TPLP **20** (2): 205–224, 2020. © Cambridge University Press 2018
doi:10.1017/S1471068418000546 First published online 4 December 2018

Technical Note
Efficiently Coupling the I-DLV Grounder with ASP Solvers

FRANCESCO CALIMERI[1], CARMINE DODARO[2], DAVIDE FUSCÀ[1], SIMONA PERRI[1] and JESSICA ZANGARI[1]
[1]*Department of Mathematics and Computer Science, University of Calabria, Rende, Italy*
(*e-mails:* calimeri@mat.unical.it, fusca@mat.unical.it, perri@mat.unical.it, zangari@mat.unical.it)
[2]*Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genova, Genova, Italy*
(*e-mail:* dodaro@dibris.unige.it)

submitted 24 March 2018; revised 4 October 2018; accepted 4 November 2018

Abstract

We present $\mathscr{I}$-DLV+$\mathscr{MS}$, a new answer set programming (ASP) system that integrates an efficient grounder, namely $\mathscr{I}$-DLV, with an automatic selector that inductively chooses a solver: depending on some inherent features of the instantiation produced by $\mathscr{I}$-DLV, machine learning techniques guide the selection of the most appropriate solver. The system participated in the latest (7th) ASP competition, winning the regular track, category *SP* (i.e., one processor allowed).

KEYWORDS: knowledge representation and nonmonotonic reasoning, constraints, technical notes and rapid communications

1 Introduction

Answer set programming (ASP) (Brewka *et al.* 2011; Gelfond and Lifschitz 1991) is a declarative programming paradigm proposed in the area of nonmonotonic 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 bodies; such programs are interpreted according to the answer set semantics (Gelfond and Lifschitz 1988; Gelfond and Lifschitz 1991). Throughout the years the availability of reliable, high-performance implementations (Calimeri *et al.* 2016; Gebser *et al.* 2016) 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 (Calimeri *et al.* 2016; Leone and Ricca 2015; Nogueira *et al.* 2001; Dodaro *et al.* 2015; Ricca *et al.* 2012; Tiihonen *et al.* 2003; Grasso *et al.* 2009; Dodaro *et al.* 2016).

The "traditional" approach to the evaluation of ASP programs relies on a grounding module (*grounder*) that generates a propositional theory semantically equivalent to the input program coupled with a subsequent module (*solver*) that uses propositional techniques for generating answer sets. There have been other attempts deviating from

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

this customary approach (Dal Palù *et al.* 2009; Lefèvre *et al.* 2017; Dao-Tran *et al.* 2012; Weinzierl 2017); nonetheless, the majority of the current solutions relies on the canonical "ground & solve" strategy, including DLV (Alviano *et al.* 2017; Leone *et al.* 2006) and CLINGO (Gebser *et al.* 2015; Gebser *et al.* 2011).

While a significant amount of solvers has been released in the latest years, as confirmed by the most recent editions of the ASP competitions (Calimeri *et al.* 2016; Gebser *et al.* 2017a; Gebser *et al.* 2017b), until recently, the alternatives for grounding basically consisted of GRINGO (Gebser *et al.* 2011) and the grounding module of the DLV system (Leone *et al.* 2006; Faber *et al.* 2012); furthermore, the DLV grounder was conceived as part of a monolithic system supporting a specific input language, thus making interfacing with other solvers difficult: this led to a wide adoption of GRINGO as a base for a number of ASP solutions. Recently, $\mathscr{I}$-DLV (Calimeri *et al.* 2017) has been released as a new ASP grounder featuring high flexibility and customizability, and a lightweight modular design for easing the incorporation of optimization techniques and updates; $\mathscr{I}$-DLV can be easily combined with different solvers, since it supports the ASP-Core-2 standard language (Calimeri *et al.* 2012) as well as the de-facto standard numerical format for ground programs (Syrjänen 2002); hence, it constitutes a valid alternative to GRINGO.

The $\mathscr{I}$-DLV performance has already been assessed, proving that it is competitive both as grounder and as deductive database system (Calimeri *et al.* 2018; Calimeri *et al.* 2017). However, the impact on ASP solvers of the instantiation it produces is still largely unexplored. We started a first experimental analysis in this respect by coupling $\mathscr{I}$-DLV with the two state-of-the-art solvers CLASP (Gebser *et al.* 2015) and WASP (Alviano *et al.* 2015). Furthermore, given that, due to possibly different heuristics, optimization techniques, data structures, or input simplifications, such solvers outperform each other depending on the ground program produced by $\mathscr{I}$-DLV given the input at hand (Gebser *et al.* 2017b), we explored possible ways of developing an efficient ASP system that was able to take advantage from peculiarities of both. To this aim, we designed an ad hoc machinery for dynamically selecting the solver to be coupled with $\mathscr{I}$-DLV on an instance basis. The resulting system features a dynamic selector that relies on specific metrics that somehow "describe" the form of the $\mathscr{I}$-DLV instantiations. In order to build the selector, we first introduced a specific set of features for defining the metrics, and then we carried out an experimental activity for actually computing them over all benchmarks submitted to the 6th ASP competition (Gebser *et al.* 2017b) as grounded by $\mathscr{I}$-DLV. We then performed an extensive experimental evaluation in order to first select the most promising features, and then choose the best performing selection algorithm among four different approaches, namely: Random forest (RF), neural network (NN), support vector machines (SVM), and AutoWEKA. The result is the new ASP system $\mathscr{I}$-DLV+$\mathscr{MS}$, which participated in the latest (7th) ASP competition (Gebser *et al.* 2017a) winning in the regular track, category *SP* (i.e., one processor allowed), and is the subject of the present work, where its performance is also assessed by means of proper experimental analyses.

In the remainder of the paper we provide the reader with a brief overview of ASP evaluation (Section 2) before presenting some insight about $\mathscr{I}$-DLV (Section 3); we then describe the design and implementation of $\mathscr{I}$-DLV+$\mathscr{MS}$ (Section 4), and the results of a proper experimental evaluation (Section 5). We discuss related work in Section 6 before illustrating ongoing works and drawing our conclusions (Section 7).

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

2 Evaluation of ASP programs

We briefly recall that the core of the ASP language consists in disjunctive Datalog with nonmonotonic negation under the stable model semantics. Nevertheless, over the years a significant amount of work has been carried out by the scientific community in order to enrich the basic language, and several extensions have been studied and proposed. Recently, the community agreed on a standard input language for ASP systems: ASP-Core-2 (Calimeri *et al.* 2012), the official language of the ASP competition series (Calimeri *et al.* 2016; Gebser *et al.* 2016; Gebser *et al.* 2017a). For a thorough reference on ASP syntax and semantics, we refer the reader to Calimeri *et al.* (2012).

The state-of-the-art strategy for the evaluation of ASP programs (Kaufmann *et al.* 2016) relies on two computational phases. In particular, this approach mimics the definition of the semantics by relying on a grounding module (*grounder* or *instantiator*) coupled with a subsequent module (*solver*) that applies proper propositional techniques for generating its answer sets. These phases are usually referred to as *instantiation* or *grounding*, and *solving* or *answer sets search*, respectively. The two phases are, in general, computationally expensive (Dantsin *et al.* 2001), and efficient ASP computation depends on proper optimization of both.

Given a (nonground) ASP program P, the grounding phase consists in producing a propositional theory G_P semantically equivalent to P, that is, such that G_P does not contain any variable but has the same answer sets as P. State-of-the-art grounders are based on semi-naïve database evaluation techniques (Ullman 1988), for avoiding duplicate work, and build predicate domains dynamically. Basically, they adopt an iterative bottom-up process that successively expands the set of variable-free terms constructible from the nonground program.

