$\mathcal{I}$-DLV+$\mathcal{MS}$: preliminary report on an automatic ASP solver selector.

Francesco Calimeri[1,2]✉, Davide Fuscà[1], Simona Perri[1], and Jessica Zangari[1]

[1] Department of Mathematics and Computer Science, University of Calabria, Italy
{calimeri,fusca, perri,zangari}@mat.unical.it
[2] DLVSystem Srl, Italy
calimeri@dlvsystem.com

Abstract. Current ASP solvers feature diverse optimization techniques that highly influence their performance, causing systems to outperform each other depending on the domain at hand. We present $\mathcal{I}$-DLV+$\mathcal{MS}$, a new ASP system that integrates an efficient grounder, namely $\mathcal{I}$-DLV, with an automatic solver selector: machine-learning techniques are applied to inductively choose the best solver, depending on some inherent features of the instantiation produced by $\mathcal{I}$-DLV. In particular, we define a specific set of features, and build our classification method for selecting the solver that is supposed to be the "best" for each input among the two state-of-the-art solvers *clasp* and *wasp*. Despite its prototypical stage, performance of the new system on benchmarks from the 6*th* ASP Competition are encouraging both against the state-of-the-art ASP systems and the best established multi-engine ASP system, ME-ASP.

Keywords: Knowledge Representation and Reasoning, Answer Set Programming, DLV, Artificial Intelligence, Deductive Database Systems, DLV, Grounding, Instantiation

1 Introduction

Answer Set Programming (ASP) [3, 16] is a declarative programming paradigm proposed in the area of non-monotonic reasoning and logic programming. The language of ASP is based on rules, allowing (in general) for both disjunction in rule heads and nonmonotonic negation in the body; such programs are interpreted according to the answer set semantics [15, 17]. Throughout the years the availability of reliable, high-performance implementations [7, 13] made ASP a powerful tool for developing advanced applications in many research areas, ranging from Artificial Intelligence to Databases and Bioinformatics, as well as in industrial contexts [7, 19, 22, 28, 30].

Although the performance of current ASP systems can be, in general, defined as good enough for a number of real-world applications, they feature several different optimization techniques, thus causing systems to outperform each other depending on the domain at hand. This is due to the different data structures, input simplification and heuristic implemented in the ASP solvers that allow

suiting well depending on the characteristic of the domains, as reported in the results of the last published ASP competition [8].

The capability to enjoy good performance over different problems domains has already been pursued by neighbour communities, by means of proper strategies of *algorithm selection* [29]; we cite here, for instance, what has been done for solving propositional satisfiability (SAT) [31] and Quantified SAT (QSAT) [25]. This approach consists of building machine learning techniques to inductively choose the "best" solver on the basis of some input program characteristics, or *features*. As for what ASP is concerned, some interesting works in this respect have already been carried out in [21]; furthermore, in [11] similar strategies have been used in order to select the "best" configuration for the solver *clasp*.

In this paper we present $\mathcal{I}$-DLV+$\mathcal{MS}$, a new ASP system that integrates an efficient grounder, namely $\mathcal{I}$-DLV [5], with an automatic solver selector: machine-learning techniques are applied to inductively choose the best solver, depending on some inherent features of the instantiation produced by $\mathcal{I}$-DLV.

We define a specific set of features, and then and carry out an experimental analysis for computing them over the ground versions of all benchmarks submitted to the 6th ASP Competition [12]; we build then our classification method for selecting the solver that is supposed to be the "best" for each input among the two state-of-the-art solvers *clasp* [10] and *wasp* [1]. Furthermore, we test $\mathcal{I}$-DLV+$\mathcal{MS}$ performance both against the state-of-the-art ASP systems and the best established multi-engine ASP system ME-ASP [21], that is the winner of the 6th ASP Competition. The tests prove that $\mathcal{I}$-DLV+$\mathcal{MS}$, even though still at a prototypical stage, already shows good performance.

