldsymbol{\theta}_a^* \in \mathbb{R}^d$ is some unknown weight vector for each algorithm $a \in \mathcal{A}$, and $\epsilon_{i,a}$ is a stochastic noise variable with zero mean. The motivation for (7) is twofold. First, theoretical properties such as positivity of the runtimes and heavy-tail properties of their distribution (by appropriate choice of the noise variables) are ensured. Second, we obtain a convenient linear model for the logarithmic runtime $y_{i,a}$ of an algorithm a on instance i, namely

$$y_{i,a} = \log(m_{i,a}) = \boldsymbol{f}_i^\intercal \boldsymbol{\theta}_a^* + \epsilon_{i,a}\,. \tag{8}$$

It is important to realize the two main implications coming with those assumption. First, up to the stochastic noise term,

[1]As we show in this work, some of their batch learning algorithms can actually be replaced by online learners.

the (logarithmic) runtime of an algorithm depends linearly on the features of the corresponding instance. This is not a big restriction, because the feature map $\boldsymbol{f}_i$ may include nonlinear transformations of "basic" features and play the role of a *linearization* (Schölkopf and Smola 2001) — the practical usefulness of non-linear models has recently been shown ,for example, by Tornede et al. (2020). Moreover, previous work on AS has also considered logarithmic runtimes for model estimation (Xu et al. 2007). Second, the model (8) suggests to estimate $\boldsymbol{\theta}_a^*$ separately for each algorithm, which is common practice but excludes the possibility to exploit (certainly existing) similarities between the algorithms. In principle, it might hence be advantageous to estimate joint models (Tornede, Wever, and Hüllermeier 2019, 2020).

Additionally, we assume that (i) $\exp(f_i^\intercal \boldsymbol{\theta}_a^*) \leqslant C$ for any $a \in \mathcal{A}$, $i \in \mathcal{I}$ and (ii) $\epsilon_{i,a}$ is normally distributed with zero mean and standard deviation $\sigma > 0$. While the first assumption is merely used for technical reasons, namely to derive theoretically valid confidence bounds for estimates of the weight vectors, the second assumption implies that $\exp(\epsilon_{i,a})$ is log-normal, which is a heavy-tail distribution yielding a heavy-tail distribution for the complete runtime, thus adhering to practical observations discussed earlier.

5 Stochastic Linear Bandits Approaches

As (8) implies $\mathbb{E}\left[\log(m_{i,a})|\boldsymbol{f}_i\right] = \boldsymbol{f}_i^\intercal \boldsymbol{\theta}_a^*$, it is tempting to apply a straightforward contextualized MAB learner designed for expected loss minimization, in which the expected losses are linear with respect to the context vector, viewing the logarithmic runtimes as the losses of the arms. This special case of contextualized MABs, also known as the *stochastic linear bandit* problem, has received much attention in the recent past (cf. Chap. 19 in Lattimore and Szepesvári (2020)). Generally speaking, such a learner tends to choose an arm having a low expected log-runtime for the given context (i.e., instance features), which in turn has a small expected loss. A prominent learner in stochastic linear bandits is LinUCB (Chu et al. 2011), a combination of online linear regression and the UCB (Auer, Cesa-Bianchi, and Fischer 2002) algorithm. UCB implements the principle of optimism in the face of uncertainty and solves the exploration-exploitation tradeoff by constructing confidence intervals around the estimated mean losses of each arm, and choosing the most optimistic arm according to the intervals.

Under the runtime assumption (8), the basic LinUCB variant (which we call BLindUCB) disregards censored observations in the OAS setting, and therefore considers the ridge-regression (RR) estimator for each algorithm $a \in \mathcal{A}$ only on the non-censored observations. Formally, in each time step t, the RR estimate $\widehat{\boldsymbol{\theta}}_{t,a}$ for the weight parameter $\boldsymbol{\theta}_a^*$ is

$$\arg\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \sum\nolimits_{j=1}^t 1_{[\![a_j=a, m_{i_j,a}\leqslant C]\!]} \left(\boldsymbol{f}_{i_j}^\intercal \boldsymbol{\theta} - y_{i_j,a}\right)^2 + \lambda \|\boldsymbol{\theta}\|^2 , \tag{9}$$

where $\lambda \geqslant 0$ is a regularization parameter. The resulting selection strategy for choosing algorithm a_t at time t is then

$$a_t = \arg\min_{a\in\mathcal{A}} \ \boldsymbol{f}_{i_t}^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot w_{t,a}(\boldsymbol{f}_{i_t}) , \tag{10}$$

where $\alpha > 0$ is some parameter controlling the exploration-exploitation trade-off, and

$$w_{t,a}(\boldsymbol{f}_{i_t}) = \|\boldsymbol{f}_{i_t}\|_{A_{t,a}}$$

the confidence width for the prediction of the (logarithmic) runtime of algorithm a for problem instance i_t based on the estimate (9).

Here, $A_{t,a} = \lambda I_d + X_{t,a}^\intercal X_{t,a}$ is the (regularized) Gram matrix, with I_d the $d \times d$ identity and $X_{t,a}$ denoting the design matrix at time step t associated with algorithm a, i.e., the matrix that stacks all the features row by row whenever a has been chosen.

The great appeal of this approach is the existence of a closed form expression of the RR estimate, which can be updated sequentially with time- and space-complexity depending only on the feature dimension but independent of the time horizon (Lattimore and Szepesvári 2020).

However, as already mentioned in Section 2.1, disregarding the right-censoring of the data often yields a rather poor performance of a regression-based learner in offline AS problems, so it might be advantageous to adapt this method to that end.

5.1 Imputation-based Upper Confidence Bounds

A simple approach to include right-censored data into BlindUCB is to impute the corresponding samples by the cut-off time C as discussed in Sec. 2.1, giving rise to the RR estimate

$$\widehat{\theta}_{t,a} = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \sum\nolimits_{j=1}^t 1_{[\![a_j=a]\!]} \left(f_{i_j}^\intercal \boldsymbol{\theta} - \widetilde{y}_{i_j,a}\right)^2 + \lambda \|\boldsymbol{\theta}\|^2 , \tag{11}$$

where $\widetilde{y}_{i_j,a} = \min(y_{i_j,a}, \log(C))$ is the possibly imputed logarithmic runtime.

