Journal of Artificial Intelligence Research 60 (2017) 41-95 Submitted 10/16; published 09/17

The Sixth Answer Set Programming Competition

Martin Gebser GEBSER@CS.UNI-POTSDAM.DE
IfICS, University of Potsdam,
August-Bebel-Str. 89, 14482 Potsdam, Germany

Marco Maratea MARCO@DIBRIS.UNIGE.IT
DIBRIS, University of Genoa,
Viale F. Causa 15, 16145 Genova, Italy

Francesco Ricca RICCA@MAT.UNICAL.IT
DeMaCS, University of Calabria,
Viale P. Bucci, Cubo 31B, 87036 Rende, Italy

Abstract

Answer Set Programming (ASP) is a well-known paradigm of declarative programming with roots in logic programming and non-monotonic reasoning. Similar to other closely related problem-solving technologies, such as SAT/SMT, QBF, Planning and Scheduling, advancements in ASP solving are assessed in competition events. In this paper, we report about the design and results of the Sixth ASP Competition, which was jointly organized by the University of Calabria (Italy), Aalto University (Finland), and the University of Genoa (Italy), in affiliation with the 13th International Conference on Logic Programming and Non-Monotonic Reasoning. This edition maintained some of the design decisions introduced in 2014, e.g., the conception of sub-tracks, the scoring scheme, and the adherence to a fixed modeling language in order to push the adoption of the ASP-Core-2 standard. On the other hand, it featured also some novelties, like a benchmark selection stage classifying instances according to their empirical hardness, and a "Marathon" track where the top-performing systems are given more time for solving hard benchmarks.

1. Introduction

Answer Set Programming (ASP) (Brewka, Eiter, & Truszczyński, 2011) is a well-known declarative programming approach to knowledge representation and reasoning, with roots in the areas of logic programming and non-monotonic reasoning (Gelfond & Lifschitz, 1991) as well as in close relationships to other formalisms such as Propositional Satisfiability (SAT), Satisfiability Modulo Theories (SMT), Quantified Boolean Formulas (QBF), Constraint Programming, Planning and Scheduling, and many others. Thanks to its high readability and expressiveness, and the availability of efficient solvers, many ASP-based applications have been presented in the literature (Erdem, Gelfond, & Leone, 2016), in several areas ranging from Artificial Intelligence (Balduccini, Gelfond, Watson, & Nogueira, 2001; Baral & Gelfond, 2000; Baral & Uyan, 2001; Brewka, Niemelä, & Syrjänen, 2002; Garro, Palopoli, & Ricca, 2006; Nogueira, Balduccini, Gelfond, Watson, & Barry, 2001) to Knowledge Management (Baral, 2003; Bardadym, 1996), Databases (Bertossi, Hunter, & Schaub, 2005; Bravo & Bertossi, 2003; Leone, Gottlob, Rosati, Eiter, Faber, Fink, Greco, Ianni, Kałka, Lembo, Lenzerini, Lio, Nowicki, Ruzzi, Staniszkis, & Terracina, 2005; Manna, Ricca, & Terracina, 2013, 2015), Phylogeny (Erdem, 2011; Koponen, Oikarinen, Janhunen, & Säilä, 2015), Bio-informatics (Dworschak, Grell, Nikiforova, Schaub, & Selbig, 2008; Gebser, Schaub, Thiele, & Veber, 2011), and industrial settings (Aschinger, Drescher, Friedrich, Gottlob, Jeavons, Ryabokon, & Thorstensen, 2011; Ielpa, Iiritano, Leone, & Ricca, 2009; Ricca, Grasso, Alviano, Manna, Lio, Iiritano, & Leone, 2012).

The ASP Competition is a biennial event organized in odd years, with the exception of the Fifth edition, which was held in 2014 in order to join the FLoC Olympic Games at the Vienna Summer of Logic (VSL, 2014). In this paper, we report about the design and the results of the Sixth ASP Competition (ASP-Comp, 2015), which was jointly organized by the University of Calabria (Italy), Aalto University (Finland), and the University of Genoa (Italy), in affiliation with the 13th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR, 2015). It maintained some of the design decisions introduced in the previous edition, e.g., (i) the conception of sub-tracks, based on the "complexity" of the encoded problems (as in past events), but also considering the language features used in encodings (e.g., choice rules, aggregates, presence of queries), (ii) the scoring scheme, which had been significantly simplified, and (iii) the adherence to a fixed modeling language in order to push the adoption of the ASP-Core-2 standard (Calimeri, Faber, Gebser, Ianni, Kaminski, Krennwallner, Leone, Ricca, & Schaub, 2012). On the other hand, it introduced novelties, some of them borrowed from past editions of the SAT and QBF Competitions, i.e., (i) a benchmark selection stage classifying instances according to their empirical hardness, in order to cover a broad range of scalability, and (ii) a "Marathon" track where the top-performing systems are given more time for solving hard benchmarks, in order to assess their success to complete challenging instances and to compare their performance in the long run. While details on the previous ASP Competition edition, held in 2014, are provided by Calimeri, Gebser, Maratea, and Ricca (2016), the main new contributions of this paper can be summarized as follows:

- We give a comprehensive overview of common ASP solving approaches, which allows us to thoroughly characterize current participant systems and analyze their relative performance.

- We introduce new benchmarks submitted to this ASP Competition edition, stemming from six application-oriented domains, and report about the encodings provided by benchmark authors as well as alternatives ones furnished for comparison.

- We describe the refined benchmark selection process applied for the first time, which consists of a classification of available instances based on empirical hardness along with a balanced selection accomplished by means of ASP.

- We present the results of the Sixth ASP Competition and analyze system performance with respect to language features, computational tasks, reference systems from the previous edition, effect of modeling, runtime limits, and sequential versus parallel processing.

- We relate the design novelties of this edition to competitions in neighboring areas and outline potential directions for future ASP Competition editions as well as ASP research in general.

The rest of the paper is structured as follows. Section 2 recalls the ASP syntax and semantics, relevant ASP program properties, and the ASP evaluation process. Then, Section 3 surveys main approaches for solving ASP programs, and presents the competition participants and systems. Section 4 introduces the setting of the Sixth ASP Competition, followed by Section 5 and 6 detailing benchmark domains and the newly introduced instance selection process, respectively. Section 7 presents the winners of competition categories and sub-tracks, and also provides a detailed analysis of performance results. The paper concludes in Section 8 and 9 by discussing similarities and differences to related competition series on the design novelties introduced as well as some considerations on future directions, respectively.

2. Background

In this section, we first review the syntax and semantics of ASP-Core-2 programs (Calimeri et al., 2012). Then, we remind some program properties that will be useful in the benchmark classification. Finally, we illustrate the typical two-step ASP evaluation process.

2.1 Syntax

Terms are composed of constants, variables, and functions. Dedicated *arithmetic* terms have the form $-(t)$ or $(t \diamond u)$ with $\diamond \in \{+, -, *, /\}$. Terms and $\prec \in \{<, \leq, =, \neq, >, \geq\}$ are used to construct three kinds of *atoms*:

- *classical* atoms $p(t_1, \dots, t_n)$ and $\neg p(t_1, \dots, t_n)$ for a predicate name p and terms $t_1, \dots, t_n$,
- *built-in* atoms $t \prec u$ for terms t and u, and
- *aggregate* atoms $\#\mathbf{agg}\{e_1; \dots; e_k\} \prec u$ for $\#\mathbf{agg} \in \{\#\mathtt{count}, \#\mathtt{sum}, \#\mathtt{max}, \#\mathtt{min}\}$, where e_i ($1 \leq i \leq k$) is an aggregate element $t_1, \dots, t_n : l_1, \dots, l_m$ in which $t_1, \dots, t_n$ are terms and $l_1, \dots, l_m$ are *naf-literals*, i.e., built-in atoms $t \prec u$ or expressions a and $\mathbf{not}\ a$ for classical atoms a.

Note that $\mathbf{not}$ stands for *default negation*, and *literals* in general include naf-literals as well as a and $\mathbf{not}\ a$ for aggregate atoms a.

ASP-Core-2 programs consist of rules, possibly accompanied by weak constraints or a query. A *rule* r is of the form $a_1 \mid \dots \mid a_m \leftarrow b_1, \dots, b_n$, where $a_1, \dots, a_m$ are classical atoms for $m \geq 0$ and $b_1, \dots, b_n$ are literals for $n \geq 0$. If $m = 1$ and $n = 0$, r is also called a *fact*, and r is a *disjunctive* rule if $m > 1$. Moreover, *choice* rules have the form $\{e_1; \dots; e_k\} \prec u \leftarrow b_1, \dots, b_n$, where e_i ($1 \leq i \leq k$) is a choice element $a : l_1, \dots, l_m$ in which a is a classical atom and $l_1, \dots, l_m$ are naf-literals. A *weak constraint* $:\sim b_1, \dots, b_n\, [w@l, t_1, \dots, t_m]$ associates literals $b_1, \dots, b_n$ with a *weight* w, a *level* l, and additional terms $t_1, \dots, t_m$ for $m \geq 0$. Finally, $a?$ is a *query* for a classical atom a. An atom, a rule, an ASP-Core-2 program, etc. is *ground* if it does not contain any variables or arithmetic terms, and non-ground otherwise.

2.2 Semantics

An *interpretation* I is a consistent set of ground classical atoms, i.e., $p(t_1, \dots, t_n)$ and $\neg p(t_1, \dots, t_n)$ must not jointly occur in I for any predicate name p and terms $t_1, \dots, t_n$. The satisfaction relation w.r.t. I is defined inductively by

- $I \models a$ for a classical atom a, if $a \in I$, otherwise $I \models \mathbf{not}\ a$;
- $I \models t \prec u$, if $t \prec u$ according to the definition in Section 2.3 by Calimeri et al. (2012);
- $I \models \#\mathbf{agg}\{e_1; \dots; e_k\} \prec u$ for $\#\mathbf{agg} \in \{\#\mathtt{count}, \#\mathtt{sum}, \#\mathtt{max}, \#\mathtt{min}\}$, if $\#\mathbf{agg}(T) \prec u$, where $T = \bigcup_{1 \leq i \leq k, e_i = t_1, \dots, t_n : l_1, \dots, l_m} \{(t_1, \dots, t_n) \mid I \models l_1, \dots, I \models l_m\}$ is the (finite)

set of tuples $(t_1, \dots, t_n)$ for aggregate elements $t_1, \dots, t_n : l_1, \dots, l_m$ whose naf-literals $l_1, \dots, l_m$ are satisfied w.r.t. I, and

- $\#\mathtt{count}(T) = |T|$,
- $\#\mathtt{sum}(T) = \sum_{(t_1,\dots,t_n) \in T \text{ with integer } t_1} t_1$,
- $\#\mathtt{max}(T) = \max\{t_1 \mid (t_1, \dots, t_n) \in T\}$, and
- $\#\mathtt{min}(T) = \min\{t_1 \mid (t_1, \dots, t_n) \in T\}$,[1]

while $I \models \mathbf{not}\ \#\mathbf{agg}\{e_1; \dots; e_k\} \prec u$ otherwise;

- $I \models \{e_1; \dots; e_k\} \prec u$, if $|\bigcup_{1 \leq i \leq k, e_i = a:l_1,\dots,l_m} \{a \in I \mid I \models l_1, \dots, I \models l_m\}| \prec u$;
- $I \models a_1 \mid \dots \mid a_m$ for classical atoms $a_1, \dots, a_m$, if $\{a_1, \dots, a_m\} \cap I \neq \emptyset$.

A rule of the form $A \leftarrow b_1, \dots, b_n$ is satisfied w.r.t. I, if $I \models b_1, \dots, I \models b_n$ implies $I \models A$. Moreover, I is a *model* of a ground ASP-Core-2 program P, if every rule in P is satisfied w.r.t. I. Following (Faber, Leone, & Pfeifer, 2004, 2011) in extending the original notion by Gelfond and Lifschitz (1991) to aggregates, the *reduct* P^I of P w.r.t. I is obtained in two steps:

1. Delete all rules $A \leftarrow b_1, \dots, b_i, \dots, b_n$ from P such that $I \not\models b_i$.
2. Replace remaining choice rules $\{e_1; \dots; a : l_1, \dots, l_m; \dots; e_k\} \prec u \leftarrow b_1, \dots, b_n$ by rules $a \leftarrow b_1, \dots, b_n, l_1, \dots, l_m$ for choice elements $a : l_1, \dots, l_m$ such that $a \in I$ and $I \models l_1, \dots, I \models l_m$.

Then, a model I of P is an *answer set* of P, if I is a $\subseteq$-minimal model of P^I. That is, all rules of P have to be satisfied w.r.t. I, and the (true) atoms in I must be "derivable" from the rules in P^I. A ground *query* $a?$ holds for P, if a belongs to every answer set of P. Moreover, let P^I_l denote the sum of integers w over all distinct tuples $(w, t_1, \dots, t_m)$ such that P contains some weak constraint $:\sim b_1, \dots, b_n\, [w@l, t_1, \dots, t_m]$ with $I \models b_1, \dots, I \models b_n$. An answer set I of P is *optimal*, if there is no answer set J of P such that $P^J_l < P^I_l$ for an integer l and $P^J_{l'} = P^I_{l'}$ for all integers $l' > l$.

