Machine Learning (2023) 112:1253–1286
https://doi.org/10.1007/s10994-022-06161-4

Algorithm selection on a meta level

Alexander Tornede[1] · Lukas Gehring[1] · Tanja Tornede[1] · Marcel Wever[2] · Eyke Hüllermeier[2]

Received: 8 March 2021 / Revised: 19 October 2021 / Accepted: 19 February 2022 /
Published online: 18 April 2022

Abstract
The problem of selecting an algorithm that appears most suitable for a specific instance of an algorithmic problem class, such as the Boolean satisfiability problem, is called instance-specific algorithm selection. Over the past decade, the problem has received considerable attention, resulting in a number of different methods for algorithm selection. Although most of these methods are based on machine learning, surprisingly little work has been done on meta learning, that is, on taking advantage of the complementarity of existing algorithm selection methods in order to combine them into a single superior algorithm selector. In this paper, we introduce the problem of meta algorithm selection, which essentially asks for the best way to combine a given set of algorithm selectors. We present a general methodological framework for meta algorithm selection as well as several concrete learning methods as instantiations of this framework, essentially combining ideas of meta learning and ensemble learning. In an extensive experimental evaluation, we demonstrate that ensembles of algorithm selectors can significantly outperform single algorithm selectors and have the potential to form the new state of the art in algorithm selection.

Keywords Algorithm selection · Meta learning · Ensemble learning · Bagging · Boosting · Stacking

Editors: Annalisa Appice, Grigorios Tsoumakas.

✉ Alexander Tornede
alexander.tornede@upb.de

Lukas Gehring
lgehring@mail.upb.de

Tanja Tornede
tanja.tornede@upb.de

Marcel Wever
marcel.wever@ifi.lmu.de

Eyke Hüllermeier
eyke@ifi.lmu.de

[1] Heinz Nixdorf Institut, Paderborn University, 33098 Paderborn, Germany

[2] University of Munich (LMU), 80538 Munich, Germany

1 Introduction

Looking at algorithmic problem classes such as Boolean satisfiability (SAT) (Xu et al., 2007, 2011), the traveling salesman problem (TSP) (Pihera & Musliu, 2014), or constraint satisfaction (CSP) (Lobjois et al., 1998), practical experience suggests that algorithms perform differently on different problem instances: while algorithm A might be better than B on a specific instance (e.g., a specific TSP), B may outperform A on another instance (e.g., another TSP). This is not very surprising and completely in line with theoretical results proving that there is "no free lunch", i.e., excluding that one algorithm uniformly dominates all others (Wolpert et al., 1997). The following task thus appears to be meaningful from a practical point of view: Given a problem class and a pool of algorithms to choose from, find a rule that automatically assigns a (presumably) most suitable algorithm to each possible problem instance. This task is called (instance-specific) algorithm selection (AS) in the literature (Rice, 1976). Here, suitability may refer to different performance criteria, such as runtime (Tornede et al., 2020c) or a measure of solution quality (Wever et al., 2021).

The problem of algorithm selection has received considerable attention over the past decade, resulting in a large set of heterogeneous algorithm selection approaches. Many of these approaches rely on machine learning, which essentially means that a rule assigning algorithms to problem instances is learned from suitable training data, for example, the performance observed in the past when running specific algorithms on specific instances. Given a new instance, a machine learning algorithm leverages such data to predict the performance of the candidate algorithms, or to predict the presumably best algorithm directly. AS approaches of that kind achieve state-of-the-art performance and typically outperform the best stand-alone algorithm, also referred to as "single best solver" (SBS) in the following, by several orders of magnitude (Kerschke et al., 2019).

Interestingly, because an algorithm selector is again an algorithm (taking an instance as input and returning a presumably best algorithm as output), the very same task of algorithm selection can also be considered on a meta level, giving rise to the following question: Given a problem instance and a set of algorithm selectors, which one should be used to predict the best algorithm? This question could be answered by an algorithm selector on the meta level, that is, by an "algorithm selector selector", which does not choose among the algorithms (or "base algorithms", to distinguish them from the AS algorithms), but among the algorithm selectors, which in turn are responsible for selecting an algorithm. Indeed, a certain complementarity among AS approaches can be observed (e.g. Tornede et al., 2020c) and the resulting meta-AS problem was first mentioned by Lindauer et al. (2019) and Kerschke et al. (2019), though without pursuing it further.

Having the choice between a set of candidate algorithm selectors, limiting oneself to choosing only *a single* one of them (which in turn chooses the final algorithm) might actually seem unnecessarily restrictive. In fact, leveraging a *composition* of selectors, which then choose the final algorithm jointly, might be a better idea. This naturally leads to *ensemble learning* (Dietterich, 2000), which is a common approach in machine learning to combine several predictors into stronger compositions. Thus, instead of using a single algorithm selector to choose an algorithm, a set of selectors is asked to evaluate the available algorithms. Subsequently, these evaluations are aggregated into a joint decision. Somewhat surprisingly, building ensembles of algorithm selectors has hardly been considered in the AS literature so far (see Sect. 7), although ensemble learning is well known to improve predictive accuracy in standard machine learning problems

such as classification and regression. One reason could be that querying multiple models obviously takes more time than querying only a single one, so that ensembling may appear counterintuitive in scenarios where runtime is considered as the target measure.

In this paper, we formalize the problem of meta algorithm selection and propose algorithmic solutions. Furthermore, we investigate their potential to make better decisions with respect to the selection of algorithms. In an extensive empirical study, we find that trying to learn the best algorithm selector, i.e., to predict which algorithm selector will pick the best algorithm for a given query, does not lead to better algorithm selection performance. On the other side, ensembling algorithm selectors helps to improve efficacy, while the additional runtime consumed for querying multiple algorithm selectors remains negligible. Of course, the improved performance comes at a higher cost of building the ensemble algorithm selector, because multiple basic algorithm selectors need to be fitted for one ensemble. However, this does not pose a problem in practice, because algorithm selectors are in general built in an offline phase prior to the actual selection process.

The remainder of the paper is structured as follows. First, we give a formal introduction to the algorithm selection problem in Sect. 2, followed by a definition of the meta AS problem in Sect. 3 and a first (still quite limited) solution to the problem in Sect. 4. As a more advanced solution, we present algorithm selection ensembles in Sect. 5. Subsequently, we present and discuss the results of our empirical evaluation in Sect. 6. Related work is discussed in Sect. 7, prior to concluding our paper in Sect. 8.

2 Algorithm selection

In the per-instance algorithm selection problem, first formalized by Rice (1976), we are faced with a space of instances $\mathcal{I}$ of an algorithmic problem class (such as SAT, where every instance is a logical formula) and a finite set of algorithms $\mathcal{A}$, which solve such instances. The goal is to find a map $s : \mathcal{I} \longrightarrow \mathcal{A}$, called algorithm selector, which assigns algorithms to instances. An assignment $a = s(i)$ is interpreted as a recommendation, suggesting that algorithm $a \in \mathcal{A}$ will perform strongly, or perhaps even best among all algorithms, on problem instance $i \in \mathcal{I}$. More formally, the goal is to optimize (expected) performance in terms of a measure $m : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}$, which is also part of the AS problem specification. Hence, the optimal algorithm selector for all instances $i \in \mathcal{I}$, also known as the *oracle* or *virtual best solver* (VBS), is defined as

$$s^*(i) = \arg\min_{a \in \mathcal{A}} \mathbb{E}[m(i, a)]\,, \tag{1}$$

where the expectation accounts for the potential randomness imposed by the algorithm. We denote the algorithm that is best on average (in expectation) on a predefined set of instances as the *single-best solver* (SBS). It constitutes the default baseline in algorithm selection.

Observe that an exhaustive evaluation of all algorithms for computing the VBS is not deemed a solution, because m is usually costly to evaluate and often even requires running the respective algorithm. For example, if runtime is the measure of interest, a single evaluation already results in a solved instance, rendering all other evaluations unnecessary. Hence, instead of performing evaluations at query time, the algorithm selector should make use of gathered knowledge to come to a decision.

2.1 Algorithm selection methods

The majority of AS approaches leverages machine learning techniques to learn (in one way or another) a surrogate performance measure $\widehat{m} : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}$ approximating m while being cheap to evaluate. With such a surrogate performance measure at hand, an exhaustive enumeration, actually excluded for the reasons explained before, does become possible and yields the canonical algorithm selector

$$s(i) := \arg\min_{a \in \mathcal{A}} \widehat{m}(i, a)\,. \tag{2}$$

For the purpose of inferring such a surrogate, the setting is usually assumed to contain a set of training instances $\mathcal{I}_D \subset \mathcal{I}$ on which some (but not necessarily all) of the algorithms in $\mathcal{A}$ have been evaluated, so that performance evaluations $m(i, a)$ are available. Note that the corresponding training performance matrix spanned by $\mathcal{I}_D$ and $\mathcal{A}$ is usually assumed to contain (sometimes many) missing values. Furthermore, instances are assumed to be representable by a set of d features generated by a feature map $f : \mathcal{I} \longrightarrow \mathbb{R}^d$. In many cases, such features are available or can be defined in a quite natural way. In the case of SAT, for example, common features include the length of a formula, the number of clauses or variables, etc. In general, the computation of features does not come for free and requires time. This should be taken into account, especially when runtime is chosen as a performance measure to be optimized.

One of the most straight-forward instantiations of the framework described above, in this paper denoted by PerAlgo, was proposed by Xu et al. (2007), where one performance surrogate $\widehat{m}_a : \mathcal{I} \longrightarrow \mathbb{R}$ is learned for each algorithm $a \in \mathcal{A}$ separately. The joint surrogate can then be defined as $\widehat{m}(i, a) = \widehat{m}_a(i)$ for all instances $i \in \mathcal{I}$.

Alternatively, the problem can be formalized as a multi-class classification problem, where each algorithm corresponds to a class, so that a multi-class classifier (here called Multiclass) of the form $s : \mathcal{I} \longrightarrow \mathcal{A}$ can be learned directly. A well-known example from this category is SATzilla'11 (Xu et al., 2011), which employs an all-pairs decomposition approach, learning a cost-sensitive classifier for each pair of algorithms and determining the selected algorithm by majority voting. Building upon the idea of pairwise comparisons of algorithms, Hanselle et al. (2020) suggest learning selectors via a combined ranking and regression approach. Similarly, Kotthoff (2012) suggests employing a stacking approach, using regression models to predict the performance of each algorithm, which is used as an additional input for a meta-learner selecting the final algorithm.

Focusing on so-called censored information present in algorithm selection data, Tornede et al. (2020c) propose a decision-theoretic approach (R2S-PAR10 and R2S-EXP), leveraging techniques from survival analysis to effectively learn from such censored information. Similarly, Hanselle et al. (2021) consider the censored information present in the data within the framework of superset learning (Hüllermeier, 2014).

Furthermore, instance-based approaches, such as SUNNY (Amadini et al., 2014) or ISAC (Kadioglu et al., 2010), have proven to successfully perform algorithm selection by exploiting performances recorded on similar instances in the training data. To this end, they employ k-nearest neighbor or clustering techniques in order to estimate the performance of an algorithm on an unseen instance.

Finally, Tornede et al. (2019, 2020a) propose the setting of "extreme algorithm selection", in which the pool of algorithms to choose from can be extremely large. They show that, by leveraging a feature representation not only for problem instances but also for algorithms, convincing selection performance can be achieved even in this setting.

