Holger Hoos | Marius Lindauer | Torsten Schaub

# claspfolio 2

advances in algorithm selection for answer set programming

Suggested citation referring to the original publication:
Theory and Practice of Logic Programming 14 (2014) 4–5, pp. 569–585
DOI https://doi.org/10.1017/S1471068414000210
ISSN (print) 1471-0684
ISSN (online) 1475-3081

*TLP* **14** (4–5): 569–585, 2014.  
doi:10.1017/S1471068414000210

# `claspfolio` 2: Advances in Algorithm Selection for Answer Set Programming

HOLGER HOOS[1], MARIUS LINDAUER[2,3] and TORSTEN SCHAUB[3]

[1]*University of British Columbia, Canada*
[2]*University of Freiburg, Germany*
[3]*University of Potsdam, Germany*

*submitted 14 February 2014; revised 18 April 2014; accepted 1 May 2014*

**Abstract**

Building on the award-winning, portfolio-based ASP solver `claspfolio`, we present `claspfolio` 2, a modular and open solver architecture that integrates several different portfolio-based algorithm selection approaches and techniques. The `claspfolio` 2 solver framework supports various feature generators, solver selection approaches, solver portfolios, as well as solver-schedule-based pre-solving techniques. The default configuration of `claspfolio` 2 relies on a light-weight version of the ASP solver `clasp` to generate static and dynamic instance features. The flexible open design of `claspfolio` 2 is a distinguishing factor even beyond ASP. As such, it provides a unique framework for comparing and combining existing portfolio-based algorithm selection approaches and techniques in a single, unified framework. Taking advantage of this, we conducted an extensive experimental study to assess the impact of different feature sets, selection approaches and base solver portfolios. In addition to gaining substantial insights into the utility of the various approaches and techniques, we identified a default configuration of `claspfolio` 2 that achieves substantial performance gains not only over `clasp`'s default configuration and the earlier version of `claspfolio`, but also over manually tuned configurations of `clasp`.

## 1 Introduction

Answer Set Programming (ASP; (Baral 2003)) has become a popular approach to declarative problem solving. This is mainly due its combination of a rich and simple modeling language with high performance solving technology. ASP decouples problem specifications from solving algorithms; however, modern ASP solvers are known to be sensitive to search configurations – a phenomenon that is common to advanced Boolean constraint processing techniques. To avoid the necessity of manual solver configuration, a substantial amount of research was thus devoted to automated algorithm configuration and selection approaches, as we detail in Section 2; in ASP, we find works by Gebser *et al.* (2011), Maratea *et al.* (2012), Silverthorn *et al.* (2012), Maratea *et al.* (2013) and Hoos *et al.* (2014), and in particular the two portfolio-based systems `claspfolio` (Gebser *et al.* 2011) and `ME-ASP` (Maratea *et al.* 2013). The idea of such portfolio-based systems is to train

classifiers on features of benchmark instances in order to predict the putatively best solver from a given solver portfolio. The portfolio of solvers used in this approach may consist of distinct configurations of the same solver or contain different solvers.

In what follows, we describe the new portfolio-based ASP system `claspfolio`, whose earlier version 1.0 won first, second, and third places at various ASP competitions. Version 0.8 of `claspfolio` was briefly described in a short paper by Gebser *et al.* (2011) and is conceptually identical to the first stable release of version 1.0. The key design features of this prototype were (i) feature generation using a light-weight version of the ASP solver `clasp`, the original `claspre` system, (ii) performance estimation of portfolio solvers via support vector regression, and (iii) a portfolio consisting of different `clasp` configurations only. In contrast to this rigid original design, the new version 2 of `claspfolio` provides a modular and open architecture (Section 3) that allows for integrating several different approaches and techniques. This includes (i) different feature generators, (ii) different approaches to solver selection, (iii) variable solver portfolios, as well as (iv) solver-schedule-based pre-solving techniques. The default setting of `claspfolio` 2 relies on an advanced version of `claspre` (Section 4), a light-weight version of `clasp` that produces statistics based on which numerous static and dynamic instance features are generated.

The flexible and open design of `claspfolio` 2 is a distinguishing factor even beyond ASP. As such, it provides a unique framework for comparing and combining existing approaches and techniques in a uniform setting. We take advantage of this and conduct an extensive experimental study comparing the influence of different options regarding (i), (ii), and (iii). In addition to gaining insights into the impact of the various approaches and techniques, we identify distinguished options showing substantial performance gains not only over `clasp`'s default configuration but moreover over manually tuned configurations of `clasp`. `claspfolio` 2 is 19-51% faster than the best known static `clasp` configuration and also 14-37% faster than `claspfolio` 1.0, as shown in Table 7 at the end of the paper. To facilitate reproducibility of our results and to promote the use of high-performance ASP solving technology, we have made `claspfolio` 2 publicly available as open-source software at `http://potassco.sourceforge.net/#claspfolio`.

## 2 Related Work

Our work continues a long line of research that can be traced back to John Rice's seminal work on algorithm selection (Rice 1976) on one side, and to work by Huberman *et al.* (1997) on parallel algorithm portfolios on the other side. Especially on SAT problems, automatic algorithm selectors have achieved impressive performance improvements in the last decade. `SATzilla` (Xu *et al.* 2008; 2007; 2009; 2011; 2012) predicted algorithm performance by means of ridge regression until 2009 and nowadays uses a pairwise voting scheme based on random forests; `ISAC` (Kadioglu *et al.* 2010) clusters instances in the instance feature space and uses a nearest neighbour approach on cluster centers for algorithm selection; `3S` (Kadioglu *et al.* 2011; Malitsky *et al.* 2013) uses $k$-NN in the feature space and introduces

pre-solving schedules computed by Integer Linear Programming and cost-sensitive clustering; SNAPP (Collautti *et al.* 2013) predicts algorithm performance based on instance features and chooses an algorithm based on the similarity of the predicted performances. All these systems are specialized on a single approach. They are highly efficient but do not provide a uniform setting, that is, different inputs and different performance metrics.