The semantics of a non-ground program P, possibly including arithmetic terms, is given by the answer sets of its ground instantiation. To this end, variables in a rule or weak constraint r are distinguished into *global* variables, appearing outside of aggregate and choice elements in r, while the remaining variables are *local*. A *ground instance* of r is obtained in two steps:

1. Apply a *global substitution* σ that maps the global variables in r to ground terms yielding a rule $r\sigma$ without global variables.
2. Replace any aggregate or choice element e in $r\sigma$ by the collection of all aggregate or choice elements $e\theta$ obtainable by applying *local substitutions* θ that map the local variables in e to ground terms.

For example, applying the global substitution $\{X \mapsto 1\}$ to the rule $\{hc(X,Y) : arc(X,Y)\} = 1 \leftarrow node(X)$ gives $\{hc(1,Y) : arc(1,Y)\} = 1 \leftarrow node(1)$. Mapping the local variable Y to the nodes 1, 2, 3, and 4 then leads to the ground instance $\{hc(1,1) : arc(1,1); hc(1,2) : arc(1,2);$

[1] By the convention in Section 2.4 by Calimeri et al. (2012), $\#\mathtt{max}(\emptyset) < u$ and $\#\mathtt{min}(\emptyset) > u$ hold for every term u, so that all aggregates $\#\mathbf{agg}(T) \prec u$ can be evaluated w.r.t. any interpretation I.

$hc(1,3) : arc(1,3); hc(1,4) : arc(1,4)\} = 1 \leftarrow node(1)$. Note that applicable substitutions σ and θ are required to be *well-formed* (see Calimeri et al., 2012), that is, the arithmetic evaluation of arithmetic terms that do not contain variables must be well-defined. Global substitutions σ violating this condition cannot be used to obtain $r\sigma$ from a rule r, while a respective local substitution θ does not yield an instance $e\theta$ of an aggregate or choice element e.

The *ground instantiation* of a program P, denoted by $grnd(P)$, is the collection of all ground instances of rules or weak constraints in P obtainable by applying well-formed substitutions and evaluating arithmetic terms. Then, the (optimal) answer sets of P are the (optimal) answer sets of $grnd(P)$, possibly subject to a query in P, where the answer to a non-ground query $a?$ consists of all ground instances of atom a that belong to every answer set of P. Taking the prerequisites of instantiation procedures like "intelligent grounding" (Faber, Leone, & Perri, 2012) into account, non-ground ASP-Core-2 programs have to comply with additional requirements (Calimeri et al., 2012), including in particular finiteness of answer sets and safety. In a nutshell, these conditions require the availability of (positive) occurrences of variables within the bodies of rules as well as aggregate or choice elements in order to restrict the relevant substitutions and enable grounders to compute a (finite) ground program, which is typically much smaller yet equivalent to $grnd(P)$.

2.3 Program Properties

In the following, we recall some program properties that are relevant for categorizing ASP programs and corresponding solving approaches.

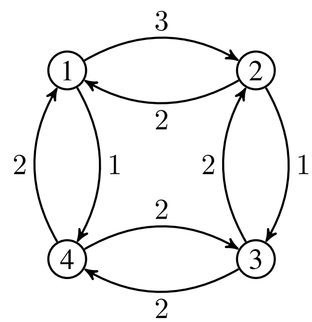
2.3.1 TIGHTNESS

Given a ground program P, the positive atom dependency graph of P contains the atoms in P as nodes and arcs from the head atoms $a_1, \dots, a_m$ of $a_1 \,|\, \dots \,|\, a_m \leftarrow b_1, \dots, b_n$ or a in $\{e_1; \dots; a : l_1, \dots, l_m; \dots; e_k\} \prec u \leftarrow b_1, \dots, b_n$ to the classical atoms among $b_1, \dots, b_n$ as well as $l_1, \dots, l_m$. We say that P is *non-tight* if some strongly connected component in its positive atom dependency graph contains an arc, and *tight* otherwise (Fages, 1994; Erdem & Lifschitz, 2003). For a tight program P, answer sets are in one-to-one correspondence with the propositional models of P's completion (Clark, 1978). In general, this property does not hold for non-tight programs. Note that our account of tightness could disregard aggregate atoms, as ASP-Core-2 restrictions on their usage (see Calimeri et al., 2012) do not permit circular (positive) dependencies through aggregates.

2.3.2 HEAD-CYCLE FREENESS

Depending on the class of programs under consideration, verifying $\subseteq$-minimality w.r.t. P^I can be tractable or computationally complex (Eiter & Gottlob, 1995). The syntactic property of *head-cycle-freeness* (Ben-Eliyahu & Dechter, 1994) allows for distinguishing such cases based on the positive atom dependency graph. A ground program P is called *head-cycle-free* (HCF), if there is no disjunctive rule $a_1 \,|\, \dots \,|\, a_m \leftarrow b_1, \dots, b_n$ in P such that two or more of the head atoms $a_1, \dots, a_m$ share some strongly connected component in the positive atom dependency graph of P; otherwise, P is *non-HCF*. Note that, if P is non-HCF, it is also non-tight, but not necessarily vice versa. The check whether a model is an answer set of P is tractable for HCF programs but coNP-complete for non-HCF programs.

(a) A directed graph



(b) Fact representation of the graph in (a)

$$
\begin{array}{lll}
arc(1,2) \leftarrow & cost(1,2,3) \leftarrow & node(1) \leftarrow \\
arc(2,1) \leftarrow & cost(2,1,2) \leftarrow & node(2) \leftarrow \\
arc(2,3) \leftarrow & cost(2,3,1) \leftarrow & node(3) \leftarrow \\
arc(3,2) \leftarrow & cost(3,2,2) \leftarrow & node(4) \leftarrow \\
arc(3,4) \leftarrow & cost(3,4,2) \leftarrow & \\
arc(4,3) \leftarrow & cost(4,3,2) \leftarrow & \\
arc(1,4) \leftarrow & cost(1,4,1) \leftarrow & \\
arc(4,1) \leftarrow & cost(4,1,2) \leftarrow &
\end{array}
$$

Figure 1: An example graph with arc costs that constitutes a TSP instance

2.4 Evaluation Process

Given a non-ground ASP program P, the computation of answer sets typically consists of two steps:

1. *Grounding* takes a non-ground ASP program as input and produces a ground program that has the same answer sets as the input program. Grounders, i.e., tools for grounding, implement techniques for generating ground programs that are often much smaller than the theoretical instantiation $grnd(P)$.

2. *Solving* takes a ground ASP program as input and computes one or more answer sets, which coincide with answer sets of the original non-ground program P.

To give some ideas how these steps work, let us consider the well-known Traveling Salesperson Problem (TSP). The directed graph with arc costs displayed in Figure 1(a) constitutes a particular problem instance. A corresponding representation in terms of ASP facts is shown in Figure 1(b), making use of the predicates $arc/2$, $cost/3$, and $node/1$ (determined by their names and arities) to specify arcs, their associated costs, and the nodes of the graph, respectively. Such facts are combined with a general encoding of TSP, which formulates conditions on (optimal) solutions for any TSP instance or set of facts, respectively. The following encoding is written in the ASP-Core-2 language:

$$
\begin{array}{rcll}
\{hc(X,Y) : arc(X,Y)\} = 1 & \leftarrow & node(X) & (1) \\
 & \leftarrow & node(Y), \#\mathtt{count}\{X : hc(X,Y)\} \neq 1 & (2) \\
reach(X) & \leftarrow & \#\mathtt{min}\{Y : node(Y)\} = X & (3) \\
reach(Y) & \leftarrow & reach(X), hc(X,Y) & (4) \\
 & \leftarrow & node(X), \mathbf{not}\ reach(X) & (5) \\
 & :\sim & hc(X,Y), cost(X,Y,C)\, [C@1, X, Y] & (6)
\end{array}
$$

Rules (1) and (2) express that, for every node, exactly one outgoing and one incoming arc must be part of a Hamiltonian cycle, i.e., a cycle that visits each node exactly once, and atoms over $hc/2$ represent the selected (outgoing) arcs. Rule (3) picks a starting node for the cycle, taken to be the lexicographically smallest one by using a $\#\mathtt{min}$ aggregate. Rule (4) derives further nodes reachable

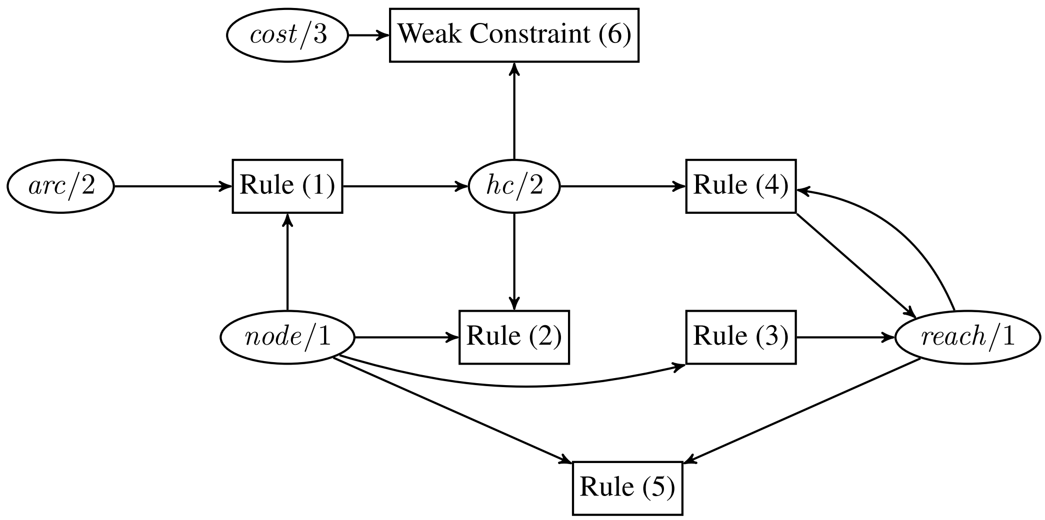


Figure 2: Predicate-rule dependency graph for the TSP encoding with rules/weak constraint (1)–(6)

via selected arcs, and rule (5), whose empty head stands for the constant $\perp$, denies isolated subcycles (indicated by a node that is not reachable). Finally, the objective of finding a Hamiltonian cycle with minimum cost is formulated by weak constraint (6), which penalizes any selected arc by its cost.

Given the facts specifying an instance and an encoding with (universally quantified) variables, the latter denoted by strings starting with uppercase letters, the grounding step consists of substituting variables by variable-free terms to obtain a ground program that has the same answer sets as the non-ground input. Virtually all ASP grounders exploit dependencies between non-ground rules, as given by the predicate-rule dependency graph in Figure 2 for the above TSP encoding, to schedule the order in which they process rules (and other constructs). The basic idea is that rules are relevant to the instantiation of another rule if instances of their head atoms may occur as preconditions in instances of the other rule. Hence, grounders process non-ground rules in topological order and use information about the head atoms of already generated ground instances for performing simplifications, still guaranteeing equivalence to the input program. For instance, facts over the predicate $arc/2$ can be used to restrict atoms over $hc/2$ in instances of rule (1) to pairs of nodes that are indeed connected by an arc, while any remaining pairs are disregarded, so that only a subset of the combinatorially feasible ground atoms over $hc/2$ is produced in practice. In fact, restricting atoms appearing in ground instances to head atoms is the primary means of ASP grounders for computing a small ground program that is equivalent to $grnd(P)$, given that underivable atoms must be false.

Regarding the graph in Figure 2, the mutual dependency between rule (4) and the predicate $reach/1$ indicates that the rule may produce head atoms that can in turn give rise to further ground instances. Existing grounding approaches differ mostly in the way they treat such positive recursion. Grounders performing *rule-wise instantiation*, including LPARSE (Syrjänen, 2001) and the first two versions of GRINGO (Gebser, Schaub, & Thiele, 2007), require that each variable in a rule occurs within some positive precondition whose predicate is topologically preceding the rule. For rule (4), this requirement is fulfilled in view of $hc(X, Y)$ in the body, and head atoms over $hc/2$ are determined before instantiating rule (4) relative to them. (The conditions required by LPARSE are slightly stricter, as each variable must be bound by a so-called domain predicate that can be deterministically evaluated during grounding. For instance, adding $arc(X, Y)$ to the body establishes this condition for rule (4).) Unlike that, the DLV grounder (Leone, Pfeifer, Faber, Eiter, Gottlob, Perri, & Scarcello, 2006; Faber et al., 2012) and recent versions of GRINGO (Gebser, Kaminski, König, & Schaub, 2011a; Gebser, Kaminski, & Schaub, 2015) are based on *semi-naive bottom-up evaluation* (cf. Abiteboul, Hull, & Vianu, 1995), only requiring safety, i.e., some occurrence in a positive precondition for each variable, to iteratively generate ground instances of recursive rules until no new head atoms are produced anymore.