The core of the process is the rule instantiation step: given a rule r and a set of ground (i.e., without variables) atoms S, which collects all the ground atoms appearing in the potential extensions of the predicates in P, it generates the ground instances of r. By potential extension of a predicate p we mean here the set of ground instances of p that are either already known to be true in all possible answer sets, or that can possibly be true in some answer set; hence, these constitute the basis for generating the "relevant" ground rules that will undergo the solving phase. In more detail, such task is performed by iterating on the positive body literals and matching them with ground atoms in S so that possible substitutions for their variables are generated. Indeed, because of the safety condition, it is enough to have a substitution for the variables occurring in the positive literals in order to generate a completely ground instance of r. Notably, the set S of predicate extensions is built dynamically: the rules are evaluated according to a specific order ensuring that the complete extension of all predicates needed for instantiating the current rule is already available. The collection of the ground rules generated from all input rules in P composes the ground program G_P. Besides this basic schema, efficient grounders employ smart optimizations techniques, geared toward efficiently producing a ground program that is considerably smaller than the full instantiation, still preserving the semantics.

The propositional program produced by the grounder is evaluated by the solver, whose role is to produce answer sets. Modern ASP solvers implement the backtracking algorithm CDCL, which is based on the pattern *choose-propagate-learn*. Intuitively, the idea is to

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

build an answer set step-by-step by starting from an empty set of literals $\mathscr{I}$. Then, the algorithm heuristically chooses a literal to be added in $\mathscr{I}$, and the deterministic consequences of this choice are propagated, that is, new literals are possibly added to $\mathscr{I}$. The propagation may lead to a conflict, that is, a literal and its negation are both in $\mathscr{I}$. In this case, the conflict is analyzed and a new constraint is added to the propositional program (learning). The conflict is then repaired, that is, $\mathscr{I}$ is restored in a consistent state, choices leading to the conflict are retracted, and a new literal is heuristically selected to be added in $\mathscr{I}$. The algorithm then iterates until $\mathscr{I}$ contains either p or *not* p, for each atom p appearing in the input program, that is, an answer set is produced, or the incoherence of the propositional program is proved, that is, no answer sets are admitted.

3 The $\mathscr{I}$-DLV grounder

The recently released system $\mathscr{I}$-DLV (Calimeri *et al.* 2017) is a brand new ASP grounder and full-fledged deductive database engine. Besides being a stand-alone system, it has been integrated as the grounder module of the new version of the popular ASP system DLV, namely DLV (Alviano *et al.* 2017). Among the most widely used ASP systems, DLV (Leone *et al.* 2006) has been one of the first solid and reliable; its project dates back a few years after the first definition of answer set semantics (Gelfond and Lifschitz 1988; Gelfond and Lifschitz 1991) and encompassed the development and the continuous enhancements of the system; DLV is widely used in academy, and in many relevant industrial applications, significantly contributing in spreading the use of ASP in real-world scenarios (Erdem *et al.* 2016).

In this section, after a brief overview on $\mathscr{I}$-DLV and its main features, we discuss the result of an experimental activity carried out in order to assess the impact of $\mathscr{I}$-DLV on modern ASP solvers.

3.1 Overview

$\mathscr{I}$-DLV has been redesigned and re-engineered from scratch in order to natively support ASP-Core-2 (Calimeri *et al.* 2012) and the latest technologies. Its grounding process is based on the basic instantiation schema briefly described above (see Section 2) that is conveniently enhanced by a number of optimization techniques explicitly designed in the context of ASP instantiation. In particular, $\mathscr{I}$-DLV employs an improved version of already known techniques having a high impact on the overall instantiation process, such as *body-reordering* and *indexing* strategies, along with several additional fine-tuning optimizations acting to different extents on the instantiation process, with the general common aim of reducing the search space and improving overall performance.

Furthermore, as the same computational problem can be encoded by means of many different ASP programs which are semantically equivalent, it is well known that real ASP systems may perform very differently when evaluating each one of them. This behavior is due, in part, to specific aspects that strictly depend on the ASP system employed, and, in part, to general "intrinsic" aspects, depending on the program at hand which could feature some characteristics that can make computation easier or harder. Thus, often, to have satisfying performance, expert knowledge is required in order to select the best encoding, and this, in a certain sense, conflicts with the declarative nature of ASP

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

that, ideally, should free the users from the burden of the computational aspects. For this reason, $\mathscr{I}$-DLV is endowed with proper means aiming at making performance less encoding dependent; intuitively, this is crucial for fostering and easing the usage of ASP in practice. One of the more effective techniques, which has been recently introduced into $\mathscr{I}$-DLV, is the *decomposition rewriting* (Calimeri *et al.* 2018). The rationale behind this technique comes from the consideration that when programs contain rules featuring long bodies, ASP systems performance might benefit from a careful split of such rules into multiple, smaller ones. However, while in some cases such decomposition is convenient, in other cases keeping the original rule is preferable; hence, $\mathscr{I}$-DLV relies on smart rewriting strategies based on *tree-decomposition* algorithms. Basically, each input rule is analyzed before the evaluation, and the system estimates whether it is convenient to decompose it into an equivalent set of smaller rules, or not; if more than one decomposition is possible, the most promising is selected. The method is general and defined so that all choices are made according to proper criteria and heuristics that can be customized: it can be tailored to different phases of the ASP computation, and it is not tied to a specific system. The $\mathscr{I}$-DLV implementation features ad hoc heuristics and criteria relying on data and statistics dynamically computed during the computation with the aim of optimizing the grounding performances. It is worth noting that, among all $\mathscr{I}$-DLV optimizations, some have a significant impact only on instantiation times, while others may impact also on solving times, as they could lead to changes in size and structure of the produced ground program; decomposition rewriting is counted among the latter.

$\mathscr{I}$-DLV comes with a general-purpose default configuration for its optimizations (for a more comprehensive list and a more thorough discussion, we refer the reader to Calimeri *et al.* (2017); however, it also provides the user with means for a fine-grained control over the whole computational process. Indeed, the long-lasting experience over the former DLV grounder (Faber *et al.* 2012) proved that a monolithic set of optimizations, most of which were activated or deactivated at the same time, does not pay in general; on the contrary, the possibility to independently set them allows to ad hoc optimize the computation and also tailor the produced instantiation to different extents. The novel possibility of *annotating ASP code* with external directives to the grounder is a further bold move in this direction. In $\mathscr{I}$-DLV, annotations allow to give explicit directions on the internal computational process. Currently, supported annotations belong to the following categories: *grounding annotations* allowing for a fine-grained customization on the grounding process, and *solving annotations* that have been integrated into DLV, and are geared toward the customization of heuristics and extension of solving capabilities (Alviano *et al.* 2017).

As additional features, $\mathscr{I}$-DLV has been endowed with means to ease the interoperability with external sources of knowledge. In particular, $\mathscr{I}$-DLV supports calls to Python functions via *external atoms* that permit to combine imperative-oriented computational aspects with ASP and connections with relational and graph databases via explicit *directives* for importing/exporting data.