Apart from SAT, there exist several algorithm selectors for other problems. Following the original claspfolio of Gebser *et al.* (2011) approach, Maratea *et al.* (2012) presented ME-ASP, a multi-engine algorithm selector for ASP with an instance feature generator for syntactic features. Similarly, AQME (Pulina and Tacchella 2007) is a multi-engine selector for QSAT. CP-Hydra (O'Mahony *et al.* 2008) selects a set of CSP solvers based on case-based reasoning and schedules them heuristically. Stone Soup (Seipp *et al.* 2012; Helmert *et al.* 2011) uses greedy hill climbing to find algorithm schedules for planning problems. aspeed (Hoos *et al.* 2014) also computes algorithm schedules, but takes advantage of the modeling and solving capabilities of ASP to find timeout-minimal schedules.

Related to our work on a more general level, Hutter *et al.* (2012) gave an overview over runtime prediction techniques, which is also used in some algorithm selection approaches, e.g., SATzilla09. A comparison of different machine learning algorithms for algorithm selection was presented by Kotthoff *et al.* (2012). Based on these results, Kotthoff (2013) introduced LLAMA, Leveraging Learning to Automatically Manage Algorithms, a flexible framework that provides functionality to train and assess the performance of different algorithm selection techniques.

## 3 Generalized Algorithm Selection Framework

claspfolio 2's new algorithm framework combines the flexibility of LLAMA with additional state-of-the-art techniques and produces an executable algorithm selection solver. As such, it provides a unique framework for comparing and combining existing approaches and techniques in a uniform setting. Furthermore, the new design of claspfolio 2 follows the idea of Level 4 of *programming by optimisation* (Hoos 2012): *"The software-development process is centered on the idea of providing design choices and alternatives in all parts of a project that might benefit from them; design choices that cannot be justified convincingly are not made prematurely."*

A further distinguishing feature of the claspfolio 2 framework is the efficient and deep integration of an algorithm scheduling system, viz. aspeed (Hoos *et al.* 2014), into an algorithm selection framework to compute a static pre-solving schedule. claspfolio 2 uses aspeed to determine the running times used within pre-solving schedules. Thereby, it considers the estimated quality of the algorithm selector to determine the running time of the complete pre-solving schedule. This also allows us to integrate the pre-solving strategies of SATzilla and 3S.

The general workflow underlying claspfolio 2 consists of collecting training data, learning a prediction model and training a pre-solving schedule; the portfolio-based ASP solver thus obtained solves a given problem instance with the pre-solving

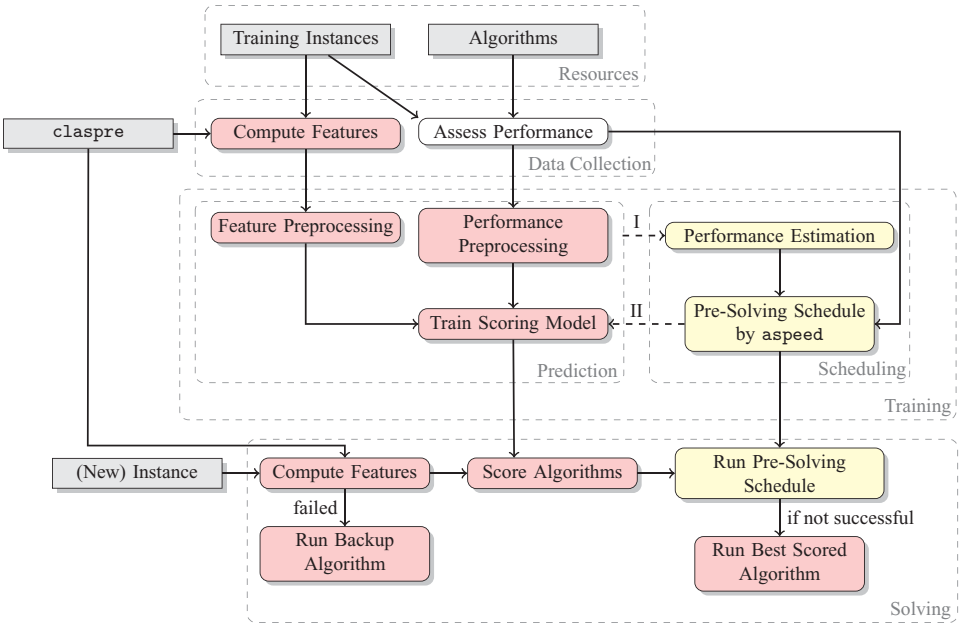


Fig. 1. (Colour online) General workflow of `claspfolio` 2. Objects such as algorithms and instances are shown as rectangles, and activities are depicted as rectangles with rounded corners. Activities related to algorithm are tinted red and activities related to algorithm schedules yellow.

schedule and a solver selected by the prediction model. In what follows, we describe how this workflow is implemented efficiently in `claspfolio` 2; see Figure 1.

*1. Resources.* To train an algorithm selector, *training instances* and a portfolio of *algorithms* are required. Algorithm selection is based on the assumption that the given *training instances* are representative for the instances to be solved using the trained algorithm selection solver. In addition, a portfolio, i.e., a set of *algorithms* with complementary strengths (e.g., high-performance solvers used in a competition), provides the basis for algorithm selectors to efficiently solve a large variety of instances.

*2. Data Collection.* An algorithm selection task is defined based on the performance of all algorithms on all training instances (*Assess Performance*), instance features for each instance (*Compute Features*) and the costs for feature computation define an algorithm selection task.`claspfolio` 2 supports several feature generators, of which `claspre` is used by default.

*3. Training.* The training phase of `claspfolio` 2 makes use of two distinct components: *Prediction* and *Scheduling*. Both components can also be used separately in `claspfolio` 2.