To see the difference between rule-wise and semi-naive approaches, assume that the arcs from node 1 to 4 and from node 2 to 3 were not included in Figure 1. Then, the nodes 3 and 4 would be disconnected from the starting node 1 determined via rule (3). Given $reach(1)$, semi-naive bottom-up evaluation traces the available arcs, expressed in terms of the predicate $hc/2$, to iteratively produce instances of rule (4), whose head atoms may identify further reachable nodes. However, in view of the omitted edges, only two rule instances can be generated before this process stops:

$$\begin{array}{rcl} reach(2) & \leftarrow & reach(1), hc(1,2) \\ reach(1) & \leftarrow & reach(2), hc(2,1) \end{array}$$

That is, the atoms $reach(3)$ and $reach(4)$ are found underivable during grounding, and rule instances that would include them as positive preconditions are not produced. Rule-wise instantiation, on the other hand, aims to process rule (4) in a single pass, while making sure that all possibly applicable ground instances are generated. Hence, the partial information about (un)derivable atoms over $reach/1$ cannot be used for simplifications, and merely the arcs indicated by $hc/2$ are available to restrict ground instances accordingly. In addition to the two rule instances above, this leads to:

$$\begin{array}{rcl} reach(1) & \leftarrow & reach(4), hc(4,1) \\ reach(2) & \leftarrow & reach(3), hc(3,2) \\ reach(3) & \leftarrow & reach(4), hc(4,3) \\ reach(4) & \leftarrow & reach(3), hc(3,4) \end{array}$$

Observe that rule-wise instantiation results in one ground instance per arc, as given by $hc/2$, and thus constitutes a combinatorial instantiation procedure w.r.t. predicates whose rules have already been processed. Although the requirement of non-recursive positive dependencies for instantiation does not make any difference for many problem encodings, there are also meaningful encodings that cannot (realistically) be processed in a strictly rule-wise fashion. For instance, the following (positive) program can be used to reverse a list, given by a fact like $list(f(d, f(o, f(g, nil))))$:

$$\begin{array}{rcl} traverse(L, nil) & \leftarrow & list(L) \\ traverse(L, f(X, R)) & \leftarrow & traverse(f(X, L), R) \\ reverse(R) & \leftarrow & traverse(nil, R) \end{array}$$

While this encoding can be processed and even be evaluated to ground facts by means of semi-naive bottom-up evaluation, it requires recursion at the first-order level to deal with inputs of arbitrary length. In such a case, rule-wise grounding approaches reach their limits and are not viable anymore. As a consequence, rule-wise instantiation came out of fashion, and state-of-the-art ASP grounders like DLV and GRINGO harness semi-naive bottom-up evaluation.

Regarding ASP solving, common systems focus on ground programs. That is, the search for (optimal) answer sets amounts to Boolean constraint solving, as pioneered by the classical Davis-Putnam-Logemann-Loveland (DPLL) procedure (Davis & Putnam, 1960; Davis, Logemann, &

Loveland, 1962) and modern Conflict-Driven Clause Learning (CDCL) (Marques-Silva & Sakallah, 1999; Zhang, Madigan, Moskewicz, & Malik, 2001) in the area of SAT. In fact, ASP solvers are based on similar backtracking procedures, yet refined to the semantics of ASP programs. This requires that (positive) recursion is addressed at the ground level. For example, reachability in the TSP encoding in (1)–(6) must trace back to a unique starting node, such as 1 for the graph in Figure 1. As a consequence, atoms representing isolated subcycles, e.g., one between the nodes 1 and 4 and another between the nodes 2 and 3, are not acceptable within answer sets, and ASP solvers implement means to detect and handle such situations. Respective approaches, especially those implemented by participant systems in the competition, are surveyed in the next section.

3. ASP Solving Approaches and Participants

This section gives an overview of principal approaches to solving ground ASP programs (Kaufmann, Leone, Perri, & Schaub, 2016), guided by their historical evolution (Lierler, Maratea, & Ricca, 2016). Then, we particularly focus on the competition participants and systems.

3.1 ASP Solving Approaches

The approaches summarized below have in common that they rely on a grounding step, after which native and translation-based methods constitute the two main approaches to ASP solving. Moreover, portfolios and/or multi-processing can serve as means to blend or parallelize such basic techniques.

3.1.1 NATIVE APPROACHES

The first full-fledged ASP solvers, DLV (Leone et al., 2006) and SMODELS (Simons, Niemelä, & Soininen, 2002), were pioneered in the late ’90s and pursued “native” approaches based on the classical DPLL procedure. While DPLL augments basic backtracking search with unit propagation on clauses, DLV and SMODELS adjust such techniques to ASP programs. In particular, this includes dedicated inference mechanisms to detect and falsify so-called unfounded sets (Van Gelder, Ross, & Schlipf, 1991; Leone, Rullo, & Scarcello, 1997), which particularly address positive recursion in case of non-tight programs. Later on, DLV was extended with backjumping techniques (Ricca, Faber, & Leone, 2006), in place of basic backtracking, and look-back heuristics (Maratea, Ricca, Faber, & Leone, 2008) that take advantage of the backjumping process (Prosser, 1993). Similarly, the SMODELSCC solver (Ward & Schlipf, 2004) extended the algorithm of SMODELS with backjumping, while further adding mechanisms for conflict-driven clause learning, as pioneered by CDCL in the area of SAT.

Going beyond DPLL-based solvers (and their extensions), the second generation of native ASP solvers, including CLASP (Gebser, Kaufmann, & Schaub, 2012) and WASP (Alviano, Dodaro, Leone, & Ricca, 2015a), integrates CDCL-style search with propagation principles dedicated to ASP programs. Implementation features shared with modern SAT solvers include, e.g., watched literals (Moskewicz, Madigan, Zhao, Zhang, & Malik, 2001), activity-based heuristics (Biere & Fröhlich, 2015), and rapid restarts (Huang, 2007). Such basic features are accompanied by techniques for dealing with unfounded sets, aggregates, and optimization in order to cover the range of modeling concepts and computational tasks available in ASP (Kaufmann et al., 2016). While the IDP system (Bruynooghe, Blockeel, Bogaerts, De Cat, De Pooter, Jansen, Labarre, Ramon, Denecker, & Verwer, 2015) has been conceived as a model generator for first-order theories extended with inductive definitions, it has much in common with the aforementioned ASP grounders and solvers. That is, it includes a grounder, GIDL (Wittocx, Mariën, & Denecker, 2008), a solver, MINISATID (Mariën, Wittocx, Denecker, & Bruynooghe, 2008), and handles (positive) recursion among atoms in inductive definitions.

For solving non-HCF programs P, where checking whether a model of P is $\subseteq$-minimal w.r.t. P^I is in general coNP-complete, native ASP solvers employ a two-level architecture in which DPLL- or CDCL-style search is used for (i) generating candidate models of P and (ii) checking for the existence of smaller counter-models of P^I. To this end, the propagation principles of the first respective solver, DLV, are capable of handling disjunctive rules (Calimeri, Faber, Leone, & Pfeifer, 2006), while the check for counter-models is delegated to a SAT solver (Koch, Leone, & Pfeifer, 2003; Pfeifer, 2004). The GNT system (Janhunen, Niemelä, Seipel, Simons, & You, 2006) pursues a corresponding approach by casting the tasks of generating and checking candidate models to ASP programs processed with separate instances of SMODELS. Similarly, CLASP (or its nowadays deprecated sibling CLASPD) and WASP couple complementary instances of their CDCL-style search engines to perform model generation or checking, respectively. Notably, both CLASP and WASP internally encode the checking task for arbitrary candidate models (Gebser, Kaufmann, & Schaub, 2013; Alviano, Dodaro, & Ricca, 2015), and then perform assumption-based reasoning (Eén & Sörensson, 2003) to check the generated candidates.

3.1.2 TRANSLATION-BASED APPROACHES

Given that answer sets of a tight program P coincide with propositional models of P's completion (a theory in which rules sharing a head atom are taken as definition for the atom), the answer sets of tight programs can be computed by running SAT solvers. In particular, the first version (Babovich & Lifschitz, 2003) of the SAT-based solver CMODELS (Giunchiglia, Lierler, & Maratea, 2006) was based on this correspondence. As a generalization to the non-tight case, Lin and Zhao (2004) proposed loop formulas whose addition to a program's completion establishes correspondence between propositional models and answer sets. Since the number of required loop formulas can be exponential (Lifschitz & Razborov, 2006), the SAT-based solvers ASSAT (Lin & Zhao, 2004) and CMODELS, from its second version on, add loop formulas incrementally to eliminate models that are no answer sets. In fact, loop formulas deny unfounded sets (Lee, 2005), which are also handled by native systems, so that there is a close proximity between native and SAT-based solvers utilizing loop formulas, and both kinds of systems are based on similar search procedures. This also carries forward to non-HCF programs (Lee & Lifschitz, 2003), where the third version (Lierler, 2005) of CMODELS utilizes SAT solvers for generating and checking candidate models.

Complementing SAT-based solvers relying on loop formulas, the translation by LP2SAT (Janhunen, 2006; Janhunen & Niemelä, 2011) is based on so-called level rankings (Niemelä, 2008) to check $\subseteq$-minimality w.r.t. the reduct of an HCF program in the non-tight case. Such level rankings are encoded a priori, rather than incrementally, and expressing them in SAT requires sub-quadratic instead of exponential space. Technically, the tool LP2ACYC (Gebser, Janhunen, & Rintanen, 2014a) instruments an ASP program such that propositional models of its completion subject to an acyclicity condition match the answer sets of the program. The required acyclicity can then be established via level rankings, where linear representations are feasible in several target formalisms, including ASP, Pseudo-Boolean Constraints/Optimization, or SAT Modulo Acyclicity (Bomanson, Gebser, Janhunen, Kaufmann, & Schaub, 2016; Gebser, Janhunen, & Rintanen, 2014b), SMT with Difference or Bit-Vector Logic (Janhunen, Niemelä, & Sevalnev, 2009; Nguyen, Janhunen, & Niemelä, 2011), and Mixed Integer Programming (Liu, Janhunen, & Niemelä, 2012). In fact, the translation-based systems participating in the Sixth ASP Competition (see Section 3.2) are based on this infrastructure, while SAT-based solvers utilizing loop formulas have come out of fashion in view of the proximity to the basic search procedures of native ASP solvers.

3.1.3 PORTFOLIO APPROACHES

Automated algorithm selection techniques (Rice, 1976) aim at robustness across a range of heterogeneous inputs. Inspired by SATZILLA (Xu, Hutter, Hoos, & Leyton-Brown, 2008) in the area of SAT, the CLASPFOLIO system (Gebser, Kaminski, Kaufmann, Schaub, Schneider, & Ziller, 2011a; Hoos, Lindauer, & Schaub, 2014) uses support vector regression to learn scoring functions approximating the performance of several CLASP variants in a training phase. Given an instance, CLASPFOLIO then extracts features and evaluates such functions in order to pick the most promising CLASP variant for solving the instance. This algorithm selection approach was particularly successful in the Third ASP Competition (Calimeri, Ianni, & Ricca, 2014), held in 2011, where CLASPFOLIO won the first place in the NP category and the second place overall (without participating in the Beyond-NP category). The ME-ASP system (Maratea, Pulina, & Ricca, 2014, 2015b) goes beyond the solver-specific setting of CLASPFOLIO and chooses among different grounders as well as solvers. Grounder selection traces back to Maratea, Pulina, and Ricca (2013), and similar to the QBF solver AQME (Pulina & Tacchella, 2009), ME-ASP uses a classification method for performance prediction. Notably, "bad" classifications can be treated by adding respective instances to the training set of ME-ASP (Maratea, Pulina, & Ricca, 2015a), which enables an adjustment to new problems or instances thereof.

In contrast to selecting a single solving strategy from a portfolio, the ASPEED system (Hoos, Kaminski, Lindauer, & Schaub, 2015) indeed runs different solvers, sequentially or in parallel, as successfully performed by PPFOLIO (Roussel, 2011) in the 2011 SAT Competition (SAT-Comp, 2011). Given a benchmark set, a fixed time limit per instance, and performance results for candidate solvers, the idea of ASPEED is to assign time budgets to the solvers such that a maximum number of instances can be completed within the allotted time. In other words, the goal is to divide the total runtime per computing core among solvers such that the number of instances on which at least one solver successfully completes its run is maximized. The portfolio then consists of all solvers assigned a non-zero time budget along with a schedule which solvers to run on the same computing core. As calculating such an optimal portfolio for a benchmark set is an Optimization problem, it can be addressed by means of ASP, and ASPEED takes advantage of corresponding problem encodings.

3.1.4 MULTI-PROCESSING SUPPORT

The aforementioned ASPEED system allows for scheduling solver runs on multiple computing cores. Moreover, translation-based systems may readily exploit parallelism provided by respective back-end solvers, as done in the 2013 and 2014 editions of the ASP Competition. We below further focus on ASP systems with dedicated multi-core/processor support.

On the one hand, parallel grounding techniques were developed as extensions of LPARSE (Balduccini, Pontelli, El-Khatib, & Le, 2005) and the DLV grounder (Perri, Ricca, & Sirianni, 2013). The former approach was designed for distributing LPARSE incarnations, working in local memory, on Beowulf clusters, while the latter aims at shared-memory parallelism on multi-core/processor machines. In particular, the parallel version of the DLV grounder allows for concurrent instantiation

Table 1: Overview of submitted systems along with categories, sub-tracks, and query support

System	Category	ST #1	ST #2	ST #3	ST #4	Query
LP2NORMAL+CLASP	**SP**	√	√	√	√	
LP2ACYCASP+CLASP	**SP**	√	√	√		
LP2ACYCPB+CLASP	**SP**	√	√	√		
LP2ACYCSAT+CLASP	**SP**	√	√	√		
LP2ACYCSAT+GLUCOSE	**SP**	√	√	√		
LP2MIP	**SP**	√	√	√		
LP2MIP-MT	**MP**	√	√	√		
LP2SAT+LINGELING	**SP**	√	√			
LP2SAT+PLINGELING-MT	**MP**	√	√			
WASP+DLV	**SP**	√	√	√	√	√
WASP	**SP**	√	√	√	√	(√)
JWASP	**SP**	√	√			(√)
ME-ASP	**SP**	√	√	√	√	√

at several levels, i.e., regarding the strongly connected components in a predicate-rule dependency graph (cf. Section 2.4), distinct rules, or even the instances of a rule. This is accompanied by techniques for granularity control and dynamic load balancing to achieve an efficient parallelization.

