Multi-Level Algorithm Selection for ASP

Marco Maratea[1], Luca Pulina[2], and Francesco Ricca[3]

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

Abstract. Automated algorithm selection techniques have been applied successfully to Answer Set Programming (ASP) solvers. ASP computation includes two levels of computation: variable substitution, called grounding, and propositional answer set search, called solving. In this paper we present ME-ASP^{ML}, an extended ASP system applying algorithm selection techniques to both levels of computation in order to choose the most promising solving strategy. Experiments conducted on benchmarks and solvers of the Fifth ASP Competition shows that ME-ASP^{ML} is able to solve more instances than state-of-the-art systems.

1 Introduction

Answer Set Programming (ASP) [8, 12, 13] is a declarative programming paradigm developed in the area of logic programming and non-monotonic reasoning. The evaluation of ASP programs usually includes two levels of computation, called grounding and solving. In the first level, a propositional program is obtained from a non-ground specification by applying intelligent techniques that eliminate variables; then, in the second level, the propositional program is fed to an ASP solver to produce answer sets.

Automated algorithm selection techniques have been applied in ASP to obtain efficient evaluation of programs. The idea is to select automatically the "best" computation strategy on the basis of features computed on a training set on instances. In the literature there are several different proposals, and among them we mention the portfolio solver CLASPFOLIO ver. 1 [10], which then evolved in a framework combining different approaches in CLASPFOLIO ver. 2 [15], the multi-engine approach implemented in ME-ASP [19], the techniques for learning heuristics orders presented in [3], and the work in [21, 14] that employ parameters tuning and/or design a solvers schedule. However, to the best of our knowledge, the application of automated selection techniques is typically limited to the evaluation of propositional programs, thus the choice of algorithms has been limited to the solving level. A preliminary contribution that exploits the features of non-ground programs was presented in [18], but also in this case the application was limited to only one level, namely the choice of the most promising grounding tool.

In this paper we present ME-ASP^{ML}, an extension of the multi-engine ASP system ME-ASP, that applies algorithm selection techniques to both levels of computation of

answer sets, with the goal of selecting the most promising computation strategy. ME-ASP^{ML} supports the new standard ASPCore 2.0 [5] and selects among the systems that participated to the Fifth ASP Competition [6].

The new architecture of ME-ASP^{ML} takes advantage from the extraction of syntactic features of non-ground programs in the first level, so to identify a number of classes of non-ground programs. Then, ME-ASP^{ML} (possibly) applies to each class identified in the first level an additional phase of algorithm selection, which exploits the features of ground programs measured after running the grounder GRINGO [11]. A key-enabler in achieving good performance in ME-ASP^{ML} is the extraction of cheap-to-compute features. These can be obtained at the price of a minimum overhead also in case of large input programs. The features employed in the first level are able to characterize a program w.r.t. the complexity class, and can even identify a class of programs where a specific grounder is the most promising (as done in [18]). The algorithm selection approach of ME-ASP [19] is then applied in the second level to the classes of programs identified in the first level, allowing for a more accurate selection of the solver to be employed, given that different classes of programs are usually characterized by different sets of meaningful features. Notably, the set of features used in the second level of ME-ASP^{ML} is a strict superset of the ones employed in ME-ASP [19], extended to deal with ASPCore 2.0 [5] programs.

An experimental analysis conducted on benchmarks and solvers of the Fifth ASP Competition shows that ME-ASP^{ML} is able to solve more instances than: (i) any solver that participated to the competition, (ii) the mere update of the (single-level) ME-ASP [19] system, and (iii) the state-of-the-art system CLASPFOLIO ver. 2.2. The results hold also considering separately each track of the competition. Such analysis, thus, suggests that the application of a multi-level algorithm selection strategy, also exploiting the features of non-ground programs, can lead to a performance that cannot be matched by any system applying algorithm selection only to propositional programs.

2 Architecture and Implementation

Architecture. Figure 1 presents the architecture of ME-ASP^{ML} (available for download at `www.mat.unical.it/ricca/downloads/measpml.tar.gz`). Looking at the figure, we can see that ME-ASP^{ML} is composed of six main modules. `NGFE` (Non-Ground Feature Extractor) aims at computing features from the input (non-ground) program that are "pragmatically" cheap-to-compute, such as the number of Head-Cycle Free components, presence of queries, and stratification property. Such features are passed to `NON-GROUND MANAGER`, that is devoted to identify the class of the input ASP program. `GROUNDER` takes as input the non-ground ASP program and returns the related grounded instance. The next module, namely `GFE` (Ground Feature Extractor), aims at computing the syntactic features of the input ground program. We used the features detailed in [19], with the addition of some ASPCore 2.0 specific features such as the number of choice rules, number of aggregates, and number of weak constraints. `GROUND MANAGER` is devoted to the prediction of the solver to run. It contains the inductive models related to the considered classes. Its working process can be divided in two steps, i.e., (i) given the input received by `NON-GROUND MANAGER`,

