Surviving Solver Sensitivity: An ASP Practitioner's Guide

BRYAN SILVERTHORN[1], YULIYA LIERLER[2], and MARIUS SCHNEIDER[3]

1 **Department of Computer Science**
The University of Texas at Austin
Austin, TX, USA
bsilvert@cs.utexas.edu

2 **Department of Computer Science**
University of Kentucky
Lexington, KY, USA
yuliya@cs.uky.edu

3 **Department of Computer Science**
University of Potsdam
Potsdam, Germany
manju@cs.uni-potsdam.de

Abstract

Answer set programming (ASP) is a declarative programming formalism that allows a practitioner to specify a problem without describing an algorithm for solving it. In ASP, the tools for processing problem specifications are called answer set solvers. Because specified problems are often NP complete, these systems often require significant computational effort to succeed. Furthermore, they offer different heuristics, expose numerous parameters, and their running time is sensitive to the configuration used. Portfolio solvers and automatic algorithm configuration systems are recent attempts to automate the problem of manual parameter tuning, and to mitigate the burden of identifying the right solver configuration. The approaches taken in portfolio solvers and automatic algorithm configuration systems are orthogonal. This paper evaluates these approaches, separately and jointly, in the context of real-world ASP application development. It outlines strategies for their use in such settings, identifies their respective strengths and weaknesses, and advocates for a methodology that would make them an integral part of developing ASP applications.

1998 ACM Subject Classification I.2.2 Automatic analysis of algorithms

Keywords and phrases algorithm configuration, algorithm selection, portfolio solving, answer set programming, algorithm portfolios

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Answer set programming (ASP) [19, 20] is a declarative programming formalism based on the answer set semantics of logic programs [9]. Its origins go back to the observation that the language of logic programs can be used to model difficult combinatorial search problems so that *answer sets* correspond to the solutions of a problem. In the declarative programming paradigm, a software engineer expresses the logic of a computation without describing its control flow or algorithm. Thus a declarative program is a description of what should be accomplished, rather than a description of how to go about accomplishing it. As a result,

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

declarative programming requires tools that process given problem specifications and find their solutions. In ASP such systems are called answer set *solvers*. They implement a difficult computational task, since the problem of deciding whether a logic program has an answer set is NP-complete. Despite the complexity of the task, ASP and its tools have proved to be useful; ASP has been used to model and solve problems in many real-world domains, including computational biology and linguistics.

The computational methods of top-performing answer set solvers such as CMODELS [10] and CLASP [8, 5] are strongly related to those of satisfiability (SAT) solvers—software systems designed to find satisfying assignments for propositional formulae [11]. It is well known that modern SAT solvers are sensitive to parameter configuration. The same holds for answer set solvers. These systems typically implement numerous heuristics and expose a variety of parameters to specify the chosen configuration. For example, the command line

```
clasp --number=1 --trans-ext=no --eq=5 --sat-prepro=0 --lookahead=no
      --heuristic=Berkmin --rand-freq=0.0 --rand-prob=no
      --rand-watches=true --restarts=100,1.5 --shuffle=0,0
      --deletion=3.0,1.1,3.0 --strengthen=yes
      --loops=common --contraction=250 --verbose=1
```

represents the default configuration of answer set solver CLASP (version 2.0.2). On one hand, a rich set of heuristics implemented in CLASP makes this solver successfully applicable to a variety of problem domains. On the other hand, given an application at hand, it is unclear how to go about picking the *best* configuration of the system. Gebser et al. [7] write:

> In fact, we are unaware of any true application on which CLASP is run in its default settings. Rather, in applications, "black magic" is used to find suitable search parameters.

We believe that *black magic* refers to manual tuning that relies on "rules of thumb", solver familiarity, and the user's domain expertise. Unfortunately, it is unreasonable to expect that a regular ASP application developer possesses enough knowledge about the internals of answer set solvers and their heuristics to understand the full implications of picking a particular configuration. Furthermore, requiring such extraordinary expertise would diminish the idea of declarative programming itself.