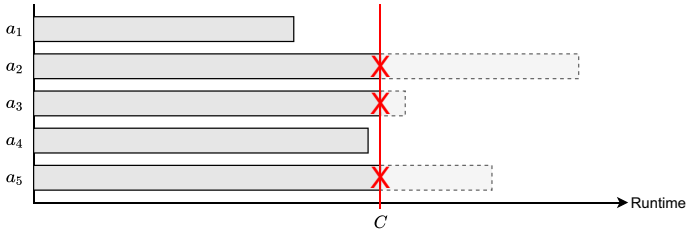

Fig. 1 This figure depicts the process of running multiple algorithms on an instance (e.g. for training data generation). If an algorithm requires longer than C to solve an instance, it is forcefully terminated and a selection of the corresponding algorithm will be punished

2.2 Loss functions

One of the most natural and interesting performance measures to consider for satisfaction problems is the time until the instance is solved, i.e., the algorithm runtime. Unfortunately, combinatorial problems often feature skewed runtime distributions, such that some algorithms are running extremely long on some instances (Gomes et al., 1997). As a consequence, algorithms are generally executed with an upper bound C on their runtime. If an algorithm does not terminate within this bound, called *cutoff*, the instance is considered unsolved and the algorithm is forcefully terminated; see Fig. 1 for an illustration. As choosing an algorithm running into a cutoff leads to an unsolved instance, such a choice should be avoided by all means. One of the most common loss functions in AS, called the penalized average runtime (*PAR10*), considers this by explicitly penalizing such timeouts. The *PAR10* over a set of instances $\mathcal{I}' \subset \mathcal{I}$, called scenario, is defined as follows, where $m(i, s(i))$ corresponds to the runtime of the algorithm $s(i)$ chosen by the algorithm selector s (and potentially the time required to compute the corresponding instance features) on instance i:

$$\begin{aligned} PAR10(s, \mathcal{I}') &= \frac{1}{|\mathcal{I}'|} \sum_{i \in \mathcal{I}'} PR10(s, i) \\ PR10(s, i) &= \begin{cases} m(i, s(i)) & \text{if } m(i, s(i)) \leq C \\ 10 \cdot C & \text{else} \end{cases} \end{aligned} \tag{3}$$

Naturally, *PAR10* scores can vary drastically across scenarios making them incomparable. To alleviate this situation, one often falls back to the normalized PAR10 score of an algorithm selector s defined as

$$nPAR10(s, \mathcal{I}') = \frac{PAR10(s, \mathcal{I}') - PAR10(oracle, \mathcal{I}')}{PAR10(SBS, \mathcal{I}') - PAR10(oracle, \mathcal{I}')}. \tag{4}$$

An *nPAR10* score of 0 corresponds to the oracle performance, a score of 1 corresponds to a performance on a par with the SBS, whereas scores above 1 indicate a deterioration in comparison to the SBS. Therefore, lower *nPAR10* scores indicate better performance, and a successful algorithm selector should definitely have a score of less than 1.

3 Meta algorithm selection

Similar to the algorithms actually solving the problem instances, the algorithm selectors also show the phenomenon of performance complementarity, as mentioned earlier. This gives rise to the question whether choosing between different algorithm selectors might be beneficial. In fact, by moving to the meta level, i.e. from the level of choosing among algorithms to the level of choosing among the algorithm selectors, we gain more freedom and can even select *multiple selectors* instead of only a single algorithm as long as we ensure to aggregate the selections made by the selectors such that a single algorithm is returned at the end. Thus, the problem of per-instance meta algorithm selection (meta AS) concerns the problem of selecting one or multiple algorithm selectors together with an aggregation, for a given instance of an algorithmic problem class. Each of the selected algorithm selectors then in turn selects an algorithm for solving the problem. Finally, these selected algorithms are aggregated such that only a single algorithm (of these) is returned. Hence, instead of directly choosing an algorithm to solve a problem instance, we take a detour by selecting one or multiple algorithm selectors and aggregating their decisions.

Formally, in the meta AS problem, we are given a set of algorithm selectors $\mathcal{S} \subseteq \{s | s : \mathcal{I} \longrightarrow \mathcal{A}\}$, which is a subset of all possible selection functions, in addition to the instance space $\mathcal{I}$, the set of algorithms $\mathcal{A}$ and the performance measure m known from the AS problem. We then seek to find a mapping

$$ass : \mathcal{I} \longrightarrow 2^{\mathcal{S}}, \tag{5}$$

called algorithm selector selector (ASS), and an aggregation function

$$agg : \mathcal{I} \times 2^{\mathcal{S}} \longrightarrow \mathcal{A}, \tag{6}$$

such that the algorithm resulting from the aggregation optimizes the original performance measure m. Accordingly, we seek to find the best pair (agg, ass) of aggregation function agg and algorithm selector selector ass, such that for all instances $i \in \mathcal{I}$ the best algorithm is returned, i.e.,

$$agg(i, ass(i)) \in \arg\min_{a \in \mathcal{A}} \mathbb{E}[m(i, a)] \ . \tag{7}$$

Observe that we principally allow the concrete aggregation to depend on the instance, thereby allowing for *learning instance-specific aggregation* functions.

Figure 2 illustrates the relation between algorithms, algorithm selectors and algorithm selector selectors. In the following, we present several instantiations of this framework.

4 Selecting single algorithm selectors through meta learning

The arguably simplest solution to the meta AS problem is achieved through meta learning (Vanschoren, 2018; Brazdil et al., 2008; Vilalta et al., 2009), namely to learn which algorithm selector takes the best decision for a given instance. More formally, one could seek to learn a map

$$s_{meta} : \mathcal{I} \longrightarrow \mathcal{S}, \tag{8}$$

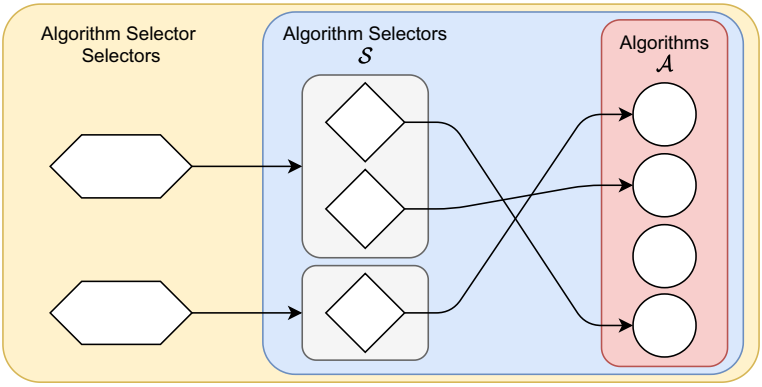

Fig. 2 Illustration of the connection between algorithms ($\mathcal{A}$), algorithm selectors ($\mathcal{S}$) and algorithm selector selectors. Algorithms solve instances of an algorithmic problem, whereas algorithm selectors are mappings from an instance to a *single* algorithm from $\mathcal{A}$. Algorithm selector selectors select *one or multiple* algorithm selectors, which in turn each select an algorithm. These selections are then aggregated using an aggregation function (not displayed here)

such that the chosen selector returns the most suitable algorithm for a given instance i, i.e.,

$$\big(s_{meta}(i)\big)(i) \in \arg\min_{a \in \mathcal{A}} \mathbb{E}[m(i,a)]. \tag{9}$$

In this case, the co-domain of the function ass in (5) is effectively restricted to singleton sets $ass(i) = \{s\} \in \mathcal{S}$ consisting of only a single algorithm selector s — we shall discuss the consequences of this self-imposed restriction in Sect. 4.1. Moreover, the aggregation agg in (6) is the identity, or, stated differently, there is actually no need for learning an aggregation function. Lastly, the instance features computed by the feature map f for the standard AS problem are also used on the meta level and thus constitute what is known as meta features in the context of meta learning. Likewise, as (8) indicates, the set of selectors $\mathcal{S}$ corresponds to the set of meta targets.

Observe that this approach is essentially a special case of the standard AS problem itself, with a very specific set of algorithms to choose from, namely algorithm selectors. Hence, standard AS methods (see Sect. 2.1) can in principle be applied. It is important to note that algorithm selection approaches not relying on a feature representation of instances do not necessarily have an advantage in terms of runtime anymore, because they may select an algorithm selector which in turn requires the feature representation. If the feature computation has to be performed either on the meta or on the base level, its time has to be taken into account as well. However, there is no need to perform the computation twice, if both the algorithm selector selector and the algorithm selector require it, because the resulting features can be shared.

4.1 Limits of learned algorithm selector selection

Limiting ourselves to choosing only a single algorithm selector for a given instance instead of leveraging multiple ones obviously has consequences in terms of achievable algorithm selection performance. To elaborate on these consequences, let us define an algorithm selector oracle (AS-oracle) as

$$ass^*(i) \in \arg\min_{s \in \mathcal{S}} \mathbb{E}[m(i, s(i))] \ . \tag{10}$$

It is important to note that the AS-oracle is in general not identical to the oracle on the base level, as the set of algorithms to choose from may change. For a better understanding, consider an example with two algorithms a_1 and a_2 and two algorithm selectors s_1 and s_2, where both always select algorithm a_1. Furthermore, assume there exists an instance for which a_2 performs better than a_1, and hence the oracle would select a_2. However, the AS-oracle can only select s_1 or s_2, which in turn both select a_1, resulting in a decrease in oracle performance.

Generally speaking, in order to preserve the original oracle, it is necessary that, for each instance, at least one algorithm selector exists that selects the best algorithm for that instance. Otherwise, the AS-oracle performance may degrade compared to the oracle. In practice, there will be at least one such instance most of the time, and hence an important question is how much the oracle performance degrades. As we show in our experimental evaluation, the degradation strongly depends on the scenario at hand, and ranges from less than 1% to over 116%.[1]

Similarly to the oracle, the SBS on the meta level changes as well, since the single best algorithm selector (SBAS), i.e., the algorithm selector which is best on average, is now an algorithm selector, making it a lot stronger baseline than the single best solver. Hence, while the SBS selects the actual problem solving algorithm that is best on average and accordingly does not depend on instance features, the SBAS does in fact depend on such features as long as it is not identical to the SBS. Observe that this results in a significant disadvantage for the SBAS in terms of achievable *PAR10* scores due to the time required to compute these instance features.

Obviously, these implications also influence the performance gains that can be achieved by algorithm selector selectors of the form (8) in comparison to algorithm selectors. As the oracle performance most likely degrades, while the SBS performance most likely improves, the gap between the two also decreases, offering less potential for algorithm selection approaches to close this gap.

5 Constructing ensembles of algorithm selectors

As mentioned earlier, the restriction to choose only a single algorithm selector seems like an unnecessary constraint and may even lead to a potential loss in achievable algorithm selection performance. Accordingly, one may think about using a *composition* of algorithm selectors, which can play to their strengths on some instances while compensating for each other's weaknesses on other instances. This idea motivates us to construct a mapping of the form (5) through *ensemble learning*.

Ensemble learning (Dietterich, 2000) presumably constitutes the most natural technique to combine several machine learning approaches into a joint one, with the goal to improve in performance. In algorithm selection, an ensemble can be thought of as a set of algorithm selectors $\mathcal{S}$, called *base algorithm selectors*, which are either trained independently or dependently on each other. At prediction time, each selector is queried for

[1] Note that on the CPMP-2015 scenario the degradation is even around 900%, but constitutes a clear outlier.

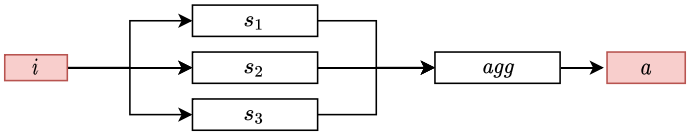

Fig. 3 This figure depicts the general process of predicting / selecting an algorithm for a given instance through a trained ensemble of algorithm selectors s_1, s_2, s_3

the given instance i, and the algorithm choices are aggregated into a final choice using an aggregation function as defined in (6). The concrete strategy used to make the selectors cooperate depends on the ensemble technique being used. Figure 3 depicts the general process of predicting / selecting an algorithm for a given instance through a trained ensemble of algorithm selectors.

As mentioned earlier, allowing for the selection of multiple algorithm selectors also requires the definition of an aggregation function in order to finally return a single algorithm. In principle, the aggregation functions can either depend on the instance, i.e., are instance-specific, or can be fixed across instances. Similarly, they can either be learned or be predefined.

In general, to be successful, ensembles require a certain degree of heterogeneity of the predictions. Therefore, the different algorithm selectors should not always coincide in their selections. Otherwise, it can easily happen that the majority of predictions made by the base selectors are identical. Hence, in such a situation, the prevalent selector (maybe with slight but negligible variations) dominates the predictions of the entire ensemble, only yielding a computationally more expensive variant of the respective dominating selector. To avoid this problem, most ensemble methods strive for a heterogeneous set of base selectors. This can be achieved through a suitable choice of base selectors given to the method, like for example in voting. Alternatively, in the case of methods such as bagging, which only work with a single base selector, different variants of the same selector can be trained on different data sets.

Intuitively, the training and querying of more than one selector might be counter-intuitive in settings where runtime is optimized, as it automatically results in larger runtime. In this regard, it is important to note that the majority of the runtime is required for training the selectors in the ensembles. In AS, we can assume this training to be performed *offline*, i.e., prior to the actual selection of algorithms. Hence, longer training times do not constitute a real disadvantage, as long as prediction (querying the ensemble members) remains fast, which is the case as most selectors are known to be extremely fast such that even compositions of them are slower, but still fast.