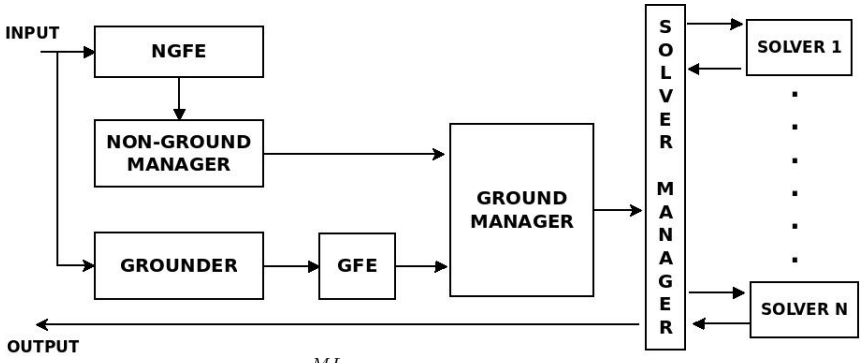

Fig. 1. The architecture of ME-ASP^{ML}. Solid boxes represent the modules, while arrows denote functional connections between them.

it selects the proper inductive model; and (ii) given the features computed in `GFE`, it outputs to `SOLVER MANAGER` the name of the predicted solver. Finally, `SOLVER MANAGER` manages the interaction with the engines. At the end of the engine computation, `SOLVER MANAGER` returns as output the result given by the solver.

Implementation. In ME-ASP^{ML}, algorithm selection is implemented by means of multinomial classification. In a few words, given a set of patterns, i.e., input vectors $X = \{\underline{x}_1, \ldots \underline{x}_k\}$ with $\underline{x}_i \in \mathbb{R}^n$, and a corresponding set of labels, i.e., output values $Y \in \{1, \ldots, m\}$, where Y is composed of values representing the m classes of the multinomial classification problem, in our modeling, the m classes are m ASP solvers. Given a set of patterns X and a corresponding set of labels Y, the task of a multinomial classifier c is to construct c from X and Y so that when we are given some $\underline{x}^\star \in X$ we should ensure that $c(\underline{x}^\star)$ is equal to $f(\underline{x}^\star)$. This task is called *training*, and the pair (X, Y) is called the *training set*. Concerning the training set, we selected instances and encodings involved in the Fifth ASP Competition [6]. The considered pool of benchmarks is composed of 26 domains organized into tracks, which are based on both complexity issues and language constructs of ASPCore 2.0. Starting from a total amount of 8572 instances, we pragmatically randomly split the amount of instances in each domain, using 50% of the total amount for training purpose, and the remaining ones for testing – the full list is available at `www.mat.unical.it/ricca/downloads/measpmlts.tar.gz`. Concerning the `NON-GROUND MANAGER` (see Figure 1), we used a list of if-then-else rules obtained running the PART decision list generator on the training instances. About the labels, we considered five program classes, namely the queries (Q) and ASP competition tracks names (in the following, for short, T_i, $i \in \{1, \ldots, 4\}$). Notice that this module does not select only the grounder as done in [18], but in principle the approach of [18] can be implemented in our architecture.
Considering `GROUNDER` module, it is actually implemented using GRINGO ver. 4. Regarding `GROUND MANAGER`, in the current version of ME-ASP^{ML} is composed of four different inductive models, i.e., models obtained training a classifier. Models are related to the program classes $T_1, \ldots, T_4$ and are computes using the training sets mentioned above. The pattern comprised in the training set is composed of the values related to

Solver	T_1	T_2	T_3	T_4	Q
CLASP [7]	✓	✓	✓	✓	–
LP2BV2+BOOLECTOR [20]		✓	–	–	–
LP2GRAPH [9]	✓		–	–	–
LP2MAXSAT+CLASP [4]		✓		–	–
LP2NORMAL2+CLASP [4]	✓	✓	✓	✓	–
LP2SAT3+GLUCOSE [16]			–	–	–
LP2SAT3+LINGELING [16]			–	–	–
WASP1 [1]			✓		
WASP1.5 [1]					✓
WASP2 [1]	✓	✓		–	–

Table 1. Considered ASP solvers that entered the Single Processor category of the Fifth ASP Competition. The first column contains the solvers, while the remaining four columns are related to the inductive models. A "✓" indicates that the solver has been selected as ME-ASP^{ML} engine. An empty cell means that the related solver has been evaluated but it is not comprised in ME-ASP^{ML}. Finally, a "–" indicates that the related solver can not compete on the program class.

