ASlib: A benchmark library for algorithm selection ☆

Bernd Bischl [a], Pascal Kerschke [b], Lars Kotthoff [d],*, Marius Lindauer [c],
Yuri Malitsky [g], Alexandre Fréchette [d], Holger Hoos [d], Frank Hutter [c],
Kevin Leyton-Brown [d], Kevin Tierney [e], Joaquin Vanschoren [f]

[a] *LMU Munich, Germany*
[b] *University of Münster, Germany*
[c] *University of Freiburg, Germany*
[d] *University of British Columbia, Vancouver, Canada*
[e] *University of Paderborn, Germany*
[f] *Eindhoven Institute of Technology, Netherlands*
[g] *IBM Research, United States*

A B S T R A C T

The task of algorithm selection involves choosing an algorithm from a set of algorithms on a per-instance basis in order to exploit the varying performance of algorithms over a set of instances. The algorithm selection problem is attracting increasing attention from researchers and practitioners in AI. Years of fruitful applications in a number of domains have resulted in a large amount of data, but the community lacks a standard format or repository for this data. This situation makes it difficult to share and compare different approaches effectively, as is done in other, more established fields. It also unnecessarily hinders new researchers who want to work in this area. To address this problem, we introduce a standardized format for representing algorithm selection scenarios and a repository that contains a growing number of data sets from the literature. Our format has been designed to be able to express a wide variety of different scenarios. To demonstrate the breadth and power of our platform, we describe a study that builds and evaluates algorithm selection models through a common interface. The results display the potential of algorithm selection to achieve significant performance improvements across a broad range of problems and algorithms.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Although NP-complete problems are widely believed to be intractable in the worst case, it is often possible to solve even very large instances of such problems that arise in practice. This is fortunate, because such problems are ubiquitous in Artificial Intelligence applications. There has thus emerged a large subfield of AI devoted to the advancement and analysis of heuristic algorithms for attacking hard computational problems. Indeed, quite surprisingly, this subfield has made consistent

and substantial progress over the past few decades, with the newest algorithms quickly solving benchmark problems that were beyond reach until recently. The results of the international SAT competitions provide a paradigmatic example of this phenomenon. Indeed, the importance of this competition series has gone far beyond documenting the progress achieved by the SAT community in solving difficult and application-relevant SAT instances—it has been instrumental in driving research itself, helping the community to coalesce around a shared set of benchmark instances and providing an impartial basis for determining which new ideas yield the biggest performance gains.

The central premise of events like the SAT competitions is that the research community ought to build, identify and reward single solvers that achieve strong across-the-board performance. However, this quest appears quixotic: most hard computational problems admit multiple solution approaches, none of which dominates all alternatives across multiple problem instances. In particular, this fact has been observed to hold across a wide variety of AI applications, including propositional satisfiability (SAT) [120], constraint satisfaction (CSP) [79], planning [42,45], and supervised machine learning [26,104,112]. An alternative is to accept that no single algorithm will offer the best performance on all instances, and instead aim to identify a portfolio of complementary algorithms and a strategy for choosing between them [85]. To see the appeal of this idea, consider the results of the sequential application (SAT+UNSAT) track of the 2014 SAT Competition.[1] The best of the 35 submitted solvers, Lingeling ayv [9], solved 77% of the 300 instances. However, if we could somehow choose the best among these 35 solvers on a per-instance basis, we would be able to solve 92% of the instances.

Research on this *algorithm selection problem* [85] has demonstrated the practical feasibility of using machine learning for this task. In fact, although practical algorithm selectors occasionally choose suboptimal algorithms, their performance can get close to that of an oracle that always makes the best choice. The area began to attract considerable attention when methods based on algorithm selection began to outperform standalone solvers in SAT competitions [118]. Algorithm selectors have since come to dominate the state of the art on many other problems, including CSP [79], planning [42], Max-SAT [71], QBF [83], and ASP [31].

To date, much of the progress in research on algorithm selection has been demonstrated in algorithm competitions originally intended for non-portfolio-based ("standalone") solvers. This has given rise to a variety of challenges for the field. First, benchmarks selected for such competitions tend to emphasize problem instances that are currently hard for existing standalone algorithms (to drive new research on solving strategies) rather than the wide range of both easy and hard instances that would be encountered in practice (which would be appropriately targeted by researchers developing algorithm selectors). Relatedly, benchmark sets change from year to year, making it difficult to assess the progress of algorithm selectors over time. Second, although competitions often require entrants to publish their source code, none require entries based on algorithm selectors to publish the code used to *construct* the algorithm selector (e.g., via training a machine learning model) or to adhere to a consistent input format. Third, overwhelming competition victories by algorithm selectors can make it more difficult for new standalone solver designs to get the attention they deserve and can thus create resentment among solver authors. Such concerns have led to a backlash against the participation of portfolio-based solvers in competitions; for example, starting in 2013 solvers that explicitly combine more than two component algorithms have been excluded from the SAT competitions. For similar reasons, there is a specific prize for non-portfolio solvers in the learning track of the International Planning Competition [107].

The natural solution to these challenges is to evaluate algorithm selectors on their own terms rather than trying to shoehorn them into competitions intended for standalone solvers. This article, written by a large set of authors active in research on algorithm selectors, aims to advance this goal by introducing a set of specifications and tools designed to standardize and facilitate such evaluations. Specifically, we propose a benchmark library, called ASlib, tailored to the cross-domain evaluation of algorithm selection techniques. In Section 3, we provide a summary of the data format specification used in ASlib that covers a wide variety of foreseeable evaluations. To date, we have instantiated this specification with benchmarks from six different problem domains, which we describe in Section 4. However, we intend for ASlib to grow and evolve over time. Thus, our article is accompanied by an online repository (http://aslib.net), which accepts submissions from any researcher. Indeed, we already included scenarios that have been submitted by contributors outside the core group of ASlib maintainers.

Our system automatically checks newly submitted datasets to verify that they adhere to the specifications and then provides an overview of the data, including the results of some straightforward algorithm selection approaches based on regression, clustering and classification. We provide some examples of these automatically-generated overviews and benchmark results in Sections 5 and 6. All code used to parse the format files, explore the algorithm selection scenarios and run benchmark machine learning models on them is publicly available in a new R package dubbed *aslib*.[2] In Section 7, we discuss two recent examples of competition settings using ASlib, along with their advantages and disadvantages.

Overall, our main objective in creating ASlib is the same as that of an algorithm competition: to allow researchers to compare their algorithms systematically and fairly, without having to replicate someone else's system or to personally collect raw data. We hope that it will help the community to obtain an unbiased understanding of the strengths and weaknesses of different methodologies and thus to improve the current state of the art in per-instance algorithm selection.

---

[1] http://www.satcompetition.org/2014/results.shtml.

[2] This package is currently hosted at https://github.com/coseal/aslib-r. We will submit it to the official R package server CRAN alongside the final version of this article.

2. Background

Rice [85] was the first to formalize the idea of selecting among different algorithms on a per-instance basis. While he referred to the problem simply as *algorithm selection*, we prefer the more precise term *per-instance algorithm selection*, to avoid confusion with the (simpler) task of selecting one of several given algorithms to optimize performance on a given set or distribution of instances.

Definition 1 *(Per-instance algorithm selection problem).* Given

- a set $\mathcal{I}$ of problem instances drawn from a distribution $\mathcal{D}$,
- a space of algorithms $\mathcal{A}$, and
- a performance measure $m : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$,

the *per-instance algorithm selection problem* is to find a mapping $s : \mathcal{I} \to \mathcal{A}$ that optimizes $\mathbb{E}_{i \sim \mathcal{D}} m(i, s(i))$, i.e., the expected performance measure for instances i distributed according to $\mathcal{D}$, achieved by running the selected algorithm $s(i)$ for instance i.

In practice, the mapping s is often implemented by using so-called instance features, i.e., characterizations of the instances $i \in \mathcal{I}$. These instance features are then mapped to an algorithm using machine learning techniques. However, the computation of instance features incurs additional costs, which have to be considered in the performance measure m.

There are many ways of tackling per-instance algorithm selection and related problems. Almost all contemporary approaches use machine learning to build predictors of the behavior of given algorithms as a function of instance features. This general strategy may involve a single learned model or a complex combination of several, which, given a new problem instance to solve, is used to decide which algorithm or which combination of algorithms to choose.

2.1. What to select and when

It is perhaps most natural to select a single algorithm for solving a given problem instance. This approach is, e.g., used in the SATzilla [77,118], ArgoSmArT [75], SALSA [22] and Eureka [19] systems. Its main disadvantage is that there is no way of mitigating a poor selection—the system cannot recover if the algorithm it chose for a problem instance exhibits poor performance.

Alternatively, we can seek a schedule that determines an ordering and time budget according to which we run all or a subset of the algorithms in the portfolio; usually, this schedule is chosen in a way that reflects the expected performance of the given algorithms (see, e.g., [44,45,56,79,83]). Under some of these approaches, the computation of the schedule is treated as an optimization problem that aims to maximize, e.g., the number of problem instances solved within a timeout. For stochastic algorithms, the further question of whether and when to restart an algorithm arises, opening the possibility of schedules that contain only a single algorithm, restarted several times (see, e.g., [18,28,37,99]). Instead of performing algorithm selection only once before starting to solve a problem, selection can also be carried out repeatedly while the instance is being solved, taking into account information revealed during the algorithm run. Such methods monitor the execution of the chosen algorithm(s) and take remedial action if performance deviates from what is expected [29,67,72], or perform selection repeatedly for subproblems of the given instance [5,64,65,90].

2.2. How to select

The kinds of decisions the selection process is asked to produce drive the choice of machine learning models that perform the selection. If only a single algorithm should be run, we can train a classification model that makes exactly that prediction. This renders algorithm selection conceptually quite simple—only a single machine learning model needs to be trained and run to determine which algorithm to choose (see, e.g., [33,39,73]).

There are alternatives to using a classification model to select a single algorithm to be run on a given instance, such as using regression models to predict the performance of each algorithm in the portfolio. This regression approach was adopted by several systems [74,77,87,92,118]. Other approaches include the use of clustering techniques to partition problem instances in feature space and make decisions for each partition separately [57,97], hierarchical models that make a series of decisions [46,116], cost-sensitive support vector machines [15] and cost-sensitive decision forests [119].