In this paper we evaluate some of the tools available for addressing the black magic problem—including portfolio solvers such as CLASPFOLIO [7] and BORG [23], and automatic algorithm configuration systems such as PARAMILS [14]—in the context of specific problem domains in ASP, highlighting real-world applications. Our goal is not to develop a novel system, but to aid ASP application developers, especially those hoping to leverage existing tools for portfolio solving and configuration tuning. We provide a case study that illustrates and evaluates alternative methodologies in three practical domains: weighted sequence [18], natural language parsing [17], and Riposte [2]. The struggle to deal with solver sensitivity in the former two domains, in fact, triggered the research described in this paper. Our hope is to identify a systematic way to remove haphazard manual tuning and performance evaluation from the process of applying ASP tools to a new domain. We focus our attention on tuning the ASP solver CLASP (version 2.0.2). The configuration space of CLASP consists of 8 binary, 7 categorical, and 25 continuous parameters, which makes us believe that this system alone is a good choice for evaluation. Nevertheless, all of the methods investigated here may accommodate any other solver of interest, as well as multiple solvers.

Six candidate strategies are studied:

1. selecting the best single configuration from among the 25 representative CLASP configurations used by CLASPFOLIO (a CLASP-based portfolio solver) for each domain;
2. constructing a "solver execution schedule" over those 25 configurations;
3. applying CLASPFOLIO, trained on its large set of ASP instances, to each domain without further modification;
4. training a portfolio specifically on each application domain using the 25 CLASP configurations of CLASPFOLIO;
5. tuning a single CLASP configuration specifically for each domain using PARAMILS; and
6. training a portfolio specifically on each application domain using multiple configurations produced by tuning CLASP, using PARAMILS, for individual instances of the domain.

We believe that these options are representative of modern approaches for dealing with solvers' configuration sensitivity, able to illustrate the strengths and weaknesses of each approach. Also, to the best of our knowledge, the evaluation of strategies 4 and 6 on individual problem domains is unique to this paper.

We start by describing the domains used in the proposed case study. Section 3 gives an overview of portfolio methods in general together with the details of the strategies 1, 2, 3, and 4. Section 4 outlines the general principles behind the algorithm configuration system PARAMILS and describes the details of strategy 5. Strategy 6 is specified in Section 5. We conclude with a thorough analysis of the methods' performance in practice.

2 Review of Application Domains

In this work we compare and contrast several methodologies on three domains that stem from different subareas of computer science. This section provides a brief overview of these applications. We believe that these domains represent a broad spectrum of ASP applications, and are thus well-suited for the case study proposed. The number of instances available for each application ranged from several hundred to several thousand. The complexity of the instances also varied. The diversity of the instances and their structure played an important role in our choice of domains.

The **weighted-sequence** (WSEQ) domain is a handcrafted benchmark problem that was used in the Third Answer Set Programming Competition[1] (ASPCOMP) [3]. Its key features are inspired by the important industrial problem of finding an optimal join order by cost-based query optimizers in database systems. In our analysis we used 480 instances of the problem, which were generated according to the metrics described by Lierler et al. [18].

The **natural language parsing** (NLP) domain formulates the task of parsing natural language, i.e., recovering the internal structure of sentences, as a planning problem in ASP. In particular, it considers the combinatory categorical grammar formalism for realizing parsing. Lierler and Schüller [17] describe the procedure of acquiring instances of the problem using CCGbank[2], a corpus of parsed sentences from real world sources. In this work we study 1,861 instances produced from the CCGbank data.

Riposte (RIP)[3] is a project in computer aided verification, where ASP is used to generate counterexamples for the FDL intermediate language of the SPARK program verification system. These counterexamples point at the problematic areas of the analyzed code. We evaluate

[1] `https://www.mat.unical.it/aspcomp2011/OfficialProblemSuite`. WSEQ was referred to as a benchmark number 28, *Weight-Assignment Tree*.
[2] `http://groups.inf.ed.ac.uk/ccg/ccgbank.html`.
[3] `https://forge.open-do.org/projects/riposte`

instances created from the application of Riposte to a SPARK implementation of the Skein hash function; these 3,133 instances were shared with us by Martin Brain in February 2012.