The system is envisioned as core part of a larger project comprising the extension of $\mathscr{I}$-DLV toward mechanisms for interoperability with other formalisms and tools: the intent is to foster the usage of ASP, and in general, of logic programming in real-world and complex applications. Moreover, $\mathscr{I}$-DLV is an open-source project: its source and binaries are available from the official repository (Calimeri *et al.* 2016). For a comprehensive

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

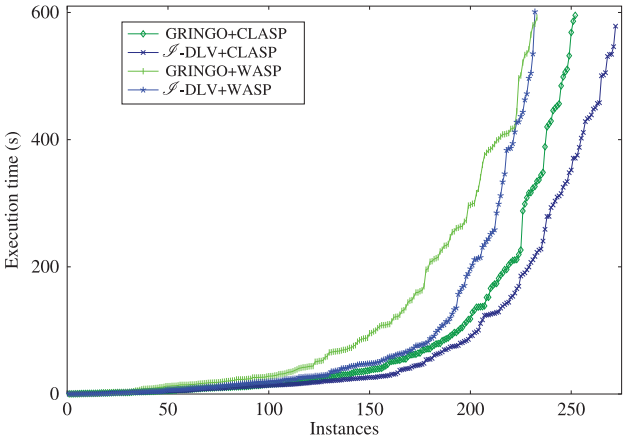


Fig. 1. Cactus plot representing the comparison of GRINGO+CLASP, $\mathscr{I}$-DLV+CLASP, GRINGO+WASP, and $\mathscr{I}$-DLV+WASP on benchmarks from the 6th ASP competition, where domains featuring queries have been removed: number of solved instances on the x-axis while running times (in seconds) are on the y-axis.

list of customizations and options, along with further details, we refer the reader to Calimeri *et al.* (2017) and to the online documentation (Calimeri *et al.* 2016).

3.2 Impact of $\mathscr{I}$-DLV on ASP solvers

In order to analyze the impact of the instantiation produced by $\mathscr{I}$-DLV on the solving phase, we carried out an ad hoc experimental activity comparing performance of $\mathscr{I}$-DLV combined with the two mainstream solvers CLASP and WASP. Furthermore, we tested also the same solvers when coupled with the well-established ASP grounder GRINGO, in order to have a solid term of comparison.

All systems have been executed with their default configuration and the versions employed are the latest available at the time of executing the experiments: $\mathscr{I}$-DLV `2017-06-01`, GRINGO, and CLASP as used by CLINGO `5.1.0`, WASP `2017-05-04`. The executables are available at Calimeri *et al.* (2017).

We relied on the whole 6th ASP competition suite (Gebser *et al.* 2017b)[1] that features 28 problems and 20 different instances per each problem; problems are categorized into three types: (i) *decision* (encodings do not contain queries or weak constraints), (ii) *optimization* (weak constraints are present in the encodings), and (iii) *query* (encodings explicitly feature queries).

All experiments reported herein and in the following have been performed on a NUMA machine equipped with two `2.8 GHz AMD Opteron 6320 CPUs` and `128 GiB` of main memory, running `Linux Ubuntu 14.04.4`; moreover, memory and time were limited to `15 GiB` and `600 s`, respectively, for each system per each single run.

A first idea on the results can be given by Figure 1 that shows a cactus plot illustrating the behavior of the four considered combinations; here, domains featuring queries

[1] The 7*th* ASP competition (Gebser *et al.* 2017a) is already over at the time of writing, but was not held yet at the time we were designing $\mathscr{I}$-DLV+$\mathscr{MS}$; furthermore, the benchmarks therein employed are not publicly available yet, at the time of writing.

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

have been ignored, as they are not supported by GRINGO: this should help at getting a more clear picture of the performance trends of the tested systems. A number of solved instances are reported on the x-axis, while running times (in seconds) are on the y-axis. It is evident that $\mathscr{I}$-DLV+CLASP is the combination that performs better on the overall, solving the highest number of instances and resulting as the most efficient in the majority of the considered domains. Furthermore, it is easy to see that, in general, both solvers benefit from the use of $\mathscr{I}$-DLV instead of GRINGO.

A more detailed analysis can be done while looking at full data, reported in Table 1, that shows average times and number of solved instances within the allotted time and memory; time outs and unsupported syntax are denoted by "TO" and "US," respectively.

Results evidence, as already noted by looking at the plot of Figure 1, good performance of the two solvers when combined with $\mathscr{I}$-DLV: indeed, CLASP and WASP solve a total of 330 and 263 instances, respectively, instead of a total of 254 and 233, respectively, when combined with GRINGO. It is worth noting that the noticeable difference is due also to the explicit support for queries, which is absent in GRINGO; however, performance of the combinations based on $\mathscr{I}$-DLV remains satisfactory even if domains with queries are not taken into account: 272 and 232 instances for CLASP and WASP, respectively. This is encouraging and suggests that $\mathscr{I}$-DLV can be used as a solid base for a competitive ASP system.

The positive impact of $\mathscr{I}$-DLV on the two solvers can be explained by the many optimizations that $\mathscr{I}$-DLV employs for manipulating input rules in order to decrease their intrinsic complexity, which often lead to the production of a smaller number of rules that feature smaller body lengths, with respect to what produced by GRINGO. This explains, for instance, the performance gain enjoyed by the solvers in the case of *Permutation Pattern Matching*, due to a careful automatic optimization of the input programs performed by $\mathscr{I}$-DLV by means of the heuristic-guided decomposition rewriting. In other situations, the difference is made by the impact of the grounding on the solving heuristics adopted during the answer set search, such as in the case of *Video Streaming* for WASP, which benefits from a different rule order produced by $\mathscr{I}$-DLV.

However, a closer look to performance of the solvers and their different behaviors allows us to make some additional considerations. We first note that CLASP outperforms WASP on 10 benchmarks: *Abstract Dialectical Frameworks*, *Combined Configuration*, *Complex Optimization*, *Graph Coloring*, *Incremental Scheduling*, *Partner Units*, *Reachability*, *Ricochet Robots*, *Strategic Companies*, and *Video Streaming*; on the other hand, WASP outperforms CLASP on *Connected Still Life*, *Crossing Minimization*, *Maximal Clique*, and *MaxSAT*. In general, CLASP results to be more efficient than WASP in problems involving cyclic disjunctive rules such as *Abstract Dialectical Frameworks*, *Complex Optimization*, and *Strategic Companies*; this difference is mainly due to the inefficient implementation of unfounded-set check performed by WASP, which is required for such problems. Another important difference between the two solvers is the default algorithm employed on optimization problems. The default configuration of CLASP used in our experiments implements a model-guided algorithm, whose idea is to search for any answer set to initialize an overestimation of the optimal cost, and then to improve the overestimation by searching new answer sets of improved cost, until the overestimation cannot further improved. The default version of WASP, instead, is based on a core-guided algorithm: the idea is to start by searching an answer set satisfying all weak constraints, which would be therefore

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

Table 1. *Comparison of* GRINGO+CLASP, GRINGO+WASP, $\mathscr{I}$-DLV+CLASP, *and* $\mathscr{I}$-DLV+WASP *on benchmarks from the 6th ASP competition (20 instances per problem): number of solved instances and average running times (in seconds). Time outs and unsupported syntax issues are denoted by TO and US, respectively*

Problem	GRINGO+CLASP		GRINGO+WASP		$\mathscr{I}$-DLV+CLASP		$\mathscr{I}$-DLV+WASP	
	No. solved	Time	No. solved	Time	No. solved	Time	No. solved	Time
Abstract Dial. Frameworks	20	8.86	12	39.75	20	6.84	11	33.79
Combined Configuration	9	252.78	1	1.17	10	177.19	0	TO
Complex Optimization	17	134.10	5	199.83	18	159.52	6	152.15
Connected Still Life	6	245.75	12	52.65	6	253.13	12	58.25
Consistent Query Ans.	0	US	0	US	20	86.37	18	87.34
Crossing Minimization	6	64.32	19	4.17	7	55.88	19	5.52
Graceful Graphs	9	70.68	4	47.69	9	136.99	6	194.57
Graph Coloring	15	142.25	8	134.27	15	169.46	8	133.42
Incremental Scheduling	12	88.65	5	130.09	14	120.34	6	77.94
Knight Tour With Holes	10	15.23	10	112.84	11	58.90	10	34.85
Labyrinth	13	108.26	10	139.45	12	148.17	10	70.98
Maximal Clique	0	TO	8	310.99	0	TO	9	353.50
MaxSAT	7	42.71	19	100.71	7	38.29	19	90.53
Minimal Diagnosis	20	8.45	20	35.87	20	8.70	20	24.45
Nomistery	7	92.30	8	58.91	8	140.89	8	36.73
Partner Units	14	35.46	10	227.77	14	20.43	5	131.61
Permut. Pattern Matching	11	165.67	20	176.26	20	16.46	20	22.31
Qualitat. Spatial Reasoning	18	116.66	16	194.69	20	125.38	13	144.68
Reachability	0	US	0	US	20	145.86	6	140.56
Ricochet Robots	7	60.76	6	87.46	9	71.48	7	94.92
Sokoban	10	90.08	9	145.24	8	78.51	8	92.85
Stable Marriage	4	402.30	7	363.72	5	387.66	2	402.24
Steiner Tree	2	52.12	1	270.88	2	40.98	1	132.88
Strategic Companies	0	US	0	US	18	151.30	7	30.30
System Synthesis	0	TO	0	TO	0	TO	0	TO
Valves Location Problem	16	42.91	15	58.69	16	42.70	15	40.01
Video Streaming	13	57.93	0	TO	13	62.61	9	8.99
Visit-all	8	17.26	8	62.91	8	14.93	8	65.43
Total Solved Instances	254		233		330		263	