In the following, we first elaborate on different aggregation strategies. Although some of these aggregation functions include learnable components, they are fixed across instances, i.e., the aggregation of predictions does not depend on the query instance. Then, we present several ensemble techniques for creating a pool of algorithm selectors, in particular voting (Dietterich, 2000), bagging (Breiman, 1996), and boosting (Schapire, 1990). We continue with a discussion of stacking (Wolpert, 1992), which can be seen as a *learned, instance-specific* aggregation method. As such, it is somehow positioned in-between ensemble and meta learning. Finally, we close this section with a methodological comparison of the presented approaches.

5.1 Aggregation strategies

One of the most natural forms of aggregation in our context is *(weighted) majority aggregation*. As the name suggests, it aggregates the algorithm choices by selecting the algorithm that was selected most frequently, potentially weighting the choices of the selectors differently. This is motivated by the idea that selectors with a strong performance should potentially be trusted more than weaker ones. More formally, weighted majority aggregation can be defined as[2]

$$agg_{(w)maj}(i, \mathcal{S}) = \arg\max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} w_s \cdot [\![s(i) = a]\!] , \tag{11}$$

where $w_s \in \mathbb{R}^+$ denotes the weight associated with selector s. With $w_s = 1$ for all $s \in \mathcal{S}$, we recover standard majority voting. To obtain proper weights, a plethora of methods are applicable in principle. However, we simply consider the *nPAR10* score of the different base algorithm selectors on the training data in order to determine corresponding weights — conducting a cross-validation on the training data for the same purpose turned out to result in similar performance while being computationally more expensive.

Up to now, we assumed that an algorithm selector only returns a single algorithm. While this is typically true in practice, the majority of approaches internally feature more nuanced predictions, often constituting some kind of loss (or score) for each algorithm in $\mathcal{A}$. Accordingly, instead of using only a concrete algorithm choice as the output of the algorithm selectors, we adapted them to return such nuanced predictions where possible.

More formally, let us assume that each *trained* algorithm selector $s \in \mathcal{S}$ cannot only be evaluated on $i \in \mathcal{I}$, but that it also allows access to $\widehat{m}_s(i, a)$, i.e., to the corresponding internal score of each algorithm $a \in \mathcal{A}$. For those approaches where such a score cannot be extracted explicitly, e.g., multi-class algorithm selectors, we define dummy losses as

$$\widehat{m}_s(i, a) = \begin{cases} 0 & \text{if } s(i) = a \\ 1 & \text{else} \end{cases} \tag{12}$$

for all instances $i \in \mathcal{I}$ and algorithms $a \in \mathcal{A}$, such that all approaches can be assumed to work as defined in (2).

With this consideration, aggregations on this more nuanced level of scores instead of the level of final choices can be made. The most straight-forward aggregation function on this level is the *arithmetic mean*, i.e.,

$$agg_{avg}(i, \mathcal{S}) = \arg\min_{a \in \mathcal{A}} \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \widehat{m}_s(i, a) . \tag{13}$$

While conceptually simple, it requires the performance surrogates of the different selectors to approximate the same function. Otherwise, the predictions are incomparable, and averaging is not a meaningful operation. For example, combining the output of a ranking loss function optimized by one selector with the estimated average *PAR10* scores of another does not make any sense. In principle, the arithmetic mean can also be turned into a weighted version as done in (11).

[2] $[\![\cdot]\!]$ denotes the indicator function evaluating to 1 if the expression is true, and to 0 otherwise.

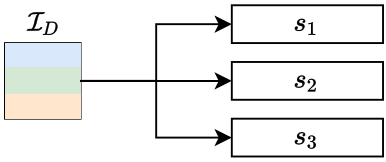

Fig. 4 This figure depicts the training process of a voting ensemble, where each base algorithm selector is trained with the same training instances. Ensemble heterogeneity is achieved by choosing a heterogeneous set of algorithm selectors in advance

In order to be able to aggregate on this more nuanced level while overcoming the weakness of the arithmetic mean, we propose to aggregate *rankings* (rank aggregation) of algorithms constructed from the algorithm scores obtained from the selectors. More precisely, we can assume that each selector s returns a ranking over the algorithms in $\mathcal{A}$ by sorting them in increasing order w.r.t. $\hat{m}_s(i, \cdot)$, such that the presumably best algorithm is put on the first position in the ranking, the second-best on the second position, etc. Having obtained such a ranking over the algorithms for each selector, they need to be aggregated in order to draw a conclusion and eventually return a single algorithm as the final choice.

A very simple method for rank aggregation is called *Borda count* (Borda, 1784). Given a ranking of n items, it assigns n points to the top item, $n-1$ points to the second-best, and so forth. This is done for each ranking to be aggregated, and the consensus ranking is obtained by sorting the items (algorithms in our case) in descending order according to their total sum of points. As pointed out by Dwork et al. (2001), the Borda count has a number of less appealing properties, at least from a theoretical point of view. On the other side, its linear time complexity makes it fast to compute. This is in sharp contrast to other rank aggregation techniques that involve intractable optimization problems (Dwork et al., 2001). Besides, Borda comes with provable approximation guarantees for several other aggregation techniques (Coppersmith et al., 2006). Overall, it seems to be a good compromise for the case of algorithm selection, where predictions are performed under tight time constraints.

Formally, we can use Borda count as an aggregation function for our setting as follows, where $rank : \mathcal{I} \times \mathcal{S} \times \mathcal{A} \to \mathbb{N}$ returns the rank of an algorithm a in the ranking returned by a selector s on an instance i:

$$agg_{borda}(i, \mathcal{S}) = \arg\min_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} rank(i, s, a) \tag{14}$$

Ties are handled by assigning to all tied algorithms the average of the block of ranks they occupy (Saari, 2000). In practice, ties can only be caused through the dummy scores introduced in (12). Therefore, they always occur at the end of the rankings. Theoretically, identical scores of $\hat{m}(i, \cdot)$ could also result in ties, but this never happened in practice.

While the aggregation techniques outlined above appear to be meaningful in the context of the algorithm selection task, we would like to point out that other aggregation techniques are of course conceivable and could be used instead.

5.2 Voting

Voting ensembles are presumably the easiest form of ensemble learning: Each algorithm selector in a set $\mathcal{S}' \subseteq \mathcal{S}$ is trained independently of the others on the same training data $\mathcal{I}_D$.

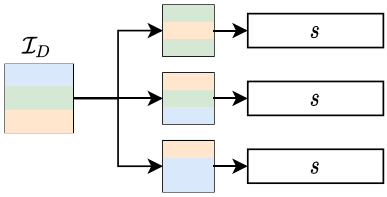

Fig. 5 This figure depicts the training process of a bagging ensemble consisting of several instantiations of the same base algorithm selector trained on bootstrapped versions of the original training data

At prediction time, all algorithm selectors in $\mathcal{S}'$ are queried, and the predictions are aggregated using one of the previously described aggregation strategies. Figure 4 depicts the training process of a voting ensemble.

As we demonstrate empirically, it is important to optimize the ensemble composition, i.e., the set of base algorithm selectors $\mathcal{S}' \subseteq \mathcal{S}$ specifying the ensemble, because the performance of a voting ensemble solely depends on this configurable parameter. Intuitively, a complete evaluation of each possible composition to check the corresponding performance might seem intractable due to the exponential (in $|\mathcal{S}|$) number of compositions. However, in practice this can be a viable option under certain circumstances. To this end, we hold back a portion of the training data $\mathcal{I}_D$ as validation data $\mathcal{I}'_D \subset \mathcal{I}_D$. Then, all base algorithm selectors can be trained on the reduced training data $\mathcal{I}_D \setminus \mathcal{I}'_D$ once, so that, in order to estimate the performance of an ensemble composition, only the predictions of the used selectors on the validation data $\mathcal{I}'_D$ need to be obtained and aggregated.[3] As the training of the selectors has to be performed only once at the beginning, and the computation of both the predictions and the aggregation can be performed in a negligible amount of time, the evaluation of all possible compositions is feasible as long as the set of algorithm selectors remains moderately large. For example, computing the training performance of each possible voting ensemble composed of up to 7 algorithm selectors required less than 5 minutes for all scenarios presented in Sect. 6. However, we want to stress that this approach still has an exponential complexity even if the corresponding predictions can be obtained quite fast as the number of ensemble compositions to evaluate is exponential in the number of algorithm selectors. Thus, if the amount of algorithm selectors becomes larger, more sophisticated ensemble pruning methods as Rokach (2009), Lazarevic and Obradovic (2001) and Hernández-Lobato et al. (2009) can be used to find good compositions.

5.3 Bagging

In contrast to voting, *bagging*[4] (Breiman, 1996) only leverages a single kind of algorithm (selector). Therefore, heterogeneity between the ensemble members has to be achieved through data manipulation techniques. To this end, bagging leverages a data resampling technique from statistics called *bootstrapping*, which works as follows. Given a set of training instances $\mathcal{I}_D$ of size $N = |\mathcal{I}_D|$, it creates a new training instance set by sampling N times from $\mathcal{I}_D$ *with replacement*. The actual ensemble is constructed by sampling k such new training instance sets $\mathcal{I}_D^{(1)}, \ldots, \mathcal{I}_D^{(k)}$ and training one

[3] We note that, although theoretically sound, we do not split up validation data for the ensemble optimization as this resulted in worse performance in practice and thus simply evaluate the performance of a composition on the same training data. Note that despite this, the final evaluation of an approach is still performed on separate test data.

[4] The term is short for short for "bootstrap aggregating".

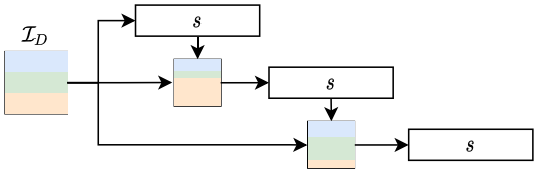

Fig. 6 This figure depicts the training process of a boosting ensemble. Similar to bagging, the ensemble constitutes several instances of the same base algorithm selector. These are subsequently trained on differently weighted versions of the training data

instantiation of the provided algorithm selector on each of the k training sets. Thus, the ensemble eventually consists of k algorithm selector instances. At prediction time, one of the previously discussed aggregation functions can be used to aggregate the predictions (selections) of the different selectors. Figure 5 depicts the training process of a bagging ensemble.

We would like to point out that we bootstrap on the level of the problem instances and not on the level of the actual training data points ((instance/algorithm)-pairs or (instance/algorithm performance)-pairs). This is done in order to allow the selection algorithms themselves to construct their training data points. In principle, this may lead to differently large training data sets for the corresponding base algorithm selectors if the number of training performance values $m(i, \cdot)$ varies across instances. However, we assume that either $m(i, a)$ is available or we know at least that $m(i, a) > C$ for all $i \in \mathcal{I}_D, a \in \mathcal{A}$, and hence can reasonably impute these missing values, thereby solving the problem of differently sized training data sets.

5.4 Boosting

While both voting and bagging fit ensemble members independently of each other (except for (partially) identical training data), boosting *successively* trains its members, each time re-weighting the training instances (Schapire, 1990). After each iteration, i.e., trained selector, the error of the previous selectors is determined and more weight is put onto those instances where a wrong algorithm selection has been performed, while the weight on correctly judged instances is reduced. Similar to bagging, boosting only uses a single selector as a basis of which it trains instantiations based on differently weighted versions of the same training instance set in order to achieve diversity w.r.t. its ensemble members. At prediction time, the predictions of each of the trained selectors are obtained and combined into a joint prediction using a weighted aggregation, using the weights that have been determined as part of the boosting algorithm during the training phase. Figure 6 illustrates the training process of a boosting ensemble.

In boosting algorithms for multi-class classification, such as SAMME (Hastie et al., 2009), and regression problems, such as AdaBoost.R2 (Drucker, 1997), one would naturally consider multi-class classification errors and regression losses, respectively, for re-weighting training instances. However, due to the inferior performance of AdaBoost.R2 in preliminary experiments, we focus on SAMME for the remainder of this paper.