When considering censoring, the least-squares formulation in (11) appears to have an important disadvantage. Those weight vectors producing overestimates of C in case of a timeout are penalized (quadratically) for predictions $C < \widehat{y} < m(i,a)$, although these predictions are actually closer to the unknown ground truth than C. In fact, one can verify this intuition theoretically by showing that, for $\lambda = 0$, the RR estimate $\widehat{\theta}_{t,a}$ is downward biased in the case of censored samples (cf. Greene (2005)). It is important to note that this bias is caused by a censoring of the runtimes, i.e. of the signal, and not by a truncation of the inputs or sparse (or non-representative) features as, for example, in (Dimakopoulou et al. 2019).

Although the imputation strategy mentioned above has been shown to work astonishingly well in practice in offline AS (Tornede et al. 2020), the bias in the RR estimates requires a bias-correction in the confidence bounds of BLindUCB to ensure that the estimate falls indeed into the bounds with a certain probability. The corresponding bias-corrected confidence bound widths are

$$w_{t,a}^{(bc)}(\boldsymbol{f}_{i_t}) = \left(1 + 2\log(C)\sqrt{N_{a,t}^{(C)}}\right) w_{t,a}(\boldsymbol{f}_{i_t}) \ , \tag{12}$$

where $N_{a,t}^{(C)}$ is the amount of timeouts of algorithm a until t (cf. Section C of the appendix). The resulting LinUCB variant, which we call BClinUCB (bias-corrected LinUCB), employs the same action rule as in (10), but uses $w_{t,a}^{(bc)}$ instead of $w_{t,a}$ and the estimate in (11).

Unfortunately, these bias-corrected confidence bounds reveal a decisive disadvantage in practice, namely, the confidence bound of an algorithm $a \in \mathcal{A}$ is usually much larger than the actually estimated (log-)runtime $\boldsymbol{f}_{i_t}^{\intercal}\widehat{\boldsymbol{\theta}}_{t,a}$ for instance i_t. Therefore, the UCB value of a — let us call it $\delta_{t,a} = \boldsymbol{f}_{i_t}^{\intercal}\widehat{\boldsymbol{\theta}}_{t,a} - w_{t,a}^{(bc)}(\boldsymbol{f}_{i_t})$ — is such that $\delta_{t,a} < 0$ if the algorithm has experienced at least one timeout. This prefers algorithms that experienced a timeout over those that did not. This in turn explains the poor performance of the BClinUCB strategies in the evaluation in Section 7.

5.2 Randomization Of Upper Confidence Bounds

One way of mitigating the problem of the bias-corrected confidence bounds is to leverage a generalized form of UCB, called randomized UCB (RandUCB) (Vaswani et al. 2020), where the idea is to multiply the bias-corrected bounds $w_{t,a}^{(bc)}(\boldsymbol{f}_{i_t})$ with a random realization of a specific distribution having positive support. RandUCB can be thought of as a mix of the classical UCB strategy, where the exploration-exploitation trade-off is tackled via the confidence bounds, and Thompson sampling (Thompson 1933; Russo et al. 2018), which leverages randomization in a clever way for the same purpose (see next section). To this end, we define randomized confidence widths

$$\widetilde{w}_{t,a}(\boldsymbol{f}_{i_t}) = w_{t,a}^{(bc)}(\boldsymbol{f}_{i_t}) \cdot r \quad , \tag{13}$$

where $r \in \mathbb{R}$ is sampled from a half-normal distribution with 0 mean and standard deviation $\widetilde{\sigma}^2$. This ensures that $r \geqslant 0$ and that the confidence widths do indeed shrink when the distribution is properly parametrized. Although this improves the performance of LinUCB as we will see later, the improvement is not significant enough to achieve competitive results.

All variants of LinUCB for OAS introduced so far can be jointly defined as in Alg. 2 in Section B of the appendix.

5.3 Bayesian Approach: Thompson Sampling

As the confidence bounds used by LinUCB seem to be a problem in practice, one may think of Thompson sampling (TS) as an interesting alternative. The idea of TS is to assume a prior loss distribution for every arm, and in each time step, select an arm (i.e. algorithm) according to its probability of being optimal, i.e., according to its posterior loss distribution conditioned on all of the data seen so far. In particular, this strategy solves the exploration-exploitation trade-off through randomization driven by the posterior loss distribution.

More specifically, let the (multivariate) Gaussian distribution with mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$ be denoted by $\mathrm{N}(\boldsymbol{\mu}, \Sigma)$. Similarly, the cumulative distribution function of a (univariate) Gaussian distribution with mean $\mu \in \mathbb{R}$ and variance σ^2 at some point $z \in \mathbb{R}$ is denoted by $\Phi_{\mu,\sigma^2}(z)$. A popular instantiation of TS for stochastic linear bandits (Agrawal and Goyal 2013) assumes a Gaussian prior distribution $\mathrm{N}(\widehat{\boldsymbol{\theta}}_{t,a}, \sigma A_{t,a}^{-1})$ for each weight vector of an algorithm a, where $\lambda, \sigma > 0$ and $\widehat{\boldsymbol{\theta}}_{t,a}$ denotes the RR estimate (11). This yields $\mathrm{N}(\widehat{\boldsymbol{\theta}}_{t+1,a}, \sigma A_{t+1,a}^{-1})$ as the posterior distribution at time step $t+1$. The choice mechanism is then defined by

$$a_t = \arg\min\nolimits_{a\in\mathcal{A}} \boldsymbol{f}_{i_t}^{\intercal}\widetilde{\boldsymbol{\theta}}_a \, , \tag{14}$$

where $\widetilde{\boldsymbol{\theta}}_a \sim \mathrm{N}(\widehat{\boldsymbol{\theta}}_{t,a}, \sigma A_{t,a}^{-1})$ for each $a \in \mathcal{A}$. Interestingly, as the experiments will show later on, this rather naïve version of Thompson sampling in the presence of censored data works astonishingly well in practice.