On the other hand, parallel solving techniques concern the search for answer sets at the ground level. The first approaches in this direction (Finkel, Marek, Moore, & Truszczyński, 2001; Pontelli, Balduccini, & Bermudez, 2003; Balduccini et al., 2005; Gressmann, Janhunen, Mercer, Schaub, Thiele, & Tichy, 2006) were based on SMODELS and divided its DPLL-style search among cluster machines or multiple threads, primarily using guiding paths (Zhang, Bonacina, & Hsiang, 1996), pioneered in the area of SAT, to process separate subproblems in parallel. Cluster and multi-threaded versions of the CDCL-based solver CLASP (Gebser, Kaminski, Kaufmann, Schaub, & Schnor, 2011b; Gebser, Kaufmann, & Schaub, 2012), however, turned out to be particularly successful when applying a parallel portfolio of CLASP variants to a common problem. Recent versions of CLASP (Gebser, Kaminski, Kaufmann, Romero, & Schaub, 2015) further extend the multi-threaded search infrastructure to non-HCF programs (Gebser et al., 2013). Moreover, Dovier, Formisano, Pontelli, and Vella (2016) provide an approach to parallel CDCL-style ASP solving utilizing GPUs.

3.2 Participants and Systems

Table 1 surveys the thirteen systems submitted to the Fifth ASP Competition along with their categories, sub-tracks, and whether they participate on Query problems (cf. Sections 4 and 5). We below briefly summarize the approaches of these systems, grouped by their respective participant teams.

3.2.1 AALTO TEAM

The Aalto team from Aalto University submitted nine systems, all using GRINGO for grounding (and not participating on Query problems due to lacking support by GRINGO), but then apply different preprocessing/translation techniques and back-end solvers at the ground level. The first system, LP2NORMAL+CLASP, represents a native approach based on CLASP, where the preprocessor LP2NORMAL (Bomanson, Gebser, & Janhunen, 2014) is run beforehand to "normalize" aggregates of small to medium size, i.e., aggregates like #count and #sum are replaced by basic subprograms without such constructs when the resulting size increase is acceptable according to heuristics.

The other eight systems utilize translations based on the tool LP2ACYC (Gebser et al., 2014a; Bomanson et al., 2016), instrumenting ASP programs with acyclicity conditions to establish correspondence between the propositional models of a program's completion and answer sets in the non-tight case (cf. Section 3.1). As such equivalence is restricted to HCF programs, the translation-based systems do not participate in the Unrestricted sub-track (#4). However, the back-end solvers (indicated after "+" or the Mixed Integer Programming solver CPLEX in case of LP2MIP) of LP2ACYCASP+CLASP, LP2ACYCPB+CLASP, LP2ACYCSAT+CLASP, LP2ACYCSAT+GLUCOSE, LP2MIP, and LP2MIP-MT work with linear representations of acyclicity in terms of ASP, Pseudo-Boolean Constraints/Optimization, or (Max)SAT Modulo Acyclicity and numerical variables for level rankings when using Mixed Integer Programming. Their back-end solvers also support optimization, so that these systems can compete in the Optimization sub-track (#3). Moreover, LP2SAT+LINGELING and LP2SAT+PLINGELING-MT rely on a sub-quadratic representation of level rankings (for non-tight programs) in plain SAT without optimization, so that they participate in the Basic and Advanced Decision sub-tracks (#1 and #2) only. All systems but those indicated by "MT" run in the single-processor (**SP**) category, while LP2MIP-MT and LP2SAT+PLINGELING-MT exploit multi-threaded versions of their back-end solvers CPLEX or LINGELING, respectively, in the multi-processor (**MP**) category.

3.2.2 WASP TEAM

The Wasp team from the University of Calabria submitted two systems based on WASP, namely WASP+DLV and WASP, as well as the proof-of-concept prototype JWASP, written in Java. The hybrid WASP+DLV system relies on GRINGO for grounding all but Query problems, where the latter are handled by means of DLV whose dedicated support for Query answering (Alviano, Faber, Greco, & Leone, 2012) allows WASP+DLV to compete in all domains. Unlike that, the WASP system exploits GRINGO also on Query problems, utilizing simple syntactic means along with Query answering functionalities at the ground level (Alviano, Dodaro, & Ricca, 2014),[2] which is still a drawback in data-driven domains (cf. Section 5). The prototype system JWASP extends the (Max)SAT solver SAT4J (Le Berre & Parrain, 2010) with ASP-specific features for dealing with aggregates and (positive) recursion. This enables JWASP to participate in the Basic and Advanced Decision as well as the Optimization sub-tracks (#1, #2, and #3), like WASP addressing Query answering at the ground level. All three systems by the Wasp team run in the **SP** category.

3.2.3 ME-ASP TEAM

The ME-ASP team from the University of Genoa, the University of Sassari, and the University of Calabria submitted the multi-engine system ME-ASP. Its approach comprises the selection among grounders, namely GRINGO and DLV, as well as solvers (cf. Section 3.1) based on first-order or propositional features of an input program, respectively. The pool of ASP systems ME-ASP chooses from is built of submissions to the Fifth ASP Competition, held in 2014, including DLV, CLASP, WASP, and translation-based solvers. This range of back-ends allows ME-ASP to participate in all sub-tracks and domains, running in the **SP** category. Notably, ME-ASP is a new entrant to the ASP

[2] Rather than passing it to GRINGO, a query is intercepted, and the WASP solver is then run with options activating dedicated techniques for cautious reasoning as well as output filtering based on the predicate name appearing in the query.

Competition series and represents a novel portfolio approach, reviving and extending the idea of the CLASPFOLIO system whose last participation was in 2013.

Finally, it is worth mentioning that, to assess the improvements in ASP solving, we also consider the CLASP version of 2014, which won the first place in accumulation over sub-tracks, for reference.

4. Format of the Sixth ASP Competition

As outlined in Section 1, the Sixth ASP Competition maintains design decisions made in the 2014 edition, but also adds some novelties. First, the scoring scheme, which was significantly simplified in 2014 (cf. Calimeri et al., 2016), remains unchanged. We also group benchmarks in sub-tracks based on language features rather than inherent computational complexity, as in the previous edition. In the following, we describe the general format of the Sixth ASP Competition along with categories and sub-tracks, review the scoring scheme, and provide details on the competition platform.

4.1 Competition Format

The competition is open to solving systems based on the ASP-Core-2 input format. However, following the positive experiences of 2014, we also organized an on-site modeling event at LPNMR 2015, in the spirit of Prolog Programming Contests (Prolog-Comp). Regarding benchmarks, the Sixth ASP Competition featured a call for new domains, whose specific goal was to attract benchmarks (i) arising from applications of practical impact and/or (ii) that are "ASP-focused" by relying on non-tight problem encodings. The new domains together with those from the previous edition form the available benchmark collection (see Section 5). The whole benchmark set was evaluated during a benchmark selection stage, classifying instances according to their empirical hardness for covering a broad range of scalability in the competition (see Section 6).

4.1.1 COMPETITION CATEGORIES

The competition consists of *two categories*, depending on the computational resources made available to participant systems:

- **SP**: One processor (core) allowed;
- **MP**: Multiple processors (cores) allowed.

While the **SP** category aims at sequential systems, parallelism can be exploited in the **MP** category.

4.1.2 COMPETITION TRACKS

Both categories feature *two tracks*, called **Regular** and **Marathon** track. The idea of the Marathon track, inspired by the 2006 QBF Competition (QBF-Comp, 2006), is to assess the performance on challenging instances in the long run by granting more time to the top-performing systems of the Regular track. In view of just two systems in the **MP** category (cf. Section 3.2), we decided to skip the Marathon track in the **MP** category and ran both tracks in the **SP** category only.

4.1.3 COMPETITION SUB-TRACKS

The categories and tracks of the competition are structured into *four sub-tracks* each:

- **Sub-track #1**: *Basic Decision*. Encodings: non-disjunctive and non-choice rules (also called *normal* rules) with classical and built-in atoms only.
- **Sub-track #2**: *Advanced Decision*. Encodings: full language with queries, excepting weak constraints and non-HCF disjunction.
- **Sub-track #3**: *Optimization*. Encodings: full language with queries and weak constraints, excepting non-HCF disjunction.
- **Sub-track #4**: *Unrestricted*. Encodings: full language.

A benchmark domain then belongs to the upmost sub-track its problem encoding is compatible with.

4.2 Scoring Scheme

We adopt the scoring scheme of the Fifth ASP Competition. In particular, it complies with the following considerations:

- All domains are weighted equally.
- If a system outputs an incorrect answer to some instance in a domain, this invalidates its score for the domain, even if other instances can be solved correctly.
- In case of Optimization problems, the scoring takes solution quality into account.

In general, 100 points can be earned in each benchmark domain. The total score of a system is the sum of points over all domains.

4.2.1 Scoring Details

For *Decision and Query problems*, the score $S(D)$ of a system S in a domain D featuring N instances is calculated as

$$S(D) \quad = \quad \frac{N_S * 100}{N}$$

where N_S is the number of instances successfully solved within the allotted time and memory limits.

For *Optimization problems*, systems are ranked by solution quality, following the approach of the MANCOOSI International Solver Competition (MISC-Comp). Given M participant systems, the score $S(D, I)$ of a system S for an instance I in a domain D featuring N instances is calculated as

$$S(D, I) \quad = \quad \frac{M_S(I) * 100}{M * N}$$

where $M_S(I)$ is

- 0, if S did neither produce a solution nor report unsatisfiability, or otherwise
- the number of participant systems that did not produce any strictly better solution than S, where a confirmed optimum solution is considered strictly better than an unconfirmed one.

The score $S(D)$ of system S in domain D is then taken as the sum of scores $S(D, I)$ over the N instances I in D. Note that, as with Decision and Query problems, $S(D)$ can range from 0 to 100.

4.2.2 SYSTEM RANKING

In each track and sub-track of the same category, the participant systems are ranked by their sums of scores over all domains, in decreasing order. In case of a draw in terms of the sum of scores, the sums of runtimes over all instances are taken into account for tie-breaking.

4.3 Competition Platform

The competition is run on a Debian Linux server (64bit kernel), equipped with 2.30GHz Intel Xeon E5-4610 v2 Processors and 128GB RAM. Time and memory for each run are limited to 20 minutes wall-clock time and 12GB, respectively. Participant systems can exploit up to eight cores (16 virtual CPUs, given that Intel Hyperthreading technology is enabled) in the **MP** category, while the execution is constrained to one core in the **SP** category. The system execution is steered by a number of scripts adopted from the previous ASP Competition edition (cf. Calimeri et al., 2016).

5. Benchmark Suite

The benchmark domains considered in the Sixth ASP Competition include those from the previous edition, summarized first. Moreover, encodings and instances were provided for six new domains, introduced afterwards.

5.1 Previous Domains

The Fifth ASP Competition, held in 2014, featured 26 benchmark domains that had been submitted to earlier editions already, mainly in 2013 when the ASP-Core-2 standard input format was specified. In some domains, however, "unoptimized" encodings submitted by benchmark authors incurred grounding bottlenecks that made participant systems fail on the majority of instances. In view of this and in order to enrich the available benchmark collection, alternative encodings were devised and empirically compared in 2014 (cf. Calimeri et al., 2016) for all but two domains dealing with Query answering, which were modeled by rather straightforward positive programs.

The first part of assembling the benchmark suite for the Sixth ASP Competition consisted in the choice of encodings for previously used domains. Table 2 gives an overview of these domains, outlining application-oriented problems, i.e., problems that abstract or encode some real-world application of ASP (cf. Calimeri et al., 2016), respective computational tasks, i.e., Decision, Optimization, or Query answering, and sub-tracks. Moreover, the fourth column reports whether the encoding provided in 2013 or the alternative one made available in 2014 has been picked for the Sixth ASP Competition, and the fifth column indicates whether resulting ground instantiations are tight. The selection was based on the results from 2014, favoring the encoding that exhibited better performance of participant systems in a benchmark domain.