3 Algorithm Portfolio Methods

In *portfolio solving*, an "algorithm portfolio method" or "portfolio solver" automatically divides computation time among a suite of solvers. SAT competitions[4] have provided a rich source of diverse solvers and benchmark instances, and have spurred the development of portfolio solving. Several different types of portfolio solvers exist. These range from simple methods that divide computational resources equally among a hand-selected suite of solvers, to more complex systems that make informed decisions by analyzing the appearance of instances. The development of the ASP portfolio solver CLASPFOLIO [7] was largely inspired by the advances of this approach in SAT, and especially by the ideas championed by the portfolio SAT solver SATZILLA [26].

Gebser et al. [6] suggest that portfolio solving in general and CLASPFOLIO in particular is a step toward overcoming the sensitivity of modern answer set solvers to parameter settings. Nevertheless, the extent to which existing portfolio solvers achieve this goal on individual application domains is an open issue. In this paper, we shed light on two questions related to it. First, how well does a general-purpose portfolio, trained on many different instance types, perform when compared against the default CLASP configuration on a particular domain? Second, what benefits are gained from moving from general-purpose to application-driven portfolios, by training a portfolio solver specifically for the application in question?

Two different portfolio systems are employed in considering these questions: the CLASPFOLIO algorithm-selection system is used as a general-purpose portfolio, and the BORG algorithm-portfolio toolkit is used to construct domain-specific portfolios based on the MAPP architecture [23]. Both of these approaches are described below.

3.1 Algorithm Selection and CLASPFOLIO

Systems for automatic algorithm selection, such as SATZILLA for SAT and CLASPFOLIO for ASP, leverage the appearance of an instance to make decisions about which solver or configuration to apply. An algorithm selection system typically involves two components:

- a suite or *portfolio* of different solvers or solver configurations, and
- a solver (or configuration) *selector*.

The selector is responsible for picking the best-performing solver for a particular instance. The definition of "best-performing" is arbitrary, but expected run time is often used. The efficiently computable properties of an instance on which these methods base their decisions are called numerical *features* of that instance.

Techniques from supervised machine learning are used to build the selector component. Thousands of runs are observed during a *training* phase, and each run is labeled with its performance score and the features of its associated instance. These examples are then used to learn a function that maps an instance, using its features, directly to a solver selection decision. Using this architecture, algorithm-selection portfolios have been top performers at the SAT and ASP competitions. For example, the portfolio answer set solver CLASPFOLIO was the winner of the NP category in the system track of ASPCOMP.

[4] `http://www.satcompetition.org/`.

The CLASPFOLIO (version 1.0.1) solver employs 25 representative configurations of CLASP (version 2.0.2), and a feature set that includes properties of an ASP instance ranging from the number of constraints to the length of clauses learned from short initial runs. The choice of configurations of CLASP that were used in building CLASPFOLIO relied on the expertise of Benjamin Kaufmann, the main designer of CLASP, and on black magic. CLASPFOLIO[5] was trained on 1,901 instances from 60 different domains. It will be used to evaluate the performance of general-purpose portfolio, one designed to operate on a wide variety of instance types. A different system, but one that exhibits comparable performance, is used to evaluate the performance of domain-specific portfolio solvers. It is described next.

3.2 Solver Scheduling, MAPP, and BORG

We utilize the BORG toolkit[6] as our experimental infrastructure. Like tools such as RUNSOLVER [21], BORG executes solvers while measuring and limiting their run time. It is also designed to collect and analyze solver performance data over large collections of instances, to compute instance feature information, and to construct different portfolio solvers.

To build domain-specific portfolios, BORG instantiates the "modular architecture for probabilistic portfolios" (MAPP) [23]. Unlike an algorithm selection method, MAPP computes the complete *solver execution schedule* that approximately maximizes the probability of solving the instance within the specified run time constraint. A solver execution schedule consists of one or more sequential calls to possibly different solvers, where the last call is allocated all remaining runtime.

