The Multi-Engine ASP Solver ME-ASP

Marco Maratea, Luca Pulina, and Francesco Ricca

[1]DIBRIS, Univ. degli Studi di Genova, Viale F.Causa 15, 16145 Genova, Italy
[2]POLCOMING, Univ. degli Studi di Sassari, Viale Mancini 5, 07100 Sassari, Italy
[3]Dipartimento di Matematica, Univ. della Calabria, Via P. Bucci, 87030 Rende, Italy
marco@dist.unige.it, lpulina@uniss.it, ricca@mat.unical.it

Abstract. In this paper we describe the new system ME-ASP, which applies machine learning techniques for inductively choosing, among a set of available ones, the "best" ASP solver on a per-instance basis. Moreover, we report the results of some experiments, carried out on benchmarks from the "System Track" of the 3rd ASP Competition, showing the state-of-the-art performance of our solver.

1 Introduction

Answer Set Programming [7] (ASP) is a truly-declarative programming paradigm proposed in the area of non-monotonic reasoning and logic programming. The idea of ASP is to represent a given computational problem by a logic program whose answer sets correspond to solutions, and then use a solver to find such solutions [12]. The language of ASP is very expressive, indeed all problems in the second level of the polynomial hierarchy are expressible in ASP [4]. Moreover, the applications of ASP nowadays belong to several fields from Artificial Intelligence to Knowledge Management [2]. The development of efficient and fast ASP systems is, thus, a crucial task made even more challenging by existing and new-coming applications.

As witnessed by the ASP competition series (see [3] for the most recent), several efficient ASP solvers have been proposed up to now, which are based on different solving techniques ranging from ASP-specific approaches to translation to SAT/Difference Logic. Inspired by the recent research results on the neighbor fields of SAT and QBF, where inductive techniques for algorithm selection were applied with success [18, 16], we have developed ME-ASP, a *multi-engine* solver for propositional ASP programs.

In this paper we describe this new system. In order to obtain a robust ASP solver, i.e., a system able to perform well across a wide set of problem domains, we leverage a number of efficient ASP systems (e.g., [6, 14, 10, 11, 9, 17]), and we apply machine learning techniques for inductively choosing, among the available ones, the "best" solver to be run on the basis of the characteristics, also called features, of the input program at hand.

We also report the results of some experiments carried out on the grounded version of all benchmarks employed in the "System Track" of the 3rd ASP Competition [3] falling in the "*NP*" and "*Beyond NP*" categories of the competition, that show the state-of-the-art performance of our multi-engine solver; indeed, ME-ASP is able to solve substantially more instances than the winner of the "System Track" of the 3rd ASP Competition.

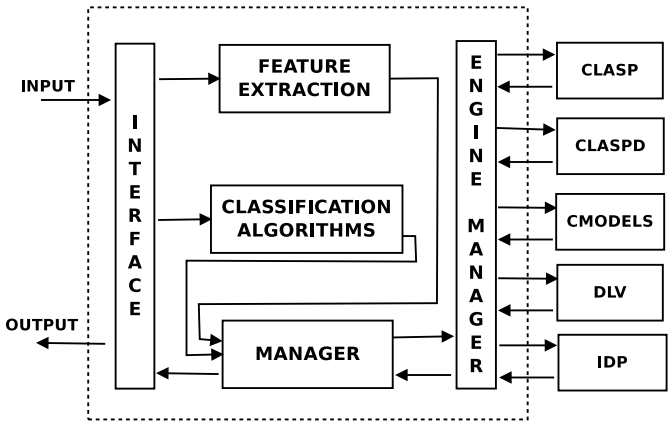

Fig. 1. The architecture of ME-ASP. The dotted box denotes the whole system and, inside it, each solid box represents its modules. Arrows denote functional connections between modules.

It is worth mentioning that, machine learning techniques have been already applied to ASP solving, i.e. CLASPFOLIO and DORS [5, 1]. In particular, the CLASPFOLIO system was conceived and implemented for selecting the "best" heuristic configuration of the CLASP solver. An important difference with ME-ASP is that the application of algorithm selection strategies is limited in CLASPFOLIO (which is, actually, a unique binary including CLASP) to the variants of a single engine; moreover, CLASPFOLIO is not able to deal with ASP programs with syntactically-unrestricted disjunction.

2 The structure of ME-ASP

Figure 1 presents the architecture of ME-ASP[1]. Looking at the figure, we can see that ME-ASP is composed of the five modules described in the following.

`INTERFACE` manages both the input received by the user and the output of the whole system. It also dispatches the input data to the remaining modules, as denoted by the outgoing arrows. In particular, `INTERFACE` collects (i) the ground ASP program in ASP-Core format [3], and (ii) the classifier type and its inductive model.

`FEATURE EXTRACTION` extracts the syntactic features of the input ground program, as detailed in [13]. The CPU time spent for the extraction is negligible.

`CLASSIFICATION ALGORITHMS` is devoted to the prediction of the engine to run. It implements five different inductive models, namely Aggregation Pheromone density based pattern Classification, Decision Rules, Decision Trees, Nearest-neighbor, and Support Vector Machine. We implemented the first one following the methodology described in [8], while the remaining ones are built on top of the RAPIDMINER library [15]. This module receives as input both the classifier type and its inductive

[1] ME-ASP is available for download at `http://www.mat.unical.it/ricca/me-asp`.