For Decision problems in the *Hanoi Tower*, *Knight Tour with Holes*, *Stable Marriage*, *Incremental Scheduling*, *Partner Units*, *Solitaire*, *Weighted-Sequence Problem*, and *Minimal Diagnosis* domains, all systems benefited from the usage of alternative 2014 encodings. Although corresponding performance results were not completely uniform, improvements of more systems or greater extent were obtained in *Graph Colouring*, *Visit-all*, *Nomystery*, *Permutation Pattern Matching*, and *Qualitative Spatial Reasoning*. In the *Sokoban* and *Complex Optimization* domains, no significant performance differences were observed, and 2014 encodings were picked as they simplify the original submissions, i.e., aggregates are omitted in *Sokoban* and redundant preconditions of rules

Table 2: Encodings selected for benchmark domains from the Fifth ASP Competition

Domain	App	Problem	Encoding	Tight	
Graph Colouring		Decision	2014	√	ST #1
Hanoi Tower		Decision	2014	√	ST #1
Knight Tour with Holes		Decision	2014		ST #1
Labyrinth		Decision	2013		ST #1
Stable Marriage		Decision	2014	√	ST #1
Visit-all		Decision	2014	√	ST #1
Bottle Filling		Decision	2013	√	ST #2
Graceful Graphs		Decision	2013	√	ST #2
Incremental Scheduling	√	Decision	2014	√	ST #2
Nomystery		Decision	2014	√	ST #2
Partner Units	√	Decision	2014	√	ST #2
Permutation Pattern Matching		Decision	2014	√	ST #2
Qualitative Spatial Reasoning		Decision	2014	√	ST #2
Reachability		Query	2013		ST #2
Ricochet Robots		Decision	2013	√	ST #2
Sokoban		Decision	2014	√	ST #2
Solitaire		Decision	2014	√	ST #2
Weighted-Sequence Problem		Decision	2014	√	ST #2
*Connected Still Life**		Optimization	2013		ST #3
Crossing Minimization	√	Optimization	2014	√	ST #3
Maximal Clique		Optimization	2014	√	ST #3
Valves Location	√	Optimization	2013		ST #3
Abstract Dialectical Frameworks		Optimization	2013		ST #4
Complex Optimization	√	Decision	2014		ST #4
Minimal Diagnosis	√	Decision	2014		ST #4
Strategic Companies		Query	2013		ST #4

dropped in *Complex Optimization*. Due to similar simplifications, the Basic Decision sub-track (#1) consists of six domains, while it previously included *Labyrinth* and *Stable Marriage* only. On the other hand, the encodings from 2013 were kept for domains in which alternative variants did not lead to improvements or even deteriorated performance, the latter applying to *Graceful Graphs*.

In view of the relative scoring of systems on Optimization problems, the selection of encodings could not be based on (uniform) improvements in terms of score here. Rather than that, we investigated runtimes, timeouts, and solution quality of the top-performing systems in 2014, thus concentrating on the feasibility of producing good but not necessarily optimal solutions. In this regard, the alternative 2014 encodings turned out to be advantageous in *Crossing Minimization* and *Maximal Clique*, while the original submissions led to better results in *Connected Still Life* and *Valves Location*, or essentially similar performance in *Abstract Dialectical Frameworks*.

Notably, the Sixth ASP Competition utilizes a revised formulation of *Connected Still Life* (thus marked by '*' in Table 2), where instances specify grid cells that must be "dead" or "alive" according to the Game of Life version considered in this domain. Such conditions are addressed by side

Table 3: New benchmark domains of the Sixth ASP Competition

Domain	App	Problem	Tight	
Combined Configuration	√	Decision		ST #2
Consistent Query Answering	√	Query	√	
MaxSAT	√	Optimization	√	ST #3
Steiner Tree	√	Optimization		
System Synthesis	√	Optimization		
Video Streaming	√	Optimization	√	

constraints added to the encoding from 2013 and enable a diversification of instances of same size, while size had been the only parameter for obtaining different instances before. In addition, the benchmark authors provided new instances of *Knight Tour with Holes*, *Stable Marriage*, *Ricochet Robots*, and *Maximal Clique*. For *Knight Tour with Holes*, the instances from 2014 were too hard for most participant systems, and too easy in the other three domains. Finally, recall that the 2013 encodings for Query problems in the *Reachability* and *Strategic Companies* domains are reused.

Six out of the 26 benchmark domains stemming from earlier ASP Competition editions are based on particular applications. In more detail, *Incremental Scheduling* (Balduccini, 2011) deals with assigning jobs to devices such that the makespan of a schedule stays within a given budget. The matching problem *Partner Units* (Aschinger et al., 2011) has applications in the configuration of surveillance, electrical engineering, computer network, and railway safety systems. *Crossing Minimization* (Gange, Stuckey, & Marriott, 2010) aims at optimized layouts of hierarchical network diagrams in graph drawing. The hydroinformatics problem *Valves Location* (Gavanelli, Nonato, & Peano, 2015) is concerned with designing water distribution systems such that the isolation in case of damages is minimized. In contrast to objective functions considered in the Optimization sub-track (#3), the *Complex Optimization* (Gebser, Kaminski, & Schaub, 2011b) domain addresses subset-minimization in the contexts of biological network repair (Gebser, Guziolowski, Ivanchev, Schaub, Siegel, Thiele, & Veber, 2010) and minimal unsatisfiable core membership (Janota & Marques-Silva, 2011). Finally, *Minimal Diagnosis* (Gebser et al., 2011) tackles the identification of subset-minimal reasons for inconsistencies between biological networks and experimental data.

5.2 New Domains

Table 3 lists the six new benchmark domains, all of which are application-oriented and half of them rely on non-tight problem encodings, that have been submitted to the Sixth ASP Competition. Their specific areas are summarized in the following:

- *Combined Configuration* (Gebser, Ryabokon, & Schenner, 2015b) is a Decision problem inspired by industrial product configuration tasks dealing with railway interlocking systems, automation systems, etc. In the considered scenario, orthogonal requirements as encountered in bin packing, graph coloring, matching, partitioning, and routing must be fulfilled by a common solution. Since the combined problem goes beyond its individual subtasks, specialized procedures for either subtask are of limited applicability, and the challenge is to integrate all requirements within general solving methods.

- *Consistent Query Answering* (Manna et al., 2013) addresses phenomena arising in the integration of data from heterogeneous sources. The goal is to merge as much information as possible, even though local inconsistencies and incompleteness typically preclude a mere data fusion. In particular, the Query problem amounts to cautious reasoning, retrieving consequences that are valid under all candidate repairs of input data.

- *MaxSAT* (Li & Manyà, 2009) is the optimization variant of SAT, where so-called soft clauses may be violated to particular costs and the sum of costs ought to be minimum. Industrial instances, taken from the 2014 MaxSAT Evaluation (MaxSAT-Comp, 2014), are represented by facts and encoded as an Optimization problem.

- *Steiner Tree* (Erdem & Wong, 2004) is concerned with connecting particular endpoints by a spanning tree. The rectilinear version of this problem, where points on a two-dimensional grid may be connected by line segments, is of practical relevance as it corresponds to wire routing in circuit design. Accumulated line segments then determine the total wire length, which ought to be minimum in the considered Optimization problem.

- *System Synthesis* (Biewer, Andres, Gladigau, Schaub, & Haubelt, 2015) deals with the allocation of parallel tasks and message routing in integrated hardware architectures for target applications. On the one hand, the capacities of processing elements are limited, so that communicating tasks must be distributed. On the other hand, network communication shall avoid long routes to reduce delays. The Optimization problem combines three lexicographically ordered objectives: balancing the allocation of processing elements, minimizing network communication, and keeping routes short.

- *Video Streaming* (Toni, Aparicio-Pardo, Simon, Blanc, & Frossard, 2014) aims at an adaptive regulation of resolutions and bit rates in a content delivery network. While the bit rates and different video formats that can be offered simultaneously are limited, service disruptions can be tolerated for a small fraction of users only. The objective of the Optimization problem is to achieve high user satisfaction with respect to particular video contents.

For each new domain, the benchmark authors provided problem instances along with an encoding in ASP-Core format. Similar to previous domains, we also furnished alternative encodings for comparison (cf. Section 7.1) and to provide options for customizing the benchmark suite in future ASP Competition editions. The only exception is the Query problem in *Consistent Query Answering*, where the encoding proposed by Manna et al. (2015) (in case of real-world instances) often enables ASP systems equipped with first-order Query answering functionalities to accomplish most of the required evaluation deterministically during grounding, and we did not further investigate the effect of modeling in this domain. For the remaining five domains, we below survey the main differences between the submitted encodings and the devised alternatives. (To keep the presentation simple, we concentrate on essential parts and simplified versions of the full encodings run in the competition (ASP-Comp, 2015).)

5.2.1 COMBINED CONFIGURATION

As mentioned above, problems in this domain involve elements of bin packing, graph coloring, matching, partitioning, and routing. In particular, each vertex of an input graph must be assigned to some color, and other subtasks focus on groups of vertices having the same color. Accordingly, a choice rule as follows forms the basis of the provided encoding:

$$\{\mathit{color}(V,C) : \mathit{col}(C)\} = 1 \quad \leftarrow \quad \mathit{vertex}(V) \tag{7}$$

As a matter of fact, the problem is such that particular colors are indistinguishable and serve only to partition the vertices at hand. However, in view of rule (7), any partition using n colors has $n!$ equivalent representations. Since such symmetry is usually not detected and explored by common ASP solving approaches (cf. Section 3), the availability of permutations leads to heavily increased combinatorics, which becomes a major bottleneck for hard instances with few or no solutions.

To avoid unnecessary guesses concerning the labels of colors, we devised an encoding variant such that colors must be assigned consecutively along a static order of input vertices, as determined by a domain predicate (i.e., a predicate that can be deterministically evaluated during grounding) $\mathit{next}/2$ in our alternative encoding. This is accomplished by means of the following subprogram:

$$\begin{aligned}
\mathit{admit}(V_2,C) &\quad\leftarrow\quad \mathit{next}(V_1,V_2), \mathit{color}(V_1,C-1), \mathit{col}(C) && (8)\\
\mathit{admit}(V_2,C) &\quad\leftarrow\quad \mathit{next}(V_1,V_2), \mathit{admit}(V_1,C) && (9)\\
&\quad\leftarrow\quad \mathit{color}(V,C), \mathit{col}(C-1), \mathbf{not}\ \mathit{admit}(V,C) && (10)
\end{aligned}$$

An atom of the form $\mathit{admit}(v_2,c)$, derived via rules (8) and (9), expresses that some vertex v_1 preceding v_2 in the given static order is assigned to color $c-1$. Note that atoms over $\mathit{next}/2$ provide direct successors, so that there is (at most) one ground instance of (8), indicating an assigned color to a direct successor, and one of (9), forwarding an assigned color along direct successors, per vertex v_2 and color c. Given this, rule (10) requires that color $c-1$ is assigned earlier than c according to the vertex order. As a consequence, any partition of vertices has a unique (rather than $n!$ many) representation in terms of colors, which narrows the number of solution candidates down. The full alternative encoding includes rules similar to (8)–(10) to disambiguate the assignment of vertices to bins as well, and further uses atoms like in the order encoding from SAT (Crawford & Baker, 1994; Tamura, Taga, Kitagawa, & Banbara, 2009) to abstract from particular colors and bins.

5.2.2 MAXSAT

Instances in this domain consist of clauses, i.e., disjunctions c of literals l given by facts of the form $\mathit{inClause}(c,l)$. A clause c can be either hard or soft, meaning that some literal in c must hold or that the violation of all contained literals incurs some cost, respectively. Then, the task is to find a consistent set of literals, including either x or $\neg x$ for each propositional variable x, such that the sum of associated costs is minimum. A literal x or $\neg x$ that holds is represented by $\mathit{pVar}(x)$ or $\mathit{nVar}(x)$, respectively, within an answer set, and an encoding has to evaluate clauses accordingly.

In order to generate consistent sets of literals, the provided encoding includes the following disjunctive rule:

$$\mathit{pVar}(X) \mid \mathit{nVar}(X) \quad \leftarrow \quad \mathit{vars}(X) \tag{11}$$

In view of the semantics of ASP programs (cf. Section 2.2), however, it turns out that instances of $\mathit{pVar}(X)$ and $\mathit{nVar}(X)$ are mutually exclusive. As a potential simplification exploiting such inverse polarity, our alternative encoding replaces (11) by:

$$\begin{aligned} \{pVar(X)\} &\leftarrow vars(X) && (12) \\ nVar(X) &\leftarrow vars(X), \mathbf{not}\ pVar(X) && (13) \end{aligned}$$

In fact, choice rule (12) expresses that any positive literal may unconditionally hold, thus reflecting the semantics of propositional logic. Negative literals are in turn derived via rule (13), defining the predicate $nVar/1$ as the complement of $pVar/1$, so that negative literals can be read off.

A second modification concerns the evaluation of (hard and soft) clauses, where

$$satisfied(C) \leftarrow clause(C), \#\texttt{count}\{L : inClause(C, L), holds(L)\} > 0 \qquad (14)$$

from the provided encoding is reformulated as follows:

$$satisfied(C) \leftarrow inClause(C, L), holds(L) \qquad (15)$$

That is, our alternative encoding omits the $\#\texttt{count}$ aggregate in (14) and uses basic rule (15) instead to check the satisfaction of clauses. With either encoding, an additional rule then denies unsatisfied hard clauses, and a weak constraint penalizes violated soft clauses.