2.3. Selection enablers

In order to make their decisions, algorithm selection systems need information about the problem instance to solve and the performance of the algorithms in the given portfolio. The extraction of this information—the features used by the machine learning techniques used for selection—incurs overhead not required when only a single algorithm is used for all instances regardless of instance characteristics. It is therefore desirable to extract information as cheaply as possible, thus ensuring that the performance benefits of using algorithm selection are not outweighed by this overhead.

Some approaches use only past performance of the algorithms in the portfolio as a basis for selecting the one(s) to be run on a given problem instance [29,92,98]. This approach has the benefit that the required data can be collected with minimal overhead as algorithms are executed. It can work well if the performance of the algorithms is similar on broad ranges of problem instances. However, when this assumption is not satisfied (as is often the case), more informative features are needed.

Turning to richer instance-specific features, commonly used features include the number of variables of a problem instance and properties of the variable domains (e.g., the list of possible assignments in constraint problems, the number of clauses in SAT, the number of goals in planning). Deeper analysis can involve properties of graph representations derived from the input instance (such as the constraint graph [33,68]) or properties of encodings into different problems (such as SAT features for SAT-encoded planning problems [25]).

In addition, features can be extracted from short runs of one or more solvers on the given problem instance. Examples of such probing features include the number of search nodes explored within a certain time, the fraction of partial solutions that are disallowed by a certain constraint or clause, the average depth reached before backtracking is required, or characteristics of local minima found quickly using local search. Probing features are usually more expensive to compute than the features that can be obtained from shallow analysis of the instance specification, but they can also be more powerful and have thus been used by many authors (see, e.g., [54,78,79,82,118]). For continuous blackbox optimization, algorithm selection can be performed based on Exploratory Landscape Analysis [15,60,74]. The approach defines a set of numerical features (of different complexities and computational costs) to describe the landscapes of such optimization problems. Examples range from simple features that describe the distribution of sampled objective values to more expensive probing features based on local search.

Finally, in the area of meta-learning (learning about the performance of machine learning algorithms; for an overview, see, e.g, [17]), these features are known as *meta-features*. They include statistical and information-theoretical measures (e.g., variable entropy), landmarkers (measurements of the performance of fast algorithms [80]), sampling landmarkers (similar to probing features) and model-based meta-features [111]. These meta-features, and the past performance measurements of many machine learning algorithms, are available from the online machine learning platform OpenML [113]. In contrast to ASlib, however, OpenML is not designed to allow cross-domain evaluation of algorithm selection techniques.

2.4. Algorithm selection and algorithm configuration

A problem closely related to algorithm selection is the algorithm configuration problem: given a parameterized algorithm A, a set of problem instances I and a performance measure m, find a parameter setting of A that optimizes m on I (see [52] for a formal definition). While algorithm selection operates on finite (usually small) sets of algorithms, algorithm configuration operates on the combinatorial space of an algorithm's parameter settings. General algorithm configuration methods, such as ParamILS [52], GGA [4], I/F-Race [11], and SMAC [50], have yielded substantial performance improvements (sometimes orders of magnitude speedups) of state-of-the-art algorithms for several benchmarks, including SAT-based formal verification [47], mixed integer programming [49], AI planning [88,109], the combined selection and hyperparameter optimization of machine learning algorithms [104], and joint architecture and hyperparameter search in deep learning [23]. Algorithm configuration and selection are complementary since configuration can identify algorithms with peak performance for homogeneous benchmarks and selection can then choose from among these specialized algorithms. Consequently, several possibilities exist for combining algorithm configuration and selection [3,27,48,57,71,89,117,119]. The algorithm configuration counterpart of ASlib is AClib [53] (http://aclib.net). In contrast to ASlib, it is infeasible in AClib to store performance data for all possible parameter configurations, which often number more than 10^{50}. Therefore, an experiment on AClib includes new (expensive) runs of the target algorithms with different configurations and hence, experiments on AClib are a lot more costly than experiments on ASlib, where no new algorithm runs are necessary.[3] Furthermore, in contrast to AClib, ASlib does not include the actual instances and binaries of the algorithms. Therefore, ASlib does not provide a way to generate new performance data, as is required in AClib as a consequence of the need to assess the performance of new target algorithm configurations arising within the configuration process. However, ASlib and AClib can be combined by generating actual performance data based on the resources in AClib and then creating an ASlib scenario which selects between different solver configurations on a per-instance basis.

A full coverage of the wide literature on algorithm selection is beyond the scope of this article, but we refer the interested reader to recent survey articles on the topic [63,91,93,108].

3. Summary of format specification

We propose a data format specification for algorithm selection scenarios, i.e., instances of the per-instance algorithm selection problem. This format and the resulting data repository allow a fair and convenient scientific evaluation and comparison of algorithm selectors.

---

[3] In algorithm configuration, this need for expensive runs indeed causes a problem for research. One way of mitigating it is offered by fast-to-evaluate surrogate algorithm configuration benchmarks [24].
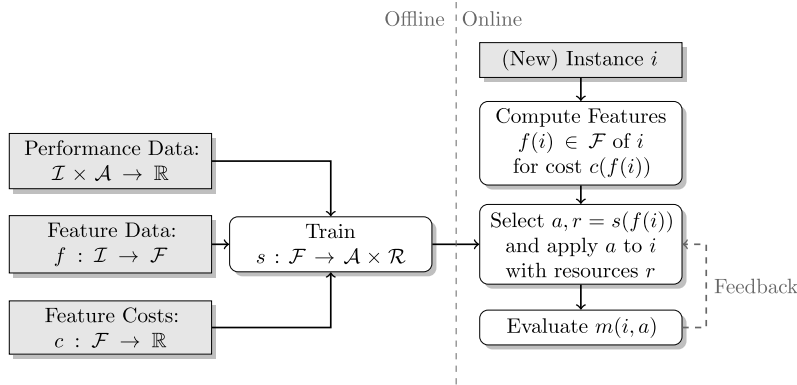
Fig. 1. Algorithm selection workflow.

The format specification assumes a generic approach to algorithm selection, depicted in Fig. 1. The general approach is as follows.

1. A vector of instance features $f(i) \in \mathcal{F}$ of i is computed. Feature computation may occur in several stages, each of which produces a group of (one or more) features. Furthermore, later stages may depend on the results of earlier ones. Each feature group incurs a cost, e.g., runtime. If no features are required, the cost is 0 (this occurs, e.g., for variants of algorithm selection that compute static schedules).
2. A machine learning technique s selects an algorithm $a \in \mathcal{A}$ based on the feature vector from Step 1.
3. The selected algorithm a is applied to i.
4. Performance measure m is evaluated, taking into account feature computation costs and the performance of the selected algorithm.
5. Some algorithm selectors do not select a single algorithm, but compute a schedule of several algorithms: they apply a to i for a resource budget $r \in \mathcal{R}$ (e.g., CPU time), evaluate the performance metric, evaluate a stopping criterion, and repeat as necessary, taking observations made during the run of a into account.[4]

The purpose of our library is to provide all information necessary for performing algorithm selection experiments using the given scenario data. The user does not need to actually run algorithms on instances, as all performance data is already precomputed. This drastically reduces the time required for executing studies, i.e., the runtime of studies is now dominated by the time required for learning s and not by applying algorithms to instances (e.g., solving SAT problems). It also means that results are perfectly reproducible; for example, the runtimes of algorithms do not depend on the hardware used; rather, they can be simply looked up in the performance data for a scenario.

Table 1 shows the basic structure of a scenario definition in ASlib; the complete specification with all details can be found in an accompanying technical report [12] and on our online platform.

4. Algorithm selection scenarios provided in ASlib release 2.0

The set of algorithm selection scenarios in release version 2.0 of our library, shown in Table 2, has been assembled to represent a diverse set of selection problem settings that covers a wide range of problem domains, types of algorithms, features and problem instances. Our scenarios include both problems that have been broadly studied in the context of algorithm selection techniques (such as SAT and CSP), as well as more recent ones (such as the container pre-marshalling problem). Most of our scenarios were taken from publications that report performance improvements through algorithm selection and consist of algorithms where the virtual best solver (VBS)[5] is significantly better than the single best solver.[6] Therefore, these are problems on which it makes sense to seek performance improvements via algorithm selection. All scenarios are available on our online platform (http://www.aslib.net/).

We now briefly describe the scenarios we included and what makes them interesting.

4.1. SAT: propositional satisfiability

The propositional satisfiability problem (SAT) is a classic NP-complete problem that consists of determining the existence of an assignment of values to variables of a Boolean formula such that the formula is true. It is widely studied, with many

---

[4] In principle, the workflow can be arbitrarily more complex, e.g., alternating between computing further features and running selected algorithms.

[5] The VBS is defined as a solver that perfectly selects the best solver from $\mathcal{A}$ on a per-instance basis.

[6] The single best solver has the best performance averaged across all instances.

Table 1
Overview of a scenario in the ASlib format.

Mandatory Data

- The *meta information file* is a global description file containing general information about the scenario, including the name of the scenario, performance measures, algorithms, features and limitations of computational resources.
- The *algorithm performance* file contains performance measurements with possible repetitions and completion status of the algorithm runs. The performance metric can be arbitrary, e.g., runtime, solution quality, accuracy or loss.
- The *instance feature* file contains the feature vectors for all instances. Another file contains technical information about errors encountered or instances solved during feature computation.
- The *cross-validation* file describes how to split the instance set into training and test sets to apply a standard machine learning approach to obtain an unbiased estimate of the performance of an algorithm selector.
- A human-readable *README* file explains the origin and meaning of the scenario, as well as the process of data generation.

Optional Data

- The *feature costs* file contains the costs of the feature groups, i.e., sets of features computed together.
- The *ground truth* file specifies information on the instances and their respective solutions (e.g., SAT or UNSAT).
- The *literature references* file in BibTeX format includes information on the context in which the data set was generated and previous studies in which it was used.

Table 2
Overview of algorithm selection scenarios in the ASLib with the number of instances $\#\mathcal{I}$, number of algorithms $\#\mathcal{A}$, number of features $\#\mathcal{F}$, number of feature processing groups $\#\mathcal{F}_g$ and availability of feature costs.