6 Expected PAR10 Loss Minimization

Due to the possibility of observing only a censored loss realization, i.e., $\mathcal{P}(C)$, it is reasonable to believe that a successful online algorithm selector needs to be able to properly incorporate the probability of observing such a realization into its selection mechanism. For this purpose, we derive the following decomposition of the expected loss under the assumptions made in Section 4 (details in Section D of the appendix):

$$\begin{aligned}\mathbb{E}\left[l_{t,a}|\boldsymbol{f}_{i_t}\right] = &\left(1 - \Phi_{\boldsymbol{f}_{i_t}^{\intercal}\boldsymbol{\theta}_a^*,\sigma^2}(\log(C))\right)\cdot(\mathcal{P}(C) - E_C)\\ &+ E_C \end{aligned}\tag{15}$$

where $C_{i_t,a}^{(1)} = {}^{(\log(C) - \boldsymbol{f}_{i_t}^{\intercal}\boldsymbol{\theta}_a^* - \sigma^2)}/_{\sigma}$, $C_{i_t,a}^{(2)} = C_{i_t,a}^{(1)} + \sigma$ and

$$E_C = E_C(\boldsymbol{f}_{i_t}^{\intercal}\boldsymbol{\theta}_a^*, \sigma) = \exp(\boldsymbol{f}_{i_t}^{\intercal}\boldsymbol{\theta}_a^* + {}^{\sigma^2}/_{2}) \cdot \frac{\Phi_{0,1}(C_{i_t,a}^{(1)})}{\Phi_{0,1}(C_{i_t,a}^{(2)})}$$

is the conditional expectation of a log-normal distribution with parameters $\boldsymbol{f}_{i_t}^{\intercal}\boldsymbol{\theta}_a^*$ and σ^2 under a cutoff C. As such, the decomposition suggests that there are two core elements driving the expected loss of an algorithm a conditioned on a problem instance i_t with features $\boldsymbol{f}_{i_t}$: its expected log-runtime $\boldsymbol{f}_{i_t}^{\intercal}\boldsymbol{\theta}_a^*$ and its probability of running into a timeout, i.e.,

$$\mathbb{P}(m(i_t, a) > C|\boldsymbol{f}_{i_t}) = \left(1 - \Phi_{\boldsymbol{f}_{i_t}^{\intercal}\boldsymbol{\theta}_a^*,\sigma^2}(\log(C))\right) \, . \tag{16}$$

6.1 LinUCB Revisited

Having the refined expected loss representation in (15), one could simply plug-in the confidence bound estimates used by LinUCB for the log-runtime predictions to obtain a selection strategy following the optimism in the face of uncertainty principle, i.e., using an estimate of the target value to be minimized (here the expected loss in (15)), which underestimates the target value with high probability. Denote by

$$o_{t,a} = \boldsymbol{f}_{i_t}^{\intercal}\widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot w_{t,a}(\boldsymbol{f}_{i_t}), \tag{17}$$

$$p_{t,a} = \boldsymbol{f}_{i_t}^{\intercal}\widehat{\boldsymbol{\theta}}_{t,a} + \alpha \cdot w_{t,a}(\boldsymbol{f}_{i_t}) \tag{18}$$

the optimistic and the pessimistic estimate used by LinUCB (or its variants), where $w_{t,a}$ is the confidence width of the corresponding LinUCB variant. With this, the selection strategy at time t is to use $a \in \mathcal{A}$ minimizing

$$\left(1 - \Phi_{p_{t,a},\sigma}(\log(C))\right)\cdot\left(\mathcal{P}(C) - \hat{E}_C^{(1)}\right) + \hat{E}_C^{(2)}, \tag{19}$$

Algorithm 1 `(bj_)Thompson_rev`

```
1: Input parameters $\sigma > 0, \lambda \geqslant 0, \mathcal{P} : \mathbb{R} \to \mathbb{R}, C$, BJ $\in$ {TRUE, FALSE}
2: for all $a \in \mathcal{A}$ do
3:     $A_{t,a} = \lambda \cdot I_{d\times d}, \boldsymbol{b}_{t,a} = 0_{d\times 1}, \widehat{\boldsymbol{\theta}}_{t,a} = 0_{d\times 1}, \tilde{\sigma}^2_{t,a} = 0$ and $\widehat{l}_{t,a} = 0$
4: end for
5: for time steps $t = 1 \ldots, T$ do
6:     Observe instance $i_t$ and its features $\boldsymbol{x}_t = f(i_t) \in \mathbb{R}^d$
7:     if $t \leqslant |\mathcal{A}|$ then
8:         Take algorithm $a_t \in \mathcal{A}$ and obtain $y_t = \min(\log(m(i_t, a_t)), \log(C))$
9:     else
10:        for all $a \in \mathcal{A}$ do
11:            $\widehat{\boldsymbol{\theta}}_{t,a} \leftarrow (A_{t,a})^{-1}\boldsymbol{b}_{t,a}$    $\tilde{\sigma}^2_{t,a} \leftarrow \sigma\|\boldsymbol{f}_{i_t}\|^2_{A_{t,a}}$
12:            Sample $\widetilde{\boldsymbol{\theta}}_a \sim \mathrm{N}\big(\widehat{\boldsymbol{\theta}}_{t,a}, \sigma(A_{t,a})^{-1}\big)$
13:            $\widehat{l}_{t,a} \leftarrow \big(1 - \Phi_{\boldsymbol{f}_{i_t}^\intercal \widetilde{\boldsymbol{\theta}}_a, \tilde{\sigma}^2_{t,a}}(\log(C))\big) \cdot \big(\mathcal{P}(C) - E_C(\boldsymbol{f}_{i_t}^\intercal \widetilde{\boldsymbol{\theta}}_a, \tilde{\sigma}_{t,a})\big) + E_C(\boldsymbol{f}_{i_t}^\intercal \widetilde{\boldsymbol{\theta}}_a, \tilde{\sigma}_{t,a})$    (RHS of (20))
14:        end for
15:        Take algorithm $a_t = \arg\min_{a\in\mathcal{A}} \widehat{l}_{t,a}$ and obtain $y_t = \min(\log(m(i_t, a_t)), \log(C))$
16:    end if
17:    if $y_t = \log(C)$ and BJ = TRUE then
18:        Sample $\check{\boldsymbol{\theta}}_a \sim \mathrm{N}\big(\widehat{\boldsymbol{\theta}}_{t,a}, \sigma(A_{t,a})^{-1}\big)$    (if $\exp(\boldsymbol{x}_t^\intercal \check{\boldsymbol{\theta}}_a) \leqslant C$ sample again)
19:        $y_t \leftarrow \log(\boldsymbol{x}_t^\intercal \check{\boldsymbol{\theta}}_a)$
20:    end if
21:    $A_{t,a} \leftarrow A_{t,a} + \boldsymbol{x}_t\boldsymbol{x}_t^\intercal$    $\boldsymbol{b}_{t,a} \leftarrow \boldsymbol{b}_{t,a} + y_t\boldsymbol{x}_t$
22: end for
```