Notably, $\mathcal{I}$-DLV+$\mathcal{MS}$ participated in the latest ($7th$) ASP competition [14], resulting as the winner in the regular track, category SP (i.e., one processor allowed).

In the remainder of the paper we introduce $\mathcal{I}$-DLV+$\mathcal{MS}$ providing the reader with an overview of the system, and we then describe the proposed classification method along with the selected features it relies on. We discuss a thorough experimental activity afterwards, and eventually draw our conclusions.

2 Answer Set Programming

We briefly recall here syntax and semantics of Answer Set Programming.

2.1 Syntax

A variable or a constant is a *term*. An *atom* is $a(t_1, \ldots, t_n)$, where a is a *predicate* of arity n and $t_1, \ldots, t_n$ are terms. A *literal* is either a *positive literal* p or a *negative literal not* p, where p is an atom. A *disjunctive rule* (*rule*, for short) r is a formula

$$a_1 \mid \cdots \mid a_n \text{ :- } b_1, \cdots, b_k, \ not \ b_{k+1}, \cdots, \ not \ b_m.$$

where $a_1, \cdots, a_n, b_1, \cdots, b_m$ are atoms and $n \geq 0$, $m \geq k \geq 0$. The disjunction $a_1 \mid \cdots \mid a_n$ is the *head* of r, while the conjunction b_1, ..., b_k, *not* b_{k+1}, ..., *not* b_m is the *body* of r. A rule without head literals (i.e. $n = 0$) is usually referred to as an *integrity constraint*. If the body is empty (i.e. $k = m = 0$), it is called a *fact*.

$H(r)$ denotes the set $\{a_1, ..., a_n\}$ of head atoms, and by $B(r)$ the set $\{b_1, ..., b_k, not\ b_{k+1}, \ldots, not\ b_m\}$ of body literals. $B^+(r)$ (resp., $B^-(r)$) denotes the set of atoms occurring positively (resp., negatively) in $B(r)$. A rule r is *safe* if each variable appearing in r appears also in some positive body literal of r.

An *ASP program* P is a finite set of safe rules. An atom, a literal, a rule, or a program is *ground* if no variables appear in it. Accordingly with the database terminology, a predicate occurring only in *facts* is referred to as an *EDB* predicate, all others as *IDB* predicates; the set of facts of P is denoted by $EDB(P)$.

2.2 Semantics

Let P be a program. The *Herbrand Universe* of P, denoted by U_P, is the set of all constant symbols appearing in P. The *Herbrand Base* of a program P, denoted by B_P, is the set of all literals that can be constructed from the predicate symbols appearing in P and the constant symbols in U_P.

Given a rule r occurring in P, a *ground instance* of r is a rule obtained from r by replacing every variable X in r by $\sigma(X)$, where σ is a substitution mapping the variables occurring in r to constants in U_P; $ground(P)$ denotes the set of all the ground instances of the rules occurring in P.

An *interpretation* for P is a set of ground atoms, that is, an interpretation is a subset I of B_P. A ground positive literal A is *true* (resp., *false*) w.r.t. I if $A \in I$ (resp., $A \notin I$). A ground negative literal *not* A is *true* w.r.t. I if A is false w.r.t. I; otherwise *not* A is false w.r.t. I. Let r be a ground rule in $ground(P)$. The head of r is *true* w.r.t. I if $H(r) \cap I \neq \emptyset$. The body of r is *true* w.r.t. I if all body literals of r are true w.r.t. I (i.e., $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$) and is *false* w.r.t. I otherwise. The rule r is *satisfied* (or *true*) w.r.t. I if its head is true w.r.t. I or its body is false w.r.t. I. A *model* for P is an interpretation M for P such that every rule $r \in ground(P)$ is true w.r.t. M. A model M for P is *minimal* if no model N for P exists such that N is a proper subset of M. The set of all minimal models for P is denoted by $\mathrm{MM}(P)$.