Scenario	$\#\mathcal{I}$	$\#\mathcal{A}$	$\#\mathcal{F}$	$\#\mathcal{F}_g$	Costs	Literature
SAT11-HAND	296	15	115	10	✓	[118]
SAT11-INDU	300	18	115	10	✓	[118]
SAT11-RAND	600	9	115	10	✓	[118]
SAT12-ALL	1614	31	115	10	✓	[121]
SAT12-HAND	767	31	115	10	✓	[121]
SAT12-INDU	1167	31	115	10	✓	[121]
SAT12-RAND	1362	31	115	10	✓	[121]
SAT15-INDU	300	28	54	1	×	–
QBF-2011	1368	5	46	1	×	[83]
QBF-2014	1254	14	46	1	×	–
MAXSAT12-PMS	876	6	37	1	✓	[71]
MAXSAT15-PMS-INDU	601	29	37	1	×	–
CSP-2010	2024	2	17	1	×	[33]
CSP-MZN-2013	4642	11	155	2	✓	[2]
PROTEUS-2014	4021	22	198	4	✓	[46]
ASP-POTASSCO	1294	11	138	5	✓	[43]
PREMAR-ASTAR-2015	527	4	22	3	×	[105]

applications including formal verification [81], scheduling [20], planning [59] and graph coloring [110]. Our SAT data mainly stems from different iterations of the SAT competition,[7] which is split into three tracks: industrial (INDU), crafted (HAND), and random (RAND).

The SAT scenarios are characterized by a high level of maturity and diversity in terms of their solvers, features and instances. Each SAT scenario involves a highly diverse set of solvers, many of which have been developed for several years. In addition, the set of SAT features is probably the best-studied feature set among our scenarios; it includes both static and probing features that are organized into as many as ten different feature groups. The instance sets used in our various SAT scenarios range from randomly-generated ones to real-world instances submitted by industry.

4.2. QBF: quantified Boolean formula

A quantified Boolean formula (QBF) is a formula in propositional logic with universal or existential quantifiers on each variable in the formula. A QBF solver finds a set of variable assignments that makes the formula true or proves that no such set can exist. This is a PSPACE-complete problem for which solvers exhibit a wide range of performance characteristics. Our QBF-2011 data set comes from the QBF Solver Evaluation 2010[8] and consists of instances from the main, small hard, 2QBF and random tracks. Our QBF-2014 data set comes from the application track of the QBF Gallery 2014.[9] The instance features were computed using the AQME system and are described in more detail by Pulina et al. [83]. The solvers for QBF-2011

---

come from the AQME system as well, whereas the solvers for QBF-2014 are the ones submitted to the application track of the QBF Gallery.

4.3. MAXSAT: maximum satisfiability

MaxSAT is the optimization version of the previously introduced SAT problem, and aims to find a variable assignment that maximizes the number of satisfied clauses. The MaxSAT problem representation can be used to effectively encode a number of real-world problems, such as FPGA routing [115], and software package installation [6], among others, as it permits reasoning about both optimality and feasibility. The particular scenarios focus on the partial MaxSAT (PMS) problem [10].

The MAXSAT12-PMS scenario is composed of a collection of random, crafted and industrial instances from the 2012 MaxSAT Evaluation [7]. The techniques used to solve the various instances in this scenario are very complementary to each other, leading to a substantial performance gap between the single best and the virtual best solver. Furthermore, because there are only six solvers with very different performance characteristics, algorithm selection approaches must be very accurate in their choices, as any mistake is heavily penalized.

The more recent MAXSAT15-PMS-INDU was built on the performance data of the industrial track on partial MAXSAT problems from the 2015 MAXSAT Evaluation.[10] With 29 algorithms, it provides a larger set of solvers than MAXSAT12-PMS. However, there are different parameterizations of the same solvers, e.g., four different variants of *ahms*, such that there are some subsets of strongly correlated solvers. The performance gap between the single best and virtual best solver is larger in MAXSAT12-PMS than in MAXSAT15-PMS-INDU.

4.4. CSP: constraint solving

The Constraint Satisfaction Problem (CSP; [100]) is concerned with finding solutions to constraint satisfaction problems—a task that is NP-complete. Learning in the context of constraint solving is a technique by which previously unknown constraints that are implied by the problem specification are uncovered during search and subsequently used to speed up the solving process.

The scenario CSP-2010 contains only two solvers: one that employs lazy learning [33,35] and one that does not [34]. The data set is heavily biased towards the non-learning solver, such that the baseline (the single best solver) is very good already. Improving on this is a challenging task and harder than in many of the other scenarios. Furthermore, both solvers share a common core, which results in a scenario that directly evaluates the efficacy of a specific technique in different contexts.

The more recent scenario CSP-MZN-2013 provides a larger set of instances, algorithms and instance features. Instances and algorithms come from the MiniZinc challenge 2012 and the International Constraint Solver Competitions (ICSC) in 2009. Specifically, the instances come from the MiniZinc 1.6 benchmark suite and the algorithms in the scenario participated in the MiniZinc Challenge 2012. Algorithms, instances and instance features are described in more detail in [1,2].

Our final CSP scenario PROTEUS comes from [46] and includes an extremely diverse mix of well-known CSP solvers alongside competition-winning SAT solvers that have to solve (converted) XCSP instances.[11] The SAT solvers can accept different conversions of the CSP problem into SAT (see, e.g., [66,101,102]), which in our format are provided as separate algorithms. This scenario is the only one in which solvers are tested with varying "views" of the same problem. The features of this scenario are also unique in that they include both the SAT and CSP features for a given instance. This potentially provides additional information to the selection approach that would normally not be available for solving CSPs. An algorithm selection system has a very high degree of flexibility here and may choose to perform only part of the possible conversions, thereby reducing the set of solvers and features, but also reducing the overhead of performing the conversions and feature computations. There are also synergies between feature computation and algorithm runs that can be exploited, e.g., if the same conversion is used for feature computation and to run the chosen algorithm then the cost of performing the conversion is only incurred once. In other cases, where features are computed on one representation and another one is solved, conversion costs are incurred both during feature computation and the running of the algorithm.

4.5. ASP: answer set programming

Answer Set Programming (ASP, [8,30]) is a form of declarative programming with roots in knowledge representation, non-monotonic reasoning and constraint solving. In contrast to many other constraint solving domains (e.g., the satisfiability problem), ASP provides a rich yet simple declarative modeling language in which problems up to Δ_3^p (disjunctive optimization problems) can be expressed. ASP has proven to be efficiently applicable to many real-world applications, e.g., product configuration [95], decision support for NASA shuttle controllers [76], synthesis of multiprocessor systems [55] and industrial team building [38].

---

[10] http://www.maxsat.udl.cat/15/results/index.html.
[11] The XCSP instances are taken from http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html as described in [46].

In contrast to the other scenarios, the algorithms in the scenario ASP-POTASSCO were automatically constructed by an adapted version of *Hydra* [117], i.e., the set of algorithms consists of complementary configurations of the solver *clasp* [32]. The instance features were generated by a light-weight version of *clasp*, including static and probing features organized into feature groups; they were previously used in the algorithm selector *claspfolio* [31,43].

4.6. PREMAR-ASTAR-2015: container pre-marshalling

The container pre-marshalling problem (CPMP) is an NP-hard container stacking problem from the container terminals literature [96]. We constructed an algorithm selection scenario from two recent A* and IDA* approaches for solving the CPMP presented in [106], using instances from the literature. The scenario is described in detail in [105].

The pre-marshalling scenario differs from other scenarios in that the set of algorithms is highly homogeneous. All of the algorithms are parameterizations of a single symmetry breaking heuristic, either using the A* or IDA* search techniques, which stands in sharp contrast to the diversity of solvers present in most other scenarios. The scenario represents a real-world, time-sensitive problem from the operations research literature, where algorithm selection techniques can have a large impact.

5. Automated exploratory data analysis

The online platform for our benchmark repository offers not only the scenario data files themselves. It also provides many tables and figures that summarize them. These pages are automatically generated and currently consist (among others) of the following parts:

- an overview table that lists, for example, the number of instances, algorithms and features for all available scenarios, similar to Table 2;
- a summary of the algorithms' performance and run status data;
- a summary of the feature values, as well as the run status and costs of the feature groups;
- benchmark results for standard machine learning models for each scenario; see Section 6.

Presenting this additional data offers the following advantages:

- Researchers can quickly understand which scenarios are available and select those best suited to their needs.
- Data can quickly be sanity-checked. It is common that data collection errors occur when scenario data is gathered and submitted for the first time.
- Interesting or challenging properties of the data sets become visible, providing the researcher with a quick and informative first impression.

The platform's **summary page for the algorithms** starts with a table listing summary statistics regarding their performance (e.g., mean values and standard variations) and run status (e.g., how many runs were successful). We also indicate whether one algorithm is dominated by another, i.e., an algorithm a_1 dominates another algorithm a_2 if and only if a_1 has performance at least equal to that of a_2 on all instances, and a_1 outperforms a_2 on at least one instance. This is useful, because there is no reason to include a dominated algorithm in a portfolio. Various visualizations, such as box plots, scatter plot matrices, correlation plots and density plots enable further inspection of the performance distribution and correlation between algorithms, allowing the reader to better understand the strengths and weaknesses of each algorithm. All of our plots can be configured to use log scales, which often improves visual understanding of heavy-tailed distributions (e.g., runtime distributions of hard combinatorial solvers [36]).

Fig. 2 shows boxplots and cumulative distribution functions for the algorithms in the QBF-2011 scenario as an example. The boxplots summarize the runtimes of an algorithm by drawing a box between the 25%- and 75%-quantile of the sample, i.e., the smallest values that are greater or equal to 25% and 75% of the runtimes. In addition, each box contains a line showing the median runtime, as well as so-called whiskers, i.e., lines that connect the box with runtimes that are within 150% of the interquartile range (the length of the box) below the 25%- or above the 75%-quantile, respectively. Observations with even more extreme runtimes are considered to be outliers and are depicted by a single point per outlier. The cumulative distribution function plots on the other hand show runtimes on all instances for the algorithm. Each point within the plot consists of the observed runtime on the *x*-axis and the corresponding cumulative density, i.e., the percentage of instances that were solved at this or a smaller runtime, on the *y*-axis.