where

$$\hat{E}_C^{(1)} = \exp(p_{t,a} + {}^{\sigma^2}\!/2) \cdot {}^{\Phi_{0,1}(\hat{C}^{(o)}_{i_t,a})}\!/_{\Phi_{0,1}(\hat{C}^{(p)}_{i_t,a}+\sigma)},$$
$$\hat{E}_C^{(2)} = \exp(o_{t,a} + {}^{\sigma^2}\!/2) \cdot {}^{\Phi_{0,1}(\hat{C}^{(p)}_{i_t,a})}\!/_{\Phi_{0,1}(\hat{C}^{(o)}_{i_t,a}+\sigma)},$$

and

$$\hat{C}^{(p)}_{i_t,a} = {}^{(\log(C)-p_{t,a}-\sigma^2)}\!/_{\sigma},$$
$$\hat{C}^{(o)}_{i_t,a} = {}^{(\log(C)-o_{t,a}-\sigma^2)}\!/_{\sigma}$$

As $o_{t,a}$ ($p_{t,a}$) underestimates (overestimates) $\boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^*$ it is easy to see that the terms in (19) are underestimating the corresponding terms occurring in (15) with high probability, respectively.

However, as our experiments will reveal later on, the issues of the LinUCB-based algorithms caused either by the wide confidence bands or the biased RR estimate remain.

6.2 Thompson Sampling Revisited

Fortunately, the refined expected loss representation in (15) can be exploited quite elegantly by Thompson Sampling using Gaussian priors as in Section 5.3. Our suggested instantiation of TS chooses algorithm $a_t \in \mathcal{A}$ which minimizes

$$\big(1 - \Phi_{\boldsymbol{f}_{i_t}^\intercal \widetilde{\boldsymbol{\theta}}_a, \tilde{\sigma}^2_{t,a}}(\log(C))\big)\big(\mathcal{P}(C) - \tilde{E}_C\big) + \tilde{E}_C, \quad (20)$$

where $\widetilde{\boldsymbol{\theta}}_a$ is a random sample of the posterior $\mathrm{N}\big(\widehat{\boldsymbol{\theta}}_{t,a}, \sigma A_{t,a}^{-1}\big)$, and $\tilde{E}_C = E_C(\boldsymbol{f}_{i_t}^\intercal \widetilde{\boldsymbol{\theta}}_a, \tilde{\sigma}_{t,a})$, $\tilde{\sigma}^2_{t,a} = \sigma\|\boldsymbol{f}_{i_t}\|^2_{A_{t,a}}$. Alg. 1 provides the pseudo code for this revisited Thompson algorithm and a variant inspired by the Buckley-James estimate we discuss in the following.

Although the TS approach just presented does involve consideration of the timeout probability, it still suffers from the problem that the estimates for $\boldsymbol{\theta}_a^*$ are downward-biased as they are based on the RR estimate obtained from imputing censored samples with the cutoff time C. In the spirit of the Kaplan-Meier estimator (Kaplan and Meier 1958) from the field of survival analysis, Buckley and James (1979) suggested to augment censored samples by their expected value according to the current model and then solve the standard least-squares problem (for an overview of alternative approaches, we refer to Miller and Halpern (1982)). This idea is particularly appealing, as it allows for an easy integration into online algorithms, due to its use of the least-squares estimator. Also, it has the potential to produce more accurate (i.e., less biased) estimates for $\boldsymbol{\theta}_a^*$. The integration is shown in lines 17–20 in Alg. 1.

7 Evaluation

We base our evaluation on the standard algorithm selection benchmark library ASlib (v4.0) (Bischl et al. 2016) and compare to the most relevant competitor approaches. ASlib is a curated collection of over 25 different algorithm selection problems, called scenarios, based on different algorithmic problem classes such as SAT, TSP, CSP. Each scenario comprises several instances for which the performance of a set of algorithms has been evaluated using a certain cutoff to avoid excessively long algorithm runs. An overview of the included scenarios and their statistics can be found in Section E of the appendix. Since ASlib was originally designed for offline AS, we do not use the train/test splits provided by the benchmark, but rather pass each instance one by one to the correspond-