Unlike an algorithm selection portfolio, then, MAPP may run more than one solver on an instance. This strategy has proved to be effective. Earlier versions of MAPP [24], built for a portfolio of pseudo-Boolean (PB) solvers, took first place in the main category of the 2010 and 2011 PB competitions.

Two different types of MAPP portfolios are evaluated:

- MAPP$^-$, which does not use instance features, and thus consistently executes a single solver execution schedule computed over the run times of all training instances, and
- MAPP$^+$, which uses instance features to tailor each execution schedule to a given instance.

We used the BORG framework to create MAPP portfolios using the same 25 configurations of CLASP employed by CLASPFOLIO. This will allow us to more fairly compare the effectiveness of the application-driven portfolio-solving approach studied here to that of the general-purpose CLASPFOLIO system. Furthermore, we use CLASPFOLIO itself to compute instance-specific features for MAPP$^+$. As a result, MAPP$^+$ tailors a solver execution schedule to each instance using the same features available to CLASPFOLIO.

We also use BORG to select the "best single" (BESTSINGLE) configuration of CLASP, from among those 25, that maximizes the probability of successfully solving an instance of the training set.

Whether they employ pure algorithm selection or solver execution scheduling, portfolio methods have repeatedly proved successful on collections of competition instances. Such collections include instances of many different problems. Section 6 evaluates their behavior instead on collections of instances drawn entirely from each of our representative domains.

The next section discusses an orthogonal methodology for handling solver sensitivity. Instead of marshalling multiple fixed configurations, it follows a local search strategy through

[5] `http://potassco.sourceforge.net/#claspfolio`
[6] `http://nn.cs.utexas.edu/pages/research/borg/`.

the configuration space of a solver, attempting to identify the best-performing single configuration on a domain.

4 Automatic Algorithm Configuration

The success of portfolio solving in competition demonstrates that selecting a solver's configuration is important. This success, however, leads to an obvious question: instead of focusing on the selection of an existing configuration, can we obtain a new configuration that performs better (or best) on a particular domain? This paper examines this possibility by applying a tool for *automatic algorithm configuration* to CLASP on our three application domains.

We take PARAMILS[7] [14] (version 2.3.5) as a representative of automatic algorithm configuration tools. Other systems of this kind include SMAC [12, 13] and GGA [1]. PARAMILS is based on iterative local search in the configuration space, and evaluates the investigated configurations on a given training set of instances. Its *focusedILS* approach allows it to focus the evaluation on a subset of the given instances, and thus to assess the quality of a configuration more quickly. This subset is adaptively extended after each update of the current suboptimal solution. The idea behind this approach is that a configuration that performs well on a small subset is also a good choice for the entire instance set. We designed our algorithm configuration experiments based on this observation.

In the experiments we tuned CLASP, with the help of PARAMILS, on a randomly sampled subset of 50 instances for each of the domains. The maximal cutoff time of each CLASP call was 1,200 seconds, the tuning time was 120,000 seconds, and the minimization of the average runtime was the optimization objective. Since PARAMILS uses a local search approach, it (i) is non-deterministic and (ii) can become trapped in a local optimum. Therefore, we ran the PARAMILS experiment ten times, independently, and afterwards chose the configuration with the best performance.

For all experiments, we used a discretized configuration space of CLASP selected by Benjamin Kaufmann. It is similar to the parameter file used in the experiments of Gebser et al. [7], and is available online at `http://www.cs.uni-potsdam.de/wv/claspfolio/`.

5 Domain-Specific Portfolio Synthesis

An automatic algorithm configuration system such as PARAMILS generates a *single* configuration tuned on a set of many instances. On the other hand, the assumption made by portfolio methods is that *multiple* configurations exhibit complementary strengths on a distribution of instances. If this assumption does not hold on some domain for a standard suite of solvers, is it possible to use automatic algorithm configuration to generate a new suite of complementary solvers? Systems such as Hydra have explored this possibility in SAT [25]. Here, we evaluate a simple strategy for doing so in ASP, leveraging PARAMILS. In this protocol, we

1. randomly sample a set of N instances from the domain,
2. use PARAMILS to tune a configuration of CLASP specifically for each instance,
3. collect training data for each configuration across the entire domain, and
4. construct a portfolio using those training data.