Such plots show the location of the mean, distribution spread, density multimodality and other properties of the distribution. In addition, they reveal how long it took an algorithm to solve the given instances. For example, for the QBF-2011 scenario in Fig. 2, one can see that the algorithm *quantor* finds a solution very quickly on a few instances, i.e., it solves approximately 5% of the instances in less than a second. However, if it does not succeed quickly, it often does not succeed at all—it solved less than 30% of all the instances. In contrast, *sSolve* usually needs longer to find a solution, but by the time it does, it is one of the best algorithms. Such behavior can indicate that the algorithm requires a 'warm-up' stage, which should be considered when deploying it.
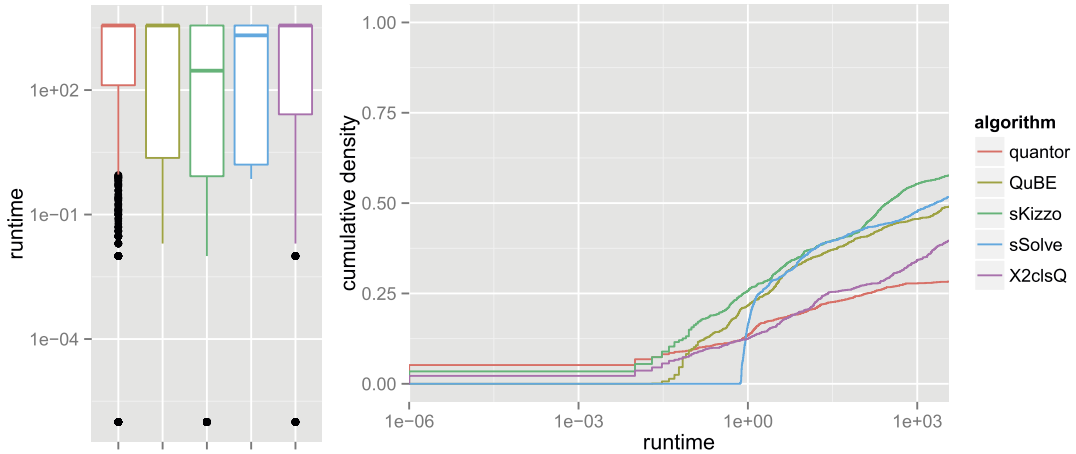
Fig. 2. Algorithm performance distributions of the QBF-2011 scenario: Boxplots (left) and cumulative distribution functions (right); both on a log scale.
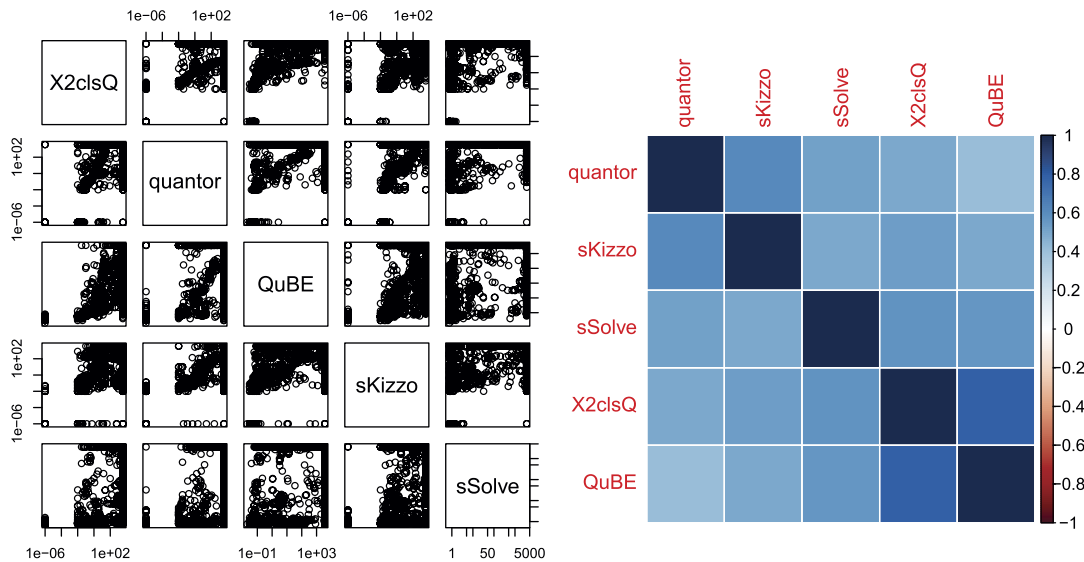

Fig. 3. Pairwise correlations among algorithms of the QBF-2011 scenario: Scatter plot matrix on a log scale (left) and correlation matrix (right). (The reader is referred to the web version of this article for the color version of this figure.)

The left panel of Fig. 3 shows pairwise scatterplots of the QBF-2011 scenario, allowing an easy comparison of algorithm pairs on all instances from a given scenario. Each point represents a problem instance within the scenario, and from the location of the point cloud one can see whether an algorithm is dominant over the majority of instances, or whether relative performance strongly varies across instances. The first case can be identified by a cloud that is located either in the upper-left or lower-right corner of a single scatterplot. In such a case, the dominated algorithm could be discarded from the portfolio. However, if this type of dominance relationship is not present, there is the potential to realize performance improvements by means of per-instance algorithm selection.

Because detecting correlation in algorithm performance is also of interest when analyzing the strengths and weaknesses of a given portfolio-based solver [120], we also present a correlation matrix, cf. Fig. 3 (right panel). Algorithms that have a (high) positive correlation are more likely to be redundant in a portfolio, whereas pairs with a (high) negative correlation are more likely to complement each other. We calculate Spearman's correlation coefficient between ranks. Blue boxes represent positive correlation, red boxes represent negative correlation, and shading indicates the strength of correlation. The algorithms are also clustered according to these values (using Ward's method [114]) and then sorted, such that similar algorithms appear together in blocks. This type of clustering allows the identification of algorithms with highly correlated performance.

Fig. 4 shows the correlation between algorithms for the SAT12-ALL scenario. The plot reveals four groups of algorithms (*minisatpsm* to *restartsat*, *sattimep* to *tnm*, *marchrw* and the three *mphaseSAT*-algorithms) with high correlations within each
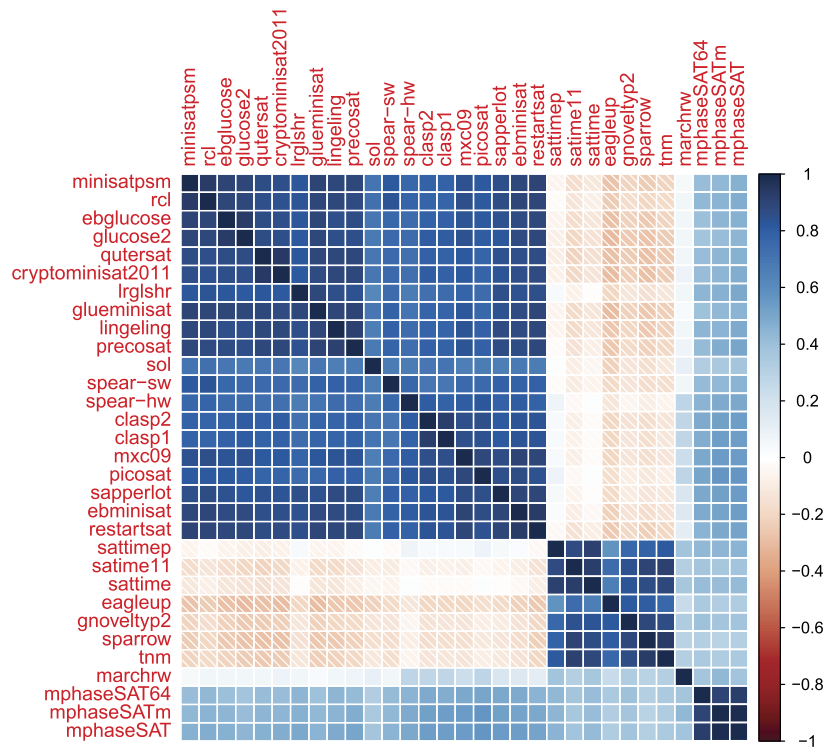
Fig. 4. Algorithm correlations on the SAT12-ALL scenario. (The reader is referred to the web version of this article for the color version of this figure.)

Table 3
Feature group summary for the SAT12-RAND scenario. The second column shows how many features depend on another feature group to be computed first. Percentages of runstatus events are followed by summary statistics for group costs.

Feature group	#	Runstatus [%]			Cost [s]			
		ok	...	crash	min	mean	max	missing [%]
Pre	115	100.00	...	0.00	0.00	0.06	1.31	0.00
Basic	14	100.00	...	0.00	0.00	0.00	0.07	0.00
KLB	20	100.00	...	0.00	0.00	0.18	6.09	0.00
CG	10	62.63	...	37.37	0.02	8.79	20.28	0.00
DIAMETER	5	100.00	...	0.00	0.00	0.60	2.11	0.00
cl	18	100.00	...	0.00	0.01	1.99	2.02	0.00
sp	18	100.00	...	0.00	0.01	0.33	3.05	0.00
ls_saps	11	100.00	...	0.00	1.36	2.12	2.51	0.00
ls_gsat	11	100.00	...	0.00	2.03	2.29	3.03	0.00
lobjois	2	100.00	...	0.00	2.00	2.00	2.27	0.00

group. It may be desirable to include just a single representative from each group, reducing the size of the entire portfolio from 31 to four algorithms.

As we do with algorithm runs, we characterize the features by giving summary statistics of the feature values, the run status and the cost of the feature groups. Table 3 shows the summary of the feature groups for the SAT12-RAND scenario. In this scenario, all 115 features have the feature group "Pre" as a requirement. While this preprocessing group succeeded in all cases, one other group did not: the feature group "CG" (which computes clause graph features) failed in 37.37% of cases due to exceeding time or memory limits, and even for instances where it succeeded, it was quite expensive (8.79 seconds on average). Such information is useful to understand the behavior of the features: how risky is it to compute a feature group, and how much time must one invest in order to obtain the corresponding features?

We also check whether instances occur with exactly the same feature values, indicating that the experimenter might have erroneously run on the same instance twice.

All of the above tables and figures and many more were generated by our online platform, and are also accessible through the R package *aslib*. The functions are highly configurable and customizable. We plan to extend our data analysis with additional techniques, such as more measures of algorithm performance [94].

Table 4
Machine learning algorithms and their parameter ranges used for our study.

