The Multi-engine ASP Solver ME-ASP: Progress Report

Marco Maratea
DIBRIS,
Univ. degli Studi di Genova,
Viale F. Causa 15, 16145 Genova, Italy
marco@dist.unige.it

Luca Pulina
POLCOMING,
Univ. degli Studi di Sassari,
Viale Mancini 5, 07100 Sassari, Italy
lpulina@uniss.it

Francesco Ricca
Dip. di Matematica ed Informatica,
Univ. della Calabria,
Via P. Bucci, 87030 Rende, Italy
ricca@mat.unical.it

Abstract

ME-ASP is a multi-engine solver for ground ASP programs. It exploits algorithm selection techniques based on classification to select one among a set of out-of-the-box heterogeneous ASP solvers used as black-box engines. In this paper we report on (i) a new optimized implementation of ME-ASP; and (ii) an attempt of applying algorithm selection to non-ground programs. An experimental analysis reported in the paper shows that (i) the new implementation of ME-ASP is substantially faster than the previous version; and (ii) the multi-engine recipe can be applied to the evaluation of non-ground programs with some benefits.

Introduction

Answer Set Programming (Baral 2003; Eiter, Gottlob, and Mannila 1997; Gelfond and Lifschitz 1988; 1991; Marek and Truszczyński 1998; Niemelä 1998) (ASP) is a declarative language based on logic programming and non-monotonic reasoning. The applications of ASP belong to several areas, e.g., ASP was used for solving a variety of hard combinatorial problems (see e.g., (Calimeri et al. 2011) and (Potsdam since 2002)).

Nowadays, several efficient ASP systems are available (Gebser et al. 2007; Janhunen, Niemelä, and Sevalnev 2009; Leone et al. 2006; Lierler 2005; Mariën et al. 2008; Simons, Niemelä, and Soininen 2002). It is well-established that, for solving empirically hard problems, there is rarely a best algorithm/heuristic, while it is often the case that different algorithms perform well on different problems/instances. It can be easily verified (e.g., by analyzing the results of the ASP competition series) that this is the case also for ASP implementations. In order to take advantage of this fact, one should be able to select automatically the "best" solver on the basis of the characteristics (called *features*) of the instance in input, i.e., one has to consider to solve an *algorithm selection problem* (Rice 1976).

Inspired by the successful attempts (Gomes and Selman 2001; O'Mahony et al. 2008; Pulina and Tacchella 2009; Xu et al. 2008) done in the neighbor fields of SAT, QSAT and CSP, the application of algorithm selection techniques to ASP solving was ignited by the release of the portfolio solver CLASPFOLIO (Gebser et al. 2011). This solver imports into ASP the SATZILLA (Xu et al. 2008) approach. Indeed, CLASPFOLIO employs inductive techniques based on *regression* to choose the "best" configuration/heuristic of the solver CLASP. The complete picture of inductive approaches applied to ASP solving includes also techniques for learning heuristics orders (Balduccini 2011), solutions to combine portfolio and automatic algorithm configuration approaches (Silverthorn, Lierler, and Schneider 2012), automatic selection of a scheduling of ASP solvers (Hoos et al. 2012) (in this case CLASP configurations), and the multi-engine approach. The aim of a multi-engine solver (Pulina and Tacchella 2009) is to select the "best" solver among a set of efficient ones used as *black-box engines*. The multi-engine ASP solver ME-ASP was proposed in (Maratea, Pulina, and Ricca 2012b), and ports to ASP an approach applied before to QBF (Pulina and Tacchella 2009).

ME-ASP exploits inductive techniques based on *classification* to choose, on a per instance basis, an engine among a selection of black-box heterogeneous ASP solvers. The first implementation of ME-ASP, despite not being highly optimized, already reached good performance. Indeed, ME-ASP can combine the strengths of its component engines, and thus it performs well on a broad set of benchmarks including 14 domains and 1462 ground instances (detailed results are reported in (Maratea, Pulina, and Ricca 2014a)).

In this paper we report on (i) a new optimized implementation of ME-ASP; and on (ii) a first attempt of applying algorithm selection to the entire process of computing answer sets of non-ground programs.