This methodology follows from the assumption that multiple distinct instance subtypes exist in the domain, and that instances belonging to these subtypes will thus be present in the

[7] `http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/`

random sample. By tuning a configuration to each instance, and therefore to each subtype, a portfolio of complementary solvers may emerge.

In our evaluations, we use $N = 20$ instances sampled from each domain to test this possibility in the set of experiments described next. We applied the same PARAMILS settings as described in previous section, with the exception of running PARAMILS only once, instead of ten times, to tune CLASP on each of the 20 instances. Afterwards, we selected 16 configurations for WSEQ, 16 for NLP and 6 for RIP, which were found on instances with runtimes longer than 0.2 seconds on average. Typically, all inspected CLASP configurations performed comparably for these easy instances. Once again we utilized the BORG toolkit to build the kind of portfolio solvers described in Section 3.2 for each of the studied domains, in this case using the configurations found by PARAMILS.

6 Experimental Results

The experiments in this section compare and contrast the approaches discussed for handling solver sensitivity in ASP. To recap, we will compare strategies 1–6, summarized in the introduction, by measuring the performance of the BESTSINGLE, MAPP, and PARAMILS-based solvers trained with various CLASP configurations and on different training sets. In addition, we will present the performance of:

- CLASPFOLIO,
- the DEFAULT configuration of CLASP, and
- the ORACLE portfolio, also called the *virtual best solver*, which corresponds to the minimal run time on each instance given a portfolio approach with perfect knowledge.

The configurations found by PARAMILS are not included in the portfolios of BESTSINGLE, CLASPFOLIO, MAPP and ORACLE. All solver runs were collected on a local cluster (Xeon X5355 @ 2.66GHz) with a timeout set to 1,200 CPU seconds.

We use the standard technique of five-fold cross validation to get an unbiased evaluation. Each collection of instances is split into pairs of training and test sets. In five-fold cross validation, these pairs are generated by dividing the collection into five disjoint test sets of equal size, with the instances left out of each test set used to form each training set.

First, to illustrate the potential effectiveness of portfolio methods, Table 1 presents the performance of CLASPFOLIO and MAPP (trained with the CLASPFOLIO configurations on the CLASPFOLIO training set) on the ASPCOMP instances from the NP category in the system track. Note that the performance of CLASPFOLIO and of the MAPP$^+$ solver appear quite similar in this situation. This performance similarity allows us to take the MAPP$^+$ approach as representative of portfolio methods in general in our evaluations. These results show that portfolios are clearly effective on *heterogeneous* collections of instances that include multiple qualitatively different problem domains. It is less clear, however, that multiple complementary solver configurations exist across instances within a single problem domain.

To answer the question of whether portfolio solving provides any benefit on individual problem domains, Figure 1 presents performance curves for experiments run on each domain separately, under five-fold cross validation. BESTSINGLE, the MAPP portfolios, and ORACLE were all trained on each specific domain (using the CLASPFOLIO configurations). PARAMILS denotes the CLASP configuration found for each domain, as described in Section 4. On a domain included in its training set (WSEQ, 5 instances), CLASPFOLIO performs well, only slightly worse than a portfolio trained specifically on that domain. On domains not included in its training set—NLP and RIP—CLASPFOLIO is less effective, beating the

	ASPCOMP	
Solver	**Solved**	**MRT (s)**
DEFAULT	75	82.99
BESTSINGLE	82	63.75
MAPP^{-}	82	63.75
MAPP^{+}	84	75.41
CLASPFOLIO	85	97.47
ORACLE	91	48.84

Table 1 The number of instances solved and the mean run time (MRT) on those solved instances for single-solver and portfolio strategies on the 125 ASPCOMP instances (with all portfolios trained on the CLASPFOLIO training set.)