Table 1. Results on the 10 grounded instances for each domain evaluated at the 3rd ASP competition. The instances of the DisjunctiveScheduling, PackingProblem and WeightAssignmentTree are not solved by any solver. The table is organized as follows. In the first column we report the benchmark, followed by three groups of columns, each one related to an evaluated solver. Each group is composed of two columns, namely "#Solved" (i.e., the total amount of solved instances within the time limit) and "Time" (i.e., the total CPU time spent on the solved instances).

Problem	ME-ASP		CLASPD		SOTA	
	#Solved	Time	#Solved	Time	#Solved	Time
GraphColouring	4	527.67	3	302.09	4	523.38
HanoiTower	9	1107.67	2	416.94	9	1041.76
KnightTour	8	755.67	8	544.21	8	728.12
Labyrinth	5	415.43	3	275.12	5	344.95
MazeGeneration	10	52.15	10	32.63	10	31.37
MinimalDiagnosis	10	1889.46	10	1859.86	10	69.01
MultiContextSystemQuerying	10	687.93	10	1177.08	10	87.45
Numberlink	8	254.01	7	47.32	8	226.06
SokobanDecision	9	1312.74	7	487.50	9	1182.24
Solitaire	5	767.98	2	57.98	8	1238.21
StrategicCcompanies	5	1290.27	3	484.14	5	1152.00
TOTAL	83	9060.98	65	5684.87	86	6624.55

model (from `INTERFACE`) and a vector of features (from `FEATURE EXTRACTION`). It returns to `MANAGER` the name of the predicted engine.

`ENGINE MANAGER` manages the interaction with the engines. It receives from `MANAGER` information about the engine to fire. At the end of the engine computation, `ENGINE MANAGER` returns to `MANAGER` the result. Finally, `MANAGER` works as a coordinator of ME-ASP modules, and it also provides the final result to `INTERFACE`.

The engines of ME-ASP, as depicted in Figure 1 (the rightmost boxes) are five state-of-the-art ASP solvers, namely CLASP [6] and its disjunctive version CLASPD, CMODELS [11], DLV [10], and IDP [14]; nonetheless, the architecture of ME-ASP is modular and allows one to easily update the engines set with additional solvers. Finally note that engines are used as "black-boxes", i.e., ME-ASP interacts with them via system calls.

3 Performance at a glance

The experiments were carried out on CyberSAR, a cluster comprised of 50 Intel Xeon E5420 blades equipped with 64 bit Gnu Scientific Linux 5.5. The resources granted to the solvers are 600s of CPU time and 2GB of memory. Time measurements were carried out using the `time` command shipped with Gnu Scientific Linux 5.5.[2] In Table 1 we report the results of the ME-ASP version using Decision Trees as classifier in comparison with CLASPD – the winner of the "System Track" of the 3rd ASP Competition – and

[2] We remind that these are different hardware setting w.r.t. the 3rd ASP competition in both computer architecture and memory limits; importantly, the inputs were pre-grounded and saved in ASP-Core format.

the state of the art (SOTA) solver, i.e., considering a problem instance, the oracle that always fares the best among available solvers.

Looking at Table 1, we can see that ME-ASP solves 18 instances more than CLASPD. More, here it is very interesting to note that its performance is very close to the SOTA solver which, we remind, has the ideal performance that we could expect on these instances with these engines. More details and additional experimental data concerning ME-ASP settings (i.e., solver selection, program features, solver training, and classification algorithms) can be found in [13].

References

1. M. Balduccini *Learning and using domain-specific heuristics in ASP solvers*. AICOM, 24:147-164, 2011.
2. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Tempe, Arizona, 2003.
3. F. Calimeri, G. Ianni, F. Ricca, et al. The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track. In *Proc. of LPNMR11.*, pages 388–403, 2011. LNCS Springer.
4. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, September 1997.
5. M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. T. Schneider, and S. Ziller. A portfolio solver for answer set programming: Preliminary report. In *Proc. of LPNMR11* , LNCS, pages 352–357, Vancouver, Canada, 2011. Springer.
6. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proc. of IJCAI-07*, pages 386–392, 2007. Morgan Kaufmann Publishers.
7. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
8. A. Halder, A. Ghosh, and S. Ghosh. Aggregation pheromone density based pattern classification. *Fundamenta Informaticae*, 92(4):345–362, 2009.
9. T. Janhunen, I. Niemelä, and M. Sevalnev. Computing stable models via reductions to difference logic. In *Proc. of LPNMR 09*, LNCS, pages 142–154, 2009. Springer.
10. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM TOCL*, 7(3):499–562, 2006.
11. Y. Lierler. Disjunctive Answer Set Programming via Satisfiability. In *Proc. of LPNMR'05*, volume 3662 of *LNCS*, pages 447–451. Springer Verlag, September 2005.
12. V. Lifschitz. Answer Set Planning. In *Proc. of ICLP'99*, pages 23–37, Las Cruces, New Mexico, USA, November 1999. The MIT Press.
13. M. Maratea, L. Pulina, and F. Ricca. Applying machine learning techniques to ASP solving. Number CVL 2012/003, page 21. University of Sassari Tech. Rep., March 2012.
14. M. Mariën, J. Wittocx, M. Denecker, and M. Bruynooghe. Sat(id): Satisfiability of propositional logic extended with inductive definitions. In *Proc. of SAT 08*, LNCS, pages 211–224, 2008. Springer.
15. I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. Yale: Rapid prototyping for complex data mining tasks. In *Proc. of KDD '06* , pages 935–940. ACM, 2006.
16. L. Pulina and A. Tacchella. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, 14(1):80–116, 2009.
17. P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138:181–234, June 2002.
18. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.