As a matter of fact, the ASP solutions available at the state of the art employing machine-learning techniques are devised to solve ground (or propositional) programs, and – to the best of our knowledge – no solution has been proposed that is able to cope directly with non-ground programs. Note that ASP programmers almost always write non-ground programs, which have to be first instantiated by a grounder. It is well-known that such instantiation phase can influence significantly the performance of the entire solving process. At the time of this writing, there are two prominent alternative implementations that are able to instantiate ASP programs: DLV (Leone et al. 2006) and GRINGO (Gebser, Schaub, and Thiele 2007). Once the peculiarities of the instantiation process are properly taken into account, both implementations can be combined in a multi-engine grounder by applying also to this phase an algorithm selection recipe, building on (Maratea, Pulina, and Ricca 2013). The entire process

of evaluation of a non-ground ASP program can be, thus, obtained by applying algorithm selection to the instantiation phase, selecting either DLV or GRINGO; and, then, in a subsequent step, evaluating the propositional program obtained in the first step with a multi-engine solver.

An experimental analysis reported in the paper shows that (i) the new implementation of ME-ASP is substantially faster than the previous version; and (ii) the straight application of the multi-engine recipe to the instantiation phase is already beneficial. At the same time, it remains space for future work, and in particular for devising more specialized techniques to exploit the full potential of the approach.

A Multi-Engine ASP system

We next overview the components of the multi-engine approach, and we report on the way we have instantiated it to cope with instantiation and solving, thus obtaining a complete multi-engine system for computing answer sets of non-ground ASP programs.

General Approach. The design of a multi-engine solver based on classification is composed of three main ingredients: (i) a set of features that are significant for classifying the instances; (ii) a selection of solvers that are representative of the state of the art and complementary; and (iii) a choice of effective classification algorithms. Each instance in a fairly-designed *training set* of instances is analyzed by considering both the features and the performance of each solvers. An inductive model is computed by the classification algorithm during this phase. Then, each instance in a *test set* is processed by first extracting its features, and the solver is selected starting from these features using the learned model. Note that, this schema does not make any assumption (other than the basic one of supporting a common input) on the engines.

The ME-ASP solver. In (Maratea, Pulina, and Ricca 2012b; 2014a) we described the choices we have made to develop the ME-ASP solver. In particular, we have singled out a set of syntactic features that are both significant for classifying the instances and cheap-to-compute (so that the classifier can be fast and accurate). In detail, we considered: the number of rules and number of atoms, the ratio of horn, unary, binary and ternary rules, as well as some ASP peculiar features, such as the number of true and disjunctive facts, and the fraction of normal rules and constraints. The number of resulting features, together with some of their combinations, amounts to 52. In order to select the engines we ran preliminary experiments (Maratea, Pulina, and Ricca 2014a) to collect a pool of solvers that is representative of the state-of-the-art solver (SOTA), i.e., considering a problem instance, the oracle that always fares the best among the solvers that entered the system track of the 3rd ASP Competition (Calimeri et al. 2011), plus DLV. The pool of engines collected in ME-ASP is composed of 5 solvers, namely CLASP, CLASPD, CMODELS, DLV, and IDP, as submitted to the 3rd ASP Competition. We experimented with several classification algorithms (Maratea, Pulina, and Ricca 2014a), and proved empirically that ME-ASP can perform better than its engines with any choice. Nonetheless, we selected the k-nearest neighbor (kNN) classifier for our new implementation: it was already used in ME-ASP (Maratea, Pulina, and Ricca 2012b), with good performance, and it was easy to integrate its implementation in the new version of the system.

Multi-engine instantiator. Concerning the automatic selection of the grounder, we selected: number of disjunctive rules, presence of queries, the total amount of functions and predicates, number of strongly connected and Head-Cycle Free(Ben-Eliyahu and Dechter 1994) components, and stratification property, for a total amount of 11 features. These features are able to discriminate the class of the problem, and are also pragmatically cheap-to-compute. Indeed, given the high expressivity of the language, non-ground ASP programs (which are usually written by programmers) contain only a few rules. Concerning the grounders, given that there are only two alternative solutions, namely DLV and GRINGO, we considered both for our implementation.

Concerning the classification method, we used an implementation of the PART decision list generator (Frank and Witten 1998), a classifier that returns a human readable model based on if-then-else rules. We used PART because, given the relatively small total amount of features related to the non-ground instances, it allows us to compare the generated model with respect to the knowledge of a human expert.