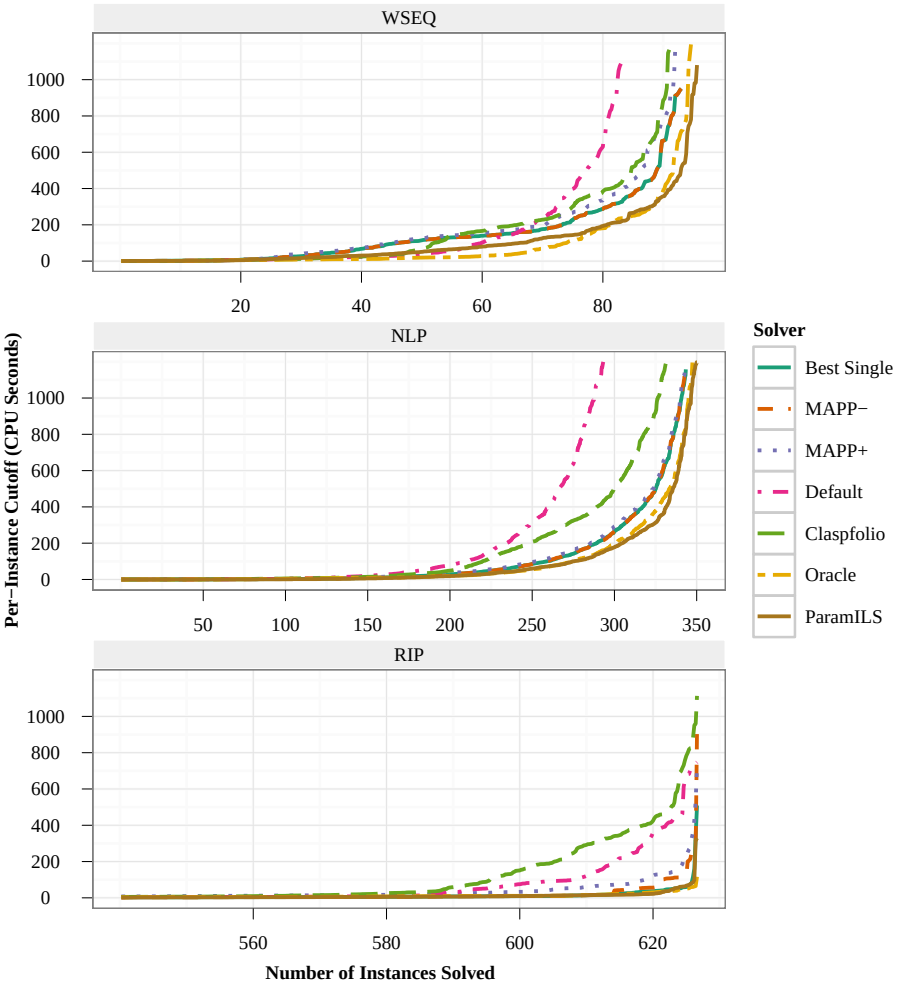

Figure 1 Cactus plots presenting the performance, under five-fold cross validation, of strategies 1-5 on the three application domains considered in this paper.

	WSEQ		NLP		RIP	
Solver	**Solved**	**MRT (s)**	**Solved**	**MRT (s)**	**Solved**	**MRT (s)**
BESTSINGLE (ILS)	93.40	163.95	343.80	95.27	626.20	1.39
MAPP$^-$ (ILS)	93.40	163.95	343.80	95.26	626.00	1.95
MAPP$^+$ (ILS)	93.80	170.52	342.40	100.61	626.00	5.48
ORACLE (ILS)	94.80	72.27	349.40	80.66	626.20	1.13

Table 2 Results summarizing the performance of "PARAMILS-based" portfolios, as described in Section 5, according to the mean number of instances solved and the mean run time on those instances. These scores were averaged over five-fold cross validation.