Technical name	Algorithm and parameter ranges	Reference
Classification		
ksvm	support vector machine	[58]
	$C \in [2^{-12}, 2^{12}]$, $\gamma \in [2^{-12}, 2^{12}]$	
randomForest	random forest	[69]
	`ntree` $\in [10, 200]$, `mtry` $\in [1, 30]$	
rpart	recursive partitioning tree, CART	[103]
Regression		
lm	linear regression	[84]
randomForest	random forest	[69]
	`ntree` $\in [10, 200]$, `mtry` $\in [1, 30]$	
rpart	recursive partitioning tree, CART	[103]
Clustering		
XMeans	extended *k*-means clustering	[40]

6. Study of algorithm selection techniques

In this section, we present an exploratory benchmark study that gives an indication of the diversity of our benchmarks. First, we evaluate the performance of algorithm selectors on our scenarios. We then perform a subset selection study to identify the important algorithms and instance features in each of the scenarios. We make no claim that the presented experimental settings are exhaustive or that we achieve state-of-the-art algorithm selection performance; rather, we provide results that can be used as a baseline comparison for other approaches. These results, and our framework in general, allow us to study which algorithm selection approaches work well for which of our scenarios.

We use the *LLAMA* toolkit [62], version 0.9.1, in combination with the *aslib* package[12] to run the algorithm selection study. *LLAMA* is an R [84] package that facilitates many common algorithm selection approaches. In particular, it enables access to classification, regression, and clustering models for algorithm selection—the three main approaches we use in our study. We use the *mlr* R package [14] as an interface to the machine learning models provided by other R packages. We parallelize all of our benchmark experiments through the *BatchExperiments* [13] R package.

In this paper, we only present aggregated benchmark results, but the interested reader can access full benchmark results at http://aslib.net. Our study is fully reproducible as the complete code to generate these results is part of the *aslib* package.

We use the subset of feature groups that are recommended by the authors of the respective scenario, called default feature set. For the feature subset selection study, we have used all feature groups. Detailed and continuously updated information (e.g., the names of the feature processing groups we selected and their average costs) is provided on the ASlib website.

6.1. Experimental setup

We consider three approaches to algorithm selection that have been studied extensively in the literature (cf. Section 2.2):

- *classification* applies a multi-class classifier to directly predict the best performing algorithm of the *k* possible algorithms;
- *regression* predicts each algorithm's performance via a regression model and then chooses the one with the best predicted performance;
- *clustering* groups problem instances in feature space, then determines the cost-optimal solver for each cluster and finally assigns to each new instance the solver associated with the instance's predicted cluster.

The specific machine learning algorithms we employed for our study are shown in Table 4. They include representatives of each of the three major approaches above.

The linear model we employ is the best-studied regression method. In its most basic version, it models the data using the linear function $f(x) = \beta^T x + \beta_0$; parameters are obtained by minimizing squared loss. The trees constructed by the CART algorithm, which can handle both classification and regression problem, are grown in a top-down manner and divide the training data into rectangular regions by axis-parallel splits at each interior node. Splits are selected by considering label impurity reduction measured by an impurity function, based, e.g., on the Gini index for classification or squared loss for regression. Leaf nodes associate the best, but constant, label with their feature region for prediction. Random forests form an ensemble of *ntree* simpler trees by bootstrapping multiple data sets from the original one and then fitting a tree for each. Predictions are made through majority voting for classification and averaging for regression. Furthermore, ensemble

---

members are decorrelated by randomly selecting only a few candidate features for each split point (controlled by parameter *mtry*) in a tree and maximally growing trees without any early stopping or pruning. Support Vector Machines perform linear classification in a transformed feature space by maximizing the margin between the positive and negative examples. Parameter C controls the trade-off between the size of the margin and classification loss. The feature mapping is implicitly built into the algorithm by substituting the regular inner product of the Euclidean space with a so-called kernel. Parameter γ is a property of the radial basis function kernel used here. The XMeans clustering algorithm is the only unsupervised learning algorithm we study. It is an extension of the well known k-means method to adaptively select the number of clusters. k-means starts with k random cluster centroids, assigns each point to the nearest centroid, and then iteratively recomputes the cluster centroids and cluster assignments until convergence. For further details on all methods the reader is referred to the standard literature [41] and for XMeans to [21].

We tuned the hyperparameters of ksvm and randomForest (classification and regression) within the listed parameter ranges, using random search with 250 iterations and a nested cross validation (with three internal folds) to ensure unbiased performance results. All other parameters were left at their default values. For the clustering algorithm, we set the (maximum) number of clusters to 30 after some preliminary experiments; the exact number of clusters is determined internally by XMeans.

6.2. Data preprocessing

We removed constant-valued (and therefore irrelevant) features and imputed missing feature values as the mean over all non-missing values of the feature.[13] For the clustering methods, we normalized the range of each feature to the interval $[-1, 1]$. The scenarios we consider in this article contain only continuous features. The other machine learning methods that require normalized data perform this internally (for example the SVMs). Missing performance values were imputed using the timeout value of the scenario.

For each problem instance, we calculated the total feature computation cost as the sum of the costs of all feature groups, in the order specified in the definition of the scenario. If the problem instance was solved during feature computation (e.g., using SLS-probing features [118]), we only considered the cost of the feature groups up to the one that solved it. Furthermore, we set the runtime for all algorithms to zero for instances solved during feature computation. If the instance was not solved during feature computation, we added the feature costs computed in this way to the runtimes of the individual algorithms on the respective instances $(c(f_i) + t(i, a))$. Given these new runtimes, we checked whether the specified timeout was now exceeded and set the run status of any corresponding algorithm accordingly. Preprocessing runtimes to include feature computation time in this way allows us to focus on an algorithm selection system's overall performance, and avoids overstating the fraction of instances that would be solved within a time budget in cases where features are expensive to compute.

Each scenario specifies a partition into 10 folds for cross-validation to ensure consistent evaluation across different methods. We used this partition in our study.

6.3. Evaluation

We evaluated different algorithm selection models using three different measures:

- the fraction of all instances solved within the timeout;
- the penalized average runtime with a penalty factor of 10, i.e., a timeout counts as 10 times the timeout;
- the average misclassification penalty, which, for a given instance, is the difference between the performance of the selected algorithm and the performance of the best algorithm.

The performance of each algorithm selection model was compared to the virtual best solver (VBS) and the single best solver. The virtual best solver selects the best solver from $\mathcal{A}$ for each instance $(\forall i \in \mathcal{I} : \arg\max_{a \in \mathcal{A}} m(i, a))$. Note that the misclassification penalty for VBS is zero by definition. The single best solver is the actual solver that has the overall best performance on the data set $(\arg\max_{a \in \mathcal{A}} \sum_{i \in \mathcal{I}} m(i, a))$. Specifically, we consider the solver with the best PAR10 score over all problem instances in a scenario.

6.4. Results

Fig. 5 presents a summary of the results of our study. In most cases, the algorithm selection approaches performed better than the single best solver. We expected this, as most of our data sets come from publications that advocate algorithm selection systems.

Nevertheless, there were significant differences between the scenarios. While almost all algorithm selection approaches outperformed the single best algorithm, there are some scenarios that seem to be much harder for algorithm selection. In particular, on the SAT12-INDU scenario, three approaches were not able to achieve a performance improvement.

---

[13] For the CSP-MZN-2013 scenario, we also removed the `gc_circuit` feature, which is almost constant.

	classif/ ksvm (0.48)	classif/ randomForest (0.52)	classif/ rpart (0.42)	regr/ lm (0.48)	regr/ randomForest (0.65)	regr/ rpart (0.34)	cluster/ XMeans (0.22)
SAT11−HAND	0.72*	0.62	0.59	0.48	0.65	0.47	0.12
SAT11−INDU	0.10	0.15*	0.11	0.13	0.01	−0.29	0.01
SAT11−RAND	0.76	0.85	0.86	0.83	0.90*	0.82	0.54
SAT12−ALL	0.63	0.70	0.06	0.43	0.72*	0.32	−0.01
SAT12−HAND	0.60	0.70	0.37	0.51	0.75*	0.45	0.08
SAT12−INDU	0.37	0.44	−0.18	0.24	0.47*	−0.01	−0.16
SAT12−RAND	−0.00	0.10	0.34*	0.28	0.25	−0.50	0.10
SAT15−INDU	0.25	0.22	0.21	0.09	0.49*	0.02	−0.10
QBF−2011	0.67	0.79	0.72	0.71	0.88*	0.70	0.43
QBF−2014	0.46	0.53	0.38	0.35	0.79*	0.52	0.07
MAXSAT12−PMS	0.62	0.65	0.62	0.63	0.90*	0.62	0.60
MAXSAT15−PMS−INDU	0.31	0.37	0.27	0.46	0.64*	0.19	0.45
CSP−2010	0.55	0.67	0.83*	0.66	0.77	0.29	0.23
CSP−MZN−2013	0.82	0.79	0.68	0.83	0.91*	0.77	0.16
PROTEUS−2014	0.73	0.73	0.47	0.77	0.81*	0.71	0.54
ASP−POTASSCO	0.39	0.43	0.47	0.55	0.80*	0.53	0.40
PREMAR−ASTAR−2015	0.23	0.16	0.23	0.16	0.29*	0.16	0.22

Fig. 5. Summary of the results of our study. We show how much of the gap between the single best and the virtual best solver in terms of PAR10 score was closed by each model. That is, a value of 0 corresponds to the single best solver and a value of 1 to the virtual best. Negative values indicate performance worse than the single best solver. Within each data set, the best model is annotated with an asterisk. The shading emphasizes that comparison: green cells correspond to values close to 1 (i.e., close to the virtual best solver), whereas red shows the models with bad performance. White shading indicates values close to 0, i.e. the model has the same performance as the single best algorithm. The arithmetic mean for each model type across all scenarios is given in parentheses after the model name. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Random regression forests stood out quite clearly as the best overall approach, achieving the best performance on 13 of the 17 datasets. This is in line with recent results showing the strong performance of this model for algorithm runtime prediction [54]. The results are also consistent with those of the original papers introducing the datasets.

XMeans performed worst on average. On some scenarios, it performed well, in particular SAT11-RAND, MAXSAT12-PMS, and PROTEUS-2014. However, on SAT12-ALL, SAT12-INDU, and SAT15-INDU XMeans performed worse than the single best solver. The default subset of instance features appears to be unfavorable for XMeans on industrial SAT instances.