optimal; if such an answer set does not exist, a subset of the weak constraints that cannot be jointly satisfied, called *unsatisfiable core*, is identified and subsequently relaxed so to search for answer sets satisfying all but one of the original weak constraints. The core-guided algorithm of WASP is complemented with the *stratification* (Alviano *et al.* 2015) technique which first considers only weak constraints of greatest weight. After all cores have been analyzed, a (suboptimal) answer set will be found, and weak constraints with the second greatest weight will be also considered. As already noted in Alviano *et al.* (2015), model-guided and core-guided algorithms show complementary behavior, and for such reason CLASP shows better performance on *Video Streaming*, while the core-guided

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

algorithm implemented by WASP obtains better results on instances of *Connected Still Life*, *Crossing Minimization*, *Maximal Clique*, and *MaxSAT*.

Therefore, none of the solvers is the best on all domains; this proves that there is room for improvements by means of proper smart selection mechanisms. Indeed, if we focus on $\mathscr{I}$-DLV combined with the two different solvers, it is possible to observe that there are 76 and 30 instances uniquely solved by CLASP and WASP, respectively.

4 Automatic coupling $\mathscr{I}$-DLV with a solver

In this section, we illustrate how $\mathscr{I}$-DLV+$\mathscr{MS}$ has been designed. The aim is obtaining a new ASP system that combines $\mathscr{I}$-DLV with an automatic solver selector that is able to automatically classify instances as "better suited for CLASP" or "better suited for WASP" on the basis of some inherent features of their instantiations as produced by $\mathscr{I}$-DLV. In other words, the goal is to perform such identification thanks to a learning phase relying on a *training set* consisting of a pair of observations on some features and membership to a category. Formally, the observations (or instances) consist of an input vector $X = \{x_1, \ldots, x_n\}$ with $x_i \in \mathbb{R}^k$, $i \in \{1 \ldots n\}$, being a vector of values over k features; the categories (or labels) consist of a vector $Y = \{y_1, \ldots, y_m\}$, where m is the number of the classes of the classification problem; in our scenario, m is the number of the ASP solvers. Hence, our training set consists of a pair (X, Y), where to each instance $x_i \in X$ is associated the "best" solver $y_j \in Y$. The *training phase* is the task of generating a classifier implementing $c : \mathbb{R}^k \to Y$ that to each $x_i \in X$ properly associates $y_j \in Y$. In order to build the classifier, we considered four different approaches that have been compared in order to choose the best to our scenario.

4.1 Data set

In order to build the data set, we started by registering the performance of the solvers over all benchmarks submitted to the 6th ASP competition (Gebser *et al.* 2017b) as grounded by $\mathscr{I}$-DLV. As usual in such cases, the data set has been divided into *training*, *validation*, and *test* sets, distributed with the ratio of 50%, 25%, and 25%, respectively, in order to tackle overfitting. Moreover, since the label distribution is unbalanced (CLASP solves significantly more instances than WASP), we used a *stratified sampling* (Esfahani and Dougherty 2014) in order to ensure that relative label frequencies are approximately preserved in each split.

4.2 Features selection

In the following we describe the features that have been considered for the classification.

The idea was to identify a set of features which could somehow "describe" the ground program at hand, and that, in the meanwhile, would be independent from any solver implementation detail, so that it could be eventually easy to test the machinery with additional systems. We discarded all features that are not computable in linear time, as we wanted to keep the overhead as little as possible: we recall here that we compute them over the ground program, whose size can be considerably large, in the general case. Then,

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

we decided to find proper descriptions for two fundamental aspects of ASP programs, namely *atoms ratio* and *rules ratio*; among several possibilities, according to the result of a thorough experimental analysis, we identified the following features as the most suitable:

– *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 (Calimeri *et al.* 2012), 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.

4.3 Choice of a classification algorithm

The considered classifiers are the following:

– *RF*: combination of tree classifiers such that each tree depends on the values of a random input vector sampled independently and with the same distribution for all trees in the forest. This classifier uses averaging to improve the predictive accuracy and control over-fitting (Breiman 2001).
– *NN*: a multi-layer perceptron (MLP) algorithm that trains using backpropagation. The classifier can learn a nonlinear function approximator for classification problem; between the input and the output layer, there can be one or more nonlinear layers, called hidden layers (Breiman 2001).
– *SVM*: a binary linear classifier that basically tries to find an optimal linear hyperplane such that the expected classification error is minimized (Smola and Schölkopf 2004).
– *AutoWEKA*: automatic classifier selection and hyperparameter optimization that try to identify the best classifier algorithm and appropriate hyperparameter settings (Kotthoff *et al.* 2017).

After the training and the hyperparameter tuning phases, we used the *precision-recall metric* to evaluate output quality of the classifiers: precision is a measure of how many relevant results appear within those that are returned, while recall is a measure of how

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

Table 2. *Results of the classifiers on test set*

	Precision	Recall	F_1 score
NN	0.94	0.87	0.89
RF	0.89	0.87	0.88
SVM	0.94	0.87	0.89
AutoWEKA	0.88	0.87	0.87

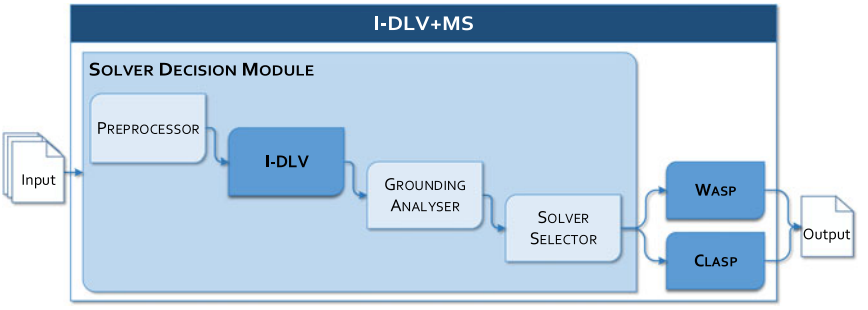


Fig. 2. $\mathscr{I}$-DLV+$\mathscr{MS}$ architecture.

many relevant results are actually returned (i.e., not missed). In particular, we computed the traditional F_1 score as the harmonic mean of precision and recall:

$$\frac{2 * Precision * Recall}{Precision + Recall}$$

Table 2 shows the precision, recall, and F_1 score for each classifiers on the test set. The overall results show good precision and recall for all candidates; however, *AutoWEKA* and *RF* feature lower precision when compared to the *NN* and the *SVM*; even if the latter two perform similarly, we decided to choose the *SVM* as in our tests it required less time to predict new instances.