The *Prediction* component of `claspfolio` 2 involves *feature pre-processing*, e.g., feature normalization and feature selection, and *performance pre-processing*,

e.g., performance score transformation and algorithm filtering[1]. Based on the preprocessed data, a *scoring model* is learned, which maps the feature vector for a given problem instance to scores for all *algorithms* such that algorithms expected to perform well on the given instances are assigned better scores.

The *Scheduling* component of `claspfolio 2` computes a timeout-minimal pre-solving schedule using `aspeed` (Hoos *et al.* 2014), where each algorithm gets a (potentially zero) time slice of the overall runtime budget available for solving a given problem instance. If the *prediction* component is not used, the schedule consists only of the given *algorithms*. If the *prediction* component is used, cross validation is used to obtain an unbiased estimate of the performance (*Performance Estimation*) of the *prediction* component (Arrow I). The resulting performance estimate of the prediction component is used as an additional simulated algorithm in the schedule generation process. All components of the schedule except the simulated one form the pre-solving schedule used in `claspfolio 2`. If the *prediction* performs well, the pre-solving schedule may be empty because the pre-solving schedule cannot perform better than a perfect predictor, i.e., the selection of the best solver. In contrast, if *prediction* performs very poorly (e.g., as a result of non-informative instance features), the simulated algorithm may be assigned a time slice of zero seconds and the prediction component is de facto ignored in the *solving* step.

Like `SATzilla` (Xu *et al.* 2008), `claspfolio 2` allows to ignore instances solved by the pre-solving schedule (Arrow II) when learning the scoring model, such that the resulting model is focused on the harder instances not solved by the pre-solvers that are actually subject to algorithm selecting during the solving phase.

*4. Solving* a *(new) instance* starts with the computation of its features. If feature computation fails, e.g., because it requires too much time, a *backup solver* is used to solve the instance. Otherwise, the scoring model is used to *score* each algorithm of the portfolio based on the computed feature vector. If the algorithm with the best score is part of the pre-solving schedule, it is removed from the schedule, because running the same algorithm twice does not increase the solving probability (when using deterministic algorithms like `clasp`). Next, the *pre-solving schedule* is executed.[2] If at the end of executing the pre-solving schedule, the instance has not been solved, the algorithm with the highest score is run for the remainder of the overall time budget.

## 4 `claspre`: Instance Features for ASP

The entire concept of algorithm selection is based on instance features which characterize benchmark instances and allow for predicting the putatively best solver

[1] Algorithm filtering removes components of the portfolio given some strategy, e.g., algorithms with a marginal contribution on virtual best solver performance of 0 can be removed. In (Xu *et al.* 2008), this is called solver subset selection and in (Maratea *et al.* 2012), solver selection.

[2] Unlike this, `SATzilla` runs the pre-solving schedule first and then computes the instance features, because the feature computation can be costly in SAT and the pre-solving schedule can solve the instance without incurring this cost. However, this does not permit removal of the selected solver from the pre-solving schedule.

Table 1. *38 static features computed by* `claspre` *(# = number, % = fraction, SCCs = Strongly Connected Components, BADG = Body-Atom-Dependency Graph)*

| | | |
|---|---|---|
| Tight | # Rules | % Integrity Constraints |
| # Problem Variables | #, % Normal Rules | # Equivalences |
| # Free problem Variables | #, % Cardinality Rules | #, % Atom-Atom |
| # Assigned problem Variable | #, % Choice Rules | Equivalences |
| # Constraints | #, % Weight Rules | #, % Body-Body |
| # Constraints / #Variables | % Negative body Rules | Equivalences |
| # Created Bodies | % Positive body Rules | #, % Other Equivalences |
| # Program Atoms | % Unary Rules | #, % Binary Constraints |
| # SCCs | % Binary Rules | #, % Ternary Constraints |
| # Nodes in positive BADG | % Ternary Rules | #, % Other Constraints |

Table 2. *25 dynamic features computed (at each restart) by* `claspre` *(# = number, % = fraction, ∅ = average, LBD = Literal Blocking Distance)*

| | | |
|---|---|---|
| # Choices | #, % Literals conflict | #, % Learnt other nogoods |
| # Conflicts / #Choices | nogoods | Longest backjump (bj) |
| ∅ conflict level | #, % Literals loop nogoods | #, ∅ Skipped levels while bj |
| ∅ LBD level | #, % Removed nogoods | running average Conflict |
| #, % Learnt conflict nogoods | #, % Learnt binary nogoods | level |
| #, % Learnt loop nogoods | #, % Learnt ternary nogoods | running average LBD level |

from a given portfolio. These instance features should be cheap-to-compute to save as much time as possible for the actual solving process, but should also provide sufficient information to distinguish between (classes of) instances for which different solvers or solver configurations work best.

For feature generation, `claspfolio` 2 uses `claspre` in its default configuration. `claspre` is a light-weight version of `clasp` (Gebser *et al.* 2011) that extracts instance features of ground ASP instances in `smodels` format (Syrjänen ), using `clasp`'s internal statistics. The features determined by `claspre` can be grouped into static and dynamic ones. The former are listed in Table 1 and include 38 properties, such as number of constraints. Beyond that, `claspre` performs a limited amount of search to collect dynamic information about solving characteristics. These dynamic features are computed after each restart of the search process, where restarts are performed after a fixed number of conflicts. Thereby, 25 dynamic features (Table 2) are extracted after each restart, such as the average number of conflict levels skipped while back-jumping.

The number of restarts performed is a parameter of `claspre`. More restarts lead to longer feature vectors that may contain more information. The number of restarts and number of conflicts between restarts determine the time used by `claspre` for feature computation We note that the pre-processing and search performed by `claspre` can actually solve a given ASP instance. The probability of this happening

increases with the length of the search performed within `claspre`; however, at the same time, long runs of `claspre` reduce the time available for running solvers from the portfolio.