DEFAULT configuration on NLP but losing to it on RIP. Lacking domain-specific training, then, CLASPFOLIO can struggle to identify good configurations. Portfolios trained for each domain (MAPP$^-$ and MAPP$^+$) consistently perform much better than DEFAULT and CLASPFOLIO. This improvement seems to be due to identifying a single good configuration: note that the MAPP$^-$ and BESTSINGLE solvers are almost identical in their performance. This hypothesis was confirmed by analyzing the solver execution schedule of MAPP$^-$: it turns out that MAPP$^-$, on these domains, may practically be identified with the BESTSINGLE solver approach. Comparing MAPP$^+$ and MAPP$^-$ performance, then, shows that feature-based prediction provides no benefit in these experiments. The feature computation overhead incurred by MAPP$^+$ and CLASPFOLIO on "easy" domains, such as RIP, is also evident. In these single-problem domains, in other words, the portfolio approach is useful for systematically identifying a good solver configuration, but struggles to make useful performance predictions from feature information.

In contrast, configurations tuned via PARAMILS perform very well. The performance of PARAMILS tracks that of the ORACLE portfolio. Both portfolios and algorithm configuration improve on the performance of DEFAULT by large margins.

Note also, however, that run time can be misleading. For example, DEFAULT is faster on some instances of the WSEQ domain, but solves fewer overall. These deceptive aspects of solver performance strongly suggest that an ASP application developer should employ a tool, such as a portfolio framework, to systematically collect and analyze solver performance. Cursory approaches, such as manually experimenting with only a few instances, can lead to the suboptimal selection of a configuration.

Since collecting training data from the entire domain incurs substantial cost, our recommendation would be to collect such data on a modest randomly sampled subset of instances. If configurations exhibit substantial differences in performance on that subset, and especially if the performance gap between the BESTSINGLE solver and the ORACLE portfolio is large, then additional training data may enable a portfolio method to make up some of that difference. Such decisions might also be made based on recently proposed formal definitions of instance set homogeneity [22].

Configurations found by PARAMILS provide substantial gains in performance on every domain. It is interesting to see that they perform nearly the same as the ORACLE portfolio of CLASPFOLIO configurations: if perfect algorithm selection were somehow available, we would not need to tune the configuration. Conversely, it is impressive that the range of configurations spanned by the CLASPFOLIO suite of solvers can be equaled by a single tuned configuration on these domains.

Table 2 presents details of the performance of portfolios obtained under the methodology described in Section 5. No further improvement in comparison to the PARAMILS configuration

could be obtained under this methodology. Either a single configuration is sufficient to achieve maximum CLASP-derived performance on these domains, or a more sophisticated approach to portfolio construction must be used—the ISAC approach [15], for example, which attempts to explicitly identify subgroups of instances within the domain, or the Hydra system, which accounts for overall portfolio performance in making tuning decisions [25]. This question is left to future investigation.

7 Conclusions

The results of this experimental study strongly recommend two courses of action for ASP application developers, one general and one specific. As a general recommendation, it is clear that significant care must be paid to solver parameterization in order to accurately characterize performance on a domain. Employing a portfolio toolkit to systematically collect run time data and select the best CLASPFOLIO configuration is a reasonable and straightforward first step. As a specific recommendation, however, the use of automatic algorithm configuration can wring more performance from a domain. Preparing such a tool, however, itself requires intimate knowledge of a specific solver. Solver authors could empower the solver's users by providing configuration files for PARAMILS or a related tool.

One final observation made clear by this work is the importance of understanding the desired solver performance objective. An ASP developer must carefully select an appropriate run time budget for their task, and must carefully weigh their desires for efficiency and consistent success. These desires may be in conflict, and the effectiveness of algorithm portfolio and configuration methods both depend on a user understanding and accurately specifying their own preferences.