Solver	Track 1		Track 2		Track 3		Track 4	
	#	Time	#	Time	#	Time	#	Time
CLASP	362	12318	1241	41049	154	4578	503	8078
LP2BV2+BOOLECTOR	205	19396	822	43124	–	–	–	–
LP2GRAPH	324	23592	1030	50341	–	–	–	–
LP2MAXSAT+CLASP	264	18537	1066	60185	74	5548	–	–
LP2NORMAL2+CLASP	342	18263	1252	60031	119	9379	496	12921
LP2SAT3+GLUCOSE	278	25149	1027	50170	–	–	–	–
LP2SAT3+LINGELING	256	23682	1108	80465	–	–	–	–
WASP1	268	15155	719	52260	88	3558	238	18951
WASP1.5	242	3782	1042	32159	23	754	238	19285
WASP2	317	13622	1146	41140	24	750	–	–
ME-ASP^{ML}	376	15391	1341	46143	231	5632	532	8960

Table 2. Results of the experiments. The first column contains the various solvers considered, plus ME-ASP^{ML}. The remaining four columns contain the results for Track 1 to Track 4. Each of these columns is then divided into two subcolumns, containing the number of solved instances within the time limit, and the sum of their CPU times in seconds, respectively. If, for a track, both sub-columns contain "–", this means that the related solver can not compete on the track.

the same features computed in GFE, while the label corresponds to the best solver – in terms of CPU time – on the given instance. In Table 1 we show the solvers that could be used as engine of ME-ASP^{ML}.[4] Considering that using all the eleven solvers altogether rises the chance of getting a bad prediction because of aliasing, for each T_i we chosen different subsets of them as follows. First, we computed the total amount of training instances solved by the state-of-the-art solver (SOTA) i.e., given an instance, the oracle that always fares the best among all the solvers. Second, we calculated the minimum number of solvers such that the total amount of instances solved by the pool is at least 90% of the SOTA solver on the training instances.

Looking at Table 1, we can see the results of this process, as well as the involved solver considering each T_i. Notice that in the case of Q we had only one "label", namely

[4] We have not considered LP2MIP2 given that we did not receive the license of CPLEX on time.

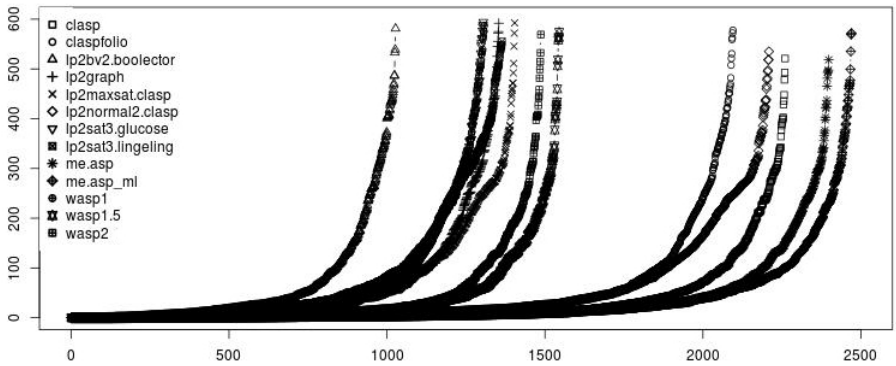

Fig. 2. Results of the solvers in Table 2, plus ME-ASP and CLASPFOLIO ver. 2.2, showed with a cactus plot. In the x-axis it is shown the total amount of solved instances, while y-axis reports the CPU time in seconds.

WASP1.5, which internally calls DLV with magic sets. Finally, the multinomial classification algorithm employed was k-Nearest Neighbors.

3 Experiments and Conclusion

We assessed the performance of ME-ASP^{ML} on the Fifth ASP Competition benchmarks. 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 results of the analysis are presented in Table 2. We first note that ME-ASP^{ML} can solve more instances than all its engines in all tracks, followed by CLASP in Tracks 1, 3 and 4, and by LP2NORMAL+CLASP in Track 2. In sum, ME-ASP^{ML} solves 2480 instances, while the second overall best, which is CLASP, solves a total of 2260 instances.