Multi-Engine System ME-ASP^{gr}. Given a (non-ground) ASP program, the evaluation workflow of the multi-engine ASP solution called ME-ASP^{gr} is the following: (i) non-ground features extraction, (ii) grounder selection, (iii) grounding phase, (iv) ground features extraction, (v) solver selection, and (vi) solving phase on ground program.

Implementation and Experiments

In this section we report the results of two experiments conceived to assess the performance of the new versions of the ME-ASP system. The first experiment has the goal of measuring the performance improvements obtained by the new optimized implementation of the ME-ASP solver. The second experiment assesses ME-ASP^{gr} and reports on the performance improvements that can be obtained by selecting the grounder first and then calling the ME-ASP solver. ME-ASP and ME-ASP^{gr} are available for download at `www.mat.unical.it/ricca/me-asp`. Concerning the hardware employed and the execution settings, all the experiments run on a cluster of Intel Xeon E31245 PCs at 3.30 GHz equipped with 64 bit Ubuntu 12.04, granting 600 seconds of CPU time and 2GB of memory to each solver. The benchmarks used in this paper belong to the suite of benchmarks, encoded in the ASP-Core 1.0 language, of the 3rd ASP Competition. Note that in the 4th ASP Competition (Alviano et al. 2013) the new language ASP-Core 2.0 has been introduced. We still rely on the language of the 3rd ASP Competition given that the total amount of solvers and grounders supporting the new standard language is very limited with respect to the number of tools supporting ASP-Core 1.0.

Assessment of the new implementation of ME-ASP. The original implementation of ME-ASP was obtained by combining a general purpose feature extractor (that we have

initially developed for experimenting with a variety of additional features) developed in Java, with a collection of Perl scripts linking the other components of the system, which are based on the *rapidminer* library. This is a general purpose implementation supporting also several classification algorithms. Since the CPU time spent for the extraction of features and solver selection has to be made negligible, we developed an optimized version of ME-ASP. The goal was to optimize the interaction among system components and further improve their efficiency. To this end, we have re-engineered the feature extractor, enabling it to read ground instances expressed in the numeric format used by GRINGO. Furthermore, we have integrated it with an implementation of the kNN algorithm built on top of the ANN library (`www.cs.umd.edu/~mount/ANN`) in the same binary developed in C++. This way the new implementation minimizes the overhead introduced by solver selection.

We now present the results of an experiment in which we compare the old implementation of ME-ASP, labelled ME-ASP^{old}, with the new one, labelled ME-ASP^{new}. In this experiment, assessing solving performance, we used GRINGO as grounder for both implementations, and we considered problems belonging to the *NP* and *Beyond NP* classes of the competition (i.e., the grounder and domains considered by ME-ASP^{old} (Maratea, Pulina, and Ricca 2014a)). The inductive model used in ME-ASP^{new} was the same used in ME-ASP^{old} (details are reported in (Maratea, Pulina, and Ricca 2014a)). The plot in Figure 1 (top) depicts the performance of both ME-ASP^{old} and ME-ASP^{new} (dotted red and solid blue lines in the plot, respectively). Considering the total amount of *NP* and *Beyond NP* instances evaluated at the 3rd ASP Competition (140), ME-ASP^{new} solved 92 instances (77 *NP* and 15 *Beyond NP*) in about 4120 seconds, while ME-ASP^{old} solved 77 instances (62 *NP* and 15 *Beyond NP*) in about 6498 seconds. We report an improvement both in the total amount of solved instances (ME-ASP^{new} is able to solve 66% of the whole set of instances, while ME-ASP^{new} stops at 51%) and in the average CPU time of solved instances (about 45 seconds against 84).

The improvements of ME-ASP^{new} are due to its optimized implementation. Once feature extraction and solver selection are made very efficient, it is possible to extract features for more instances and the engines are called in advance w.r.t. what happens in ME-ASP^{old}. This results in more instances that are processed and solved by ME-ASP^{new} within the timeout.

Assessment of the complete system. We developed a preliminary implementation of a grounder selector, which combines a feature extractor for non-ground programs written in Java, and an implementation of the PART decision list generator, as mentioned in the previous section. The grounder selector is then combined with ME-ASP^{new}.