## 5 Empirical Performance Analysis

As previously described, `claspfolio` 2's modular and open architecture (Section 3) allows for integrating several different approaches and techniques, including (i) different feature generators, (ii) different approaches to solver selection, as well as (iii) variable solver portfolios. Taking advantages of this flexibility, we conducted an extensive experimental study to assess the efficacy of the various choices on large and representative sets of ASP instances.

Training data of `claspfolio` 2 is stored in the algorithm selection data format developed by the *COSEAL Group*,[3] an international group of experts in the field of algorithm selection and configuration. Detailed experimental results and the source code of `claspfolio` 2 are available at `http://www.cs.uni-potsdam.de/claspfolio`. Our empirical analysis makes use of commonly used techniques from statistics and machine learning (see, e.g., (Bishop 2007)).

### 5.1 Setup

All our experiments were performed on a computer cluster with dual Intel Xeon E5520 quad-core processors (2.26 GHz, 8192 KB cache) and 48 GB RAM per node, running Scientific Linux (2.6.18-308.4.1.el5). Each algorithm run was limited to a runtime cutoff of 600 CPU seconds and to a memory cutoff of 6 GB. Furthermore, we used permutation tests with 100 000 permutations and significance level $\alpha = 0.05$ to our performance metrics, the (0/1) timeout scores, the PAR10 scores and the PAR1 scores,[4] to asses the statistical significance of observed performance differences.

### 5.2 Instance Sets

We used all instances submitted to the 2013 ASP Competition in the `NP` category that could be grounded with `gringo` (3.0.5) within 600 CPU seconds and 6 GB memory. The resulting instance set consists of 2214 instances from 17 problem classes; we call it `Comp-13-Set`. As an even more heterogeneous instance set, we used the ASP `Potassco-Set` introduced by Hoos *et al.* (2013); it consists of 2589 instances from 105 problem classes and includes instances from the ASP competitions organized in 2007 (SLparse track), 2009 (with the encodings of the Potassco group) and 2011 (decision NP-problems from the system track), as well as several instances from the ASP benchmark collection platform *asparagus*.[5] All instances were grounded with `gringo`, and the grounding time was not counted towards solving the instances.

[3] `https://code.google.com/p/coseal`
[4] PAR$X$ is the penalized average runtime penalizing timeouts by $X$ times the runtime cutoff.
[5] `http://asparagus.cs.uni-potsdam.de`

Each instance set was randomly split into equally sized, disjoint training and test set; only the training sets were used in the process of building algorithm portfolios. The resulting `claspfolio` 2 solvers were evaluated on the hold-out test sets. We also used the training instances to determine the best `claspfolio` 2 configuration (Subsection 5.3). To assess the performance of `claspfolio` 2 (Subsection 5.6), we used a 10-fold cross validation on the test set. Notice that we cannot use the training set for `claspfolio` 2 to obtain an unbiased learned model, because the algorithm portfolios have an optimistic performance estimation on the training set on which they were build.

### *5.3 Building Algorithm Portfolios*

In addition to a set of training instances, a portfolio (i.e., a set) of algorithms is required to construct a portfolio solver. `claspfolio` 2 can handle portfolios containing different solvers as well as different configurations of a given solver, all of which are viewed as individual ASP solvers. We investigated the following portfolios of ASP solvers:

- `Expert-portfolio` of four `clasp` (2.1.3) configurations designed by Benjamin Kaufmann (configurations: *frumpy* (default), *jumpy*, *handy* and *crafty*)
- `SOTA-portfolio` (Maratea *et al.* 2012): non-portfolio solvers participating in the 2013 ASP Competition[6] and in addition, the well-established solvers `cmodels` and `smodels`; in detail: `clasp` (Gebser *et al.* 2011), `cmodels` (Giunchiglia *et al.* 2006), `lp2bv` (Nguyen *et al.* 2013), `lp2mip` (Liu *et al.* 2012), `lp2sat` (Janhunen 2006), `smodels` (Simons *et al.* 2002), and `wasp` (Alviano *et al.* 2013)
- `Hydra-like-portfolio` (Xu *et al.* 2010; Xu *et al.* 2011) of `clasp` (2.1.3) configurations
- `ISAC-like-portfolio` (Kadioglu *et al.* 2010) of `clasp` (2.1.3) configurations

`Expert-portfolio` and `SOTA-portfolio` are portfolios manually constructed by experts. In contrast, `Hydra` and `ISAC` are automatic methods for constructing portfolios using algorithm configurators, e.g., `ParamILS` (Hutter *et al.* 2007), `GGA` (Ansótegui *et al.* 2009) or `SMAC` (Hutter *et al.* 2011). They generate a portfolio of configurations of a given solver by determining configurations that complement each other well on a given set of training instances, with the goal of optimizing the performance of the portfolio under the idealized assumption of perfect selection; this performance is also called the virtual best solver (vbs) or oracle performance of the portfolio.

An implementation of `Hydra` that can be applied to solvers for arbitrary problems has not yet been published by Xu *et al.*; therefore, we have implemented our own version of `Hydra` (in consultation with the authors), which we refer to as `Hydra-like-portfolio` in the following. Also, since the only published version of `ISAC` (2.0) does not include algorithm configuration, we reimplemented the part of

[6] `IDP3` was removed from the portfolio because it was strongly dominated by all other solvers.

Table 3. *Virtual best solver (VBS) performance of portfolio building approaches on test sets. Results shown in boldface were statistically significantly better than all others within the respective column (according to a permutation test with* 100 000 *permutations and* $\alpha = 0.05$*).*

| | Comp-13-Set | | | Potassco-Set | | |
|---|---|---|---|---|---|---|
| | #TOs | PAR10 | PAR1 | #TOs | PAR10 | PAR1 |
| Expert-portfolio | 360 | 2169 | 255 | 100 | 491 | 74 |
| SOTA-portfolio | 335 | 1866 | 231 | 111 | 538 | 75 |
| Hydra-like-portfolio | 326 | 1798 | 207 | **82** | **400** | **58** |
| ISAC-like-portfolio | **313** | **1724** | **196** | 99 | 476 | 63 |

ISAC responsible for portfolio generation, dubbed ISAC-like-portfolio. In contrast to the original ISAC, which performs g-means clustering, ISAC-like-portfolio uses k-means clustering, where the number of clusters is determined by using cross-validation to optimize the scoring function of the k-means procedure (following Hoos *et al.* (2013)).