6.5. Algorithm and feature subset selection

To provide further insight into our algorithm selection scenarios, we applied forward selection [61] to the algorithms and features to determine whether smaller subsets still achieve comparable performance. We performed forward search independently for algorithms and features for each scenario. Forward selection is an iterative selection algorithm whose the first iteration starts with an empty set of algorithms and features; in each subsequent iteration, it greedily adds the algorithm or feature to the set which most improves the cross-validated score (PAR10) of the predictor. The selection process terminates when the score does not improve by at least ε. We set $\varepsilon = 1$, which roughly corresponds to an improvement of 1 second per instance. In all other aspects, the experimental setup was the same as described before.

We used random regression forests,[14] as it was the best overall approach so far. We note that the selection results use standard cross validation rather than the nested version, which may result in overconfident performance estimates for the selected subsets [16]. We accept this caveat since our goal here is to study the ranking of the features and the size of the selected sets, and a more complex, nested approach would have resulted in multiple selected sets.

Tables 5 and 6 present the results of forward selection for algorithms and features on all scenarios. Usually, the number of selected features is very small compared to the complete feature set. This is consistent with the observations of Roberts and Howe [86] and Hutter et al. [51] who found in their experiments that only a few instance features are necessary to reliably predict the runtime of their algorithms. For example, on SAT12-RAND, the only three features selected were a balance feature concerning the ratio of positive and negative occurrences of each variable in each clause and two features based on survey propagation.

---

[14] We used random forests with default parameters, as the tuning was done for the set of features specified by the scenario authors and the full set of solvers.

Table 5
PAR10 scores for the set of all algorithms and the reduced set, along with the number of all algorithms and the size of the reduced portfolio.

Scenario	PAR10 full set	Number	PAR10 reduced set
SAT11-HAND	16943.49	15 → 8	15919.09
SAT11-INDU	13246.70	18 → 4	12127.05
SAT11-RAND	10253.09	9 → 4	10180.39
SAT12-ALL	971.45	31 → 11	979.17
SAT12-HAND	4303.81	31 → 10	4187.13
SAT12-INDU	2763.37	31 → 7	2775.61
SAT12-RAND	3241.42	31 → 2	3153.48
SAT15-INDU	3845.52	28 → 6	3604.57
QBF-2011	9232.49	5 → 4	9198.01
QBF-2014	2090.02	14 → 8	2040.33
MAXSAT12-PMS	3370.22	6 → 3	3299.15
MAXSAT15-PMS-INDU	1752.57	29 → 7	1479.31
CSP-2010	6541.20	2 → 2	6516.57
CSP-MZN-2013	4204.58	11 → 9	4168.35
PROTEUS-2014	5905.71	22 → 8	5725.50
ASP-POTASSCO	525.55	11 → 5	509.61
PREMAR-ASTAR-2015	5154.40	4 → 3	4954.45

Table 6
PAR10 scores for the set of all features, the default feature set and the reduced set, along with the number of all features and the size of the reduced feature set.

Scenario	Full set	Default set	Number	Reduced set
SAT11-HAND	17249.59	16943.49	113 → 6	15743.04
SAT11-INDU	13111.61	13246.70	112 → 4	10951.00
SAT11-RAND	10496.39	10253.09	112 → 3	9854.11
SAT12-ALL	994.25	971.45	113 → 9	815.37
SAT12-HAND	4298.00	4303.81	113 → 6	4092.58
SAT12-INDU	2881.97	2763.37	113 → 6	2506.25
SAT12-RAND	3196.28	3241.42	113 → 3	3088.72
SAT15-INDU	3970.56	3845.52	51 → 3	3620.06
QBF-2011	9229.99	9232.49	46 → 5	8995.62
QBF-2014	2102.79	2090.02	46 → 4	2032.50
MAXSAT12-PMS	3321.22	3370.22	30 → 4	3296.52
MAXSAT15-PMS-INDU	1696.69	1752.57	29 → 5	1520.77
CSP-2010	6514.37	6541.20	69 → 3	6415.23
CSP-MZN-2013	4192.82	4204.58	115 → 5	4119.36
PROTEUS-2014	6120.13	5905.71	193 → 6	5700.05
ASP-POTASSCO	516.47	525.55	134 → 4	472.84
PREMAR-ASTAR-2015	5289.96	5154.40	22 → 3	4619.49

The number of algorithms after forward selection is also substantially reduced on most scenarios. On the SAT scenarios, we expected to see this because the scenarios consider a huge set of SAT solvers that were not pre-selected in any way. Xu et al. [120] showed that many SAT solvers are strongly correlated and make only very small contributions to the VBS, a finding that is corroborated by our results (see Fig. 4 in Section 5). For example, on the SAT12-RAND scenario, only two solvers were selected: sparrow and eagleup. We did not expect the set of algorithms to be reduced on the ASP-POTASSCO scenario, as the portfolio was automatically constructed using algorithm configuration to obtain a set of complementary parameter settings that are particularly amenable to portfolios; nevertheless only 5 of 11 configurations were chosen by the forward selection.

Our results indicate that in real-world settings, selecting the most predictive features and the solvers that make the highest contributions can be important. More detailed and continuously updated results can be found on the ASlib website.

7. Competitions on ASlib

As described before and illustrated in Section 6, we designed ASlib to enable easy and fair comparison of different algorithm selection approaches. The next step to get unbiased performance comparisons of algorithm selectors is to organize competitions based on ASlib. In this section, we will briefly describe two exemplary competition settings based on ASlib.

On-going evaluation on ASlib. In the on-going evaluation on ASlib,[15] every participant can simply submit his/her performance for each scenario (using the provided cross validation splits) and the source code of their algorithm selector. The latter will

---

[15] The most recent results of the on-going evaluation can be found on the ASlib homepage aslib.net.

only be used to verify the results in case of doubt. The results (i.e., (penalized) average performance on each scenario) will be added in an overview table and the system will be linked.

In this setting, every system that can read the ASlib format can easily participate and no deadlines for submission are required. Therefore, the newest systems and results can always be added on-the-fly such that the on-going evaluation always reflects the most recent known state-of-the-art systems and their performances. Disadvantages of this setting are:

1. the different participants use different amounts of computational resources to compute the results—for example, two well-performing systems in the on-going evaluation are *SATzilla* [118] and *AutoFolio* [70] but it is also well-known that these two systems use a lot more computation resources (several CPU days) than other systems;
2. since the test and training data are published, the system will tend to overfit the scenarios if we will not regularly provide new scenarios to reveal such overfitting.

ICON challenge on algorithm selection. The ICON Challenge on Algorithm Selection[16] provided a comparative evaluation of state-of-the-art algorithm selection systems. The winner of the challenge was the zilla system [121]. In this competition, the algorithm selectors needed to be submitted before a fixed deadline and each system was executed on the organizers' hardware with some limitations (e.g., at most 12 CPU hours for training on one scenario). Although the used scenarios were also already published before (i.e., ASlib version 1.0.1), the organizers did not reveal the training-test splits to avoid overly strong overfitting on these scenarios. Furthermore, the ICON challenge assessed the performance of the algorithm selectors based on 3 different performance metrics (i.e., average number of solved instances, PAR10, and misclassification penalty (MCP)) which revealed some strengths and weaknesses of algorithm selectors, e.g., systems that used an algorithm schedule had better performance on solved instances and PAR10, but wasted some time with respect to MCP.

8. Summary

We have introduced ASlib, a benchmark library for algorithm selection, a rapidly growing field of research with substantial impact on various sub-communities in artificial intelligence. ASlib facilitates research on algorithm selection methods by providing a common set of benchmarks and tools for working with these. Similar to solver competitions, it enables principled comparative empirical performance assessment. It also considerably lowers the otherwise rather high barrier for researchers to work on algorithm selection, since anyone using the benchmark scenarios we provide does not have to perform actual runs of the solvers contained in them. Since our library provides performance data for the solvers and problem instances included in each selection scenario, using ASlib also substantially reduces the computational burden of performance assessments. Otherwise, these data would have to be produced, at considerable computational cost, by anyone working with that scenario. We carefully selected the set of scenarios included in release version 2.0 of ASlib to challenge algorithm selection methods in various ways and thus provide a solid basis for developing and assessing such methods.

Release version 2.0 of the library contains 17 algorithm selection scenarios from six different areas with a focus on (but not a limitation to) constraint satisfaction problems. We discussed the format of new algorithm selection scenarios and showed examples of the automated exploratory data analysis that will run for each new scenario submitted to our online platform http://aslib.net/. Finally, exploratory studies with various algorithm selection approaches demonstrated the performance that algorithm selection systems can achieve on our scenarios.

References

[1] R. Amadini, M. Gabbrielli, J. Mauro, An enhanced features extractor for a portfolio of constraint solvers, in: Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea – March 24–28, 2014, pp. 1357–1359.
[2] R. Amadini, M. Gabbrielli, J. Mauro, SUNNY: a lazy portfolio approach for constraint solving, Theory Pract. Log. Program. 14 (2014) 509–524.
[3] C. Ansótegui, Y. Malitsky, M. Sellmann, MaxSAT by improved instance-specific algorithm configuration, in: Proceedings of the Twenty-Eighth National Conference on Artificial Intelligence, 2014, pp. 2594–2600.
[4] C. Ansótegui, M. Sellmann, K. Tierney, A gender-based genetic algorithm for the automatic configuration of algorithms, in: Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP'09), 2009, pp. 142–157.
[5] A. Arbelaez, Y. Hamadi, M. Sebag, Continuous search in constraint programming, in: Proceedings of the Twenty-Second IEEE International Conference on Tools with Artificial Intelligence, 2010, pp. 53–60.

---

[16] http://challenge.icon-fet.eu/.

[6] J. Argelich, D.L. Berre, I. Lynce, J. Marques-Silva, P. Rapicault, Solving Linux upgradeability problems using Boolean optimization, in: Proceedings of the International Workshop on Logics for Component Configuration, 2010, pp. 11–22.

[7] J. Argelich, C. Li, F. Manyà, J. Planes, Seventh MaxSAT evaluation, http://www.maxsat.udl.cat/12/, 2012.

[8] C. Baral, Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2003.

[9] A. Biere, Yet another local search solver and Lingeling and friends entering the SAT competition 2014, in: Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions, 2014, pp. 39–40.

[10] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009.

[11] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated F-race: an overview, in: Empirical Methods for the Analysis of Optimization Algorithms, Springer, 2010.