Given a ground program P and an interpretation I, the *reduct* of P w.r.t. I is the subset P^I of P, which is obtained from P by deleting rules in which a body literal is false w.r.t. I. Note that the above definition of reduct, proposed in [9], simplifies the original definition of Gelfond-Lifschitz (GL) transform [17], but is fully equivalent to the GL transform for the definition of answer sets [9].

Let I be an interpretation for a program P. I is an *answer set* (or stable model) for P if $I \in \mathrm{MM}(P^I)$ (i.e., I is a minimal model for the program P^I) [24, 17]. The set of all answer sets for P is denoted by $ANS(P)$.

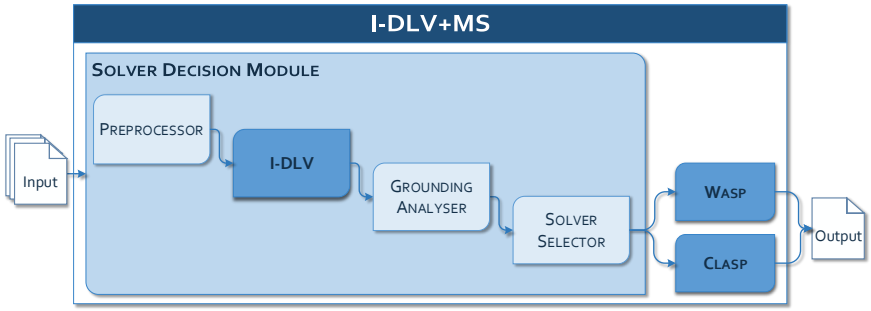

Fig. 1. $\mathcal{I}$-DLV+$\mathcal{MS}$ Architecture.

3 $\mathcal{I}$-DLV+$\mathcal{MS}$ Overview

The architecture of $\mathcal{I}$-DLV+$\mathcal{MS}$ is reported in Figure 1. The Preprocessor module analyzes the input program P, and interacts with the $\mathcal{I}$-DLV system in order to determine if the input program is non-disjunctive and stratified, as these kinds of programs are completely evaluated by $\mathcal{I}$-DLV without the need for a solver. If this is not the case, the Grounding analyser module extracts the intended features from the ground program produced by $\mathcal{I}$-DLV and passes them to the classification module. Then, the Solver Selector, based on proper classification algorithms, tries to foresee, among the available solvers, which one would perform better, and selects it. This module is based on *Decision Trees*, a non-parametric supervised learning method used for classification [26]; this classifier aims at creating a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. We use an optimized version of tree algorithm implemented in *scikit-learn* library [23], namely *CART* (Classification and Regression Trees) [2]. The algorithm is a variant of C4.5 [27], and differs from it as it supports numerical target variables and does not compute rule sets.

$\mathcal{I}$-DLV+$\mathcal{MS}$ currently supports the two state-of-the-art ASP solvers *clasp* and *wasp*. Nonetheless, the modular architecture of $\mathcal{I}$-DLV+$\mathcal{MS}$ easily allows one to update the solvers or even add additional ones. Clearly, such changes would require the prediction model to be retrained with appropriate statistics on the new solvers.

$\mathcal{I}$-DLV+$\mathcal{MS}$ is freely available at [6].

3.1 Features

As already introduced, the machine-learning technique herein employed selects the best solver according to some specific features of the input program. In this work, we selected several features with the aim of catching two fundamental aspects of ASP programs:

– *Atoms ratios.* We considered five different ratios that represent the type of atoms and a raw measure of their distribution in the input ground program:

$$(a) : \frac{F}{R} \qquad (b) : \frac{PA}{R} \qquad (c) : \frac{NA}{R} \qquad (d) : \frac{PA}{BA} \qquad (e) : \frac{NA}{BA}$$

where F is the total number of facts and always true atoms, R the total number of ground rules, PA/NA the total number of positive/negative atoms and BA the total number of atoms appearing in rule bodies.