Using this approach, ISAC-like-portfolio found 15 clusters for Comp-13-Set and 11 clusters for Potassco-Set, inducing 15 and 11 configuration tasks, respectively. To obtain a fair comparison, we allocated the same time budget to Hydra-like-portfolio and allowed it to perform 15 and 11 iterations, respectively (each consisting of one configuration task). The configuration process performed by SMAC (2.06.01; Hutter *et al.* 2011) on each cluster and in each Hydra iteration, respectively, was allocated 120 000 CPU seconds, i.e., 200 times the target algorithm cutoff time, and 10 independent repetitions, from which the result with the best PAR10 score on the given training set was selected. SMAC optimized PAR10.

Table 3 shows the performance of the virtual best solvers (i.e., the performance of a perfect algorithm selector) for the different considered portfolios. Interestingly, the results differ qualitatively between two benchmark sets. While SOTA-portfolio performs better than Expert-portfolio on Comp-13-Set, Expert-portfolio is better on Potassco-Set. Furthermore, while for both sets, the automatic generation methods found better performing portfolios than the the manual selected methods, on the Comp-13-Set, ISAC-like-portfolio produced a better results than Hydra-like-portfolio, and the opposite holds for Potassco-Set. Furthermore, unlike conjectured by Maratea *et al.* (2012), a set of configurations of the same, highly parameterized solver (Expert-portfolio, ISAC-like-portfolio and Hydra-like-portfolio) generally did not yield worse performance than a mixed portfolio, such as SOTA-portfolio.

While we gave Hydra the same time budget as ISAC to find portfolios, the components added by Hydra-like-portfolio in its final three iterations decreased the number of timeouts only by one on our training and test sets. Following Xu *et al.* (2010), Hydra would be terminated when the performance does not improve on the training set after an iteration. Hence, Hydra-like-portfolio not only produced a better portfolio on Potassco-Set than ISAC, but also does so using less configuration time than ISAC.

Table 4. *Time required for computing the features of a single ASP instance in CPU seconds, with a* 600 *seconds runtime cutoff. We report minimum (Min), 25% quantile ($Q_{0.25}$), median and 75% quantile ($Q_{0.75}$) of the distribution over the respective instance set, as well as the percentage of timeouts (%TOs).*

| | Comp-13-Set | | | | | Potassco-Set | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Min | $Q_{0.25}$ | Median | $Q_{0.75}$ | %TOs | Min | $Q_{0.25}$ | Median | $Q_{0.75}$ | %TOs |
| claspre(s) | 0.04 | 1.43 | 1.72 | 8.83 | 16.2 | 0.13 | 0.91 | 1.38 | 1.72 | 1.0 |
| claspre(s+d) | 0.07 | 1.36 | 1.72 | 13.94 | 16.2 | 0.18 | 0.87 | 1.48 | 1.81 | 1.1 |
| ME-ASP | 0.04 | 1.18 | 1.97 | 15.97 | 3.2 | 0.06 | 0.83 | 1.10 | 1.79 | 0.1 |
| lp2sat | 0.08 | 24.88 | 484.85 | 600 | 49.4 | 0.04 | 3.81 | 21.82 | 91.13 | 14.6 |

### *5.4 Feature Sets*

In addition to the claspre feature set presented in Section 4, we considered a set of ASP features introduced by Maratea *et al.* (2013) that is focussed on very efficiently computable syntactic features, such as number of variables. The published version of their feature generator supports only the ASPCore 1.0 (Calimeri *et al.* 2011) language of the 2011 ASP Competition. Our Comp-13-Set consists of instances of the 2013 ASP Competition in ASPCore 2.0, which introduced further language constructs. Therefore, we re-implemented this feature generator with the help of Maratea *et al.* to be compatible with ASPCore 2.0.[7]

One of the most established and investigated feature generators for SAT is provided as part of SATzilla (Xu *et al.* 2008). ASP instances can be translated to SAT with techniques by Janhunen (2006), using his tool lp2sat. We use a combination of lp2sat[8] with the feature generator of SATzilla to generate a set of instance features for ASP instances; this is the first time, these features are studied in the context of ASP. Since the full set of SATzilla features is very expensive to compute and our SAT encodings can get quite large, we decided to only use the efficiently computable base features.

Table 4 shows the runtime statistics for claspre with static features, claspre(s), claspre with static and dynamic features, claspre(s+d), with 4 restarts and 32 conflicts between the restarts, the (re-implemented) feature generator of ME-ASP and the combination of lp2sat and SATzilla's feature generator on our full benchmark sets (training + test instances). claspre(s) is only slightly faster than claspre with additional dynamic features, since its search was limited to 128 conflicts. To solve typical ASP instances, searches well beyond 100 000 conflicts are often required; nevertheless, claspre(s) solved 51 instances through pre-processing, and claspre(s+d) solved 123 instances on Comp-13-Set, 9 and 400 instances on Potassco-Set, respectively. The feature generation of ME-ASP was faster, but (unsurprisingly, considering the nature of these features) did not solve any instance.

[7] The new feature generator is implement in Python, whereas the original generator was implemented in C++, which induced an overhead of a factor 2 in terms of running time on average on ASPCore 1.0 instances from the 2011 ASP Competition.

[8] lp2sat was used as submitted at the 2013 ASP Competition.