[12] B. Bischl, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, P. Kerschke, K. Leyton-Brown, J. Vanschoren, Algorithm selection format specification, Technical report, available at: http://www.aslib.net/, 2014.

[13] B. Bischl, M. Lang, O. Mersmann, J. Rahnenführer, C. Weihs, BatchJobs and BatchExperiments: abstraction mechanisms for using R in batch environments, J. Stat. Softw. 64 (2015) 1–25.

[14] B. Bischl, M. Lang, J. Richter, J. Bossek, L. Judt, T. Kuehn, E. Studerus, L. Kotthoff, Z. Jones, mlr: machine learning in R, R package version 2.7, https://github.com/mlr-org/mlr, 2015.

[15] B. Bischl, O. Mersmann, H. Trautmann, M. Preuss, Algorithm selection based on exploratory landscape analysis and cost-sensitive learning, in: Proceedings of the Fourteenth Annual Conference on Genetic and Evolutionary Computation, 2012, pp. 313–320.

[16] B. Bischl, O. Mersmann, H. Trautmann, C. Weihs, Resampling methods for meta-model validation with recommendations for evolutionary computation, Evol. Comput. 20 (2012) 249–275.

[17] P. Brazdil, C. Giraud-Carrier, C. Soares, R. Vilalta, Metalearning: Applications to Data Mining, 1st ed., Springer, 2008.

[18] V.A. Cicirello, S.F. Smith, The max k-armed bandit: a new model of exploration applied to search heuristic selection, in: Proceedings of the Twentieth National Conference on Artificial Intelligence, AAAI Press, 2005, pp. 1355–1361.

[19] D.J. Cook, R.C. Varnell, Maximizing the benefits of parallel search using machine learning, in: Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI Press, 1997, pp. 559–564.

[20] J.M. Crawford, A.B. Baker, Experimental results on the application of satisfiability algorithms to scheduling problems, in: Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994, pp. 1092–1097.

[21] A.M. Dan Pelleg, X-means: extending k-means with efficient estimation of the number of clusters, in: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 2000, pp. 727–734.

[22] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R.C. Whaley, K. Yelick, Self-adapting linear algebra algorithms and software, Proc. IEEE 93 (2005) 293–312.

[23] T. Domhan, J.T. Springenberg, F. Hutter, Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI), 2015.

[24] K. Eggensperger, F. Hutter, H.H. Hoos, K. Leyton-Brown, Efficient benchmarking of hyperparameter optimizers via surrogates, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.

[25] C. Fawcett, M. Vallati, F. Hutter, J. Hoffmann, H. Hoos, K. Leyton-Brown, Improved features for runtime prediction of domain-independent planners, in: Proceedings of the International Conference on Automated Planning and Scheduling, 2014.

[26] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: Advances in Neural Information Processing Systems, vol. 28, 2015, pp. 2944–2952.

[27] M. Feurer, J.T. Springenberg, F. Hutter, Initializing Bayesian hyperparameter optimization via meta-learning, in: B. Bonet, S. Koenig (Eds.), Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25–30, 2015, Austin, Texas, USA, AAAI Press, 2015, pp. 1128–1135.

[28] M. Gagliolo, J. Schmidhuber, Learning restart strategies, in: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI), 2007, pp. 792–797.

[29] M. Gagliolo, V. Zhumatiy, J. Schmidhuber, Adaptive online time allocation to search algorithms, in: Proceedings of European Conference on Machine Learning, Springer, 2004, pp. 134–143.

[30] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers, 2012.

[31] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M.T. Schneider, S. Ziller, A portfolio solver for answer set programming: preliminary report, in: Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2011, pp. 352–357.

[32] M. Gebser, B. Kaufmann, T. Schaub, Multi-threaded ASP solving with clasp, Theory Pract. Log. Program. 12 (2012) 525–545.

[33] I. Gent, C. Jefferson, L. Kotthoff, I. Miguel, N. Moore, P. Nightingale, K. Petrie, Learning when to use lazy learning in constraint solving, in: Proceedings of the Nineteenth European Conference on Artificial Intelligence, IOS Press, 2010, pp. 873–878.

[34] I.P. Gent, C.A. Jefferson, I. Miguel, MINION: a fast, scalable, constraint solver, in: Proceedings of the European Conference on Artificial Intelligence, 2006, pp. 98–102.

[35] I.P. Gent, I. Miguel, N.C.A. Moore, Lazy explanations for constraint propagators, in: Proceedings of the Twelfth International Symposium on Practical Aspects of Declarative Languages, 2010, pp. 217–233.

[36] C. Gomes, B. Selman, N. Crato, H. Kautz, Heavy-tailed phenomena in satisfiability and constraint satisfaction problems, J. Autom. Reason. 24 (2000) 67–100.

[37] C.P. Gomes, B. Selman, Algorithm portfolios, Artif. Intell. 126 (2001) 43–62.

[38] G. Grasso, S. Iiritano, N. Leone, V. Lio, F. Ricca, F. Scalise, An ASP-based system for team-building in the Gioia–Tauro seaport, in: Proceedings of the Twelfth International Symposium on Practical Aspects of Declarative Languages, 2010, pp. 40–42.

[39] A. Guerri, M. Milano, Learning techniques for automatic algorithm portfolio selection, in: Proceedings of the Sixteenth European Conference on Artificial Intelligence, IOS Press, 2004, pp. 475–479.

[40] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: an update, ACM SIGKDD Explor. Newsl. 11 (2009) 10–18.

[41] T.J. Hastie, R.J. Tibshirani, J.H. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer Series in Statistics, Springer, New York, 2009.

[42] M. Helmert, G. Röger, E. Karpas, Fast downward stone soup: a baseline for building planner portfolios, in: Proceedings of the Workshop on Planning and Learning at the Twenty-First International Conference on Automated Planning and Scheduling, 2011, pp. 28–35.

[43] H. Hoos, M. Lindauer, T. Schaub, Claspfolio 2: advances in algorithm selection for answer set programming, Theory Pract. Log. Program. (2014) 569–585.

[44] H.H. Hoos, R. Kaminski, M. Lindauer, T. Schaub, aspeed: solver scheduling via answer set programming, Theory Pract. Log. Program. (2014) 1–26.

[45] A.E. Howe, E. Dahlman, C. Hansen, M. Scheetz, A. von Mayrhauser, Exploiting competitive planner performance, in: Proceedings of the Fifth European Conference on Planning, Springer, 1999, pp. 62–72.

[46] B. Hurley, L. Kotthoff, Y. Malitsky, B. O'Sullivan, Proteus: a hierarchical portfolio of solvers and transformations, in: Proceedings of the Eleventh International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 2014, pp. 301–317.

[47] F. Hutter, D. Babić, H.H. Hoos, A.J. Hu, Boosting verification by automatic tuning of decision procedures, in: Formal Methods in Computer Aided Design, IEEE Computer Society, 2007, pp. 27–34.

[48] F. Hutter, Y. Hamadi, H.H. Hoos, K. Leyton-Brown, Performance prediction and automated tuning of randomized and parametric algorithms, in: Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming, 2006, pp. 213–228.

[49] F. Hutter, H.H. Hoos, K. Leyton-Brown, Automated configuration of mixed integer programming solvers, in: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 2010, pp. 186–202.

[50] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proceedings of the International Conference on Learning and Intelligent Optimization, 2011, pp. 507–523.

[51] F. Hutter, H.H. Hoos, K. Leyton-Brown, Identifying key algorithm parameters and instance features using forward selection, in: LION 7, 2013.

[52] F. Hutter, H.H. Hoos, K. Leyton-Brown, T. Stützle, ParamILS: an automatic algorithm configuration framework, J. Artif. Intell. Res. 36 (2009) 267–306.

[53] F. Hutter, M. López-Ibáñez, C. Fawcett, M. Lindauer, H. Hoos, K. Leyton-Brown, T. Stützle, Aclib: a benchmark library for algorithm configuration, in: Proceedings of the International Conference on Learning and Intelligent Optimization, 2014, pp. 36–40.

[54] F. Hutter, L. Xu, H.H. Hoos, K. Leyton-Brown, Algorithm runtime prediction: methods & evaluation, Artif. Intell. 206 (2014) 79–111.

[55] H. Ishebabi, P. Mahr, C. Bobda, M. Gebser, T. Schaub, Answer set vs. integer linear programming for automatic synthesis of multiprocessor systems from real-time parallel programs, Int. J. Reconfigurable Comput. (2009).

[56] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, M. Sellmann, Algorithm selection and scheduling, in: Proceedings of the International Conference on Principles and Practice of Constraint Programming, in: Lecture Notes in Computer Science, vol. 6876, Springer, 2011, pp. 454–469.

[57] S. Kadioglu, Y. Malitsky, M. Sellmann, K. Tierney, ISAC – instance-specific algorithm configuration, in: Proceedings of Nineteenth European Conference on Artificial Intelligence, IOS Press, 2010, pp. 751–756.

[58] A. Karatzoglou, A. Smola, K. Hornik, A. Zeileis, kernlab – an S4 package for kernel methods in R, J. Stat. Softw. 11 (2004) 1–20.

[59] H. Kautz, B. Selman, Unifying SAT-based and graph-based planning, in: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1999, pp. 318–325.

[60] P. Kerschke, M. Preuss, C. Hernández, O. Schütze, J.Q. Sun, C. Grimme, G. Rudolph, B. Bischl, H. Trautmann, Cell mapping techniques for exploratory landscape analysis, in: Proceedings of the EVOLVE 2014: A Bridge Between Probability, Set Oriented Numerics, and Evolutionary Computation, Springer, 2014, pp. 115–131.

[61] R. Kohavi, G.H. John, Wrappers for feature subset selection, Artif. Intell. 97 (1997) 273–324.

[62] L. Kotthoff, LLAMA: leveraging learning to automatically manage algorithms, Technical report, 2013, arXiv:1306.1031.

[63] L. Kotthoff, Algorithm selection for combinatorial search problems: a survey, AI Mag. 35 (2014) 48–60.

[64] M. Lagoudakis, M. Littman, Algorithm selection using reinforcement learning, in: Proceedings of the Seventeenth International Conference on Machine Learning, 2000, pp. 511–518.

[65] M. Lagoudakis, M. Littman, Learning to select branching rules in the DPLL procedure for satisfiability, in: Proceedings of the International Conference on Satisfiability, 2001, pp. 344–359.