We now present the results of an experiment in which we compare ME-ASP^{gr} with ME-ASP^{new}, and the SOTA solver. ME-ASP^{new} coupled with DLV (resp. GRINGO) is denoted by ME-ASP^{new} (dlv) (resp. ME-ASP^{new} (gringo)). In this case we considered all the benchmark problems of the 3rd ASP Competition, including the ones belonging to the *P* class. Indeed, in this case we are interested also in

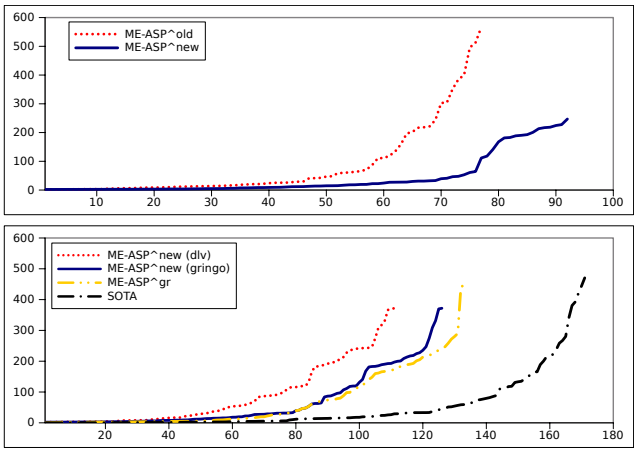

Figure 1: Performance of ME-ASP^{old} and ME-ASP^{new} on *NP* and *Beyond NP* instances evaluated at the 3rd ASP Competition (top); performance of ME-ASP^{gr}, its engines and SOTA on the complete set of instances evaluated at the 3rd ASP Competition (bottom). In the x-axis it is shown the total amount of solved instances, while y-axis reports the CPU time in seconds.

grounders' performance, which is crucial in the *P* class.

The plot in Figure 1 (bottom) shows the performance of the aforementioned solvers. In the plot, we depict the performance of ME-ASP^{new} (dlv) with a red dotted line, ME-ASP^{new} (gringo) with a solid blue line, ME-ASP^{gr} with a double dotted dashed yellow line, and, finally, with a dotted dashed black line we denote the performance of the SOTA solver. Looking at the plot, we can see that ME-ASP^{new} (gringo) solves more instances that ME-ASP^{new} (dlv) – 126 and 111, respectively – while both are outperformed by ME-ASP^{gr}, that is able to solve 134 instances. The average CPU time of solved instances for ME-ASP^{new} (dlv), ME-ASP^{new} (gringo) and ME-ASP^{gr} is 86.86, 67.93 and 107.82 seconds, respectively. Looking at the bottom plot in Figure 1, concerning the performance of the SOTA solver, we report that it is able to solve 173 instances out of a total of 200 instances (evaluated at the 3rd ASP Competition), highlighting room for further improving this preliminary version of ME-ASP^{gr}. Indeed, the current classification model predicts GRINGO for most of the *NP* instances, but having a more detailed look at the results, we notice that CLASP and IDP with GRINGO solve both 72 instances, while using DLV they solve 93 and 92 instances, respectively. A detailed analysis of the performance of the various ASP solvers with both grounders can be found in (Maratea, Pulina, and Ricca 2013).

It is also worth mentioning that the output formats of GRINGO and DLV differ, thus there are combinations grounder/solver that require additional conversion steps in our implementation. Since the new feature extractor is designed to be compliant with the numeric format produced by GRINGO, if DLV is selected as grounder then the non-ground program is instantiated twice. Moreover, if DLV is selected as grounder, and it is not selected also as solver, the produced propositional program is fed in gringo to be

converted in numeric format. These additional steps, due to technical issues, result in a suboptimal implementation of the execution pipeline that could be further optimized in case both grounders would agree on a common output format.

Conclusion. In this paper we presented improvements to the multi-engine ASP solver ME-ASP. Experiments show that (i) the new implementation of ME-ASP is more efficient, and (ii) the straight application of the multi-engine recipe to the instantiation phase is already beneficial. Directions for future research include exploiting the full potential of the approach by predicting the pair grounder+solver, and importing policy adaptation techniques employed in (Maratea, Pulina, and Ricca 2014b).