5.2.3 STEINER TREE

In the rectilinear version of the Steiner Tree problem, a number of crossing points can be placed on a two-dimensional grid to form a spanning tree connecting given endpoints and the crossing points, so that the total length of the contained edges is minimum. The amount of possible crossing points is a crucial issue, where, e.g., 100×100-grids yield about 10000 possible locations. In this respect, the provided encoding leads to large ground instantiations, as it includes rules like the following:

$$\begin{aligned} \{cross(I, X, Y)\} &\leftarrow id(I), grid(x, X), grid(y, Y) && (16) \\ point(X, Y) &\leftarrow cross(I, X, Y) && (17) \\ point(X, Y) &\leftarrow given(X, Y) && (18) \\ \{edge(X_1, Y_1, X_2, Y_2)\} &\leftarrow point(X_1, Y_1), point(X_2, Y_2) && (19) \end{aligned}$$

Given a 100×100-grid, rules (16) and (17) for guessing and propagating the location of a crossing point roughly yield 10000 ground instances each, while the instantiation of rule (18) remains proportional to the endpoints specified by an instance. Moreover, rule (19) is particularly critical as its ground instances make the cross product of grid locations explicit, consisting of one million pairs for a 100×100-grid. To avoid such size blow-up, our alternative encoding splits the locations of crossing points by horizontal and vertical coordinates, and also uses a different edge representation:

$$\begin{aligned} \{cross(I, C, Z)\} &\leftarrow id(I), grid(C, Z) && (20) \\ \{edge(e(X_1, Y_1), e(X_2, Y_2))\} &\leftarrow given(X_1, Y_1), given(X_2, Y_2) && (21) \\ \{edge(e(X, Y), c(I))\} &\leftarrow given(X, Y), id(I) && (22) \\ \{edge(c(I_1), c(I_2))\} &\leftarrow id(I_1), id(I_2) && (23) \end{aligned}$$

The idea of rule (20) is to pick x- and y-coordinates of crossing points separately, which results in 200 ground instances per point for a 100×100-grid. In fact, the length of edges, taken as the endpoints' Manhattan distance and subject to minimization, can be determined coordinate-wise, so that there is no need to combine x- and y-coordinates, and the full alternative encoding makes use of order encoding techniques (Crawford & Baker, 1994; Tamura et al., 2009) for a compact representation. Most importantly, choice rules (21)–(23) express edges in terms of identifiers for their endpoints to avoid referring to the (guessed) coordinates of crossing points. To this end, the function term $e(x, y)$ denotes a given point at location (x, y), as specified by an instance, while $c(i)$ refers to a crossing point, which may be placed anywhere on the grid, in terms of its identifier. As a consequence, (21)–(23) yield $(m + n)^2$ ground instances for m given endpoints and a maximum number n of additional crossing points. In particular, the n^2 instances obtained from rule (23) constitute a crucial reduction compared to the cross product of grid locations expressed by (19), so that instantiations of the alternative encoding are substantially smaller. Notably, our alternative encoding also breaks symmetries among identifiers for crossing points in the same way as with *Combined Configuration* above, thus establishing a unique representation for any placement of crossing points.

5.2.4 SYSTEM SYNTHESIS

The Optimization problem in this domain, dealing with the design of integrated hardware architectures, combines three lexicographically ordered objectives: balancing the allocation of processing elements, minimizing network communication, and keeping routes short. In the provided encoding, these objectives are expressed by rules and weak constraints as follows:

$$
\begin{aligned}
\mathit{limit}(1000) &\leftarrow && (24)\\
\{\mathit{limit}(L_2)\} &\leftarrow \mathit{limit}(L_1), \mathit{next}(L_1, L_2) && (25)\\
&\leftarrow \mathit{proc}(P), \mathit{limit}(L), \#\mathtt{sum}\{U, T : \mathit{task}(T, U), \mathit{bind}(T, P)\} > L && (26)\\
&:\sim \mathit{next}(L_1, L_2), \mathbf{not}\ \mathit{limit}(L_2)\ [1@3, L_2] && (27)\\
&:\sim \mathit{route}(M, S, R)\ [1@2, M] && (28)\\
&:\sim \mathit{route}(M, S, R)\ [1@1, M, S, R] && (29)
\end{aligned}
$$

Rules (24)–(26) along with weak constraint (27) encode the first objective, where the predicate $\mathit{limit}/1$ provides candidate thresholds for the utilization of processing elements. While the threshold 1000 (representing 100.0%) is hard, gradually decreasing thresholds can be generated via choice rule (25), provided that the accumulated utilization by tasks on any processing element does not exceed the threshold, as checked in (26). Given that a missing instance of $\mathit{limit}(L_2)$ in an answer set is penalized by (27), a balanced distribution of tasks among processing elements is thus preferable. In addition, network communication, reflected by instances of $\mathit{route}(M, S, R)$ for directly connected senders S and receivers R along the route of a message M, is penalized by weak constraints (28) and (29), the former referring to routed messages M (i.e., communicating tasks are allocated to distinct processing elements) and the latter to direct connections along routes (reflecting route lengths).

Our alternative encoding varies the formulation of objectives and replaces (24)–(29) by:

$$
\begin{aligned}
\mathit{receive}(M, R) &\leftarrow \mathit{route}(M, S, R) && (30)\\
\mathit{exceed}(L) &\leftarrow \mathit{proc}(P), \mathit{threshold}(L), \#\mathtt{sum}\{U, T : \mathit{task}(T, U), \mathit{bind}(T, P)\} > L && (31)\\
&\leftarrow \mathit{exceed}(1000) && (32)\\
&:\sim \mathit{exceed}(L)\ [1@3, L] && (33)\\
&:\sim \mathit{receive}(M, R)\ [1@2, M] && (34)\\
&:\sim \mathit{receive}(M, R)\ [1@1, M, R] && (35)
\end{aligned}
$$

The idea of rule (31) is to avoid the guess of candidate thresholds in (25), but rather derive thresholds that are exceeded on some processing element, where rule (32) denies a utilization beyond 1000 or 100.0%, respectively, and weak constraint (33) penalizes excesses of lower thresholds. Moreover, auxiliary instances of $receive(M, R)$, derived via rule (30), drop senders and provide the receivers R of a message M only. (Although not shown here, the basic formulation of routes in the provided and our alternative encoding is such that any receiver gets a message from one sender only, which makes sense in view of the objective to keep routes short and thus omit redundant routing connections.) The weak constraints (34) and (35), addressing routed messages or direct connections, respectively, are then based on $receive/2$ instead of $route/3$, so that the number of ground instances is smaller.

5.2.5 VIDEO STREAMING

The objective in this Optimization problem is to achieve high satisfaction of users interested in videos of particular contents (cartoons, movies, sports, etc.), while the number of different video formats to broadcast and the available bandwidth are limited. Such conditions are formulated in the following encoding:

$$\{cast(V, B, S) : poss(V, B, S)\} = L \quad \leftarrow \quad limit(L) \tag{36}$$
$$\{assign(U, V, B, S)\} \quad \leftarrow \quad user(U, V, C, M), cast(V, B, S), B \leq C \tag{37}$$
$$\leftarrow \quad width(W), \#\texttt{sum}\{B, U, V : assign(U, V, B, S)\} > W \tag{38}$$
$$\leftarrow \quad serve(N), \#\texttt{count}\{U, V : assign(U, V, B, S)\} < N \tag{39}$$
$$:\sim \quad user(U, V, C, M), assign(U, V, B, S)\ [M - S@1, U, V] \tag{40}$$

Choice rule (36) expresses that exactly the given limit many instances of $cast(V, B, S)$ shall be selected among all possible video formats, where V stands for particular video contents, B for a bit rate, and S for the respective user satisfaction. Given this, choice rule (37) allows for assigning a selected format to a user, indicated by U, requesting the video contents, provided that the bit rate B does not exceed the capacity C of the user's connection. Two observations can be made regarding this pattern to assign video formats to users. On the one hand, a format selected via rule (36) does not necessarily have to be assigned to any user via (37), as some other format (providing higher satisfaction) or nothing at all may be picked instead. On the other hand, several selected formats of the same contents may in principle be assigned to the same user, and only bandwidth limitations and primarily penalties in view of the objective suppress such redundant assignments. In fact, rule (38) uses a $\#\texttt{sum}$ aggregate to make sure that video formats assigned to users do not exceed the available bandwidth, and (39) checks that at least a given number of users receive their requested video contents in some format. Finally, weak constraint (40) penalizes gaps $M - S$ between the satisfaction S associated with a video format assigned to a user and a fixed value M, where instances are built such that M represents the maximum satisfaction among formats that can be assigned in view of the capacity of the user's connection. This choice of M means that the assignment of any but the format providing highest satisfaction incurs some penalty, so that each user will be assigned at most one format of a contents in an optimal solution or answer set, respectively.

Our alternative encoding in this domain primarily addresses the selection of video formats to assign to users as well as the penalization of formats not providing highest satisfaction:

$$\{assign(U, V, B, S)\} \quad \leftarrow \quad user(U, V, C, M), poss(V, B, S), B \leq C \tag{41}$$
$$leq(U, V, S) \quad \leftarrow \quad assign(U, V, B, S) \tag{42}$$

$$\begin{array}{rcll} leq(U,V,S_1) & \leftarrow & next(U,V,S_1,S_2), leq(U,V,S_2) & (43) \\ & \leftarrow & next(U,V,S_1,S_2), leq(U,V,S_2), assign(U,V,B,S_1) & (44) \\ & \leftarrow & limit(L), \#\texttt{count}\{V,B,S : assign(U,V,B,S)\} > L & (45) \\ & :\sim & next(U,V,S_1,S_2), leq(U,V,S_2)\ [S_1 - S_2@1, U, V, S_1] & (46) \end{array}$$

Rather than picking formats first, choice rule (41) directly provides potential assignments with respect to users' connection capacities. Then, rules (42) and (43) derive instances of $leq(U,V,S)$, expressing that a format of satisfaction S or lower is assigned to a user U requesting video contents V. To this end, instances of $next(U,V,S_1,S_2)$, where $next/4$ is a domain predicate defined by the full alternative encoding, chain successive satisfaction levels S_1 and S_2 for video contents V and formats assignable to user U in decreasing order. Given this, rule (44) denies an assignment of several formats (with distinct satisfaction levels) to the same user, thus handling the inherent redundancy of such assignments by a hard constraint rather than leaving it as subject to optimization. Moreover, the limit on formats that can be broadcast simultaneously is checked by rule (45), which replaces the original choice in (36), while rules (38) and (39) remain unchanged and are not repeated above for brevity. Finally, weak constraint (46) utilizes instances of $leq(U,V,S_2)$, indicating an assigned as well as higher satisfaction levels S_2, to implement a differential penalization scheme based on gaps $S_1 - S_2$ between successive levels S_1 and S_2. This approach reflects higher satisfaction of a user in terms of a subset of penalized atoms in an answer set, which can sometimes lead to an improved solving performance (see, e.g., Gebser, Kaminski, Kaufmann, & Schaub, 2012).

6. Benchmark Selection

In the following, we describe the selection of benchmark instances that were run in the Sixth ASP Competition. The goal of this step was to make an informed selection balancing the hardness of instances per domain, even in cases where the majority of available instances, provided by benchmark authors, tend to be too easy or overly hard and thus rather uninformative to rank participant systems. To this end, we applied an instance selection strategy inspired by the 2014 SAT Competition (SAT-Comp, 2014), going beyond the random selection approaches of earlier editions (cf. Calimeri et al., 2014, 2016). First, we evaluated the empirical hardness of all available instances by running the top-performing systems from the previous ASP Competition edition, held in 2014, and by classifying the instances according to the performance of these reference systems. Then, the instance selection aims at balancing the obtained hardness classes per domain, using an ASP program to compensate for underpopulated classes by adjusting the numbers of instances to pick from each class accordingly. The two steps, instance classification by hardness and balanced selection among classes, are detailed below.

6.1 Instance Classification

To evaluate the hardness of instances, we ran the top-performing system per team participating in the Fifth ASP Competition, corresponding to the systems taking the first three places in 2014, i.e., CLASP, LP2NORMAL2+CLASP, and WASP-1.5. This choice comes close to the virtual state-of-the-art solver that matches the top-performing system on each instance. Each run was limited to 20 minutes for grounding and 40 minutes for solving, granting 12GB memory for each of the two steps. The allotted resources exceed those of actual competition runs, where 20 minutes and 12GB are available for both grounding and solving, in order to also capture runs that a reference system completes within some tolerance. The obtained performance results are then taken to classify instances into hardness categories by picking the upmost class such that some of the following conditions applies:

[non-groundable] Instances such that none of the three systems finished grounding in 20 minutes.

[very easy] Instances completed by all three systems in less than 20 seconds solving time.

[easy] Instances completed by all three systems in less than 2 minutes solving time.

[medium] Instances completed by all three systems in less than 20 minutes solving time.

[hard] Instances completed by at least one system in 40 minutes (twice the timeout) solving time.

[too hard] Instances such that none of the three systems finished solving in 40 minutes.

Running the reference systems exhaustively on all available instances, belonging to the 32 benchmark domains introduced in Section 5, where the encodings indicated in Table 2 are used for previous domains and those provided by benchmark authors for new domains, took about 212 CPU days on the competition platform. Given that participant systems in the Sixth ASP Competition are based on similar solving approaches as the three reference systems (cf. Section 3), non-groundable and very easy instances are likely to remain uninformative regarding the ranking of systems, and we decided to drop such instances. In four domains, *Bottle Filling*, *Hanoi Tower*, *Solitaire*, and *Weighted-Sequence Problem*, this left too few suitable instances, so that we had to exclude these domains. The actual competition thus featured 28 domains, among which 12 are application-oriented.

6.2 Instance Selection

For each of the 28 domains used for ranking participant systems, the selection task was to pick 20 instances to run in the competition. As mentioned above, non-groundable and very easy instances were excluded, so that the selection is made among easy, medium, hard, and too hard instances. Perfect balancing would then consist of picking four instances per hardness class and another four instances freely in order to guarantee that each hardness class contributes at least 20% of the instances in a domain. The objective of this division is to cover a range of scalability, including instances that are in reach for newcomer or yet unoptimized participant systems as well as challenging instances that may indicate the progress of and differentiate state-of-the-art systems.