Table 5. *Algorithm selection mechanism supported by* `claspfolio 2`.

| | Approach | Feat. Norm. | Pre-Solver | Pre-Solver Time [sec] |
|---|---|---|---|---|
| `aspeed` | static schedule | none | $\leqslant \infty$ | $\leqslant \infty$ |
| `claspfolio-1.0-like` | support vector regression | $z$-score | 0 | 0 |
| `ME-ASP-like` | nearest neighbor | none | 0 | 0 |
| `ISAC-like` | $k$-means clustering | linear | 0 | 0 |
| `3S-like` | $k$-NN | linear | $\leqslant \infty$ | $\leqslant$ cutoff/10 |
| `SATzilla'09-like` | ridge regression | $z$-score | $\leqslant 2$ | $\leqslant 20$ |
| `SATzilla'11-like` | voting with random forest | $z$-score | $\leqslant 3$ | $\leqslant 30$ |

Because of the substantial overhead of generating translations from ASP to SAT, the combination of `lp2sat` and `SATzilla`'s feature generator turned out to be substantially slower than the other approaches and failed to compute the feature vectors of 1094 instances on `Comp-13-Set` and 377 instances on `Potassco-Set` within the given cutoff time.

### *5.5 Algorithm Selection Approaches*

As previously mentioned, `claspfolio` 2 was explicitly designed to easily integrate several state-of-the-art algorithm selection approaches. This not only permits us to optimize the performance of `claspfolio` 2, but also to compare the considered algorithm selection approaches within a controlled environment. Although our re-implementations may not reproduce the original implementations in all details (something that would be difficult to achieve, considering that sources are not available for some published approaches), they provide the only freely available, open-source implementations of some of these systems and thus provide a basis for further analysis and improvements.[9]

Table 5 gives an overview of the approaches available within `claspfolio` 2. These differ with respect to (i) the algorithm selection method, (ii) the feature normalization technique, (iii) the maximal number of pre-solvers used and (iv) the maximal running time allocated to the pre-solving schedule. In all cases, the pre-solving schedules were computed by `aspeed`, and hyperparameters of the machine learning techniques were set using grid search on training data.

### *5.6 Results*

We have assessed the performance of `claspfolio` 2 on all 112 combinations of our 4 feature sets, 4 portfolios and 7 algorithm selection approaches, using a cross

[9] As with `Hydra` and `ISAC` above, published and trainable, general-purpose implementations of `3S` and `ME-ASP` are not available.

Table 6. *Statistics ($\mu$ = average, $\sigma$ = standard deviation, min = minimum) of PAR10 performance over all combinations except for the one kept fixed to assess its impact.*

| | Impact of feature set | | | |
|---|---|---|---|---|
| | Comp-13-Set | | Potassco-Set | |
| | $\mu_{PAR10} \pm \sigma_{PAR10}$ | $\min_{PAR10}$ | $\mu_{PAR10} \pm \sigma_{PAR10}$ | $\min_{PAR10}$ |
| claspre(s) | **2116.3** $\pm$ 128.7 | 1927.0 | 638.9 $\pm$ 81.1 | 490.6 |
| claspre(s+d) | 2127.6 $\pm$ 122.6 | 1931.3 | **630.8** $\pm$ 78.1 | **480.0** |
| ME-ASP | 2138.4 $\pm$ 127.7 | **1919.4** | 661.0 $\pm$ 108.8 | 486.0 |
| lp2sat | 2240.3 $\pm$ 81.3 | 2056.9 | 688.3 $\pm$ 45.6 | 610.3 |
| | Impact of portfolio | | | |
| | Comp-13-Set | | Potassco-Set | |
| | $\mu_{PAR10} \pm \sigma_{PAR10}$ | $\min_{PAR10}$ | $\mu_{PAR10} \pm \sigma_{PAR10}$ | $\min_{PAR10}$ |
| Expert-portfolio | 2251.8 $\pm$ 55.0 | 2165.0 | 679.1 $\pm$ 47.7 | 621.6 |
| SOTA-portfolio | 2172.4 $\pm$ 60.6 | 2072.9 | 691.9 $\pm$ 55.3 | 614.7 |
| Hydra-like-portfolio | 2141.5 $\pm$ 160.4 | 1943.7 | **609.6** $\pm$ 103.5 | **480.0** |
| ISAC-like-portfolio | **2056.9** $\pm$ 111.3 | **1919.4** | 638.3 $\pm$ 90.9 | 526.7 |
| | Impact of selection mechanism | | | |
| | Comp-13-Set | | Potassco-Set | |
| | $\mu_{PAR10} \pm \sigma_{PAR10}$ | $\min_{PAR10}$ | $\mu_{PAR10} \pm \sigma_{PAR10}$ | $\min_{PAR10}$ |
| aspeed | 2292.8 $\pm$ 66.1 | 2222.0 | 731.2 $\pm$ 40.8 | 672.6 |
| claspfolio-1.0-like | 2152.7 $\pm$ 108.0 | 1978.6 | 650.3 $\pm$ 58.3 | 519.3 |
| ME-ASP-like | 2245.3 $\pm$ 77.3 | 2091.8 | 753.3 $\pm$ 76.7 | 656.8 |
| ISAC-like | 2100.1 $\pm$ 113.5 | 1939.5 | 608.4 $\pm$ 65.7 | 490.6 |
| 3S-like | 2092.0 $\pm$ 109.2 | 1927.0 | 596.0 $\pm$ 57.6 | 489.1 |
| SATzilla'09-like | 2120.3 $\pm$ 99.4 | 1932.6 | 652.7 $\pm$ 48.2 | 544.0 |
| SATzilla'11-like | **2086.4** $\pm$ 125.9 | **1919.4** | **591.1** $\pm$ 62.5 | **480.0** |

validation on both test sets. To study the effect of each design choice, we collected statistics over the distribution of results by keeping one choice fixed and varying all remaining components; the results are shown in Table 6. The top part of the table shows results obtained for using each of the feature sets, in terms of average PAR10 performance, standard deviation in PAR10 performance and best PAR10 performance over all 28 combinations of portfolios and selection approaches. The subsequent parts of Table 6 show analogous results for different portfolios and selection approaches.