– *Rules ratios.* We considered five different ratios that represent the type of rules and a raw measure of their distribution in the input ground program, taking into account also advanced constructs of the ASP-Core-2 standard language [4], such as choices, aggregates, and weak constraints:

$$(f) : \frac{C}{R} \qquad (g) : \frac{W}{R} \qquad (h) : \frac{SR}{R} \qquad (i) : \frac{CR}{R} \qquad (j) : \frac{WR}{R}$$

where C is the total number of strong constraints, W the total number of weak constraints, SR the total number of standard rules, CR the total number of choice rules and WR the total number of weight rules. Please note that with standard rules we denote rules without aggregate or choice atoms; weight and choice rules, instead, handle aggregate literals and choice atoms generated by the grounder.

4 Experimental Evaluation

In this section we report the results of an experimental activity performed in order to evaluate the approach implemented into $\mathcal{I}$-DLV+$\mathcal{MS}$. In particular, we performed two distinct sets of experiments, that are discussed in the following.

Experiments have been performed on a NUMA machine equipped with two `2.8GHz AMD Opteron 6320` and `128 GiB` of main memory, running `Linux Ubuntu 14.04.4` . Binaries were generated with the `GNU C++ compiler 4.9.0`. As for memory and time limits, we allotted `15 GiB` and `600` seconds for each system per each single run.

4.1 Efficacy of Solver Selection

With the first set of experiments, we aimed at preliminarly assessing the machine-learning-based model for solver selection, i.e., checking the quality of the choice made for each input. To this end, we took the benchmarks from the latest available ASP Competition [12] and ran $\mathcal{I}$-DLV+$\mathcal{MS}$ along with two distinct combinations of the $\mathcal{I}$-DLV grounder with the latest available versions, at the time of writing, of *clasp* and *wasp* solvers, respectively: *clasp* version 3.2.1 and *wasp* version `2017-05-04`.

Basically, this allows us to easily compare the performance of the solver chosen by $\mathcal{I}$-DLV+$\mathcal{MS}$ against the best one for each set of benchmarks. A system equipped with a perfect selector would ideally match the performance of the best solver for each benchmark, net of possible overheads.

Results are reported in Table 1: first column shows the name of the benchmark, while the next pairs report the solved instances and average times per each tested system (each benchmark featured 20 instances). The best performing combination among $\mathcal{I}$-DLV +*clasp* and $\mathcal{I}$-DLV +*wasp* is highlighted in bold, on a benchmark basis. Results show that $\mathcal{I}$-DLV+$\mathcal{MS}$ performance is very close to the best solver; in particular, the system show the same performance of the best solver in 17 domains out of 24, suggesting that the defined measures, along with the chosen model, lead to good choices. Furthermore, the total number of instances solved by $\mathcal{I}$-DLV+$\mathcal{MS}$ is 327, while for $\mathcal{I}$-DLV +*clasp* and $\mathcal{I}$-DLV +*wasp* is 292 and 256, respectively.

Table 1. $\mathcal{I}$-DLV+$\mathcal{MS}$, $\mathcal{I}$-DLV +*clasp* and $\mathcal{I}$-DLV +*wasp*: number of solved instances and average running times (in seconds) on benchmarks from the 6th ASP Competition (20 instances per problem). In bold is outlined the top-performing system among $\mathcal{I}$-DLV +*clasp* and $\mathcal{I}$-DLV +*wasp*.