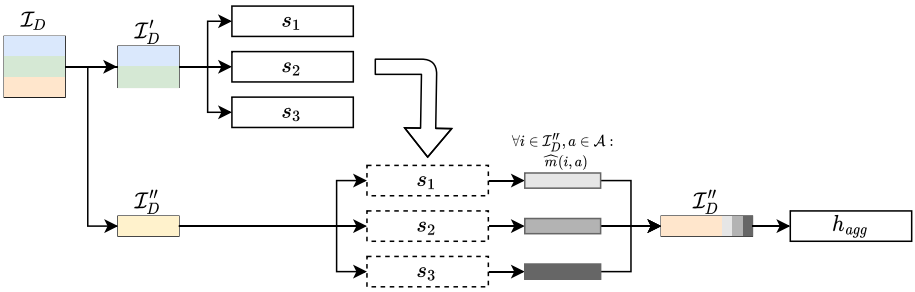

Fig. 7 This figure depicts the general idea behind a stacking ensemble. Each ensemble member is trained with the same subset of training instances and the remaining instances are augmented with the corresponding predictions of the trained selectors. Then, a meta-learner, i.e. an additional algorithm selector, h_{agg} is trained on this augmented data, which decides on the algorithm to select

5.5 Stacking

In the previous ensemble techniques, the aggregation strategy is always fixed from the beginning and independent of the actual instance at hand. The idea of stacking is to learn the aggregation, i.e., how to best aggregate the predictions of the base algorithm selectors for a given instance. Therefore, a meta-learner

$$\mathbf{h}_{agg} : \mathcal{I} \times \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|} \to \mathcal{A} \tag{15}$$

is fitted and used to aggregate the predicted performances $\widehat{m}(i,a)$ of each algorithm selector $s \in \mathcal{S}$ for a given instance $i \in \mathcal{I}$ and each algorithm $a \in \mathcal{A}$ into a joint decision. To avoid any bias in the training data for the meta-learner, it needs to be ensured that this data is disjoint from the training data of the base algorithm selectors. Therefore, the set of training instances $\mathcal{I}_D$ is normally split into a set of base algorithm selector training instances $\mathcal{I}'_D \subset \mathcal{I}_D$ and a set of meta-learner training instances $\mathcal{I}''_D \subset \mathcal{I}_D$ such that $\mathcal{I}'_D \cap \mathcal{I}''_D = \emptyset$.[5] As all possible base algorithm selectors are used, each can be trained independently on the same subset of training instances $\mathcal{I}'_D$ as a first step such that the training data for the meta-learner can be built. Then, the meta-learner is trained based on the features $f(i) \in \mathbb{R}^d$ of each training instance $i \in \mathcal{I}''_D$ extended by the predictions $\widehat{m}_s(i,\cdot)$ of all base algorithm selectors $s \in \mathcal{S}$ on these instances. At prediction time, each base algorithm selector $s \in \mathcal{S}$ is queried, its predictions $\widehat{m}_s(i,\cdot)$ are concatenated and attached to the instance features $f(i) \in \mathbb{R}^d$ of instance i, based on which the meta-learner predicts which algorithm to choose. As the meta-learner is an algorithm selector itself, any of the base algorithm selectors can be used. Figure 7 depicts the general idea of a stacking ensemble.

Since stacking is working on an (extended) feature representation, standard feature selection techniques can be used to reduce the number of features and help the meta-learner achieve better prediction performance. Thus, the ensemble composition does not require any optimization upfront. For an overview of feature selection methods, we refer to Guyon and Elisseeff (2003).

[5] Although theoretically correct, we did actually not do that split in our experimental evaluation in Sect. 6, where is led to worse empirical performance.

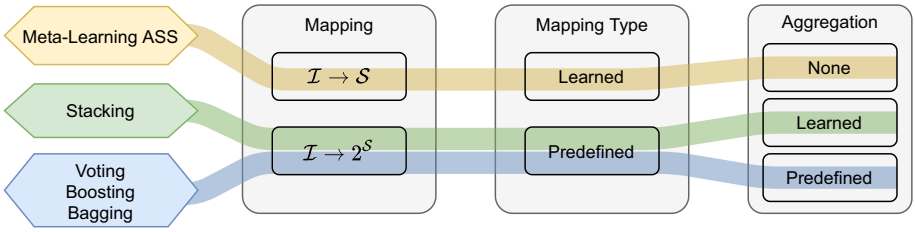

Fig. 8 Illustration of the different approaches w.r.t. the kind of mapping they model, how this mapping is constructed, and how the required aggregation is obtained

5.6 Comparison of the approaches

To put the approaches presented so far into the broader context of meta AS, we close this section by revisiting them w.r.t. to their most important properties. Figure 8 provides an overview and illustrates how the approaches relate to each other. It clarifies what kind of mapping these approaches model, how this mapping is constructed, and how the required aggregation function is constructed.

As an important observation, note that some approaches involve learning on the meta level while others do not. The former most obviously holds for learning an algorithm selector selector (cf. Sect. 4), where the modeled mapping is learned directly. On the other side, most ensemble approaches (cf. Sect. 5) do not require any learning on the meta level, because their mapping is essentially predefined. Stacking is somehow in-between these two groups: the mapping itself is predefined, but the aggregation function is learned on the meta level.

6 Experimental evaluation

In this section, we provide an empirical evaluation of the ideas presented in the preceding sections. It is organized into four main parts. First, we introduce our experiment setup. Second, we investigate the chance for performance improvements when learning algorithm selector selectors and evaluate the performance of standard algorithm selectors working as algorithm selector selectors. Third, we evaluate the performance of the different ensemble methods presented earlier and discuss the results. We end this section by drawing a broader conclusion from these results.

6.1 Experiment setup

All evaluations are run on a subset of the scenarios from the ASlib v4.0 benchmark suite (Bischl et al., 2016) with a 10-fold cross-validation, where the folds are provided by the benchmark. ASlib is a curated collection of algorithm selection problems spanning a variety of different problems such as the Boolean satisfiability problem (SAT), the quantified Boolean formula problem (QBF) and others. Most of these problems focus on runtime as a measure of interest as such problems often exhibit the property of performance

Table 1 Overview of examined ASlib scenarios including their number of instances (#I), unsolved instances (#U), algorithms (#A), provided features (#F), and the cutoffs (C)

	ASP-POTASSCO	BNSL-2016	CPMP-2015	CSP-2010	CSP-MZN-2013	CSP-Minizinc-Time-2016	GLU-HACK-18	MAXSAT-PMS-2016	MAXSAT-WPMS-2016	MAXSAT12-PMS	MAX-SAT15-PMS-INDU	PRO-TEUS-2014	QBF-2011
#I	1294	1179	527	2024	4642	100	353	601	630	876	601	4021	1368
#U	82	2	0	253	944	17	116	45	89	129	46	456	314
#A	11	8	4	2	11	20	8	19	18	6	29	22	5
#F	138	86	22	86	155	95	50	37	37	37	37	198	46
C	600	7200	3600	5000	1800	1200	5000	1800	1800	2100	2100	3600	3600

	QBF-2014	QBF-2016	SAT03-16_INDU	SAT11-HAND	SAT11-INDU	SAT11-RAND	SAT12-ALL	SAT12-HAND	SAT12-INDU	SAT12-RAND	SAT15-INDU	SAT18-EXP
#I	1254	825	2000	296	300	600	1614	767	1167	1362	300	353
#U	241	55	269	77	47	108	20	229	209	322	19	67
#A	14	24	10	15	18	9	31	31	31	31	28	37
#F	46	46	483	115	115	115	115	115	115	115	54	50
C	900	1800	5000	5000	5000	5000	1200	1200	1200	1200	3600	5000

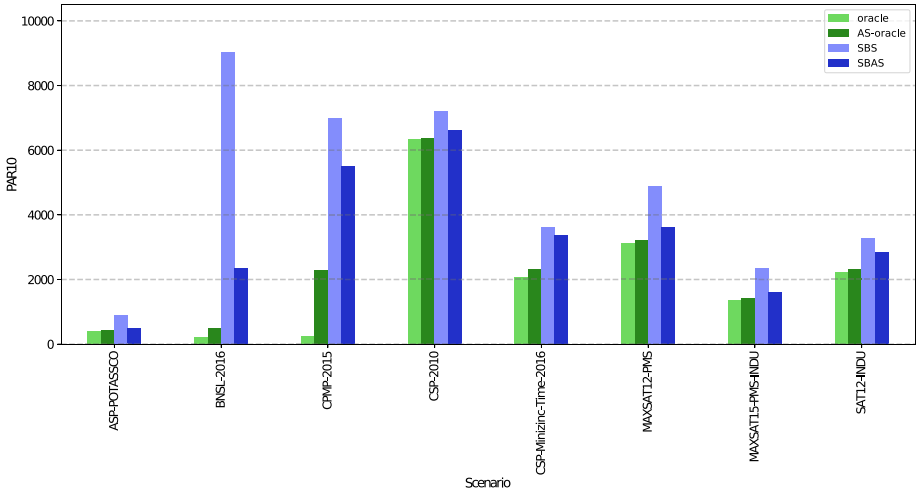

Fig. 9 This figures shows the PAR10 scores of the oracle, AS-oracle, SBS and SBAS on a subset of the ASlib v4.0 benchmark scenarios as bar charts

complementarity motivating the AS problem. Table 1 shows the scenarios used with their corresponding characteristics. Details regarding each scenario can be found in the corresponding README file of the ASlib GitHub repository[6] and in Bischl et al. (2016).

The performance of the approaches is measured in terms of the *normalized penalized average runtime* (*nPAR10*) metric as defined in (4) if not mentioned otherwise. Recall that a value of 0 indicates oracle performance, values below 1 an improvement over the SBS, and values above 1 a degradation compared to the SBS. To allow for a better visual interpretation, we sometimes illustrate results aggregated over all scenarios. Needless to say, such aggregations have to be treated with care, because (differences between) performance degrees are not easily comparable across scenarios.

The set of algorithm selectors used for the evaluation consists of $\mathcal{S} = \{$PerAlgo, SATzilla'11, R2S-Exp, R2S-PAR10, SUNNY, ISAC, Multiclass$\}$, which all have been described in Sect. 2. These are used both as meta-learners, but also as base algorithm selectors for the ensembles. Furthermore, we compare all ensemble variants against the *single best algorithm selector* (SBAS), i.e., the algorithm selector which performs best across all scenarios in terms of average or median *nPAR10* performance. Lastly, we note that in general we leave out instances from the test sets where all algorithms run into the cutoff as no sensible selection is possible for those. However, we do include these instances for the meta learning experiments in Sect. 6.2 as the set of instances in the test sets would otherwise vary between the base level and the meta level yielding incomparable results. This is the case, as we would potentially need to leave out an instance on the meta level (if none of the algorithm selectors chose an algorithm solving it before the cutoff), which we might have included on the base level (since there exists an algorithm solving it before the cutoff time). This problem is very much related to the degradation in oracle performance, which was previously discussed.

[6] https://github.com/coseal/aslib_data.

All experiments were run on machines featuring Intel Xeon E5-2695v4@2.1GHz CPUs with 16 cores and 64GB RAM. In the interest of reproducibility of our results, all code, including detailed documentation of the experiments and execution instructions, is available at GitHub.[7]

6.2 Meta learning for selecting an algorithm selector

Figure 9 shows the PAR10 scores of the oracle, AS-oracle, SBS and SBAS on a subset of the ASlib v4.0 benchmark scenarios. As one can see, several of the implications we noted in Sect. 4.1 can be validated empirically. Firstly and most importantly, although the SBS/oracle gap is a lot larger than the SBAS/AS-oracle gap, the SBAS/AS-oracle gaps are non-negligible, and hence constructing an algorithm selector selector can in principle make sense. For example, consider scenarios BNSL-2016 or CPMP-2015 with large SBAS/AS-oracle gaps.

As we noted earlier, the reason why these gaps become smaller is that the oracle performance degrades when moving to the meta level for all scenarios, whereas the SBS performance tends to improve, because the SBAS is essentially an algorithm selector. While the degradation in oracle performance is moderate for the majority of scenarios (less than 10%), the improvement of the SBAS over the SBS is non-negligible, as the more successful the algorithm selectors considered by the algorithm selector selectors are, the larger this performance gain is.

Table 2 shows the *nPAR10* scores of all algorithm selectors and the corresponding algorithm selector selectors of form (8). Moreover, for the algorithm selector selectors, the values in brackets (*a*/*b*) indicate that the approach achieves a performance better or equal to *a* base approaches and is worse than *b* base approaches.