An aggregate picture of the performance of competing systems is presented in the cactus plot of Figure 2. This plot also includes ME-ASP and CLASPFOLIO ver. 2.2 [5] for a direct comparison with approaches of algorithm selection that only exploit ground features. From the figure we can see that ME-ASP^{ML} solves more instances also in comparison with its previous version ME-ASP and the state-of-the-art system CLASPFOLIO ver. 2.2, other than all its engines. In particular, CLASPFOLIO solves 358, 1148, 118 e 471 instances on the four tracks, respectively. We can note that, consistently with the information provided in the CLASPFOLIO web page, CLASPFOLIO performance are not optimized on Track 3: indeed, this is the track where it shows the biggest performance gap (as percentage of solved instances) w.r.t. ME-ASP^{ML}.

To sum up, the extended approach implemented in ME-ASP^{ML}, which applies algorithm selection to both levels of computation, performs very well, being able to solve more instances than (i) its engines, (ii) its previous version ME-ASP, and (iii) CLASPFOLIO ver. 2.2, in all tracks of the Fifth ASP Competition.

[5] CLASPFOLIO has been run with its default setting, and with CLASP ver. 3 as a back-end solver. This improved version has been provided by Marius Lindauer, who is thanked.

References

1. Alviano, M., Dodaro, C., Faber, W., Leone, N., Ricca, F.: WASP: A native ASP solver based on constraint learning. In: Cabalar, P., Son, T. (eds.) Proc. of LPNMR 2013. LNCS, vol. 8148, pp. 54–66. Verlag (2013)
2. Alviano, M., Faber, W., Greco, G., Leone, N.: Magic sets for disjunctive datalog programs. Artificial Intellegence 187, 156–192 (2012)
3. Balduccini, M.: Learning and using domain-specific heuristics in ASP solvers. AI Communications 24(2), 147–164 (2011)
4. Bomanson, J., Janhunen, T.: Normalizing cardinality rules using merging and sorting constructions. In Proc. of LPNMR 2013. LNCS, vol. 8148, pp. 187–199. Springer (2013)
5. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: Asp-core-2 input language format (since 2013), https://www.mat.unical.it/aspcomp2013/ASPStandardization
6. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: The design of the fifth answer set programming competition. ICLP 2014 TC abs/1405.3710 (2014), http://arxiv.org/abs/1405.3710
7. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-Driven Disjunctive Answer Set Solving. In Proc. of KR 2008. pp. 422–432. AAAI Press (2008)
8. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM TODS 22(3), 364–418 (Sep 1997)
9. Gebser, M., Janhunen, T., Rintanen, J.: Answer set programming as sat modulo acyclicity. In Proc. of ECAI 2014. FAIA vol. 263, pp. 351–356. IOS Press (2014)
10. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M.T., Ziller, S.: A portfolio solver for answer set programming: Preliminary report. In Proc. of LPNMR 2011. LNCS, vol. 6645, pp. 352–357 (2011)
11. Gebser, M., Schaub, T., Thiele, S.: GrinGo : A New Grounder for Answer Set Programming. In Proc. of LPNMR 2007. vol. 4483, pp. 266–271 (2007)
12. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Logic Programming: Proceedings Fifth Intl Conference and Symposium. pp. 1070–1080. MIT Press, Cambridge, Mass. (1988)
13. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. NGC 9, 365–385 (1991)
14. Hoos, H., Kaminski, R., Schaub, T., Schneider, M.T.: ASPeed: Asp-based solver scheduling. In Proc. of ICLP 2012. LIPIcs, vol. 17, pp. 176–187.
15. Hoos, H., Lindauer, M.T., Schaub, T.: claspfolio 2: Advances in algorithm selection for answer set programming. TPLP 14(4-5), 569–585 (2014)
16. Janhunen, T.: Some (in)translatability results for normal logic programs and propositional theories. Journal of Applied Non-Classical Logics 16, 35–86 (2006)
17. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL 7(3), 499–562 (Jul 2006)
18. Maratea, M., Pulina, L., Ricca, F.: Automated selection of grounding algorithm in answer set programming. In Proc. of AI*IA 2013. LNCS vol. 8249, pp. 73–84. (2013)
19. Maratea, M., Pulina, L., Ricca, F.: A multi-engine approach to answer-set programming. TPLP 14(6), 841–868 (2014), http://dx.doi.org/10.1017/S1471068413000094
20. Nguyen, M., Janhunen, T., Niemelä, I.: Translating answer-set programs into bit-vector logic. In Proc. of INAP/WLP 2011 Revised Selected Papers. LNCS, vol. 7773, pp. 105–116. (2011)
21. Silverthorn, B., Lierler, Y., Schneider, M.: Surviving solver sensitivity: An asp practitioner's guide. In ICLP 2012. LIPIcs, vol. 17, pp. 164–175.