Approach	bj_thompson		thompson_rev		degroote_ϵ-greedy_LR	
Scenario						
ASP-POTASSCO	949.38	± 62.38	**902.64**	± 78.43	1047.13	± 46.50
BNSL-2016	9638.04	± 378.05	**9467.01**	± 252.52	12510.26	± 1291.03
CPMP-2015	8241.01	± 1164.85	8158.72	± 1268.83	**6991.97**	± 501.36
CSP-2010	8295.76	± 699.43	7892.67	± 692.83	**7593.13**	± 208.94
CSP-MZN-2013	8207.06	± 532.70	8171.21	± 594.49	**8034.62**	± 113.78
CSP-Minizinc-Time-2016	4811.54	± 409.79	**4759.50**	± 306.03	5258.70	± 406.91
GRAPHS-2015	4.1e+07	± 4.4e+06	4.2e+07	± 3.4e+06	**3.5e+07**	± 1.4e+06
MAXSAT-PMS-2016	2853.44	± 210.21	**2808.51**	± 218.55	3279.54	± 133.00
MAXSAT-WPMS-2016	6304.15	± 166.98	6592.87	± 210.25	**6287.21**	± 541.69
MAXSAT12-PMS	5347.39	± 291.87	5408.40	± 482.42	**5308.11**	± 129.30
MAXSAT15-PMS-INDU	3046.05	± 128.34	**3032.08**	± 90.71	3867.70	± 255.98
MIP-2016	**8081.57**	± 845.74	8746.73	± 1159.36	10644.68	± 3405.18
PROTEUS-2014	**13484.34**	± 541.83	14115.69	± 768.16	15622.29	± 784.60
QBF-2011	15708.25	± 784.81	15178.86	± 904.72	**13912.24**	± 356.69
QBF-2014	**3629.40**	± 220.68	3679.96	± 256.03	4116.15	± 116.27
QBF-2016	5082.59	± 718.71	**5045.16**	± 848.59	5346.29	± 210.05
SAT03-16_INDU	**11980.15**	± 193.67	12154.46	± 221.01	12754.50	± 200.55
SAT11-HAND	30484.08	± 1379.35	30085.51	± 764.32	**29544.70**	± 952.78
SAT11-INDU	17540.58	± 530.82	17028.84	± 479.15	**17018.24**	± 647.90
SAT11-RAND	**18061.78**	± 2770.70	19061.88	± 2522.11	21008.77	± 530.22
SAT12-ALL	**4720.22**	± 432.14	5132.48	± 395.74	5650.32	± 214.36
SAT12-HAND	**7443.01**	± 180.51	7509.02	± 199.39	7634.24	± 267.89
SAT12-INDU	**4511.68**	± 76.33	4945.79	± 228.37	4755.52	± 206.95
SAT12-RAND	**4008.79**	± 206.59	4523.33	± 170.56	5023.73	± 174.68
SAT15-INDU	**7700.27**	± 310.65	7856.08	± 522.84	8220.22	± 525.13
SAT18-EXP	25201.41	± 681.42	**24906.56**	± 540.36	25272.35	± 881.19
TSP-LION2015	**1226.11**	± 309.42	1411.06	± 329.16	1634.79	± 112.29
avgrank	**1.814815**		1.888889		2.296296	

Table 1: Average PAR10 scores and standard deviation of Thompson variants and Degroote.

ing online approaches, ask them to select an algorithm and return the corresponding feedback. To increase evaluation robustness, we randomly shuffle the instances of each scenario, repeat the evaluation ten times with different seeds and always report average or median aggregations across those ten repetitions. As ASlib contains missing feature values for some instances in some scenarios, we imputed these using the mean feature value of all instances seen until that point. Moreover, features were scaled to unit vectors by dividing by their norm. If the according variant does not self-impute censored values, these were imputed with the cutoff time.

All experiments were run on machines featuring Intel Xeon E5-2695v4@2.1GHz CPUs with 16 cores and 64GB RAM, where each approach was limited to a single-core. All code including detailed documentation and the appendix itself can be found on on GitHub[2]. The corresponding hyperparameter settings used for the experiments can be found in Section F of the appendix and in the repository, parameter sensitivity analyses in Section G.

Instead of directly reporting PAR10 scores, we sometimes resolve to reporting a normalized version called rePAR10 (relative PAR10), which is comparable across scenarios and defined with respect to the oracle[3]. The rePAR10 is simply defined as the PAR10 score of the corresponding approach divided by the PAR10 score of the oracle, i.e., the smaller the rePAR10, the better. Moreover, we will explicitly analyze the "prediction time", i.e., the time an approach requires for making a single selection and updating its model with the corresponding feedback.

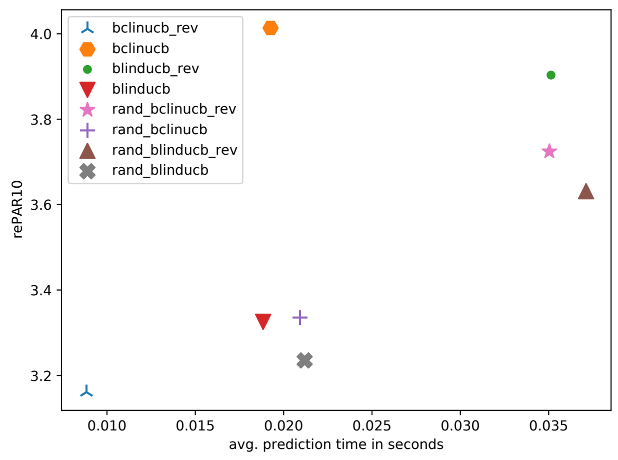
7.1 Ablation Study

First, we analyze how the different LinUCB and Thompson variants perform in terms of rePAR10 performance, when some of their components are activated or deactivated.