Unsurprisingly, most algorithm selector selectors are able to consistently improve over the SBS. However, moving to the meta level proves to be beneficial for only seven scenarios and these improvements are even distributed across different algorithm selector selectors. To explain this moderate result, we speculate that the considered AS approaches are not able to unleash their full potential on the meta level, although considerable SBAS/AS-oracle gaps exist, as we have seen previously. However, the win/loss scores in brackets indicate that moving to the meta level is beneficial in the sense that a more robust performance across several scenarios can be achieved.

6.3 Voting ensembles

Figure 10 shows the average/median performance in terms of *nPAR10* (over all scenarios) of all possible voting ensemble compositions as violin plots grouped by the aggregation strategy being used. The dashed line indicates the performance of the SBAS, the black dot indicates the performance of the best composition w.r.t. the training performance, whereas the red dot indicates the performance of the ensemble with all base algorithm selectors.

First of all, it is important to note that voting ensembles offer a lot of optimization potential in terms of both mean and median performance in comparison to the SBAS. While a concrete optimization of the ensemble composition (black dots) does not seem to be beneficial, simply using all possible base algorithm selectors as ensemble

[7] https://github.com/alexandertornede/as_on_a_meta_level.

Table 2 PAR10 scores of all base- and algorithm selector selectors normalized wrt. the standard oracle and SBS. The result of the best approach is marked in bold for each scenario. Moreover, for the meta-algorithm selectors the values in brackets (*a*/*b*) indicate that that the approach achieves a performance better or equal to *a* base-approaches and is worse than *b* base-approaches

Level	Algorithm Selectors							Algorithm Selector Selectors (Meta)						
Approach	R2SExp	R2SPAR10	ISAC	MCC	PAReg	SATzilla'11	SUNNY	R2SExp	R2SPAR10	ISAC	MCC	PAReg	SATzilla'11	SUNNY
Scenario														
ASP-POTASSCO	0.30	0.32	0.60	0.64	0.34	0.47	**0.17**	0.24 (6/1)	0.19 (6/1)	0.24 (6/1)	0.36 (3/4)	0.32 (5/2)	0.31 (5/2)	0.26 (6/1)
BNSL-2016	**0.18**	0.21	0.84	0.31	**0.18**	**0.18**	0.25	0.22 (3/4)	0.21 (4/3)	0.19 (4/3)	0.28 (2/5)	0.22 (3/4)	0.28 (2/5)	0.27 (2/5)
CPMP-2015	0.76	**0.69**	0.90	0.85	0.78	0.70	0.94	0.78 (4/3)	0.78 (4/3)	0.89 (2/5)	0.81 (3/4)	0.77 (4/3)	0.81 (3/4)	0.89 (2/5)
CSP-2010	0.13	0.15	0.31	0.80	0.25	0.13	0.34	0.05 (7/0)	**0.04** (7/0)	0.19 (4/3)	0.13 (7/0)	0.46 (1/6)	0.18 (4/3)	0.09 (7/0)
CSP-MZN-2013	0.11	0.11	0.35	0.31	0.13	0.21	0.13	0.11 (7/0)	**0.10** (7/0)	0.13 (5/2)	0.15 (3/4)	0.13 (5/2)	0.19 (3/4)	0.14 (3/4)
CSP-Minizinc-Time-2016	0.43	**0.27**	0.83	0.36	0.67	0.34	0.37	0.51 (2/5)	0.51 (2/5)	0.76 (1/6)	0.60 (2/5)	0.67 (2/5)	0.35 (5/2)	0.51 (2/5)
GLUHACK-18	0.43	0.46	0.69	0.41	0.46	0.42	0.51	**0.40** (7/0)	0.45 (4/3)	0.41 (7/0)	0.49 (2/5)	0.57 (1/6)	0.47 (2/5)	0.46 (4/3)
MAXSAT-PMS-2016	0.60	**0.36**	0.82	1.06	0.77	0.62	0.41	0.64 (3/4)	0.65 (3/4)	0.60 (5/2)	0.71 (3/4)	0.82 (2/5)	0.98 (1/6)	0.75 (3/4)
MAXSAT-WPMS-2016	0.44	0.37	0.76	0.85	0.52	0.31	**0.16**	0.37 (5/2)	0.39 (4/3)	0.62 (2/5)	0.60 (2/5)	0.54 (2/5)	0.43 (4/3)	0.44 (4/3)
MAXSAT12-PMS	0.22	0.23	0.47	0.40	0.28	0.24	0.29	0.25 (4/3)	0.25 (4/3)	**0.20** (7/0)	0.21 (7/0)	0.32 (2/5)	0.22 (7/0)	0.21 (7/0)
MAXSAT15-PMS-INDU	0.34	0.44	0.89	1.06	0.55	0.39	**0.24**	0.36 (5/2)	0.57 (2/5)	0.33 (6/1)	0.39 (5/2)	0.40 (4/3)	0.51 (3/4)	0.26 (6/1)
PRO-TEUS-2014	**0.41**	**0.41**	0.64	0.84	0.45	0.58	0.47	0.47 (4/3)	0.47 (4/3)	0.48 (3/4)	0.48 (3/4)	0.53 (3/4)	0.62 (2/5)	0.53 (3/4)
QBF-2011	0.21	0.20	0.37	0.35	0.18	**0.15**	0.22	0.20 (5/2)	0.21 (4/3)	0.22 (3/4)	0.21 (4/3)	0.29 (2/5)	0.25 (2/5)	0.26 (2/5)
QBF-2014	**0.26**	0.28	0.51	0.59	0.32	0.31	0.31	0.31 (5/2)	0.30 (5/2)	0.32 (3/4)	0.36 (2/5)	0.41 (2/5)	0.39 (2/5)	0.36 (2/5)
QBF-2016	0.52	0.51	0.65	0.69	0.61	0.61	**0.49**	0.55 (4/3)	0.55 (4/3)	0.52 (5/2)	0.53 (4/3)	0.62 (2/5)	0.57 (4/3)	0.58 (4/3)

Table 2 (continued)

Level	Algorithm Selectors							Algorithm Selector Selectors (Meta)						
Approach	R2SExp	R2SPAR10	ISAC	MCC	PAReg	SATzilla'11	SUNNY	R2SExp	R2SPAR10	ISAC	MCC	PAReg	SATzilla'11	SUNNY
Scenario														
SAT03-16-INDU	**0.71**	0.76	0.98	0.99	0.77	0.82	0.82	0.92 (2/5)	0.90 (2/5)	0.80 (4/3)	0.79 (4/3)	0.81 (4/3)	0.84 (2/5)	0.86 (2/5)
SAT11-HAND	**0.34**	**0.34**	0.65	0.57	0.46	0.44	0.60	0.42 (5/2)	0.47 (3/4)	0.42 (5/2)	0.44 (5/2)	0.50 (3/4)	0.45 (4/3)	0.56 (3/4)
SAT11-INDU	0.69	0.69	1.08	0.71	0.63	0.79	0.76	0.78 (2/5)	0.89 (1/6)	0.84 (1/6)	**0.61** (7/0)	0.79 (2/5)	0.73 (3/4)	0.85 (1/6)
SAT11-RAND	0.13	**0.06**	0.59	0.17	0.09	0.39	0.12	0.15 (3/4)	0.12 (5/2)	0.17 (3/4)	0.18 (2/5)	0.18 (2/5)	0.30 (2/5)	0.20 (2/5)
SAT12-ALL	**0.36**	**0.36**	0.67	0.38	0.37	0.44	0.38	0.37 (5/2)	0.40 (2/5)	0.39 (2/5)	0.39 (2/5)	0.40 (2/5)	0.40 (2/5)	0.43 (2/5)
SAT12-HAND	0.34	0.34	0.64	0.41	0.37	**0.27**	0.43	0.34 (6/1)	0.34 (6/1)	0.31 (6/1)	0.38 (3/4)	0.39 (3/4)	0.39 (3/4)	0.38 (3/4)
SAT12-INDU	0.70	0.73	1.02	0.94	0.79	**0.59**	0.78	0.62 (6/1)	0.63 (6/1)	0.75 (4/3)	0.73 (5/2)	0.65 (6/1)	0.65 (6/1)	0.66 (6/1)
SAT12-RAND	0.96	**0.86**	0.91	5.20	1.17	0.93	1.14	0.92 (5/2)	1.02 (3/4)	0.94 (4/3)	1.00 (3/4)	1.25 (1/6)	1.23 (1/6)	1.05 (3/4)
SAT15-INDU	0.95	0.83	0.76	0.91	0.74	0.75	1.00	0.68 (7/0)	0.88 (3/4)	1.00 (1/6)	0.96 (1/6)	**0.65** (7/0)	0.85 (3/4)	0.81 (4/3)
SAT18-EXP	0.61	0.68	0.62	0.65	0.64	0.59	0.63	0.66 (1/6)	0.67 (1/6)	0.61 (6/1)	0.58 (7/0)	0.61 (6/1)	**0.54** (7/0)	0.59 (7/0)

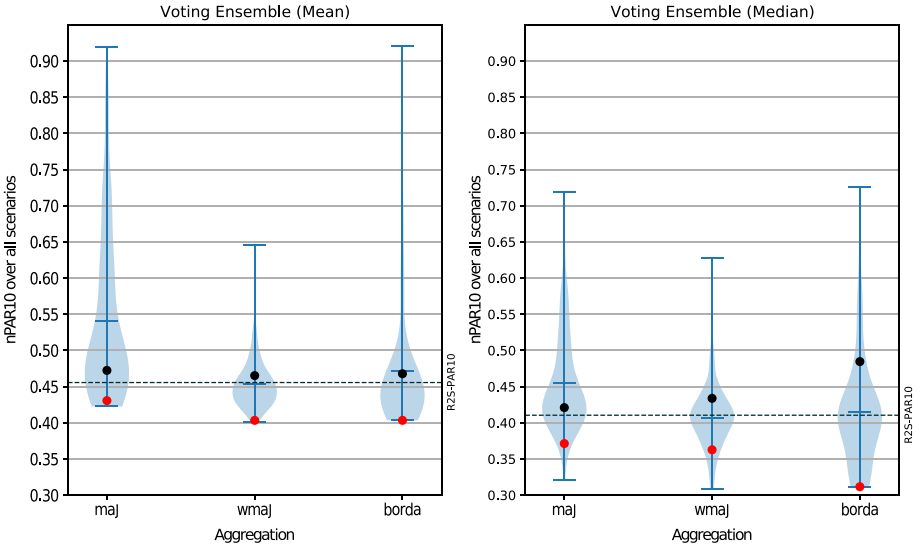

Fig. 10 Mean/median performance in terms of *nPAR10* (over all scenarios) of all possible voting ensemble compositions as violin plots grouped by the aggregation strategy being used. The dashed line indicates the performance of the SBAS, the black dot indicates the performance of the best composition w.r.t. to the training performance, whereas the red dot indicates the performance of the ensemble with all base algorithm selectors

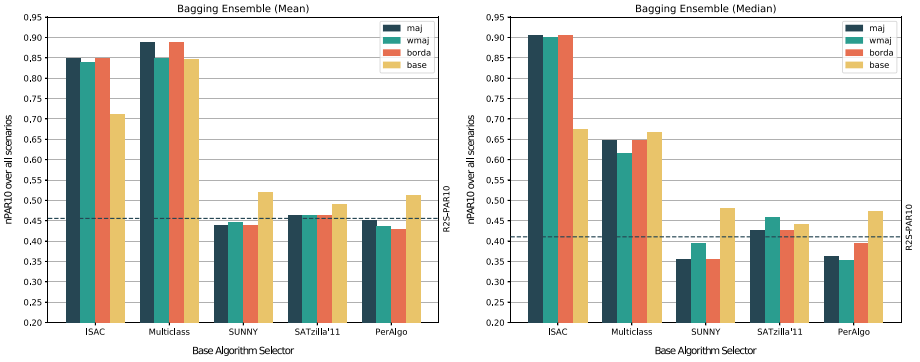

Fig. 11 Average/median *nPAR10* performance over all scenarios of each bagging ensemble with 10 instantiations of the corresponding base algorithm selector and different aggregation functions. Moreover, the performance of the corresponding base algorithm selector is shown. Once again, the dashed line indicates the performance of the SBAS

members often comes close to the lower performance bound of the voting ensemble strategy. Independent of the aggregation strategy, a voting ensemble with all base algorithm selectors is always able to improve over the best single algorithm selector, sometimes even drastically (e.g., Borda aggregation in terms of median performance). Overall, the weighted majority and the Borda aggregation seem to be on a par in terms of performance when considering the mean *nPAR10* score, while Borda is superior in the median case.