4.4 The $\mathscr{I}$-DLV+ $\mathscr{MS}$ system

The architecture of $\mathscr{I}$-DLV+$\mathscr{MS}$ is reported in Figure 2. The Preprocessor module analyzes the input program P and interacts with the $\mathscr{I}$-DLV system in order to determine if the input program is nondisjunctive and stratified, as these kinds of programs are completely evaluated by $\mathscr{I}$-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 $\mathscr{I}$-DLV and passes them to the classification module. Then, the Solver Selector tries to foresee, among the available solvers, which one would perform better, and selects it.

Even if the current version of $\mathscr{I}$-DLV+$\mathscr{MS}$ supports just clasp and wasp, its modular architecture 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. In particular, it is worth mentioning that, since clasp and wasp offer several command-line options that modify some aspects of their computation, in order to find the best way of coupling them with $\mathscr{I}$-DLV, we first

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

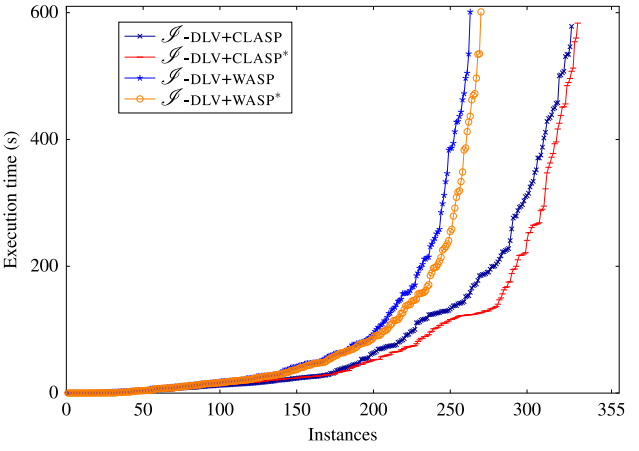


Fig. 3. Experiments for the empirical selection of invocation options for CLASP and WASP when coupled with $\mathscr{S}$-DLV, performed on benchmarks from the 6th ASP competition.

performed an experimental activity by testing the two solvers with different combinations of such command-line options, before the selector was tuned. For the sake of readability, we omit to extensively report here the results obtained in this preliminary step; we just point out that a proper analysis of the results suggested to select the option `-configuration=trendy` for CLASP, which basically changes the heuristic parameters used during the search, and options `-shrinking- strategy=progression -shrinking-budget=10 -trim-core -enable- disjcores` for WASP. It is worth noting that such options force WASP to use two techniques tailored to optimization problems, namely *disjoint cores preprocessing* and *core-shrinking* (Alviano and Dodaro 2016), and that are not supported by CLASP; even if core-shrinking is now supported also by CLASP since version 3.3.0, it was not available at the submission time of the 7th ASP competition, when $\mathscr{I}$-DLV+$\mathscr{MS}$ was being designed and implemented.

The cactus plot in Figure 3 evidences the benefits coming from the usage of the discussed options instead of the default configurations: it compares (i) $\mathscr{I}$-DLV+CLASP (CLASP invoked with the default configuration), (ii) $\mathscr{I}$-DLV+CLASP* (CLASP invoked with the selected configuration), (iii) $\mathscr{I}$-DLV+WASP (WASP invoked with the default configuration), and (iv) $\mathscr{I}$-DLV+WASP* (WASP invoked with the selected configuration); the number of solved instances is on the x-axis while running times (in seconds) are on the y-axis. Consequently, the selector has been actually trained on the basis of the performance of CLASP* and WASP*, which, in turn have been incorporated into the system as depicted in Figure 2.

$\mathscr{I}$-DLV+$\mathscr{MS}$ is freely available at Calimeri *et al.* (2017).

5 Experimental evaluation of $\mathscr{I}$-DLV+$\mathscr{MS}$

In this section, we report the results of an experimental analysis conducted in order to empirically assess performance of $\mathscr{I}$-DLV+$\mathscr{MS}$.

We first compare $\mathscr{I}$-DLV+$\mathscr{MS}$ against $\mathscr{I}$-DLV combined with the two solvers configured as described in Section 4.4, that is, CLASP* and WASP*. The results of

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

Table 3. *Comparison of $\mathscr{I}$-DLV+$\mathscr{MS}$ against $\mathscr{I}$-DLV+CLASP* and $\mathscr{I}$-DLV+WASP* on benchmarks from the 6th ASP competition (20 instances per problem): number of solved instances and average running times (in seconds). Time outs and unsupported syntax issues are denoted by TO and US, respectively*

Problem	$\mathscr{I}$-DLV+CLASP*		$\mathscr{I}$-DLV+WASP*		$\mathscr{I}$-DLV+$\mathscr{MS}$	
	No. solved	Time	No. solved	Time	No. solved	Time
Abstract Dial. Frameworks	20	8.46	14	27.01	20	9.61
Combined Configuration	4	46.60	0	TO	4	51.14
Complex Optimization	19	115.11	6	152.15	19	118.28
Connected Still Life	5	290.71	10	98.31	10	87.05
Consistent Query Ans.	20	86.37	18	87.34	20	102.42
Crossing Minimization	7	71.20	19	1.02	19	2.30
Graceful Graphs	9	125.49	6	194.57	8	68.63
Graph Coloring	16	102.89	8	133.42	16	103.66
Incremental Scheduling	11	36.75	6	77.94	11	41.68
Knight Tour With Holes	14	26.37	10	34.85	14	28.43
Labyrinth	10	125.90	10	70.98	10	128.60
Maximal Clique	0	TO	15	152.43	15	157.62
MaxSAT	9	97.56	19	98.78	18	99.34
Minimal Diagnosis	20	14.81	20	24.45	20	17.65
Nomistery	8	25.38	8	36.73	8	26.66
Partner Units	14	25.24	5	131.61	14	26.36
Permutation Pattern Matching	20	19.20	20	22.31	20	24.82
Qual. Spatial Reasoning	20	108.87	13	144.68	20	115.41
Reachability	20	145.86	6	140.56	20	155.86
Ricochet Robots	10	101.86	7	94.92	11	144.63
Sokoban	9	78.53	8	92.85	8	91.68
Stable Marriage	9	333.31	2	402.24	8	418.16
Steiner Tree	3	179.96	0	TO	3	180.19
Strategic Companies	18	151.30	7	30.30	18	154.29
System Synthesis	0	TO	0	TO	0	TO
Valves Location Problem	16	12.45	16	69.44	16	14.95
Video Streaming	14	64.40	9	0.11	14	65.05
Visit-all	8	16.78	8	65.43	9	81.88
Total Solved Instances	**333**		**270**		**373**	

such comparison are reported in Table 3. $\mathscr{I}$-DLV+$\mathscr{MS}$ performs better than both $\mathscr{I}$-DLV+CLASP* and $\mathscr{I}$-DLV+WASP*; in particular, it solves 40 and 103 instances more than $\mathscr{I}$-DLV+CLASP* and $\mathscr{I}$-DLV+WASP*, respectively. More in detail, when comparing $\mathscr{I}$-DLV+$\mathscr{MS}$ against $\mathscr{I}$-DLV+CLASP*, even if $\mathscr{I}$-DLV+$\mathscr{MS}$ "misses" a solvable instance in *Graceful Graphs*, *Sokoban*, and *Stable Marriage* with respect to $\mathscr{I}$-DLV+CLASP*, a noticeable advantage emerges in several other domains, such as *Connected Still Life*, *Crossing Minimization*, *Maximal Clique*, *MaxSAT*, *Ricochet Robots*, *Visit-all*, where $\mathscr{I}$-DLV+$\mathscr{MS}$ solves 5, 12, 15, 9, 1, 1 additional instances, respectively. When comparing $\mathscr{I}$-DLV+$\mathscr{MS}$ against $\mathscr{I}$-DLV+WASP*, the improvements achieved by $\mathscr{I}$-DLV+$\mathscr{MS}$ are even more evident, with peaks of 13 instances for *Complex Optimization*, 9 instances for