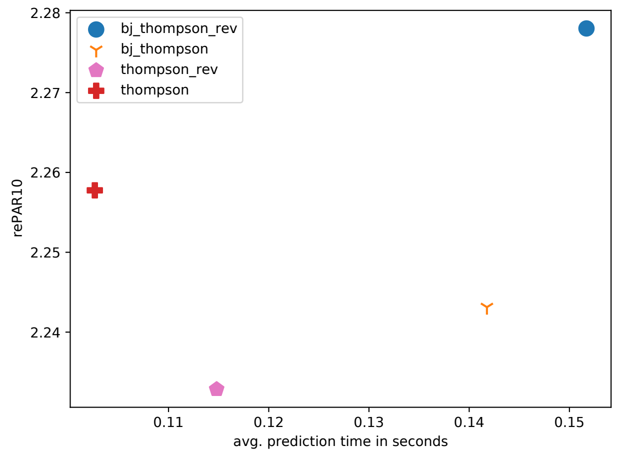
LinUCB Recall that we differentiate in principle between BlindUCB and BClinUCB. Both the randomization idea (denoted by a 'rand_' prefix) and the expected PAR10 loss mini-

[2]https://github.com/alexandertornede/online_as

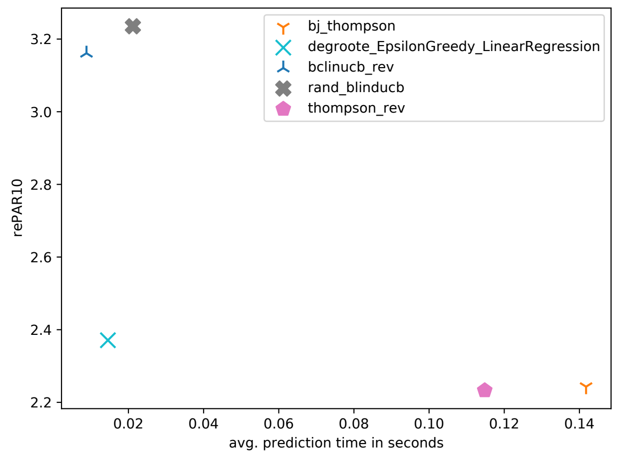
[3]Although in standard AS one usually uses the nPAR10 which is defined wrt. to both the oracle and algorithm best on average (aka. SBS), we abstain from using it as the SBS cannot be as easily defined as in the standard setting since instead of offline training data only the performance of the selected algorithm (and not of all) in the current time step is available to update the underlying model.



(a) LinUCB variants



(b) Thompson variants



(c) Degroote vs. this work

Figure 1: rePAR10 score averaged over all scenarios of approaches plotted against their average prediction time in seconds.

mization (denoted by '_rev' suffix) can in principle be incorporated into both yielding a total of 8 variants.

Figure 1a shows the median rePAR10 score over all scenarios of the corresponding variant plotted against its prediction time in seconds. First of all, it is very clear that all of the LinUCB variants are at least 3.1 times as worse as the oracle. A closer look at the selections made by the corresponding models shows two things. First, although BlindUCB heavily underestimates runtimes as it completely ignores censored samples, its estimates yield some of the best algorithm selections among all LinUCB variants. Second, except for the revisited versions, BClinUCB yields worse results than the corresponding BlindUCB variant in all cases. As hinted at earlier, BClinUCB suffers from very large confidence bounds due to the correction, yielding suboptimal selections in many cases. Moreover, one can see that directly minimizing the expected PAR10 loss does not prove to be very beneficial (except for the pure BClinUCB variant) and can even worsen the performance for some variants. From a methodological point of view, this is not surprising for BlindUCB, as it would technically require a different treatment of the expected PAR10 loss based on a truncated (instead of a censored) linear model (cf. Greene (2005)). However, for the BClinUCB variants, this is rather disappointing. In contrast, the randomization (i.e. RandUCB) yields consistent improvements (except for one case), making some of the randomized variants the best among all LinUCB variants. This also coincides with our observation that the poor selection performance is caused by large confidence widths due to the correction, which are decreased through randomization.

Thompson We presented both a naïve and a revisited form of Thompson incorporating expected PAR10 loss minimization ('_rev' suffix). Moreover, both versions can be equipped with the Buckley-James imputation strategy discussed at the end of Section 6.2 ('bj_' prefix), yielding a total of 4 variants.

Figure 1b shows the median rePAR10 score over all scenarios of the corresponding variant plotted against its average prediction time per instance. As expected, the more components are active, the longer the prediction time becomes. However, the average prediction time per instance still remains below 0.16s. Both the revisited and the Buckley-James variant yield an improvement over the plain Thompson variant. A combined variant worsens the performance, meaning that the revisited variant achieves the best performance. However, overall one has to note that all variants behave rather similar with only small differences in performance.

7.2 Comparison To Competitors

In the following, we only compare two UCB and Thompson variants to the competitors to avoid overloading the evaluation. In particular, we compare to an approach from Degroote et al. (cf. Section 3). Their approaches essentially employ batch machine learning models (linear regression or random forests) on the runtime, which are fully retrained after each seen instance. The selection is either done via a simple ϵ-greedy strategy (Sutton and Barto 2018, Chapter 2) or using a UCB strategy, where the confidence bounds are estimated using the standard deviation extracted from the underlying random forest by means of the Jackknife (Sexton and Laake 2009) method. In fact, the Degroote approaches cannot be considered true online algorithms due to their dependence on the time horizon — they become intractable with an increasing number of instances. Although one can update the underlying models in principle less often (e.g., every ten instances as in the original paper), we abstain here from doing so, because our approaches also incorporate every sample immediately.

As we only consider linear models in this work, we only compare to the linear ϵ-greedy strategy presented by Degroote et al. and abstain from comparing against the random forest versions to avoid that the model complexity becomes a confounding factor in the evaluation.

Figure 1c illustrates the rePAR10 value in comparison to the prediction time in seconds of our most successful bandit algorithms and the linear ϵ-greedy Degroote approach. First, it is easy to see that the Thompson variants largely outperform the LinUCB variants in terms of performance at the cost of being slightly slower in terms of prediction time. Second, the Thompson variants improve around 6% in terms of performance upon the Degroote approach. Interestingly, the latter can compete with all online algorithms in terms of prediction time, and even outperforms the Thompson variants. This is mainly because of the limited size of the data, and because the batch linear regression of the library used for implementation of the Degroote approach is extremely efficient, making batch training affordable. Besides, the Thompson variants require sampling from a multi-variate normal distribution, taking up most of the prediction time. Nevertheless, as already said, batch learning will necessarily become impracticable with an increasing number of observations, and sooner or later get slower than the incremental Thompson approach.