The need for studies such as that conducted in this paper has also been expressed by Karp [16]. By looking at worst-case asymptotic performance over the space of all possible inputs, theoretical computer science typically predicts the intractability in general of the computational tasks exemplified by ASP or SAT. In practice, however, these challenging tasks can often be solved, thus driving the need for an experimental approach to the task of finding and evaluating algorithms for difficult search problems on specific domains. Karp writes:

> A tuning strategy [searches] the space of concrete algorithms consistent with the algorithmic strategy to find the one that performs best on the training set. Finally, an evaluation method compares the chosen algorithm with its competitors on a verification set of instances drawn from the same distribution as the training set.

The case study presented in this work, as well as the methodologies it explored, are steps toward refining such an experimental approach—an approach that appears essential to enabling a practitioner to evaluate and apply increasingly powerful, increasingly sensitive parameterized solvers.

Acknowledgments

We are grateful to Vladimir Lifschitz, Peter Schüller, and Miroslaw Truszczynski for useful discussions related to the topic of this work. Martin Brain, Peter Schüller, and Shaden Smith assisted us with the instances used in this case study. Yuliya Lierler was supported by a CRA/NSF 2010 Computing Innovation Fellowship.

References

1 C. Ansótegui, M. Sellmann, and K. Tierney. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. In I. Gent, editor, *Proceedings of the CP'09*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer-Verlag, 2009.

2 M. Brain and F. Schanda. Riposte: Supporting development in spark using counter-examples. Unpublished manuscript, 2012.

3 F. Calimeri, G. Ianni, F. Ricca, M. Alviano, A. Bria, G. Catalano, S. Cozza, W. Faber, O. Febbraro, N. Leone, M. Manna, A. Martello, C. Panetta, S. Perri, K. Reale, M. Carmela Santoro, M. Sirianni, G. Terracina, and P. Veltri. The third answer set programming competition: Preliminary report of the system competition track. In Delgrande and Faber [4], pages 388–403.

4 J. Delgrande and W. Faber, editors. *Proceedings of the LPNMR'11*, volume 6645 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2011.

5 M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):105–124, 2011.

6 M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Challenges in answer set solving. In M. Balduccini and T. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond*, volume 6565, pages 74–90. Springer-Verlag, 2011.

7 M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. Schneider, and S. Ziller. A portfolio solver for answer set programming: Preliminary report. In Delgrande and Faber [4], pages 352–357.

8 M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proceedings of the IJCAI'07*, pages 386–392. MIT Press, 2007.

9 M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of the ICLP'88*, pages 1070–1080. MIT Press, 1988.

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

11 C. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, pages 89–134. Elsevier, 2008.

12 F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION'11*, pages 507–523, 2011.

13 F. Hutter, H. Hoos, and K. Leyton-Brown. Parallel algorithm configuration. In *Proceedings of the LION'12*, 2012. To appear.

14 F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

15 S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. ISAC—instance-specific algorithm configuration. In *Proceedings of the ECAI'10*, 2010.

16 R. Karp. Heuristic algorithms in computational molecular biology. *Journal of Computer and System Sciences*, 77(1):122–128, 2011.

17 Y. Lierler and P. Schüller. Parsing combinatory categorial grammar with answer set programming: Preliminary report. In *Workshop on Logic programming (WLP)*, 2011.

18 Y. Lierler, S. Smith, M. Truszczynski, and A. Westlund. Weighted-sequence problem: Asp vs casp and declarative vs problem oriented solving. In *Proceedings of the PADL'12)*, 2012.

19 V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

20 I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

21 O. Roussel. Controlling a Solver Execution with the runsolver Tool. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(4):139–144, 2011.

22 M. Schneider and H. Hoos. Quantifying homogeneity of instance sets for algorithm configuration. In Y. Hamadi and M. Schoenauer, editors, *Proceedings of the LION'12*, 2012. Submitted for Post-Proceedings.

23 B. Silverthorn. *A Probabilistic Architecture for Algorithm Portfolios*. PhD thesis, The University of Texas at Austin, 2012.

24 B. Silverthorn and R. Miikkulainen. Latent class models for algorithm portfolio methods. In *Proceedings of the AAAI'10*, 2010.

25 L. Xu, H. Hoos, and K. Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI'10*, 2010.

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