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

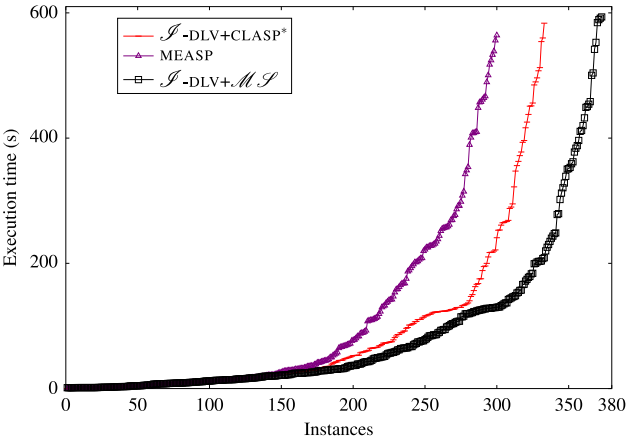


Fig. 4. Experimental evaluation of $\mathscr{I}$-DLV+$\mathscr{M}\mathscr{S}$ on benchmarks from the 6th ASP competition.

Partner Units, 14 instances for *Reachability*, and 11 instances for *Strategic Companies*; the only exception is *MaxSAT* in which $\mathscr{I}$-DLV+WASP* solves one instance more than $\mathscr{I}$-DLV+$\mathscr{M}\mathscr{S}$.

A more careful analysis of Table 3 might lead to the conclusion that the number of instances solved by $\mathscr{I}$-DLV+$\mathscr{M}\mathscr{S}$, for each domain, is limited by the best configuration $\mathscr{I}$-DLV+CLASP* and $\mathscr{I}$-DLV+WASP*, that is, as if it chooses the same solver for all instances of a given domain. This is actually the case for some benchmarks, as, for instance, *Combined Configuration*, where it is evident that the selector always elects CLASP*. This is not surprising, as most of the ASP competition instances of a given benchmark problem, when coupled with the encoding, produce ground programs that feature similar structure; hence, same values for the metrics employed by the selector are induced. Nevertheless, this is not true in general, as witnessed by cases such as *Ricochet Robots*, *Graceful Graphs*, and others.

The smart selection machinery causes an overhead; this is expected, as it requires to analyze the produced instantiation in order to first compute the features needed for selecting the solver and then properly invoking it. However, such overhead is definitely acceptable, as it can be clearly observed by taking into account domains for which only one among $\mathscr{I}$-DLV+CLASP* and $\mathscr{I}$-DLV+WASP* is able to solve at least one instance. For example, when dealing with *Combined Configuration* and *Steiner Tree*, $\mathscr{I}$-DLV+$\mathscr{M}\mathscr{S}$ performs analogously to $\mathscr{I}$-DLV+CLASP*, while when dealing with *Maximal Clique* it reflects the behavior of $\mathscr{I}$-DLV+WASP*, solving the same number of instances with an overhead that goes from less than 1% to no more than 9%.

Further experiments have been performed for testing $\mathscr{I}$-DLV+$\mathscr{M}\mathscr{S}$ against (i) $\mathscr{I}$-DLV+CLASP*, which is the best combination of $\mathscr{I}$-DLV with a solver obtainable without the use of a smart selection machinery, and (ii) the latest available version of MEASP[2], which represents the state-of-the-art among multi-engine ASP solvers, and was also the winner of the 6th ASP competition. Results of such comparison are depicted in the cactus plot reported in Figure 4, where the number of solved instances is on the x-axis while

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

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

running times (in seconds) are on the y-axis. In general, $\mathscr{I}$-DLV+$\mathscr{MS}$ outperforms both $\mathscr{I}$-DLV+CLASP* and MEASP. While, according to the consideration above, the results of the comparison against $\mathscr{I}$-DLV+CLASP* are expected, some additional considerations deserve to be made about MEASP.

Differently from $\mathscr{I}$-DLV+$\mathscr{MS}$, MEASP employs machine-learning techniques for the inductive choice of the best solver on a per instance basis. Moreover, even if there are similarities with $\mathscr{I}$-DLV+$\mathscr{MS}$, there are some other key differences, starting from the nature and the number of ASP systems employed. Indeed, in order to choose the solver, MEASP relies on the instantiation produced by GRINGO; furthermore, it chooses among five solvers, way more than the mere two taken into account by $\mathscr{I}$-DLV+$\mathscr{MS}$. On the one hand, given that different engines use evaluation strategies that can be substantially different, the use of a larger pool of solvers allows to gain a significant number of instances that are uniquely solvable by one solver. On the other hand, such differences imply that a high price might be paid in case of a wrong choice; in fact, if the space for choices is narrowed, the probability of making a wrong one decreases, and this might lead to a more consistent behavior, as in the case of $\mathscr{I}$-DLV+$\mathscr{MS}$.

In addition, it might be interesting to note that $\mathscr{I}$-DLV+CLASP* outperforms MEASP. This can be partially explained by observing that, as already mentioned, MEASP relies on GRINGO, for grounding, and the best configuration overall obtainable from its pool of solvers consists of GRINGO and CLASP; however, we already noted in Section 3 how CLASP performance look to increase when working on $\mathscr{I}$-DLV instantiations instead of on GRINGO ones.

A final consideration concerns overfitting, which can be an important issue when dealing with inductive machineries that rely on training. As pointed out in Section 4.1, we specifically designed the data set in order to prevent the problem, and the system appears to be able to adapt to unseen situations: indeed, the selector was trained on the basis of the data available from the 6th ASP competition and participated in the 7th, which featured significant differences with respect to the previous one in terms of new domains, new encodings for already present domains, and new instances. Nevertheless, $\mathscr{I}$-DLV+$\mathscr{MS}$ was able to perform well and even win the regular track.

6 Related work

Traditional evaluation strategy of ASP systems is based on two stages and mimics the intended semantics by first performing *grounding* and then executing *solving*; for both phases, several efficient systems have been proposed during the years.

As for instantiation, the most widespread grounders are (the one in) DLV (Faber *et al.* 2012) and GRINGO (Gebser *et al.* 2011); both systems are based on seminaïve database evaluation techniques (Ullman 1988) for avoiding duplicate work during grounding, and accept safe programs as input (Faber *et al.* 2012; Calimeri *et al.* 2017; Gebser *et al.* 2011). Other approaches have been implemented in LPARSE (Syrjänen 2001) and in earlier versions of GRINGO (Gebser *et al.* 2007) that take as input ω- or λ-restricted programs, respectively. $\mathscr{I}$-DLV (Calimeri *et al.* 2017) has been released as the new version of the DLV grounder; it shares with DLV and GRINGO the support to safe programs and the general evaluation schema based on seminaïve; however, it significantly differs in the optimization

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

strategies from both of them and, furthermore, differently from DLV, it fully supports the ASP-Core-2 (Calimeri *et al.* 2012) language standard.

Concerning ASP solvers, the first generation, that is, SMODELS (Simons *et al.* 2002) and DLV (Leone *et al.* 2006; Maratea *et al.* 2008), was based on a DPLL-like algorithm extended with inference rules specific to ASP. Modern ASP solvers, including CLASP (Gebser *et al.* 2016; Gebser *et al.* 2012; Gebser *et al.* 2015) and WASP (Dodaro *et al.* 2011; Alviano *et al.* 2015), also include mechanisms for conflict-driven clause learning and for nonchronological backtracking.