Table 1 illustrates a more nuanced comparison between the best Thompson variants and Degroote, where the best value for each scenario is printed in bold and the second best is underlined.

Overall, one can verify that Thompson sampling is a much more successful strategy than both ϵ-greedy and LinUCB in OAS. Moreover, directly optimizing the expected PAR10 score (_rev variants) and thereby incorporating the right-censoring of the data often proves beneficial, yielding one of the best OAS approaches in this work in the form of Thompson_rev. Nevertheless, as the large rePAR10 scores indicate, there is still room for improvement.

8 Conclusion And Future Work

In this paper, we revisited several well-known contextual bandit algorithms and discussed their suitability for dealing with the OAS problem under censored feedback. As a result of the discussion, we adapted them towards runtime-oriented losses, assuming partially censored data while keeping a space- and time-complexity independent of the time horizon. Our extensive experimental study shows that the combination of considering right-censored data in the selection process and an appropriate choice of the exploration strategy leads to better performance.

As future work, we plan to investigate whether online adaptations of non-parametric survival analysis methods (such as Cox-regression) are possible. Furthermore, results from offline algorithm selection suggest that an extension of our approaches to non-linear models, random forests in particular, seems useful to further improve performance. Moreover, motivated by the recent success of meta algorithm selection (Tornede et al. 2021a; Tornede, Wever, and Hüllermeier 2020), we plan to investigate, whether exploiting a possible heterogeneity across the different OAS approaches we presented is beneficial.

Environmental Impact Of Experiments

In the spirit of Green AutoML (Tornede et al. 2021b), we acknowledge the environmental footprint left by the work on this paper. In total, all experiments executed for this paper consumed roughly 5376 CPU hours, which corresponds to an estimate of about 38.64 kWh or 10.43 kg CO2e (computed at Paderborn University), when considering only the energy consumption of the CPU. The final evaluation presented in this work caused about 384 CPU hours, i.e. 2.76 kWh / 0.75 kg CO2e of the aforementioned consumption.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901/3 project no. 160364472). The authors gratefully acknowledge support of this project through computing time provided by the Paderborn Center for Parallel Computing (PC^2).

References

Agrawal, S.; and Goyal, N. 2013. Thompson sampling for contextual bandits with linear payoffs. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, 127–135.

Armstrong, W.; Christen, P.; McCreath, E.; and Rendell, A. P. 2006. Dynamic algorithm selection using reinforcement learning. In *2006 International Workshop on Integrating AI and Data Mining*, 18–25. IEEE.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3): 235–256.

Biedenkapp, A.; Bozkurt, H. F.; Eimer, T.; Hutter, F.; and Lindauer, M. 2020. Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, 427–434.

Biedenkapp, A.; Bozkurt, H. F.; Hutter, F.; and Lindauer, M. 2019. Towards white-box benchmarks for algorithm control. *CoRR*, abs/1906.07644.

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

Breslow, N. E. 1972. Contribution to discussion of paper by DR Cox. *Journal of the Royal Statistical Society*, 34: 216–217.

Buckley, J.; and James, I. 1979. Linear regression with censored data. *Biometrika*, 66(3): 429–436.

Chu, W.; Li, L.; Reyzin, L.; and Schapire, R. E. 2011. Contextual bandits with linear payoff dunctions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011*, 208–214.

Cicirello, V. A.; and Smith, S. F. 2005. The max *K*-armed bandit: A new model of exploration applied to search heuristic selection. In *Proceedings of The Twentieth National Conference on Artificial Intelligence*, 1355–1361.

Cox, D. R. 1972. Regression models and life tables (with discussion). *Journal of the Royal Statistical Society*, 34(2): 187–220.

Degroote, H. 2017. Online algorithm selection. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, 5173–5174.

Degroote, H.; Bischl, B.; Kotthoff, L.; and Causmaecker, P. D. 2016. Reinforcement learning for automatic online algorithm selection - an empirical study. In *Proceedings of the 16th ITAT Conference Information Technologies - Applications and Theory*, 93–101.

Degroote, H.; Causmaecker, P. D.; Bischl, B.; and Kotthoff, L. 2018. A regression-based methodology for online algorithm selection. In *Proceedings of the Eleventh International Symposium on Combinatorial Search, SOCS 2018*, 37–45.

Dimakopoulou, M.; Zhou, Z.; Athey, S.; and Imbens, G. 2019. Balanced linear contextual bandits. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, 3445–3453.

Eggensperger, K.; Haase, K.; Müller, P.; Lindauer, M.; and Hutter, F. 2020. Neural model-based optimization with right-censored observations. *CoRR*, abs/2009.13828.

Eggensperger, K.; Lindauer, M.; Hoos, H. H.; Hutter, F.; and Leyton-Brown, K. 2018. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107(1): 15–41.

El Mesaoudi-Paul, A.; Weiß, D.; Bengs, V.; Hüllermeier, E.; and Tierney, K. 2020. Pool-based realtime algorithm configuration: A preselection bandit approach. In *International Conference on Learning and Intelligent Optimization , LION 2020*, 216–232. Springer.

Fitzgerald, T.; Malitsky, Y.; and O'Sullivan, B. 2015. ReACTR: Realtime algorithm configuration through tournament rankings. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, 304–310. AAAI Press.

Fitzgerald, T.; Malitsky, Y.; O'Sullivan, B.; and Tierney, K. 2014. ReACT: Real-time algorithm configuration through tournaments. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014*. AAAI Press.

Gagliolo, M.; and Legrand, C. 2010. Algorithm survival analysis. In *Experimental Methods for the Analysis of Optimization Algorithms*, 161–184. Springer.