[66] D. Le Berre, I. Lynce, CSP2SAT4J: a simple CSP to SAT translator, in: Proceedings of the Second International CSP Solver Competition, 2008, pp. 43–54.

[67] R. Leite, P. Brazdil, J. Vanschoren, Selecting classification algorithms with active testing, in: Machine Learning and Data Mining in Pattern Recognition, Springer, 2012, pp. 117–131.

[68] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, Y. Shoham, A portfolio approach to algorithm selection, in: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 2003, pp. 1542–1543.

[69] A. Liaw, M. Wiener, Classification and regression by randomForest, R News 2 (2002) 18–22.

[70] M. Lindauer, H. Hoos, F. Hutter, T. Schaub, Autofolio: an automatically configured algorithm selector, J. Artif. Intell. 53 (2015) 745–778.

[71] Y. Malitsky, D. Mehta, B. O'Sullivan, Evolving instance specific algorithm configuration, in: The Sixth Annual Symposium on Combinatorial Search, 2013.

[72] Y. Malitsky, B. O'Sullivan, A. Previti, J. Marques-Silva, A portfolio approach to enumerating minimal correction subsets for satisfiability problems, in: Proceedings of the Eleventh International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming, 2014.

[73] Y. Malitsky, A. Sabharwal, H. Samulowitz, M. Sellmann, Non-model-based algorithm portfolios for SAT, in: Proceedings of the Fourteenth International Conference on Theory and Applications of Satisfiability Testing, Springer, 2011, pp. 369–370.

[74] O. Mersmann, B. Bischl, H. Trautmann, M. Wagner, J. Bossek, F. Neumann, A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem, Ann. Math. Artif. Intell. (2013) 1–32.

[75] M. Nikolić, F. Marić, P. Janičić, Instance-based selection of policies for SAT solvers, in: Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing, Springer, 2009, pp. 326–340.

[76] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, M. Barry, An A-Prolog decision support system for the Space Shuttle, in: Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages, Springer, 2001, pp. 169–183.

[77] E. Nudelman, K. Leyton-Brown, G. Andrew, C. Gomes, J. McFadden, B. Selman, Y. Shoham, Satzilla 0.9, 2003.

[78] E. Nudelman, K. Leyton-Brown, H.H. Hoos, A. Devkar, Y. Shoham, Understanding random SAT: beyond the clauses-to-variables ratio, in: Principles and Practice of Constraint Programming – CP 2004, Springer, 2004, pp. 438–452.

[79] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, B. O'Sullivan, Using case-based reasoning in an algorithm portfolio for constraint solving, in: Proceedings of the Nineteenth Irish Conference on Artificial Intelligence and Cognitive Science, 2008.

[80] B. Pfahringer, H. Bensusan, C. Giraud-Carrier, Meta-learning by landmarking various learning algorithms, in: Proceedings of the Seventeenth International Conference on Machine Learning, 2000, pp. 743–750.

[81] M.R. Prasad, A. Biere, A. Gupta, A survey of recent advances in SAT-based formal verification, Int. J. Softw. Tools Technol. Transf. 7 (2005) 156–173.

[82] L. Pulina, A. Tacchella, A multi-engine solver for quantified Boolean formulas, in: Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming, Springer, 2007, pp. 574–589.

[83] L. Pulina, A. Tacchella, A self-adaptive multi-engine solver for quantified Boolean formulas, Constraints 14 (2009) 80–116.

[84] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2014, http://www.R-project.org/.

[85] J.R. Rice, The algorithm selection problem, Adv. Comput. 15 (1976) 65–118.

[86] M. Roberts, A. Howe, Learning from planner performance, Artif. Intell. J. 173 (2009) 536–561.

[87] M. Roberts, A.E. Howe, Learned models of performance for many planners, in: Proceedings of the Workshop on AI Planning and Learning at the Seventeenth International Conference on Automated Planning and Scheduling, 2007.

[88] M. Roberts, A.E. Howe, B. Wilson, M. desJardins, What makes planners predictable?, in: ICAPS, 2008, pp. 288–295.

[89] A. Sabharwal, H. Samulowitz, M. Sellmann, Y. Malitsky, Boosting sequential solver portfolios: knowledge sharing and accuracy prediction, in: LION 7, 2013.

[90] H. Samulowitz, R. Memisevic, Learning to solve QBF, in: Proceedings of the Twenty-Second National Conference on Artificial Intelligence, AAAI Press, 2007, pp. 255–260.
[91] F. Serban, J. Vanschoren, J.U. Kietz, A. Bernstein, A survey of intelligent assistants for data analysis, ACM Comput. Surv. 45 (2013) 1–35.
[92] B. Silverthorn, R. Miikkulainen, Latent class models for algorithm portfolio methods, in: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010, pp. 167–172.
[93] K.A. Smith-Miles, Cross-disciplinary perspectives on meta-learning for algorithm selection, ACM Comput. Surv. 41 (2008) 6:1–6:25.
[94] K.A. Smith-Miles, D. Baatar, B.J. Wreford, R. Lewis, Towards objective measures of algorithm performance across instance space, Comput. Oper. Res. 45 (2014) 12–24.
[95] T. Soininen, I. Niemelä, Developing a declarative rule language for applications in product configuration, in: Proceedings of the First International Workshop on Practical Aspects of Declarative Languages, Springer, 1999, pp. 305–319.
[96] R. Stahlbock, S. Voß, Operations research at container terminals: a literature update, OR Spektrum 30 (2008) 1–52.
[97] K. Stergiou, Heuristics for dynamically adapting propagation in constraint satisfaction problems, AI Commun. 22 (2009) 125–141.
[98] M.J. Streeter, D. Golovin, S.F. Smith, Combining multiple heuristics online, in: Proceedings of the Twenty-Second National Conference on Artificial Intelligence, AAAI Press, 2007, pp. 1197–1203.
[99] M.J. Streeter, D. Golovin, S.F. Smith, Restart schedules for ensembles of problem instances, in: Proceedings of the Twenty-Second National Conference on Artificial Intelligence, AAAI Press, 2007, pp. 1204–1210.
[100] P.J. Stuckey, T. Feydy, A. Schutt, G. Tack, J. Fischer, The MiniZinc challenge 2008–2013, AI Mag. 35 (2014) 55–60.
[101] N. Tamura, T. Tanjo, M. Banbara, System description of a SAT-based CSP solver sugar, in: Proceedings of the Third International CSP Solver Competition, 2008, pp. 71–75.
[102] T. Tanjo, N. Tamura, M. Banbara, Azucar: a SAT-based CSP solver using compact order encoding, in: Theory and Applications of Satisfiability Testing – SAT 2012, Springer, 2012, pp. 456–462.
[103] T. Therneau, B. Atkinson, B. Ripley, rpart: recursive partitioning and regression trees, http://CRAN.R-project.org/package=rpart, 2014, R package version 4.1-8.
[104] C. Thornton, F. Hutter, H. Hoos, K. Leyton-Brown, Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms, in: Proceedings of the Nineteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2013, pp. 847–855.
[105] K. Tierney, Y. Malitsky, An algorithm selection benchmark of the container pre-marshalling problem, in: C. Dhaenens, L. Jourdan, M.E. Marmion (Eds.), Learning and Intelligent Optimization, Springer International Publishing, 2015, pp. 17–22.
[106] K. Tierney, D. Pacino, S. Voß, Solving the pre-marshalling problem to optimality with A* and IDA*, Technical Report Working Paper #1401, Decision Support & Optimization Lab, University of Paderborn, 2014.
[107] M. Vallati, L. Chrpa, M. Grzes, T.L. McCluskey, M. Roberts, S. Sanner, The 2014 international planning competition: progress and trends, AI Mag. 36 (2015) 90–98.
[108] M. Vallati, L. Chrpa, D. Kitchin, Portfolio-based planning: state of the art, common practice and open challenges, AI Commun. 28 (2015) 717–733.
[109] M. Vallati, C. Fawcett, A. Gerevini, H.H. Hoos, A. Saetti, Automatic generation of efficient domain-optimized planners from generic parametrized planners, in: International Symposium on Combinatorial Search (SoCS), 2013.
[110] A. Van Gelder, Another look at graph coloring via propositional satisfiability, Discrete Appl. Math. 156 (2008) 230–243.
[111] J. Vanschoren, Understanding machine learning performance with experiment databases, Ph.D. thesis, University of Leuven, 2010.
[112] J. Vanschoren, H. Blockeel, B. Pfahringer, G. Holmes, Experiment databases. A new way to share, organize and learn from experiments, Mach. Learn. 87 (2012) 127–158.
[113] J. Vanschoren, J.N. van Rijn, B. Bischl, L. Torgo, OpenML: networked science in machine learning, ACM SIGKDD Explor. Newsl. 15 (2013) 49–60.
[114] J. Ward, Hierarchical grouping to optimize an objective function, J. Am. Stat. Assoc. 22 (1963) 236–244.
[115] H. Xu, R. Rutenbar, K. Sakallah, Sub-SAT: a formulation for relaxed Boolean satisfiability with applications in routing, in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2003, pp. 814–820.
[116] L. Xu, H.H. Hoos, K. Leyton-Brown, Hierarchical hardness models for SAT, in: Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming, Springer, 2007, pp. 696–711.
[117] L. Xu, H.H. Hoos, K. Leyton-Brown, Hydra: automatically configuring algorithms for portfolio-based selection, in: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI Press, 2010, pp. 210–216.
[118] L. Xu, F. Hutter, H.H. Hoos, K. Leyton-Brown, SATzilla: portfolio-based algorithm selection for SAT, J. Artif. Intell. Res. 32 (2008) 565–606.
[119] L. Xu, F. Hutter, H.H. Hoos, K. Leyton-Brown, Hydra-MIP: automated algorithm configuration and selection for mixed integer programming, in: Proceedings of the RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the Twenty-Second International Joint Conference on Artificial Intelligence, 2011.
[120] L. Xu, F. Hutter, H.H. Hoos, K. Leyton-Brown, Evaluating component solver contributions to portfolio-based algorithm selectors, in: Proceedings of the Fifteenth International Conference on Theory and Applications of Satisfiability Testing, Springer, 2012, pp. 228–241.
[121] L. Xu, F. Hutter, J. Shen, H.H. Hoos, K. Leyton-Brown, Satzilla2012: improved algorithm selection based on cost-sensitive classification models, in: Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions, 2012, pp. 55–58.