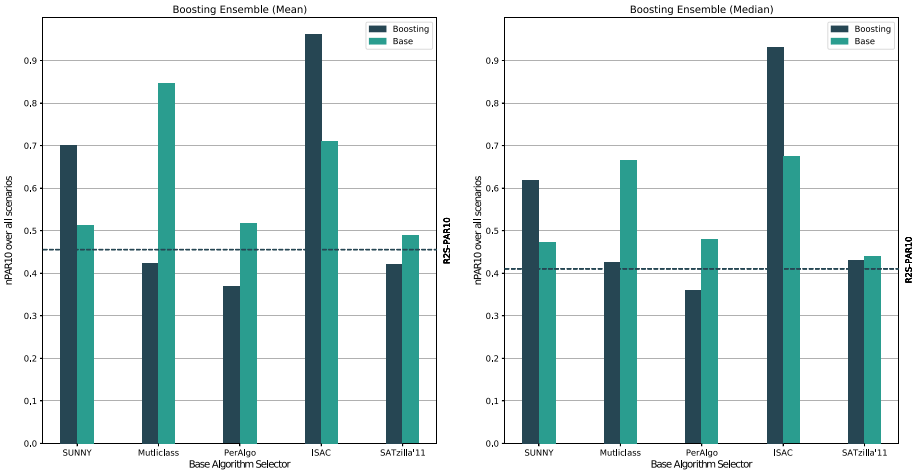

Fig. 12 Average/median *nPAR10* performance over all scenarios of each boosting ensemble with 20 iterations and different aggregation functions. Moreover, the performance of the corresponding base algorithm selector is shown. Once again, the dashed line indicates the performance of the SBAS

It is important to understand the scope of the improvement depicted here. Although R2S-PAR10 already offers a remarkable performance and represents the state of the art in algorithm selection, it is beaten by around 15% (mean) and 32% (median), which constitute tremendous improvements.

6.4 Bagging ensembles

Figure 11 shows the average / median *nPAR10* performance over all scenarios of each bagging ensemble with 10 instantiations of the corresponding base algorithm selector and different aggregation functions. Moreover, the performance of the corresponding base algorithm selector is shown. Once again, the dashed line indicates the performance of the SBAS.

While both ensemble variants equipped with ISAC or Multiclass as a base algorithm selector deteriorate in terms of performance compared to the SBAS, SUNNY, SATzilla'11, and PerAlgo are able to improve both in terms of mean and median performance if the right aggregation is chosen. Surprisingly, none of the aggregation functions seems to be dominating the others. Furthermore, it can be seen that bagging improves the performance of SUNNY, SATzilla'11 and PerAlgo, but mostly worsens the performance for ISAC and offers mixed results for Multiclass.

In light of the general experience with bagging in machine learning, the performance deterioration of the ISAC ensemble in comparison to its base selector may appear surprising. We conjecture that the negative effect of ensembling is due to the specific characteristics of this method. ISAC applies a clustering technique in order to form clusters over the training instances and computes a threshold t based on the average distances of all instances to their corresponding cluster centroid and the standard deviation over these values. At prediction time, ISAC finds the centroid which is closest to the new instance and returns the algorithm performing best on the cluster, if the

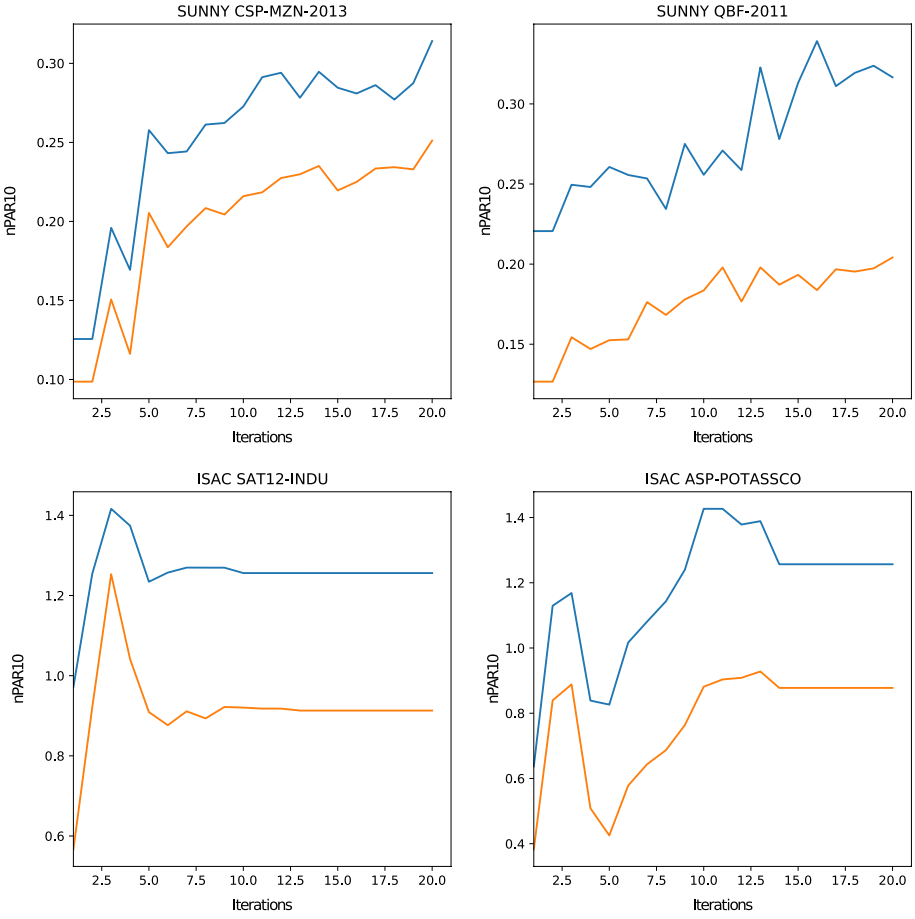

Fig. 13 Learning curves featuring training (orange) and testing (blue) *nPAR10* scores of the SAMME boosting algorithm with SUNNY (top two) and ISAC (bottom two) as a base selector on two instances

distance to the centroid is below the aforementioned threshold. If this is not the case, the SBS is returned. Thus, the threshold can be seen as a fail-safe in case ISAC considers the closest cluster to be too different to draw any reasonable conclusions. After careful investigation, we found that the threshold t decreases for the ensemble members trained on bootstrapped training instance sets as both the average distance and the standard deviation decreases. As a result, the ensemble members mostly deteriorate to the SBS and suggest the SBS on a majority of the instances. This explains the decrease in performance and the similar results of the different aggregation strategies.

We note that Run2Survive was left out as a base algorithm selector for bagging as it cannot easily be trained with bootstrapped instance training sets on scenarios with many censored samples. In such cases, bootstrapping often leads to training data sets consisting of censored samples only, which the approach cannot handle.

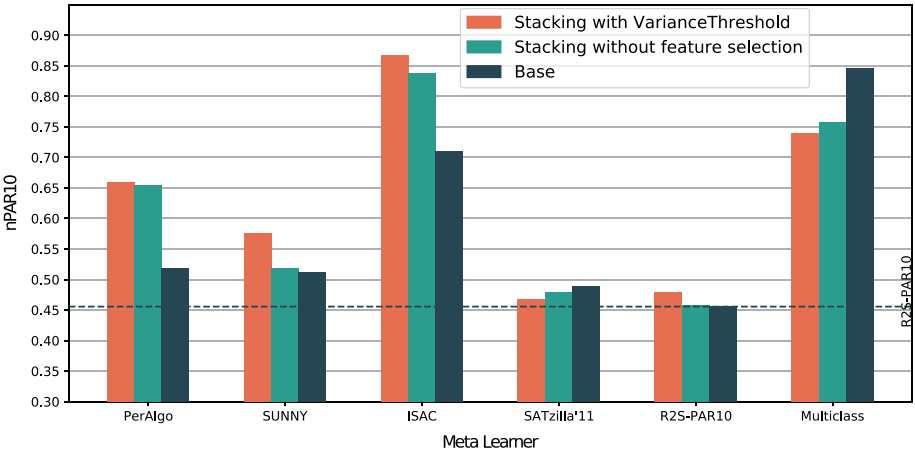

Fig. 14 This figure shows the average *nPAR10* performance of stacking variants where h_{agg}, i.e. the meta-learner, is instantiated through different algorithm selectors with and without a variance threshold feature selection approaches

6.5 Boosting ensembles

Figure 12 shows the average / median *nPAR10* performance over all scenarios of each boosting ensemble with 20 iterations and different aggregation functions.

While the performance of the PerAlgo, Multiclass and SATzilla'11 algorithm selectors improves through boosting, the performance of SUNNY and ISAC degrades. Once again, the degradation of ISAC can be explained by the same phenomenon as in the case of bagging: the instance weighting required by boosting was implemented through data sampling, whence ISAC mostly degenerates to the SBS. We chose to do so, since not all of the base algorithm selectors inherently support instance weights, but we wanted to investigate boosting variants powered by as many base algorithm selectors as possible. The degradation of the performance of SUNNY can also be explained in a similar fashion. Recall that SUNNY essentially is a similar *k*-nearest neighbor algorithm, which, given a new instance, returns the algorithm which performs best in terms of PAR10 performance on the *k* nearest instances in the training data. However, this training data mostly consists of instances with a high weight as all others have a lower chance of being sampled. As a consequence, SUNNY will return the algorithm performing best on average on exactly these instances, while completely ignoring all other instances. This results in degenerate boosting learning curves as depicted in Fig. 13. The problem is less dominant for selectors that generalize in a more sophisticated way across the features, such as PerAlgo or Multiclass. For instance-based approaches such as SUNNY or ISAC, different forms of boosting specialized for k-NN approaches (García-Pedrajas & Ortiz-Boyer, 2009) or clustering (Frossyniotis et al., 2004) might be more promising and should be investigated in future work.

6.6 Stacking

Figure 14 shows the average *nPAR10* performance of stacking variants, where the meta-learner h_{agg} is instantiated through different algorithm selectors with and without a variance threshold feature selection approach. Each variant uses all base algorithm selectors to

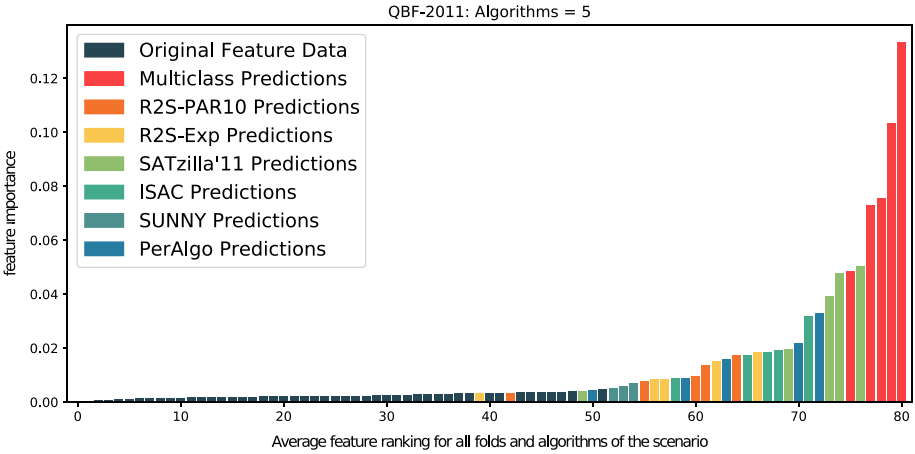

Fig. 15 This figure portrays a ranking over the features w.r.t. their feature importance values extracted from the multi-class classification meta-learner (instantiated with a one-vs-all decomposition equipped with a random forest classifier) for the QBF-2011 scenario

generate additional features. The variance threshold method selects all features with a variance larger than a given threshold, which was set to 0.16 for these experiments. The dotted line indicates the average performance of the SBAS.