On average, the best feature set was claspre(s) (the static claspre features) on Comp-13-Set, followed by claspre(s+d) (the static + dynamic claspre features), the feature sets of ME-ASP and lp2sat. However, the best claspfolio 2 configuration on Comp-13-Set used ME-ASP. The fact that claspre(s+d) gave worse results than claspre(s), although the former is superset of the latter, indicates that

Table 7. *Comparison of two* `clasp` *configurations, the* `Single Best` *solver in all portfolios (cf. Subsection 5.3),* `claspfolio 1.0`*, the* `claspfolio 2` *with* `claspre(s+d)` *features,* `Hydra-like-portfolio` *and* `SATzilla'11-like` *approach. The significantly best performances (except VBS) are shown in boldface (according to a permutation test with* 100 000 *permutations and significance level* $\alpha = 0.05$*).*

| | Comp-13-Set | | | Potassco-Set | | |
|---|---|---|---|---|---|---|
| | *#TOs* | PAR10 | PAR1 | *#TOs* | PAR10 | PAR1 |
| clasp (default) | 577 | 3168 | 351 | 287 | 1347 | 176 |
| clasp (ASP Comp 13) | 421 | 2329 | 273 | 150 | 723 | 97 |
| Single Best | 414 | 2333 | 268 | 150 | 723 | 97 |
| claspfolio 1.0 | 403 | 2237 | 269 | 134 | 658 | 99 |
| claspfolio 2 | **353** | **1960** | **237** | **97** | **480** | **75** |
| best known VBS | 313 | 1724 | 196 | 82 | 400 | 58 |

not all features were useful and that feature selection should be used to identify a subset of features with highest information content. On Potassco-Set, the best average performance and the best performance of any claspfolio 2 configuration was consistently obtained by using claspre(s+d). We believe that the additional dynamic features are necessary to distinguish between the larger number of different problem classes in Potassco-Set.

The results on the impact of the portfolio of algorithms used as a basis for algorithm selection confirm our assumption that the best potential performance, i.e., best VBS performance, is a good indicator of the actual performance achieved by a high-performance selection approach. On Comp-13-Set, ISAC-like-portfolio achieved the best performance, while on Potassco-Set, Hydra-like-portfolio yielded even better results. Furthermore, the portfolios obtained using the two automatic portfolio generation methods, ISAC and Hydra, yielded better results than the manually created ones, Expert-portfolio and SOTA-portfolio.

As shown in the lower part of Table 6, the SATzilla'11-like approach performed best on both benchmark sets, followed closely by 3S-like and ISAC-like. SATzilla'09-like and claspfolio-1.0-like showed similar, but weaker performance results, followed by the ME-ASP-like approach and the pure algorithm schedules of aspeed.

Overall, the best combination both on the training and test sets of Comp-13-Set was the ME-ASP features, ISAC-like-portfolio and SATzilla'11-like selection approach, and claspre(s+d) features, Hydra-like-portfolio and SATzilla'11-like selection approach for Potassco-Set.

## 6 Conclusions and Future Work

Our new, modular claspfolio 2 ASP solver architecture comprises a diverse set of portfolio-based algorithm selection techniques, including feature extractors,

manually and automatically constructed base algorithm portfolios, algorithm selection mechanisms and solver-schedule-based pre-solving techniques. As seen from the high-level overview of empirical performance results in Table 7, on standard, diverse and heterogeneous sets of ASP benchmarks, `claspfolio 2` is substantially more robust than the default configuration of `clasp`, the manual tuned configuration of `clasp` of the 2013 ASP Competition, and than all other assessed individual solvers; in fact, its performance in terms of PAR10-score lies only about 20% and 15% above that of the best known oracle on `Potassco-Set` and `Comp-13-Set` benchmark sets, respectively. The reimplementation of `claspfolio` 1.0 in `claspfolio 2`, which had a similar performance in preliminary experiments than the original implementation, achieves also about $14 - 37\%$ higher PAR10-score than `claspfolio 2`. While the best configuration of `claspfolio 2` varies between these two benchmark sets, the performance differences are relatively minor: on `Comp-13-Set`, the best configuration of `claspfolio 2` for `Potassco-Set` – which we also chose as the default configuration for `claspfolio 2` – achieves a PAR10-score only about 2.1% lower than the best configuration for `Comp-13-Set`, and on `Potassco-Set`, its PAR10-score is about 9.6% higher. This configuration uses the `claspre(s+d)` feature set in combination with the `Hydra-like-portfolio` base algorithm portfolio construction approach and the `SATzilla'11-like` algorithm selection mechanism, but other feature sets, base algorithm portfolios and algorithm selection mechanisms also achieve very strong performance.

## Acknowledgments

T. Schaub and M. Lindauer were supported by the DFG projects under SCHA 550/8-3. H. Hoos was supported by an NSERC Discovery Grant.

## References

Alviano, M., Dodaro, C., Faber, W., Leone, N., and Ricca, F. 2013. WASP: A native asp solver based on constraint learning. In *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*, P. Cabalar and T. Son, Eds. Lecture Notes in Artificial Intelligence, vol. 8148. Springer-Verlag, 54–66.

Ansótegui, C., Sellmann, M., and Tierney, K. 2009. 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)*, I. Gent, Ed. Lecture Notes in Computer Science, vol. 5732. Springer-Verlag, 142–157.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Bessiere, C., Ed. 2007. *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP'07)*. Lecture Notes in Computer Science, vol. 4741. Springer-Verlag.

Bishop, C. 2007. *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. 2006. Corr. 2nd printing ed. Springer.

Calimeri, F., Ianni, G., and Ricca, F. 2011. Third ASP competition - file and language formats. Tech. rep., Università della Calabria.

Collautti, M., Malitsky, Y., Mehta, D., and O'Sullivan, B. 2013. SNAPP: Solver-based nearest neighbor for algorithm portfolios. In *Proceedings of the Twenty-Fourth European Conference on Machine Learning (ECML'13)*, F. Zelezny, Ed. Lecture Notes in Computer Science. Springer-Verlag.