In practice, however, the division must also account for the availability of instances and compensate for underpopulated hardness classes. To this end, we increased the numbers of instances to pick from other hardness classes whenever a class lacks the four intended instances. In more detail, let the labels e, m, h, and t denote the classes of easy, medium, hard, and too hard instances. For any subset C of $\{e, m, h, t\}$, by $\#C$ we refer to the number of instances belonging to some of the hardness classes in C. Moreover, we arrange subsets of $\{e, m, h, t\}$ in a binary tree as follows:

$$\begin{tikzcd}
& & & \{e,m,h,t\} \arrow[dll] \arrow[drr] & & & \\
& \{e,m\} \arrow[dl] \arrow[dr] & & & & \{h,t\} \arrow[dl] \arrow[dr] & \\
\{m\} & & \{e\} & & \{h\} & & \{t\}
\end{tikzcd}$$

For the three inner nodes C (the non-singletons), let $\mathit{left}(C)$ and $\mathit{right}(C)$ be the child nodes of C, where the idea is that instances belonging to $\mathit{left}(C)$ are preferred for compensation. In particular, this applies to medium instances in $\mathit{left}(\{e,m\}) = \{m\}$ in case a domain lacks the intended eight hard or too hard instances. The following equations make this balancing scheme precise:

$$\begin{array}{rcl} \mathit{target}(\{e,m,h,t\}) & = & 16 \\ \mathit{target}(\mathit{left}(C)) & = & \min\{\#\mathit{left}(C), \mathit{target}(C) - \min\{\#\mathit{right}(C), |\mathit{right}(C)| * 4\}\} \\ \mathit{target}(\mathit{right}(C)) & = & \min\{\#\mathit{right}(C), \mathit{target}(C) - \mathit{target}(\mathit{left}(C))\} \end{array}$$

After doing the calculation, the numbers of instances to pick from hardness classes are read off the leaves of the above binary tree. For example, when a domain includes 12 easy, 6 medium, 1 hard, and 5 too hard instances, we obtain $\mathit{target}(\{e,m\}) = 10$, $\mathit{target}(\{h,t\}) = 6$, $\mathit{target}(\{m\}) = 6$, $\mathit{target}(\{e\}) = 4$, $\mathit{target}(\{h\}) = 1$, and $\mathit{target}(\{t\}) = 5$, where two additional medium and one too hard instance compensate for the lack of three hard instances. Given the availability of at least 20 instances per domain in appropriate hardness classes, our balancing scheme makes sure that 16 instances will be picked based on hardness and augmented with four freely chosen instances.

As a secondary criterion beyond hardness, we consider the satisfiability of instances, where respective information is available whenever at least one of the three reference systems completed its run on an instance (or produced some solution in case of Optimization problems). The idea is to split the number of instances to pick from a hardness class evenly, provided that sufficiently many instances have been determined as satisfiable or unsatisfiable, respectively. Letting $\#C[\mathit{sat}]$ and $\#C[\mathit{unsat}]$ denote the corresponding satisfiable or unsatisfiable, respectively, instances belonging to a set C of hardness classes, the goal of an even division of instances to pick from C can be expressed as follows, where $s \in \{\mathit{sat}, \mathit{unsat}\}$:

$$\mathit{divide}(C,s) \quad = \quad \min\{\#C[s], \left\lfloor \frac{\mathit{target}(C)}{2} \right\rfloor\}$$

However, such secondary division based on satisfiability is neutral except for instances of Decision and Query problems that are not too hard, as the available instances of Optimization problems are satisfiable (or there would be nothing to optimize), and the reference systems do not yield satisfiability information for too hard instances otherwise. Also, $\mathit{divide}(C,s) > \max\{0, \mathit{target}(C) - \#C[\overline{s}]\}$ has to hold for $\overline{\mathit{sat}} = \mathit{unsat}$ and $\overline{\mathit{unsat}} = \mathit{sat}$ to yield a non-trivial secondary condition.

We encoded the balanced instance selection by an ASP program in ASP-Core-2 format, where the results of running the three reference systems are provided by facts, $\mathit{target}(C)$ and $\mathit{divide}(C,s)$ are determined by domain predicates (deterministically evaluated during grounding), and an additional domain predicate $\mathit{instance}/3$ represents the file names of instances in a domain along with their hardness and satisfiability. The following combinatorial encoding part then expresses conditions on the selection of instances:

$$\{\mathit{select}(I,C,S) : \mathit{instance}(I,C,S)\} = 20 \quad \leftarrow \tag{47}$$

$$\leftarrow \quad c \in \{e,m,h,t\}, \#\mathtt{count}\{I : \mathit{select}(I,c,S)\} < \mathit{target}(\{c\}) \tag{48}$$

$$\leftarrow \quad c \in \{e,m,h,t\}, s \in \{\mathit{sat}, \mathit{unsat}\}, \#\mathtt{count}\{I : \mathit{select}(I,c,s)\} < \mathit{divide}(\{c\}, s) \tag{49}$$

That is, choice rule (47) generates candidate sets of 20 instances in a domain, while rules (48) and (49) check that at least the number of target instances are picked per hardness class and that the latter are evenly divided based on satisfiability, provided that a respective condition is non-trivial.

The higher the number of appropriate instances in a domain, the more loose are the conditions expressed by (47)–(49), and an answer set found is solver-specific. Pragmatically, we picked the instances provided in the first answer set computed by CLASP, using its options *rand-freq*, *sign-def*, and *seed* for guaranteeing reproducible randomization. In particular, we fixed the random seed to the concatenation of winning numbers in the EuroMillions lottery of 23rd June 2015. The resulting instances run in the competition and tools to reproduce the instance selection are available on-line (ASP-Comp, 2015).

7. Competition Results

This section presents the results of the Sixth ASP Competition. We first announce the winners in the **SP** category and analyze their performance, and then proceed to systems in the **MP** category.

7.1 Results in the SP Category

After summarizing the results of the Regular track, we provide details on each sub-track. We further assess the improvements in solving performance by comparing the top-performing systems in this ASP Competition edition to those from the previous edition, held in 2014. Moreover, we evaluate the alternative encodings for five of the six new benchmark domains introduced in Section 5.2. Finally, we turn to the Marathon track in which the top-performing systems were given more time.

7.1.1 REGULAR TRACK

Figure 3(a) shows the scores of participant systems, ordered by their sums of scores over all sub-tracks. Accordingly, the first three places in the Regular track go to the systems:

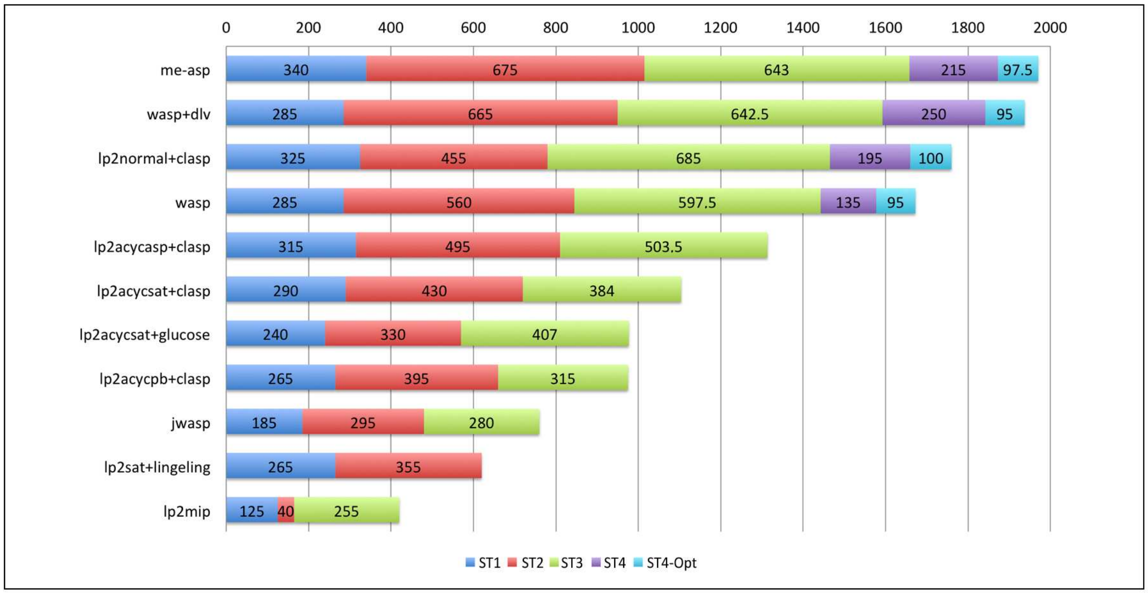
1. ME-ASP, by the ME-ASP team, with 1971 points;
2. WASP+DLV, by the Wasp team, with 1938 points;
3. LP2NORMAL+CLASP, by the Aalto team, with 1760 points.

That is, the first place is taken by a system pursuing the portfolio approach outlined in Section 3, and the following two places by native systems. Notably, the overall winner ME-ASP is a newcomer participating for the first time in this ASP Competition edition.

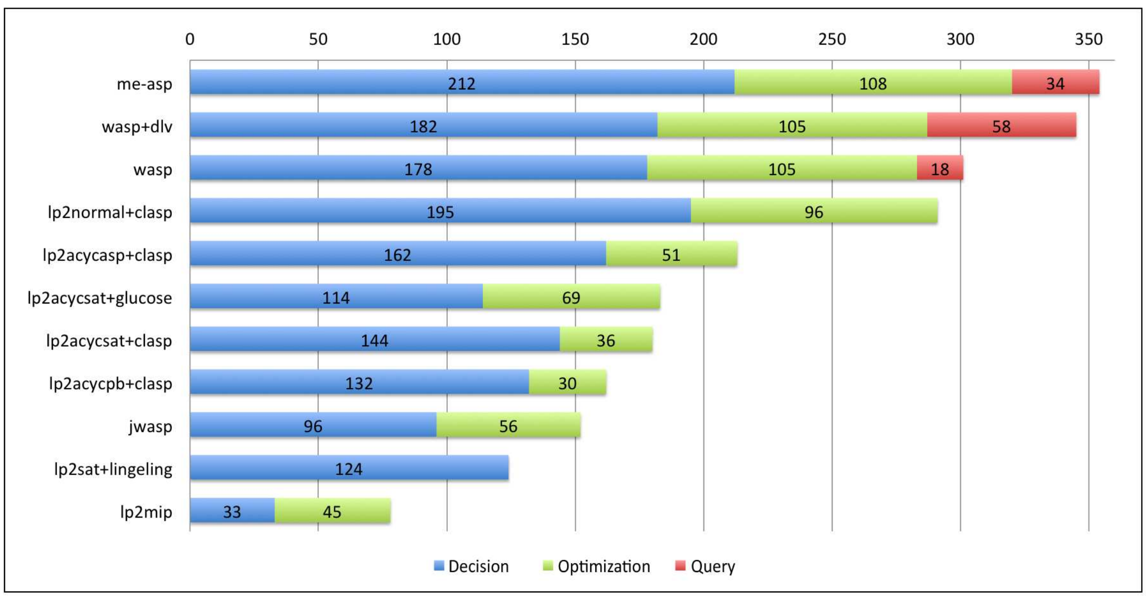
Within sub-tracks, the three top-performing systems overall take the first places as well:

- Sub-track #1 (Basic Decision): ME-ASP with 340 points;
- Sub-track #2 (Advanced Decision): ME-ASP with 675 points;
- Sub-track #3 (Optimization): LP2NORMAL+CLASP with 685 points;
- Sub-track #4 (Unrestricted): WASP+DLV with 345 points.

Note that Figure 3(a) distinguishes the Optimization problem *Abstract Dialectical Frameworks* in sub-track #4 in view of its dedicated scoring scheme, as described in Section 4. The total scores in sub-track #4, however, include all four domains listed in Table 2, given that up to 100 points can be earned in each of them.



(a) Score acquisition per sub-track



(b) Number of solved instances (confirmed optima) per computational task

Figure 3: Overall results of the Regular track

Figure 3(b) views solving performance from the perspective of successfully completed runs relative to computational tasks, i.e., Decision, Optimization, and Query answering. For Decision and Query problems (belonging to sub-tracks #1, #2, and #4), the numbers of solved instances are directly proportional to scores, and we observe that ME-ASP performed particularly well on Decision and WASP+DLV on Query problems. Regarding Optimization problems (in sub-tracks #3 and #4), the correlation between runs completed with a confirmed optimum solution and scores is not that immediate, since the quality of best solutions found comes into play in case of timeouts. In fact, ME-ASP, WASP+DLV, and WASP completed more runs on Optimization problems than LP2NORMAL+CLASP, which still achieved the highest scores, as given in Figure 3(a). This divergence is due to the use of different optimization strategies, namely model- versus core-guided approaches (Morgado, Heras, Liffiton, Planes, & Marques-Silva, 2013; Alviano et al., 2015a; Alviano, Dodaro, Marques-Silva, & Ricca, 2015b; Gebser et al., 2015), where the former are geared for producing good-quality solutions and the latter for confirming optimum solutions.[3] As ME-ASP, WASP+DLV, and WASP utilize core-guided optimization, they are able to complete more runs than LP2NORMAL+CLASP, whose model-guided approach yields better solutions in case of timeouts. That is, the choice of an appropriate optimization strategy depends on the expected hardness of instances as well as the objective whether to find good-quality solutions or having optimality confirmed. In this respect, the available benchmark collection includes a significant portion of instances for which confirmed optimum solutions are beyond reach, so that the scoring scheme favors model-guided optimization. On the other hand, the number of suitable instances and possibly domains were heavily reduced in case runs on Optimization problems had to be completed to achieve a non-zero score, which is the primary reason for taking solution quality into account.