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

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

Gama, J. 2012. A survey on learning from data streams: Current and future trends. *Progress in Artificial Intelligence*, 1(1): 45–55.

Gomes, C. P.; Selman, B.; and Crato, N. 1997. Heavy-tailed distributions in combinatorial search. In *International Conference on Principles and Practice of Constraint Programming*, 121–135.

Greene, W. H. 2005. Censored data and truncated distributions. *NYU Working Paper*.

Gupta, R.; and Roughgarden, T. 2017. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3): 992–1017.

Hanselle, J.; Tornede, A.; Wever, M.; and Hüllermeier, E. 2020. Hybrid ranking and regression for algorithm selection. In *KI 2020: Advances in Artificial Intelligence - 43rd German Conference on AI*, 59–72.

Hanselle, J.; Tornede, A.; Wever, M.; and Hüllermeier, E. 2021. Algorithm selection as superset learning: Constructing algorithm selectors from imprecise performance data. In *Advances in Knowledge Discovery and Data Mining - 25th Pacific-Asia Conference, PAKDD 2021*, 152–163.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Bayesian optimization with censored response data. In *NIPS workshop on Bayesian Optimization, Sequential Experimental Design and Bandits*.

Kaplan, E. L.; and Meier, P. 1958. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282): 457–481.

Kerschke, P.; Hoos, H. H.; Neumann, F.; and Trautmann, H. 2019. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1): 3–45.

Kleinbaum, D. G.; and Klein, M. 2010. *Survival Analysis*, volume 3. Springer.

Lagoudakis, M. G.; and Littman, M. L. 2000. Algorithm selection using reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning, ICML 2000*, 511–518.

Laroche, R.; and Féraud, R. 2017. Algorithm selection of off-policy reinforcement learning algorithm. *CoRR*, abs/1701.08810.

Lattimore, T.; and Szepesvári, C. 2020. *Bandit Algorithms*. Cambridge University Press.

Lindauer, M.; Bergdoll, R.; and Hutter, F. 2016. An empirical study of per-instance algorithm scheduling. In *International Conference on Learning and Intelligent, LION 2016*, 253–259.

Lissovoi, A.; Oliveto, P. S.; and Warwicker, J. A. 2020. Simple hyper-heuristics control the neighbourhood size of randomised local search optimally for LeadingOnes*. *Evolutionary Computation*, 28(3): 437–461.

Miller, R.; and Halpern, J. 1982. Regression with censored data. *Biometrika*, 69(3): 521–531.

Pimpalkhare, N.; Mora, F.; Polgreen, E.; and Seshia, S. A. 2021. MedleySolver: Online SMT algorithm selection. In *International Conference on Theory and Applications of Satisfiability Testing, SAT 2021*, 453–470.

Rossi, A. L. D.; de Carvalho, A. C. P. L. F.; and Soares, C. 2012. Meta-learning for periodic algorithm selection in time-changing data. In *2012 Brazilian Symposium on Neural Networks*, 7–12.

Russo, D.; Roy, B. V.; Kazerouni, A.; Osband, I.; and Wen, Z. 2018. A tutorial on Thompson sampling. *Foundations and Trends in Machine Learning*, 11(1): 1–96.

Schmee, J.; and Hahn, G. J. 1979. A simple method for regression analysis with censored data. *Technometrics*, 21(4).

Schölkopf, B.; and Smola, A. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

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

Shaker, A.; and Hüllermeier, E. 2014. Survival analysis on data streams: Analyzing temporal events in dynamically changing environments. *International Journal of Applied Mathematics and Computer Science*, 24(1): 199–212.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT press.

Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4): 285–294.

Tornede, A.; Gehring, L.; Tornede, T.; Wever, M.; and Hüllermeier, E. 2021a. Algorithm selection on a meta level. *CoRR*, abs/2107.09414.

Tornede, A.; Wever, M.; and Hüllermeier, E. 2019. Algorithm selection as recommendation: From collaborative filtering to dyad ranking. In *29th Workshop Computational Intelligence, Dortmund 2019*.

Tornede, A.; Wever, M.; and Hüllermeier, E. 2020. Extreme algorithm selection with dyadic feature representation. In *International Conference on Discovery Science, DS 2020*, 309–324.

Tornede, A.; Wever, M.; and Hüllermeier, E. 2020. Towards meta-algorithm selection. In *Workshop on Meta-Learning (MetaLearn 2020) @ NeurIPS 2020*.

Tornede, A.; Wever, M.; Werner, S.; Mohr, F.; and Hüllermeier, E. 2020. Run2Survive: A decision-theoretic approach to algorithm selection based on survival analysis. In *Proceedings of the 12th Asian Conference on Machine Learning, ACML 2020*, 737–752.

Tornede, T.; Tornede, A.; Hanselle, J.; Wever, M.; Mohr, F.; and Hüllermeier, E. 2021b. Towards green automated machine learning: Status quo and future directions. *CoRR*, abs/2111.05850.

van Rijn, J. N.; Holmes, G.; Pfahringer, B.; and Vanschoren, J. 2014. Algorithm selection on data streams. In *International Conference on Discovery Science, DS 2014*, 325–336.

van Rijn, J. N.; Holmes, G.; Pfahringer, B.; and Vanschoren, J. 2015. Having a blast: Meta-learning and heterogeneous ensembles for data streams. In *2015 IEEE International Conference on Data Mining, ICDM 2015*, 1003–1008.

van Rijn, S.; Doerr, C.; and Bäck, T. 2018. Towards an adaptive CMA-ES configurator. In *Parallel Problem Solving from Nature - PPSN XV*, 54–65.

Vaswani, S.; Mehrabian, A.; Durand, A.; and Kveton, B. 2020. Old dog learns new tricks: Randomized UCB for bandit problems. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020*, 1988–1998.

Wolpert, D. H.; and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1): 67–82.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2007. SATzilla-07: The design and analysis of an algorithm portfolio for SAT. In *International Conference on Principles and Practice of Constraint Programming*, 712–727. Springer.