Dovier, A. and Santos Costa, V., Eds. 2012. *Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12)*. Vol. 17. Leibniz International Proceedings in Informatics (LIPIcs).

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Schneider, M. 2011. Potassco: The Potsdam answer set solving collection. *AI Communications 24,* 2, 107–124.

Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M., and Ziller, S. 2011. A portfolio solver for answer set programming: Preliminary report. In *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, J. Delgrande and W. Faber, Eds. Lecture Notes in Artificial Intelligence, vol. 6645. Springer-Verlag, 352–357.

Giunchiglia, E., Lierler, Y., and Maratea, M. 2006. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning 36,* 4, 345–377.

Helmert, M., Röger, G., and Karpas, E. 2011. Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*. 28–35.

Hoos, H. 2012. Programming by optimisation. *Communications of the ACM 55*, 70–80.

Hoos, H., Kaminski, R., Lindauer, M., and Schaub, T. 2014. aspeed: Solver scheduling via answer set programming. *Theory and Practice of Logic Programming First View*, 1–26. Available at `http://arxiv.org/abs/1401.1024`.

Hoos, H., Kaufmann, B., Schaub, T., and Schneider, M. 2013. Robust benchmark set selection for boolean constraint solvers. See Pardalos and Nicosia (2013), 138–152.

Huberman, B., Lukose, R., and Hogg, T. 1997. An economic approach to hard computational problems. *Science 275*, 51–54.

Hutter, F., Hoos, H., and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*. Lecture Notes in Computer Science, vol. 6683. Springer-Verlag, 507–523.

Hutter, F., Hoos, H., and Stützle, T. 2007. Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-second National Conference on Artificial Intelligence (AAAI'07)*. AAAI Press, 1152–1157.

Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. 2012. Algorithm runtime prediction: The state of the art. *Artificial Intelligence*.

Janhunen, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics 16,* 1-2, 35–86.

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. 2011. Algorithm selection and scheduling. In *Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP'11)*, J. Lee, Ed. Lecture Notes in Computer Science, vol. 6876. Springer-Verlag, 454–469.

Kadioglu, S., Malitsky, Y., Sellmann, M., and Tierney, K. 2010. ISAC – instance-specific algorithm configuration. In *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI'10)*, H. Coelho, R. Studer, and M. Wooldridge, Eds. IOS Press, 751–756.

Kotthoff, L. 2013. LLAMA: leveraging learning to automatically manage algorithms. Tech. rep., Cork Constraint Computation Centre. published at arXiv.

Kotthoff, L., Gent, I. P., and Miguel, I. 2012. An evaluation of machine learning in algorithm selection for search problems. *AI Communications 25,* 3, 257–270.

Liu, G., Janhunen, T., and Niemel, I. 2012. Answer set programming via mixed integer programming. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, G. Brewka, T. Eiter, and S. McIlraith, Eds. AAAI Press, 32–42.

Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. 2013. Boosting sequential solver portfolios: Knowledge sharing and accuracy prediction. See Pardalos and Nicosia (2013), 153–167.

Maratea, M., Pulina, L., and Ricca, F. 2012. Applying machine learning techniques to ASP solving. See Dovier and Santos Costa (2012), 37–48.

Maratea, M., Pulina, L., and Ricca, F. 2013. A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming First View*, 1–28.

Nguyen, M., Janhunen, T., and Niemelä, I. 2013. Translating answer-set programs into bit-vector logic. In *Proceedings of the Nineteenth International Conference on Applications of Declarative Programming and Knowledge Management (INAP'11) and the Twenty-fifth Workshop on Logic Programming (WLP'11)*, H. Tompits, S. Abreu, J. Oetsch, J. Pührer, D. Seipel, M. Umeda, and A. Wolf, Eds. Lecture Notes in Computer Science, vol. 7773. Springer-Verlag, 105–116.

O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., and O'Sullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of the Nineteenth Irish Conference on Artificial Intelligence and Cognitive Science (AICS'08)*, D. Bridge, K. Brown, B. O'Sullivan, and H. Sorensen, Eds.

Pardalos, P. and Nicosia, G., Eds. 2013. *Proceedings of the Seventh International Conference on Learning and Intelligent Optimization (LION'13)*. Lecture Notes in Computer Science, vol. 7997. Springer-Verlag.

Pulina, L. and Tacchella, A. 2007. A multi-engine solver for quantified boolean formulas. See Bessiere (2007), 574–589.

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

Seipp, J., Braun, M., Garimort, J., and Helmert, M. 2012. Learning portfolios of automatically tuned planners. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS'12)*, L. McCluskey, B. Williams, J. R. Silva, and B. Bonet, Eds. AAAI.

Silverthorn, B., Lierler, Y., and Schneider, M. 2012. Surviving solver sensitivity: An ASP practitioner's guide. See Dovier and Santos Costa (2012), 164–175.

Simons, P., Niemelä, I., and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1-2, 181–234.

Syrjänen, T. Lparse 1.0 user's manual.

Xu, L., Hoos, H., and Leyton-Brown, K. 2007. Hierarchical hardness models for SAT. See Bessiere (2007), 696–711.

Xu, L., Hoos, H., and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, M. Fox and D. Poole, Eds. AAAI Press, 210–216.

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

Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. 2009. SATzilla2009: An automatic algorithm portfolio for SAT. In *SAT 2009 competitive events booklet: preliminary version*, D. Le Berre, O. Roussel, L. Simon, V. Manquinho, J. Argelich, C. Li, F. Manyà, and J. Planes, Eds. 53–55. Available at `http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf`.

Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. 2011. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI'11)*.

Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. 2012. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proceedings of the Fifteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, A. Cimatti and R. Sebastiani, Eds. Lecture Notes in Computer Science, vol. 7317. Springer-Verlag, 228–241.