Benchmark	$\mathcal{I}$-DLV+$\mathcal{MS}$		$\mathcal{I}$-DLV +*clasp*		$\mathcal{I}$-DLV +*wasp*	
	solved	time	solved	time	solved	time
AbstractDialecticalFrameworks	20	10,91	**20**	**7,17**	14	80,38
CombinedConfiguration	13	103,83	**12**	**108,78**	6	52,93
ComplexOptimizationOfAnswerSets	19	127,68	**19**	**105,12**	5	127,58
ConnectedMaximim-densityStillLife	10	86,02	5	283,53	**8**	**120,60**
CrossingMinimization	19	2,46	6	91,88	**19**	**0,65**
GracefulGraphs	9	122,36	**10**	**59,77**	5	22,12
GraphColouring	16	104,05	**16**	**117,28**	6	137,67
IncrementalScheduling	11	42,09	**11**	**35,74**	8	113,75
KnightTourWithHoles	14	29,95	**14**	**25,87**	10	34,76
Labyrinth	10	134,82	**11**	**78,49**	11	134,06
MaximalCliqueProblem	14	49,06	0	-	**15**	**143,18**
MaxSAT	19	17,55	8	47,70	**20**	**50,91**
MinimalDiagnosis	20	27,55	**20**	**14,50**	20	24,86
Nomistery	8	26,21	**8**	**25,26**	8	37,09
PartnerUnits	14	23,29	**14**	**45,03**	8	210,47
PermutationPatternMatching	20	117,40	**20**	**16,86**	20	129,84
QualitativeSpatialReasoning	20	106,06	**20**	**113,75**	12	124,32
RicochetRobots	10	95,43	**12**	**130,96**	7	166,09
Sokoban	8	343,63	9	23,71	**10**	**153,95**
StableMarriage	6	111,30	**9**	**268,98**	7	397,86
SteinerTree	8	14,60	**8**	**84,82**	4	0,15
ValvesLocationProblem	16	6,99	**16**	**12,10**	16	37,04
VideoStreaming	15	18,89	**15**	**5,63**	9	1,94
Visit-all	8	162,88	**9**	**57,18**	8	69,74

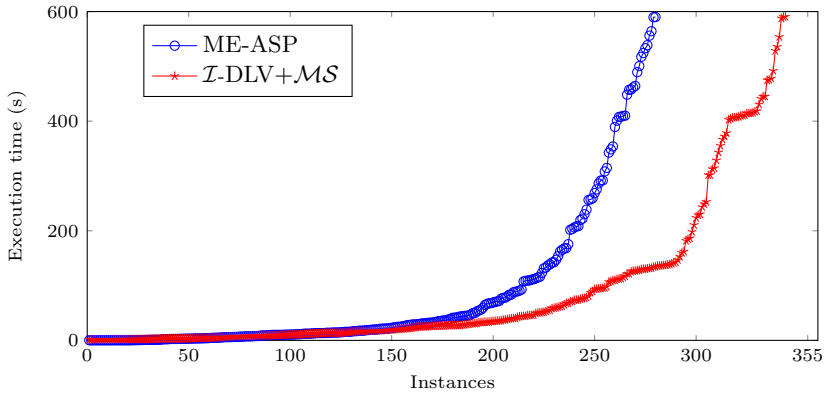

Fig. 2. $\mathcal{I}$-DLV+$\mathcal{MS}$ and ME-ASP comparison on benchmarks from the 6th ASP Competition.

It is worth noting that, even when the choice was right, $\mathcal{I}$-DLV+$\mathcal{MS}$ pays some overhead. This is due to the fact that the ground program produced by $\mathcal{I}$-DLV is not directly passed to the solver; rather, it must be analyzed in order to extract the features needed for making the choice: intuitively, huge ground programs might hence cause a loss of performance during the features extraction.

4.2 Comparison to the State of the Art

In the second set of experiments we compared $\mathcal{I}$-DLV+$\mathcal{MS}$ against the latest available version of ME-ASP[3]), the state-of-the-art among multi-engine ASP solvers, resulted as the winner of the 6th ASP Competition. Differently from $\mathcal{I}$-DLV+$\mathcal{MS}$, ME-ASP employs machine-learning techniques to inductively choose the best ASP solver on a per-instance basis.

Figure 2 reports the cactus plot comparing $\mathcal{I}$-DLV+$\mathcal{MS}$ and ME-ASP. Interestingly, even though $\mathcal{I}$-DLV+$\mathcal{MS}$ is a prototypical system, on the overall, it showed encouraging performance: indeed, it solved 327 instances, 49 more instances with respect to what ME-ASP did.