Firstly, we would like to note that no general recommendation on the use of feature selection can be made, as the effect seems to depend very much on the meta-learner. However, while all stacking ensemble variants do not improve over the best single algorithm selector, the variants deploying SATzilla'11 and Multiclass as a meta-learner can slightly improve in performance. We find this quite disappointing, because the additional features provided to the meta-learner seem to carry valuable information. This is confirmed by the feature importance analysis portrayed in Fig. 15. It shows a ranking over the features w.r.t. their feature importance values extracted from the multi-class classification meta-learner (instantiated with a random forest classifier) for the QBF-2011 scenario. Clearly, the additional features in the form of the predictions of the ensemble members carry the biggest part of the information contained in the data.

6.6.1 Overall comparison

Table 3 displays *nPAR10* values of a subset of all evaluated ensemble variants and all base algorithm selectors broken down to the different scenarios. The best result for each scenario is marked in bold, and a line above a result of an ensemble approach indicates that it is better than the result of the best base algorithm selector on the corresponding scenario.

Overall, ensembles of algorithm selectors achieve a performance superior to single algorithm selectors. There are only two scenarios (ASP-POTASSCO, MAXSAT-WPMS-2016) for which none of the selected ensemble variants was able to improve over the base algorithm selector, performing best on that particular scenario, and another three scenarios where a competitive performance was achieved (MAXSAT15-PMS-INDU, SAT11-HAND, SAT12-HAND). For all other scenarios, at least one of the ensemble variants achieved a new state-of-the-art performance. While some of these improvements are rather small (CSP-MNZ-2013, where an improvement from 0.11 to 0.10 is recorded), there

Table 3 *nPAR10* values of the best ensemble variants and all base algorithm selectors broken down to the different scenarios. The best result for each scenario is marked in bold and a line above a result indicates beating all base algorithm selectors

Ensemble	Voting		Bagging		Stacking		Boosting								
Aggregation	wmaj	borda	wmaj	borda	R2S-Exp	SATzilla'11 (VT)	wmaj	wmaj							
Base selector	All	All	SUNNY	PerAlgo	All	All	Multiclass	PerAlgo	R2S-Exp	R2S-PAR10	ISAC	Multiclass	PerAlgo	SATzilla'11	SUNNY
Scenario															
ASP-POTASSCO	0.26	0.24	0.21	0.29	0.31	0.31	0.44	0.44	0.3	0.34	0.64	0.67	0.34	0.45	**0.17**
BNSL-2016	0.16	0.17	0.25	**0.15**	0.16	0.18	0.32	0.3	0.2	0.22	0.84	0.31	0.2	0.18	0.25
CPMP-2015	0.81	0.87	0.83	0.82	0.88	0.76	0.51	**0.47**	0.97	0.81	0.98	0.94	0.9	0.81	1.05
CSP-2010	0.24	0.24	0.33	**0.23**	**0.23**	0.24	0.43	0.42	0.26	0.26	0.38	0.78	0.36	0.24	0.4
CSP-MZN-2013	**0.1**	0.11	0.12	**0.1**	0.14	0.2	0.39	0.36	0.11	0.11	0.34	0.31	0.13	0.22	0.13
CSP-Minizinc-Time-2016	**0.21**	0.31	0.51	0.51	0.46	0.4	0.39	0.35	0.46	0.46	0.7	0.61	0.61	0.41	0.52
GLUHACK-18	0.44	0.44	0.47	0.49	0.45	0.43	0.4	**0.36**	0.47	0.5	0.6	0.39	0.44	0.41	0.52
MAXSAT-PMS-2016	0.55	0.58	0.39	0.47	0.76	0.7	0.42	**0.38**	0.57	0.41	1.05	1.18	0.79	0.6	0.49
MAXSAT-WPMS-2016	0.34	0.28	0.26	0.33	0.43	0.45	0.41	0.38	0.46	0.38	0.76	0.84	0.49	0.37	**0.24**
MAXSAT12-PMS	0.27	0.27	**0.17**	0.21	0.28	0.33	0.38	0.36	0.27	0.29	0.55	0.37	0.33	0.24	0.28
MAXSAT15-PMS-INDU	0.36	**0.24**	0.31	0.4	0.34	0.3	0.37	0.36	0.39	0.46	1.0	1.24	0.58	0.43	**0.24**
PRO-TEUS-2014	0.42	0.42	0.43	0.39	0.41	0.58	0.41	**0.36**	0.41	0.41	0.62	0.84	0.45	0.58	0.47
QBF-2011	0.18	0.17	0.16	**0.1**	0.16	0.15	0.39	0.34	0.19	0.19	0.33	0.33	0.2	0.16	0.22
QBF-2014	0.28	0.26	0.36	**0.25**	0.28	0.36	0.4	0.32	0.3	0.31	0.51	0.63	0.31	0.36	0.4

Table 3 (continued)

Ensemble	Voting		Bagging		Stacking		Boosting								
Aggregation	wmaj	borda	wmaj	borda	R2S-Exp	SATzilla'11 (VT)	wmaj	wmaj							
Base selector	All	All	SUNNY	PerAlgo	All	All	Multiclass	PerAlgo	R2S-Exp	R2S-PAR10	ISAC	Multiclass	PerAlgo	SATzilla'11	SUNNY
Scenario															
QBF-2016	0.42	0.41	0.44	0.42	0.56	0.63	0.41	**0.33**	0.47	0.49	0.59	0.68	0.65	0.62	0.51
SAT03-16_INDU	0.73	0.71	0.7	0.75	0.66	0.81	0.44	**0.36**	0.72	0.76	0.94	0.99	0.89	0.84	0.85
SAT11-HAND	0.37	**0.36**	0.45	0.45	0.46	0.44	0.43	**0.36**	0.51	**0.36**	0.69	0.57	0.48	0.49	0.7
SAT11-INDU	0.66	0.65	0.97	0.66	0.71	0.69	0.45	**0.38**	0.66	0.74	0.98	0.76	0.62	0.83	0.85
SAT11-RAND	0.1	0.1	0.1	**0.07**	0.11	0.29	0.44	0.37	0.14	0.08	0.61	0.17	0.11	0.36	0.13
SAT12-ALL	0.3	0.3	0.36	**0.29**	0.36	0.38	0.43	0.36	0.37	0.35	0.67	0.37	0.37	0.44	0.4
SAT12-HAND	0.28	**0.27**	0.34	0.3	0.29	0.28	0.43	0.35	0.35	0.34	0.64	0.41	0.38	**0.27**	0.42
SAT12-INDU	0.61	0.58	0.71	0.73	0.73	0.61	0.45	**0.35**	0.73	0.75	0.97	0.94	0.81	0.61	0.81
SAT12-RAND	0.87	0.89	0.91	1.06	0.87	0.9	0.48	**0.37**	1.0	0.9	1.01	5.32	1.18	0.99	1.11
SAT15-INDU	0.65	0.7	0.79	0.85	0.9	0.68	0.5	**0.39**	1.01	0.79	0.72	0.86	0.72	0.72	1.04
SAT18-EXP	0.47	0.52	0.57	**0.38**	0.52	0.59	0.5	0.4	0.6	0.67	0.61	0.65	0.62	0.6	0.63
Mean	0.4	0.4	0.45	0.43	0.46	0.47	0.42	**0.37**	0.48	0.46	0.71	0.85	0.52	0.49	0.51
Median	0.36	**0.31**	0.39	0.39	0.43	0.43	0.43	0.36	0.46	0.41	0.67	0.67	0.48	0.44	0.47
Avg. Rank	4.32	**4.2**	6.92	5.28	6.96	7.56	7.72	5.8	8.2	7.72	13.48	12.88	10.16	8.44	10.32

are also various scenarios with a > 1.5 fold improvement (e.g., CSP-Minizinc-Time-2016, SAT03_16_INDU, QBF-2011). This is especially remarkable as only very few improvements have been made in the last two years.

In terms of median, and average rank performance across all scenarios, the Borda voting ensemble variant achieves the best result and improves over the previous state of the art by more than 32% (median performance). Thus, it demonstrates a very robust performance across all scenarios. The voting ensemble with a Borda aggregation (13), the bagging ensemble with the PerAlgo base selector and a Borda aggregation (11), and the boosting ensemble with the PerAlgo base selector and a weighted majority aggregation (13) all consistently outperform the best single algorithm selector on 11 to 13 of 25 scenarios and, thus, achieve an impressive performance.

6.7 Discussion of results

We end the experimental evaluation by discussing both the scope of the presented results and the hardness of meta learning.

6.7.1 Scope of results

As the composition of the ASlib benchmark and existing literature show, most of the algorithm selection research is centered around constraint satisfaction problems, where the measure to optimize is algorithm runtime or a penalized version thereof such as the *PAR10*. This has several reasons: First, constraint satisfaction problems play a very important role in industry while, despite the large amount of research committed to these kinds of problems over the last century, they remain hard to solve in general. Second, the phenomenon of performance complementarity among algorithms, which is the main motivation for the AS problem as discussed earlier, is very present for these problems. More precisely, algorithms for solving constraint satisfaction problems are known to exhibit heavy-tailed runtime distributions (Gomes et al., 1997), i.e., they need very long to solve some instances while other algorithms might solve the same much faster. Overall, the potential for algorithm selection is very large on these kinds of problems, while sometimes lower for other problems such as selecting machine learning algorithms for a dataset. For that particular example, both random forests and gradient boosting often show strong performance and, thus, constitute strong SBS, which in principle can be improved upon as for example shown in Thornton et al. (2013), but often to a smaller degree.

Correspondingly, as common in the AS literature, the results presented so far focus on scenarios optimizing algorithm runtime. However, in order to at least give an idea about the applicability of the proposed framework for other algorithmic problem classes, we would also like to present results on two other scenarios from ASlib, which focus on optimizing solution quality instead of algorithm runtime. In particular, we present results on the OPENML-WEKA-2017 and the TTP-2016 scenarios. While the former is concerned with the selection of machine learning algorithms for different datasets, the latter deals with selecting algorithms for instances of the traveling thief problem (Bonyadi et al., 2013). As the Run2Survive models are specifically tailored towards AS wrt. algorithm runtime instead of performance, we leave them out of the comparison here.

Table 4 shows the results for the ensemble methods and the base algorithm selectors including both the SBS and the oracle as reference points. These reference points are included as this table does not show *nPAR10* scores, but a performance score in the unit

Table 4 Performance values (OPENML-WEKA-2017: accuracy, TTP-2016: TTP objective function Wagner et al., 2018) of the best ensemble variants and all base algorithm selectors broken down to the respective scenarios. The best result for each scenario is marked in bold and a line above a result indicates beating all base algorithm selectors

Ensemble	Voting		Bagging		Stacking		Boosting								
Aggregation	wmaj	borda	wmaj	borda	R2S-Exp	SATzilla'11 (VT)	wmaj	wmaj							
Base selector	All	All	SUNNY	PerAlgo	All	All	Multiclass	PerAlgo	PerAlgo	SUNNY	ISAC	SATzilla'11	Multiclass	sbs	Oracle
Scenario															
OPENML-WEKA-2017	0.85	0.85	0.85	0.85	0.85	0.85	0.85	**0.86**	0.85	0.84	0.85	**0.86**	0.85	**0.86**	0.88
TTP-2016	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.96	1.0

interval where 1 is the optimum since the scenarios are concerned with solution quality optimization as noted earlier. While the base algorithm selectors are able to achieve a slight improvement over the SBS on the TTP-2016 scenario, none of them can beat the SBS on the OPENML-WEKA-2017 scenario. Similarly, none of the ensemble approaches is able to improve over a base selector on the two scenarios. Hence, the empirical results corroborate the essence of the discussion above: On both scenarios the SBS is a strong baseline, which is quite close to the oracle in terms of performance and hence, there exists hardly any potential for algorithm selection in general let alone the meta level.[8]

Overall, algorithm selection on the meta level is only sensible in cases where (a) a considerable gap between the performances of standard algorithm selection approaches and the oracle exists and (b) performance complementarity among the algorithm selectors can be exploited. In contrast to the scenarios concerned with runtime, at least the first condition is not met for the two additional scenarios here, making an application not worthwhile for these cases.

6.7.2 Is meta learning harder than learning?

Recall our taxonomy of the approaches presented in Fig. 8, regarding which kind of mapping they model, how this mapping is constructed, and how the required aggregation function is obtained. Drawing an overall conclusion from the results presented in this work, we cautiously conclude that the presumably simpler problem of learning a mapping (8) from the instances to the set of algorithm selectors yields worse results than solving the presumably more complicated problem of finding both a mapping from instances to a set of selectors and a corresponding aggregation function. While we observed remarkable performance improvements for all ensemble approaches, the meta learning approach could essentially achieve no improvement. Although ensembles are known to often yield better results than single approaches and thus an improvement is to be expected, we believe that the degree of improvement in a well-researched field such as algorithm selection is truly remarkable. Moreover, it is surprising that the meta learning essentially fails and hence, classic AS approaches cannot exploit performance complementarity on the meta level.