Interestingly, the attempt to find ways of enjoying good performance over different problem domains by means of smart choices has already been pursued by neighboring communities, by means of proper strategies of *algorithm selection* (Rice 1976); we cite here, for instance, what has been done for solving propositional satisfiability (SAT) (Xu *et al.* 2008) and Quantified SAT (QSAT) (Pulina and Tacchella 2009). As for what ASP is concerned, some interesting works in this respect have already been carried out in Maratea *et al.* (2014) and Hoos *et al.* (2014): the former proposal is the well-established multi-engine ASP system MEASP, the winner of the 6th ASP competition, while the latter relies on a single solver, that is, CLASP, and uses similar strategies in order to select the "best" configuration for it. Some considerations about similarities and differences between such proposals and $\mathscr{I}$-DLV+$\mathscr{MS}$ follow.

$\mathscr{I}$-DLV+$\mathscr{MS}$ is a portfolio solver and its working principles are similar to the ones of the multi-engine ASP solver MEASP (Maratea *et al.* 2014). Indeed, both solvers are based on a classification of instances according to their internal features. In particular, the ones computed by MEASP can be classified in four different categories: (i) *problem size*, that is, number of rules r, number of atoms a, ratios (r/a), $(r/a)^2$, $(r/a)^3$, (a/r), $(a/r)^2$, $(a/r)^3$; (ii) *balance*, that is, fraction of unary, binary, and ternary rules; (iii) *proximity to Horn*, that is, fraction of Horn rules and number of occurrences in a Horn rule for each atom; (iv) *ASP specific*, that is, number of rules with empty bodies, fraction of normal rules and constraints. As a design choice, all the features considered by MEASP can be easily computed, that is, they are computable in linear time in the size of the input program. $\mathscr{I}$-DLV+$\mathscr{MS}$ considers a different set of features, as detailed in Section 4, still linear-time computable; they also include rule types, for example, choice and weight rules. Additional differences concern the pool of engines employed by MEASP and $\mathscr{I}$-DLV+$\mathscr{MS}$. As already mentioned in Section 5, the former uses CLASP (and its variant CLASPD; Gebser *et al.* (2015)), CMODELS (Lierler and Maratea 2004), DLV (Leone *et al.* 2006), and IDP (Bruynooghe *et al.* 2015), while $\mathscr{I}$-DLV+$\mathscr{MS}$ uses only CLASP (Gebser *et al.* 2015) and WASP (Alviano *et al.* 2015).

$\mathscr{I}$-DLV+$\mathscr{MS}$ is also related to CLASPFOLIO (Hoos *et al.* 2014), which selects the *best* configuration for the solver CLASP. The first difference between the two solvers is related to the inductive model; indeed, the one of CLASPFOLIO is based on *regression* techniques, while the inductive model of $\mathscr{I}$-DLV+$\mathscr{MS}$ is based on *classification* algorithms. Moreover, CLASPFOLIO uses dynamic features that are computed at each restart, for example, the number of choices performed by the solver, while $\mathscr{I}$-DLV+$\mathscr{MS}$ only considers static features. Finally, since CLASPFOLIO is tailored to CLASP, it takes into account features that cannot be computed by WASP, for example, the number of equivalences. On the contrary, $\mathscr{I}$-DLV+$\mathscr{MS}$ considers the underlying solvers as black boxes, therefore it can be extended for including also other solvers other than CLASP and WASP.

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

7 Conclusions and future work

In this work we presented $\mathscr{I}$-DLV+$\mathscr{MS}$, a new ASP system that relies on a modern ASP grounder, namely $\mathscr{I}$-DLV, and of a smart selection machinery to inductively choose a solver depending on the features of the input at hand. We reported the rationales and the details about design and implementation of the system, along with a number of experimental analyses. The system shows good performance, both when compared against MEASP, which was the winner of the 6th ASP competition and makes use of similar inductive approaches for selecting ASP solvers, and when compared against combinations of $\mathscr{I}$-DLV with the mainstream ASP solvers CLASP and WASP. Good performance is confirmed also by the success in the latest (7th) ASP competition (Gebser *et al.* 2017a), where the system won the regular track in the *SP* category.

Note that all systems employed and tested in this work refer to the respective versions available at the time of designing, implementing, and training the herein proposed system, that is, before the 7th ASP competition officially started. At the time of finalizing this manuscript, new versions came out for many of such systems, both solvers and grounders. As future work, we plan to update $\mathscr{I}$-DLV+$\mathscr{MS}$ by taking into account the new versions, as results and considerations might significantly change. Furthermore, we will take into account additional ASP solvers with different parameterizations and explore more features for improving classification capabilities and achieve better overall performance. Eventually, we are studying the possibility of designing a built-in automatic configurator for $\mathscr{I}$-DLV that, using machine-learning techniques, properly sets a bunch of optimization parameters among the several currently available in $\mathscr{I}$-DLV; this would make the grounder capable to automatically adapt to the problem at hand.

References

Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P., and Zangari, J. 2017. The ASP system DLV2. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 10377, Espoo, Finland, 3–6 Jul. 2017, M. Balduccini and T. Janhunen, Eds. Springer, 215–221.

Alviano, M. and Dodaro, C. 2016. Anytime answer set optimization via unsatisfiable core shrinking. *Theory and Practice of Logic Programming 16,* 5–6, 533–551.

Alviano, M., Dodaro, C., Leone, N. and Ricca, F. 2015. Advances in WASP. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 9345, Lexington, KY, USA, 27–30 Sept. 2015, F. Calimeri, G. Ianni and M. Truszczynski, Eds. Springer, 40–54.

Alviano, M., Dodaro, C., Marques-Silva, J. and Ricca, F. 2015. Optimum stable model search: Algorithms and implementation. *Journal of Logic and Computation*.

Breiman, L. 2001. Random forests. *Machine Learning 45,* 1, 5–32.

Brewka, G., Eiter, T. and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM 54,* 12, 92–103.

Bruynooghe, M., Blockeel, H., Bogaerts, B., De Cat, B., Pooter, S. D., Jansen, J., Labarre, A., Ramon, J., Denecker, M. and Verwer, S. 2015. Predicate logic as a modeling language: Modeling and solving some machine learning and data mining problems with *IDP3*. *Theory and Practice of Logic Programming 15,* 6, 783–817.

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F. and Schaub, T. 2012. Asp-core-2: Input language format. *ASP Standardization Working Group*. Technical Report.

Calimeri, F., Fuscà, D., Perri, S. and Zangari, J. since 2016. $\mathscr{I}$-DLV homepage. URL: https://github.com/DeMaCS-UNICAL/I-DLV/wiki. [Accessed on March 20, 2018].

Calimeri, F., Fuscà, D., Perri, S. and Zangari, J. since 2017. $\mathscr{I}$-DLV+ $\mathscr{MS}$ homepage. URL: https://github.com/DeMaCS-UNICAL/IDLV-MS. [Accessed on March 20, 2018].

Calimeri, F., Fuscà, D., Perri, S. and Zangari, J. 2017. I-DLV: The new intelligent grounder of DLV. *Intelligenza Artificiale 11,* 1, 5–20.

Calimeri, F., Fuscà, D., Perri, S. and Zangari, J. 2018. Optimizing answer set computation via heuristic-based decomposition. In *International Symposium on Practical Aspects of Declarative Languages*. Lecture Notes in Computer Science, vol. 10702, Los Angeles, CA, USA, 8–9 Jan. 2018, F. Calimeri, K. W. Hamlen and N. Leone, Eds. Springer, 135–151.

Calimeri, F., Gebser, M., Maratea, M. and Ricca, F. 2016. Design and results of the fifth answer set programming competition. *Artificial Intelligence 231*, 151–181.

Dal Palù, A., Dovier, A., Pontelli, E. and Rossi, G. 2009. GASP: Answer set programming with lazy grounding. *Fundamenta Informaticae 96,* 3, 297–322.

Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Survey 33,* 3, 374–425.