The two systems are similar, yet they feature some key differences; in particular, the main differences are due to the nature and the number of systems used. First of all, ME-ASP computes features of the input program at hand over a ground program produced by *gringo* system, while $\mathcal{I}$-DLV+$\mathcal{MS}$ makes use of $\mathcal{I}$-DLV. Furthermore, the main interesting difference is due to the fact that ME-ASP manages five solvers, way more than the mere two taken into account by $\mathcal{I}$-DLV+$\mathcal{MS}$. On the one hand, the strategy of using a large pool of solver engines, as in the case of ME-ASP, allows to solve a significant number of instances uniquely, i.e., instances solved by only one solver, as the different engines use evaluation strategies that can be substantially different; nevertheless, such differences imply that a high price is paid in case of a wrong choice. On the other hand, when the space for choices is narrowed, the probability of picking

[3] http://aspcomp2015.dibris.unige.it/participants

the wrong solver decreases, and this might lead to a more consistent behaviour, as in the case of $\mathcal{I}$-DLV+$\mathcal{MS}$.

5 Ongoing Works

Notwithstanding the good performance of $\mathcal{I}$-DLV+$\mathcal{MS}$, the system is still in a prototype phase. As future work, we aim to test additional supervised learning method and also several frameworks for automatic algorithm configuration, like Autofolio [20] or Auto-WEKA [18]. We also plan to significantly extend experiments over additional domains and analyze the possible overfitting of the model and try different splits of the dataset for the train and test set among the available problems. Moreover, we aim to both include additional ASP solvers with different parameterizations, and explore more features for improving the classification capabilities and achieve better overall performance.

In addition, we are studying the possibility of taking advantage from machine-learning techniques for improving performance of ASP grounding engines; in particular, we plan to develop a built-in automatic algorithm selector within the $\mathcal{I}$-DLV system (which $\mathcal{I}$-DLV+$\mathcal{MS}$ is based on), thus opening up the possibility to dynamically adapt all the optimization strategies to the problem at hand.

6 Conclusions

In this work, we made use of machine-learning techniques in the ASP solving scenario, with the aim of designing and producing an efficient automatic ASP solver selector. To this end, we first selected syntactic features that represent specific characteristics of ground ASP programs, in order to be able to perform accurate classifications. Then, we trained the model and performed an extensive experimental evaluation on problems from the 6th ASP Competition. Experiments are very promising, as $\mathcal{I}$-DLV+$\mathcal{MS}$ showed very good performance despite its prototypical nature. Indeed, our system outperforms both its ASP component solvers and the winner of the 6th ASP Competition. Such good performance is confirmed by the success in the latest ($7th$) ASP competition [14], where the system won the regular track in the SP category.

References

1. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: LPNMR. LNCS, vol. 9345, pp. 40–54. Springer (2015)
2. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA (1984)
3. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM 54(12), 92–103 (2011)
4. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2: 4th ASP Competition Official Input Language Format (2013), https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf
5. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: I-DLV: the new intelligent grounder of DLV. Intelligenza Artificiale 11(1), 5–20 (2017), https://doi.org/10.3233/IA-170104

6. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: I-DLV-MS (since 2017), {https://github.com/DeMaCS-UNICAL/IDLV-MS}
7. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the fifth answer set programming competition. Artificial Intelligence 231, 151–181 (2016), http://dx.doi.org/10.1016/j.artint.2015.09.008
8. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the fifth answer set programming competition. Artificial Intelligence 231, 151–181 (2016)
9. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) Proceedings of the 9th European Conference on Artificial Intelligence (JELIA 2004). Lecture Notes in AI (LNAI), vol. 3229, pp. 200–212. Springer Verlag (Sep 2004)
10. Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., Schaub, T.: Progress in clasp series 3. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings. LNCS, vol. 9345, pp. 368–383. Springer (2015), http://dx.doi.org/10.1007/978-3-319-23264-5_31
11. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M.T., Ziller, S.: A portfolio solver for answer set programming: Preliminary report. In: Delgrande, J.P., Faber, W. (eds.) Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6645, pp. 352–357. Springer (2011), https://doi.org/10.1007/978-3-642-20895-9_40
12. Gebser, M., Maratea, M., Ricca, F.: The design of the sixth answer set programming competition – report. In: LPNMR. LNCS, vol. 9345, pp. 531–544 (2015), http://dx.doi.org/10.1007/978-3-319-23264-5_44
13. Gebser, M., Maratea, M., Ricca, F.: What's hot in the answer set programming competition. In: Schuurmans, D., Wellman, M.P. (eds.) Proc. of the 13th AAAI Conference on Artificial Intelligence, Feb 12-17, 2016, Phoenix, Arizona, USA. pp. 4327–4329. AAAI Press (2016), http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12233
14. Gebser, M., Maratea, M., Ricca, F.: The design of the seventh answer set programming competition. In: Balduccini, M., Janhunen, T. (eds.) Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10377, pp. 3–9. Springer (2017), https://doi.org/10.1007/978-3-319-61660-5_1
15. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, WA, Aug 15-19, 1988 (2 Volumes). pp. 1070–1080. MIT Press, Cambridge, Mass. (1988)
16. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
17. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9(3/4), 365–385 (1991), http://dx.doi.org/10.1007/BF03037169
18. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. Journal of Machine Learning Research 17, 1–5 (2016)
19. Leone, N., Ricca, F.: Answer set programming: A tour from the basics to advanced development tools and industrial applications. In: Reasoning Web. LNCS, vol. 9203, pp. 308–326. Springer (2015)

20. Lindauer, M., Hoos, H.H., Hutter, F., Schaub, T.: Autofolio: An automatically configured algorithm selector. Journal of Artificial Intelligence Research 53, 745–778 (2015)
21. Maratea, M., Pulina, L., Ricca, F.: A multi-engine approach to answer-set programming. TPLP 14(6), 841–868 (2014), https://doi.org/10.1017/S1471068413000094
22. Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., Barry, M.: An A-Prolog Decision Support System for the Space Shuttle. In: Ramakrishnan, I. (ed.) Practical Aspects of Declarative Languages, Third International Symposium (PADL 2001). Lecture Notes in Computer Science, vol. 1990, pp. 169–183. Springer (2001)
23. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. Journal of Machine Learning Research 12(Oct), 2825–2830 (2011)
24. Przymusinski, T.C.: Stable Semantics for Disjunctive Programs. New Generation Computing 9, 401–424 (1991)
25. Pulina, L., Tacchella, A.: A multi-engine solver for quantified boolean formulas. In: Bessiere, C. (ed.) Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4741, pp. 574–589. Springer (2007), https://doi.org/10.1007/978-3-540-74970-7_41
26. Quinlan, J.R.: Induction of decision trees. Machine learning 1(1), 81–106 (1986)
27. Quinlan, J.R.: C4. 5: programs for machine learning. Elsevier (2014)
28. Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., Leone, N.: Team-building with answer set programming in the gioia-tauro seaport. Theory and Practice of Logic Programming. *Cambridge University Press* 12(3), 361–381 (2012)
29. Rice, J.R.: The algorithm selection problem. Advances in Computers 15, 65–118 (1976), https://doi.org/10.1016/S0065-2458(08)60520-3
30. Tiihonen, J., Soininen, T., Niemelä, I., Sulonen, R.: A practical tool for mass-customising configurable products. In: Proceedings of the 14th International Conference on Engineering Design (ICED'03). pp. 1290–1299 (2003)
31. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: Portfolio-based algorithm selection for sat. CoRR abs/1111.2249 (2011), http://dblp.uni-trier.de/db/journals/corr/corr1111.html#abs-1111-2249