As a possible reason, note that the meta learning approach heavily relies on the instance features, which are required for learning on the meta level. On the contrary, ensembles of algorithm selectors do not use these features on the meta level directly (except for stacking), but only aggregate the predictions of multiple selectors. Thus, we speculate that the information contained in the features does not allow for an improvement in performance through moving to the meta level, while the predictions of the selectors do carry enough information to do so. This hypothesis is corroborated by the feature analysis conducted as part of the experiments around stacking (cf. Fig. 15), which indicate that much more information is present in the predictions of the base selectors than in the original instance features. We attribute stacking's ability to perform successful learning on the meta level (aggregation) to the same reason. While stacking was able to achieve improvements, the arguably most simple ensemble approach in the form of voting, which involves no learning on the meta level at all, achieved by far the best results. Overall, learning on the meta level appears to be a very hard problem.

[8] As the ensemble approaches performed much better than the direct meta learning approach in the runtime experiments, we focused on the ensemble variants here.

7 Related work

In the following, we give an overview of the most related work regarding the use of ensemble methods in algorithm selection. As mentioned earlier, this work is surprisingly sparse. For a general overview of work on algorithm selection, we refer to Kerschke et al. (2019).

We presented a preliminary version of the meta AS problem in a preprint (Tornede et al., 2020b), which aimed at constructing a more effective algorithm selector by leveraging multiple existing selectors. The idea presented there is identical to the idea presented here in Sect. 4. In this work, we define the problem in a more general fashion, present a framework for solving this problem and show several instantiations of this framework. Accordingly, the work presented in the preprint is subsumed by this work.

In algorithm selection, it is normally assumed that the set of algorithms $\mathcal{A}$ to choose from is predefined, although the composition of this set can have an influence on the selectors. Therefore, Kordík et al. (2018) propose to not simply use all available algorithms as a basis to choose from, but to employ ensemble techniques in order to construct algorithms constituting this set. Thus, Kordík et al. (2018) build ensembles on the level of algorithms, whereas we ensemble on the level of selectors with the goal to create a better combined algorithm selector.

Last but not least, and perhaps indeed most related, both Malone et al. (2017) and Kotthoff (2012) suggest a stacking approach: First, a regression model is learned per algorithm to estimate the performance on a given instance, and second, the estimated performances are used as input for a multi-class classification model that eventually selects the algorithm. While Kotthoff (2012) only uses the outputs of the performance estimators as input of the meta-learner, Malone et al. (2017) use these in addition to the original features. Moreover, Malone et al. (2017) suggest to also include uncertainty information obtained from the performance estimators as input for the meta-learner. Both variants are very specific instantiations of the general idea presented in this paper, using stacking as an ensemble technique and a specific selector as a base algorithm selector. While the approach presented by Malone et al. (2017) resulted in the last spot in the open algorithm selection competition of 2017 (Lindauer et al., 2019), Kotthoff (2012) considered a setting, where the goal was to select the best machine learning algorithm for a dataset. He showed that stacking a classifier on top of the pure performance estimation does yield indeed an improvement in most cases over choosing the algorithm based on the performance estimates only.

8 Conclusion

In this paper, we revisited the problem of algorithm selection from a meta perspective. We defined the problem of meta algorithm selection and proposed a general methodological framework for this problem. Moreover, we considered several concrete learning methods as instantiations of this framework and compared them conceptually and empirically. In an extensive experimental study on an established benchmark for algorithm selection, we have shown that the meta algorithm selection problem can be solved efficiently, and that solutions can provide remarkable improvements in performance, often significantly better than the hitherto state of the art. Finally, we set the results into a broader context, concluding that learning algorithm selector selectors seems to be harder and less promising than defining them through well-established concepts from ensemble learning.

In future work, more effort should be invested in understanding why learning algorithm selector selectors appears to be a hard problem, while manually defined algorithm selection ensembles can achieve good performance. In particular, investigations of this phenomenon on a theoretical level would be of interest. Another possible direction for future work might be to focus more on learning instance-specific aggregation functions (Melnikov & Hüllermeier, 2016) to be used inside the ensembles, because this would allow one to leverage the information of which algorithm did indeed perform best on a given instance, instead of using an a priori fixed aggregation function. As seen with stacking, this works at least in principle. Yet another direction for future work is to adapt the idea of ensembles to the field of algorithm scheduling, where the recommendation target is no longer a single algorithm, but a complete algorithm schedule. One of the main challenges here is the aggregation of schedules.

Acknowledgements 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) and the German Federal Ministry of Education and Research (ITS.ML project no. 01IS18041D). The authors gratefully acknowledge support of this project through computing time provided by the Paderborn Center for Parallel Computing (PC^2).

Author Contributions AT motivated this work and took the lead in writing. All authors contributed to the content and the writing. The implementation and experiments were conducted by AT and LG.

Funding Open Access funding enabled and organized by Projekt DEAL. 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) and the German Federal Ministry of Education and Research (ITS.ML project no. 01IS18041D).

Availability of data and material All experiments are based on an existing public benchmark, which can be found at https://github.com/coseal/aslib_data.

Availability of code All code used for the experiments presented in this manuscript is available at https://github.com/alexandertornede/as_on_a_meta_level.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

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

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123–140.

Frossyniotis, D., Likas, A., & Stafylopatis, A. (2004). A clustering method based on boosting. *Pattern Recognition Letters, 25*(6), 641–654.
García-Pedrajas, N., & Ortiz-Boyer, D. (2009). Boosting k-nearest neighbor classifier by means of input space projection. *Expert Systems with Applications, 36*(7), 10570–10582.
Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*, 1157–1182.
Hastie, T., Rosset, S., Zhu, J., & Zou, H. (2009). Multi-class adaboost. *Statistics and its Interface, 2*(3), 349–360.
Hernández-Lobato, D., Martínez-Muñoz, G., & Suárez, A. (2009). Statistical instance-based pruning in ensembles of independent classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 31*(2), 364–369.
Kerschke, P., Hoos, H. H., Neumann, F., & Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. *Evolutionary Computation, 27*(1), 3–45.
Kordík, P., Cerný, J., & Frýda, T. (2018). Discovering predictive ensembles for transfer learning and meta-learning. *Machine Learning, 107*(1), 177–207.
Lindauer, M., van Rijn, J. N., & Kotthoff, L. (2019). The algorithm selection competitions 2015 and 2017. *Artificial Intelligence, 272*, 86–100.
Rokach, L. (2009). Collective-agreement-based pruning of ensembles. *Computational Statistics & Data Analysis, 53*(4), 1015–1026.
Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*(2), 197–227.
Wagner, M., Lindauer, M., Misir, M., Nallaperuma, S., & Hutter, F. (2018). A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics, 24*(3), 295–320. https://doi.org/10.1007/s10732-017-9328-y
Wever, M., Tornede, A., Mohr, F., & Hüllermeier, E. (2021). Automl for multi-label classification: Overview and empirical evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 43*(9), 3037–3054.
Wolpert, D. H. (1992). Stacked generalization. *Neural Networks, 5*(2), 241–259.
Amadini, R., Gabbrielli, M., & Mauro, J. (2014). SUNNY: A lazy portfolio approach for constraint solving. *Theory and Practice of Logic Programming, 14*(4–5).
Bonyadi, M. R., Michalewicz, Z., & Barone, L. (2013). The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20–23, 2013* (pp. 1037–1044). https://doi.org/10.1109/CEC.2013.6557681.
Borda, J. D. (1784). *Mémoire sur les élections au scrutin. Histoire de l'Academie Royale des Sciences pour 1781*.
Brazdil, P., Carrier, C. G., Soares, C., & Vilalta, R. (2008). *Metalearning: Applications to data mining*. Springer Science & Business Media.
Coppersmith, D., Fleischer, L., & Rudra A. (2006). Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *ACM-SIAM symposium on discrete algorithms (SODA)* (pp. 776–782).
Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Proceedings of multiple classifier systems, first international workshop, MCS 2000, Cagliari, Italy, June 21–23, 2000* (pp. 1–15).
Drucker, H. (1997). Improving regressors using boosting techniques. In *ICML* (Vol. 97, pp. 107–115). Citeseer
Dwork, C., Kumar, R., Naor, M., & Sivakumar, D. (2001). Rank aggregation methods for the web. In *Proceedings of the tenth international world wide web conference, WWW 10, Hong Kong, China, May 1–5, 2001* (pp. 613–622).
Gomes, C. P., Selman, B., & Crato, N. (1997). Heavy-tailed distributions in combinatorial search. In *Proceedings of Principles and practice of constraint programming—CP97, third international conference, Linz, Austria, October 29–November 1, 1997* (pp. 121–135).
Hanselle, J., Tornede, A., Wever, M., & Hüllermeier, E. (2020). Hybrid ranking and regression for algorithm selection. In *KI 2020: Advances in artificial intelligence*.
Hanselle, J., Tornede, A., Wever, M., & Hüllermeier, E. (2021). Algorithm selection as superset learning: Constructing algorithm selectors from imprecise performance data. *In The 25th Pacific-Asia conference on knowledge discovery and data mining (PAKDD-2021), May 11–14, 2021*.
Hüllermeier, E. (2014). Learning from imprecise and fuzzy observations: Data disambiguation through generalized loss minimization. *International Journal of Approximate Reasoning, 55*(7), 1519–1534 (special issue: Harnessing the information contained in low-quality data sources).
Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). ISAC—instance-specific algorithm configuration. In *ECAI*.

Kotthoff, L. (2012). Hybrid regression-classification models for algorithm selection. In *ECAI 2012—20th European conference on artificial intelligence*.

Lazarevic, A., & Obradovic, Z. (2001). Effective pruning of neural network classifier ensembles. In *Proceedings of IJCNN'01. International joint conference on neural networks* (Vol. 2, pp. 796–801). IEEE (Cat. No. 01CH37222).

Lobjois, L., & Lemaître, M. (1998). Branch and bound algorithm selection by performance prediction. In *AAAI/IAAI* (pp. 353–358).

Malone, B., Kangas, K., Järvisalo, M., Koivisto, M., & Myllymäki, P. (2017). as-asl: Algorithm selection with auto-sklearn. In *Open algorithm selection challenge 2017, PMLR* (pp. 19–22).

Melnikov, V., & Hüllermeier, E. (2016). Learning to aggregate using uninorms. In *Joint European conference on machine learning and knowledge discovery in databases* (pp. 756–771). Springer.

Pihera, J., & Musliu, N. (2014). 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, 2014* (pp. 47–54). IEEE Computer Society.

Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers* (Vol. 15, pp. 65–118). Elsevier.

Saari, D. G. (2000). The mathematics of voting: Democratic symmetry. *Economist, 83*.

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *The 19th ACM SIGKDD international conference on knowledge discovery and data mining, KDD 2013, Chicago, IL, USA, August 11–14, 2013* (pp. 847–855).

Tornede, A., Wever, M., & Hüllermeier, E. (2019). Algorithm selection as recommendation: From collaborative filtering to dyad ranking. In *CI Workshop*.

Tornede, A., Wever, M., & Hüllermeier, E. (2020a). Extreme algorithm selection with dyadic feature representation. In *Discovery science*.

Tornede, A., Wever, M., & Hüllermeier, E. (2020b). Towards meta-algorithm selection. In *Workshop on meta-learning (MetaLearn 2020) @ NeurIPS 2020*.

Tornede, A., Wever, M., Werner, S., Mohr, F., & Hüllermeier, E. (2020c). Run2survive: A decision-theoretic approach to algorithm selection based on survival analysis. In *ACML*.

Vanschoren, J. (2018). Meta-learning: A survey. CoRR arxiv:1810.03548.

Vilalta, R., Giraud-Carrier, C., & Brazdil, P. (2009). Meta-learning-concepts and techniques. In *Data mining and knowledge discovery handbook* (pp. 717–731). Springer.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *Evolutionary Computation, 1*(1).

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2007). Satzilla-07: the design and analysis of an algorithm portfolio for sat. In *CP*. Springer.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2011). Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop @ IJCAI*.