Dao-Tran, M., Eiter, T., Fink, M., Weidinger, G., and Weinzierl, A. 2012. Omiga: An open minded grounding on-the-fly answer set solver. In *European Conference on Logics in Artificial Intelligence*. Lecture Notes in Computer Science, vol. 7519, Toulouse, France, 26–28 Sept. 2012, L. F. del Cerro, A. Herzig and J. Mengin, Eds. Springer, 480–483.

Dodaro, C., Alviano, M., Faber, W., Leone, N., Ricca, F. and Sirianni, M. 2011. The birth of a WASP: Preliminary report on a new ASP solver. In *Italian Conference on Computational Logic*. CEUR Workshop Proceedings, vol. 810, Pescara, Italy, Aug. 31 - Sept. 2, F. Fioravanti, Ed. CEUR-WS.org, 99–113.

Dodaro, C., Gasteiger, P., Leone, N., Musitsch, B., Ricca, F. and Schekotihin, K. 2016. Combining answer set programming and domain heuristics for solving hard industrial problems (application paper). *Theory and Practice of Logic Programming 16,* 5–6, 653–669.

Dodaro, C., Leone, N., Nardi, B. and Ricca, F. 2015. Allotment problem in travel industry: A solution based on ASP. In *International Conference on Web Reasoning and Rule Systems*. Lecture Notes in Computer Science, vol. 9209, Berlin, Germany, 4–5 Aug. 2015, B. T. Cate and A. Mileo, Eds. Springer, 77–92.

Erdem, E., Gelfond, M. and Leone, N. 2016. Applications of answer set programming. *AI Magazine 37,* 3, 53–68.

Esfahani, M. S. and Dougherty, E. R. 2014. Effect of separate sampling on classification accuracy. *Bioinformatics 30,* 2, 242–250.

Faber, W., Leone, N. and Perri, S. 2012. The intelligent grounder of DLV. In *Correct Reasoning–Essays on Logic-Based AI in Honour of Vladimir Lifschitz*. Lecture Notes in Computer Science, vol. 7265. Springer, 247–264.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T. and Wanko, P. 2016. Theory solving made easy with clingo 5. In *Technical Communications of the International Conference on Logic Programming*. OASICS, vol. 52, New York, NY, USA, 16–21 Oct. 2016, M. Carro, A. King, N. Saeedloei and M. De Vos, Eds. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2:1–2:15.

Gebser, M., Kaminski, R., Kaufmann, B., Romero, J. and Schaub, T. 2015. Progress in clasp series 3. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, Lexington, KY, USA, 27–30 Sept. 2015, F. Calimeri, G. Ianni and M. Truszczynski, Eds. vol. 9345. Springer, 368–383.

Gebser, M., Kaminski, R., König, A. and Schaub, T. 2011. Advances in *gringo* series 3. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

in Computer Science, vol. 6645, Vancouver, Canada, 16–19 May 2011, J. P. Delgrande and W. Faber, Eds. Springer, 345–351.

Gebser, M., Kaufmann, B. and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence 187*, 52–89.

Gebser, M., Maratea, M. and Ricca, F. 2016. What's hot in the answer set programming competition. In *AAAI Conference on Artificial Intelligence*, Phoenix, AZ, USA, 12–17 Feb. 2016, D. Schuurmans and M. P. Wellman, Eds. AAAI Press, 4327–4329.

Gebser, M., Maratea, M. and Ricca, F. 2017a. The design of the seventh answer set programming competition. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 10377, Espoo, Finland, 3–6 Jul. 2017, M. Balduccini and T. Janhunen, Eds. Springer, 3–9.

Gebser, M., Maratea, M. and Ricca, F. 2017b. The sixth answer set programming competition. *Journal of Artificial Intelligence Research 60*, 41–95.

Gebser, M., Schaub, T. and Thiele, S. 2007. Gringo : A new grounder for answer set programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 4483, Tempe, AZ, USA, May 15–17 2007, C. Baral, G. Brewka and J. S. Schlipf, Eds. Springer, 266–271.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming In *International Conference and Symposium on Logic Programming*, Seattle, WA, USA, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9,* 3/4, 365–386.

Grasso, G., Iiritano, S., Leone, N. and Ricca, F. 2009. Some DLV applications for knowledge management. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 5753, Potsdam, Germany, 14–18 Sept. 2009. E. Erdem, F. Lin and T. Schaub, Eds. Springer, 591–597.

Hoos, H., Lindauer, M. T. and Schaub, T. 2014. claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming 14,* 4–5, 569–585.

Kaufmann, B., Leone, N., Perri, S. and Schaub, T. 2016. Grounding and solving in answer set programming. *AI Magazine 37,* 3, 25–32.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F. and Leyton-Brown, K. 2017. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research 18*, 25:1–25:5.

Lefèvre, C., Béatrix, C., Stéphan, I. and Garcia, L. 2017. Asperix, a first-order forward chaining approach for answer set computing. *Theory and Practice of Logic Programming 17,* 3, 266–310.

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 Transaction on Computational Logic 7,* 3, 499–562.

Leone, N. and Ricca, F. 2015. Answer set programming: A tour from the basics to advanced development tools and industrial applications. In *International Summer School on Reasoning Web*. Lecture Notes in Computer Science, vol. 9203, Berlin, Germany, Jul. 31-Aug. 4 2015, W. Faber and A. Paschke, Eds. Springer, 308–326.

Lierler, Y. and Maratea, M. 2004. Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 2923, Fort Lauderdale, FL, USA, 6–8 Jan. 2004, V. Lifschitz and I. Niemelä, Eds. Springer, 346–350.

Maratea, M., Pulina, L. and Ricca, F. 2014. A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming 14,* 6, 841–868.

Maratea, M., Ricca, F., Faber, W. and Leone, N. 2008. Look-back techniques and heuristics in DLV: Implementation, evaluation, and comparison to QBF solvers. *Journal of Algorithms 63,* 1–3, 70–89.

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press

Nogueira, M., Balduccini, M., Gelfond, M., Watson, R. and Barry, M. 2001. An a-prolog decision support system for the space shuttle. In *International Symposium on Practical Aspects of Declarative Languages*. Lecture Notes in Computer Science, vol. 1990, Las Vegas, NV, USA, 11–12 Mar. 2001, I. V. Ramakrishnan, Eds. Springer, 169–183.

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

Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S. and Leone, N. 2012. Team-building with answer set programming in the gioia-tauro seaport. *Theory and Practice of Logic Programming 12,* 3, 361–381.

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

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1–2, 181–234.

Smola, A. J. and Schölkopf, B. 2004. A tutorial on support vector regression. *Statistics and Computing 14,* 3, 199–222.

Syrjänen, T. 2001. Omega-restricted logic programs. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 2173, Vienna, Austria, 17–19 Sept. 2001, T. Eiter, W. Faber and M. Truszczynski, Eds. Springer, 267–279.

Syrjänen, T. 2002. Lparse 1.0 user's manual. URL: http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz. [Accessed on March 20, 2018].

Tiihonen, J., Soininen, T., Niemelä, I. and Sulonen, R. 2003. A practical tool for mass-customising configurable products. In *International Conference on Engineering Design*, Stockholm, Sweden, 19–21 Aug. 2003, A. Folkeson, K. Gralen, M. Norell and U. Sellgren, Eds.

Ullman, J. D. 1988. *Principles of Database and Knowledge-Base Systems*, vol. I. Computer Science Press, New York, NY, USA.

Weinzierl, A. 2017. Blending lazy-grounding and CDNL search for answer-set solving. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 10377, Espoo, Finland, 3–6 Jul. 2017, M. Balduccini and T. Janhunen, Eds. Springer, 191–204.

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

https://doi.org/10.1017/S1471068418000546 Published online by Cambridge University Press