Acknowledgments. This research has been partly supported by Regione Calabria under project PIA KnowRex POR FESR 2007- 2013 BURC n. 49 s.s. n. 1 16/12/2010, the Italian Ministry of University and Research under PON project "Ba2Know S.I.-LAB" n. PON03PE_0001, the Autonomous Region of Sardinia (Italy) and the Port Authority of Cagliari (Italy) under L.R. 7/2007, Tender 16 2011 project "DESCTOP", CRP-49656.

References

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

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming*, 1070–1080. Cambridge, Mass.: MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *NGC* 9:365–385.

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive Datalog. *ACM TODS* 22(3):364–418.

Frank, E., and Witten, I. H. 1998. Generating accurate rule sets without global optimization. In *ICML'98*, 144.

Marek, V. W., and Truszczyński, M. 1998. Stable models and an alternative logic programming paradigm. *CoRR* cs.LO/9809032.

Niemelä, I. 1998. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. In *CANR 98 Workshop*, 72–79.

Gomes, C. P., and Selman, B. 2001. Algorithm portfolios. *Artificial Intelligence* 126(1-2):43–62.

Potsdam, U. since 2002. asparagus homepage. `http://asparagus.cs.uni-potsdam.de/`.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence* 138:181–234.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Tempe, Arizona: CUP.

Lierler, Y. 2005. Disjunctive Answer Set Programming via Satisfiability. In *LPNMR 05*, LNCS 3662, 447–451.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM TOCL* 7(3):499–562.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *IJCAI-07*, 386–392.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. GrinGo : A New Grounder for Answer Set Programming. In *LPNMR 2007*, LNCS 4483, 266–271.

Mariën, M.; Wittocx, J.; Denecker, M.; and Bruynooghe, M. 2008. Sat(id): Satisfiability of propositional logic extended with inductive definitions. In *SAT 08*, LNCS, 211–224.

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 *ICAICS 08*.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *JAIR* 32:565–606.

Janhunen, T.; Niemelä, I.; and Sevalnev, M. 2009. Computing stable models via reductions to difference logic. In *LPNMR 09*, LNCS, 142–154.

Pulina, L., and Tacchella, A. 2009. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints* 14(1):80–116.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Schaub, T.; Schneider, M. T.; and Ziller, S. 2011. A portfolio solver for answer set programming: Preliminary report. In *LPNMR 11*, LNCS 6645, 352–357.

Balduccini, M. 2011. Learning and using domain-specific heuristics in ASP solvers. *AICOM* 24(2):147–164.

Ben-Eliyahu R.; Dechter R. 1994. Propositional Semantics for Disjunctive Logic Programs *Annals of Mathematics and Artificial Intelligence*. 12:53–87, Science Publishers.

Calimeri, F.; Ianni, G.; Ricca, F.; et al. 2011. The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track. In *Proc. of LPNMR11.*, 388–403 LNCS.

Hoos, H.; Kaminski, R.; Schaub, T.; and Schneider, M. T. 2012. ASPeed: Asp-based solver scheduling. In *Tech. Comm. of ICLP 2012*, volume 17 of *LIPIcs*, 176–187.

Maratea, M.; Pulina, L.; and Ricca, F. 2012b. The multi-engine asp solver me-asp. In *JELIA 2012.*, LNCS 7519, 484–487.

Silverthorn, B.; Lierler, Y.; and Schneider, M. 2012. Surviving solver sensitivity: An asp practitioner's guide. In *Tech. Comm. of ICLP 2012*, volume 17 of *LIPIcs*, 164–175.

Alviano, M.; Calimeri, F.; Charwat, G.; et al. 2013. The fourth answer set programming competition: Preliminary report. In *LPNMR*, LNCS 8148, 42–53.

Maratea, M.; Pulina, L.; and Ricca, F. 2013. Automated selection of grounding algorithm in answer set programming. In *AI* IA 2013*. International Publishing. 73–84.

Maratea, M.; Pulina, L.; and Ricca, F. 2014a. A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming*. DOI: `http://dx.doi.org/10.1017/S1471068413000094`

Maratea, M.; Pulina, L.; and Ricca, F. 2014b. Multi-engine asp solving with policy adaptation. *JLC*. In Press.