The cactus plot in Figure 4 further displays the solved instances of Decision and Query problems, whose number increases along the x-axis, within the runtime given on the y-axis. The curves allow for distinguishing four groups of systems by their performance. First, ME-ASP outperforms the other participants in terms of the number of instances solved within a given runtime, yet closely followed by WASP+DLV. In fact, the detailed results provided in Tables 4 and 5 in the appendix yield excellent performance of ME-ASP on six Decision problems and of WASP+DLV on three Decision as well as three Query problems. The latter advantage is based on dedicated techniques for cautious reasoning (Alviano et al., 2014), while ME-ASP relies on stand-alone DLV for Query answering. Query problems also separate the first from the second group, consisting of WASP and LP2NORMAL+CLASP, where WASP applies the same cautious reasoning techniques as WASP+DLV to "unoptimized" instantiations at ground level and LP2NORMAL+CLASP provides no support at all. The third group of systems comprises JWASP as well as the translation-based systems by the Aalto team that are limited to HCF programs. Among them, LP2ACYCASP+CLASP performs best, given that it merely instruments non-tight programs with acyclicity conditions. The detailed results on *Knight Tour with Holes* in Tables 4 and 5 show that such conditions can sometimes be advantageous, as also confirmed by the comparable performance of LP2ACYCPB+CLASP and LP2ACYCSAT+CLASP, which vary the underlying problem representation formalism, in this domain. While LP2SAT+LINGELING, utilizing the plain SAT solver LINGELING, has disadvantages on non-tight programs, it is interesting to note that it is the only system solving all (tight) instances of *Visit-all*. The back-end solver of LP2ACYCSAT+GLUCOSE turns out to be less successful on the competition benchmarks, so that it falls behind the other translation-based systems in the third group. It is still ahead of the prototype system JWASP, which is implemented in Java and thus not tuned for low-level performance. The LP2MIP system, relying on Mixed Integer Programming, constitutes the last group, as the approach of its back-end solver CPLEX is different from DPLL- or CDCL-style search and aims primarily at problems including numerical variables, rather than Boolean variables only.

[3]Model-guided approaches take the quality of a best solution found as (strict) upper bound for the next solution to produce, and thus improve solution quality "from above". Unlike that, core-guided approaches successively admit more penalties based on the unsatisfiability of tighter quality bounds, aiming to converge to an optimal solution "from below".

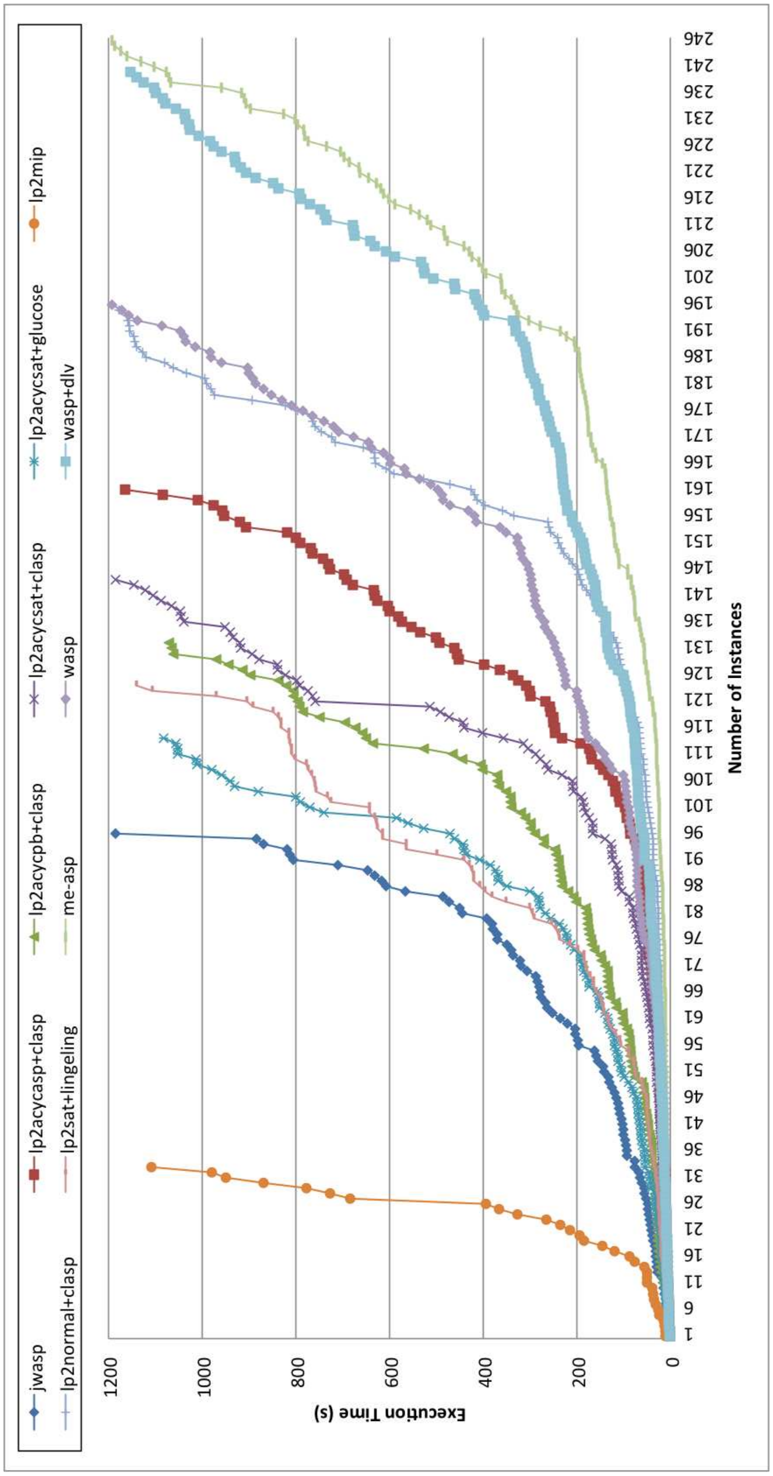


Figure 4: Cactus plot for Decision and Query problems

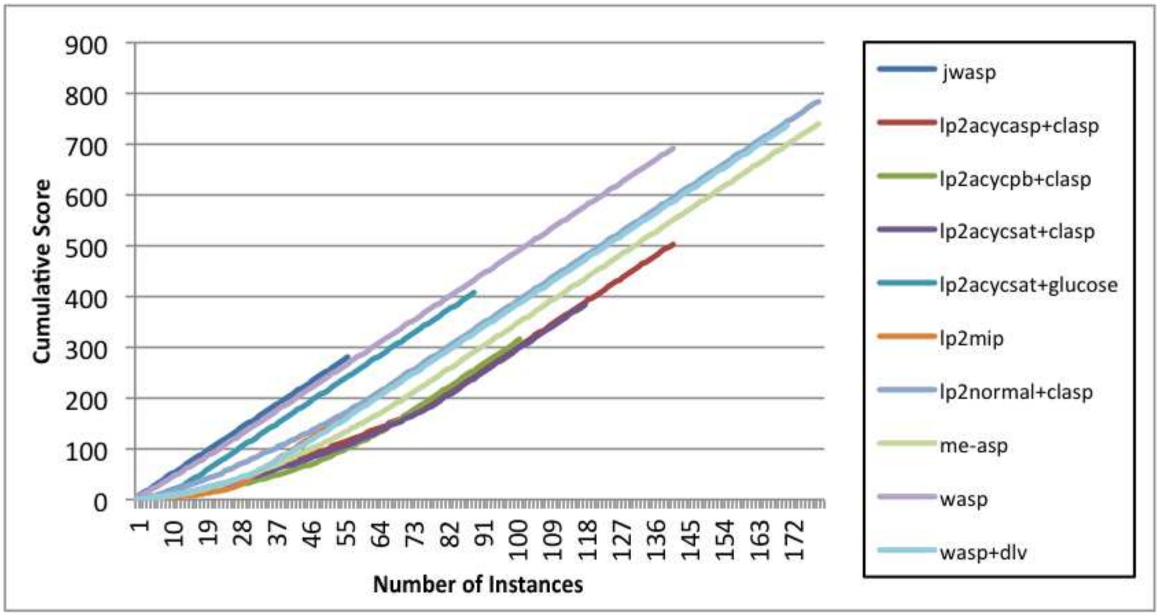


Figure 5: Score acquisition plot for Optimization problems

Figure 5 plots the score acquisition on Optimization problems, where the non-zero scores of a system are arranged in increasing order and the cumulative sum over the number of instances on the x-axis is displayed on the y-axis. Right-most points thus yield total scores achieved on Optimization problems belonging to sub-tracks #3 and #4. The steep curves of JWASP, WASP, and LP2ACYCSAT+GLUCOSE indicate that these systems tend to produce either a confirmed optimum solution or no solution at all, which matches the typical behavior of core-guided optimization approaches. Unlike that, the three top-performing systems on Optimization problems (and overall), LP2NORMAL+CLASP, ME-ASP, and WASP+DLV, exhibit a gentler slope, given that their optimization strategies put stronger focus on finding good-quality solutions in case a confirmed optimum solution is beyond reach. In particular, WASP+DLV combines model- and core-guided optimization approaches, LP2NORMAL+CLASP pursues a model-guided strategy, and ME-ASP frequently picks the model-guided CLASP system from 2014 for solving. The remaining translation-based systems, LP2ACYCASP+CLASP, LP2ACYCPB+CLASP, LP2ACYCSAT+CLASP, and LP2MIP, are less successful than the native LP2NORMAL+CLASP system, even though three of them are based on the same back-end solver. In fact, Tables 4 and 5 show that the large ground instantiations faced in the *Steiner Tree* domain constitute a bottleneck for translation tools and lead to memory outs. Moreover, aggregates involving large weights, as in the *Valves Location* domain, as well as lexicographically ordered objectives, as encountered in *System Synthesis*, impose issues for the translations made by LP2ACYCPB+CLASP, LP2ACYCSAT+CLASP, and LP2MIP.[4] However, systems that may seem disadvantageous over all domains may still be useful in particular cases, e.g., LP2ACYCSAT+GLUCOSE and LP2MIP are the only systems achieving perfect scores in *Connected Still Life* and *Crossing Minimization*, respectively.

[4] Due to technical problems in the handling of large weights that could not be fixed before running the competition, LP2ACYCPB+CLASP had to be disqualified in *Valves Location* and LP2ACYCSAT+GLUCOSE as well as LP2MIP-MT in *Video Streaming*.

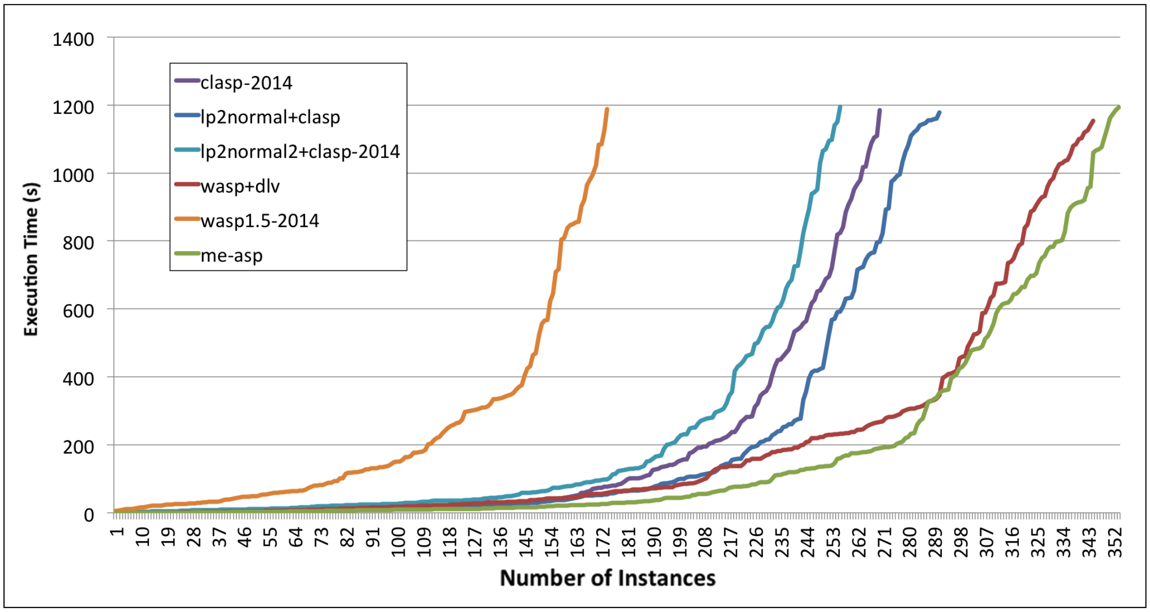


Figure 6: Comparison to reference systems from 2014

7.1.2 ADVANCEMENTS IN ASP SOLVING

The cactus plot in Figure 6 compares the top-performing systems from the previous edition, namely CLASP, LP2NORMAL2+CLASP, and WASP-1.5 (indicated by the suffix "-2014"), to those of the Sixth ASP Competition: ME-ASP, WASP+DLV, and LP2NORMAL+CLASP. While the winner system from 2014, CLASP, could complete 270 instances (confirmed optima in case of Optimization problems) out of those run in this edition, LP2NORMAL+CLASP, WASP+DLV, and ME-ASP solved 21, 75, or 84 instances more, respectively. When comparing LP2NORMAL+CLASP and WASP+DLV to their earlier versions from 2014, LP2NORMAL2+CLASP and WASP-1.5, the improvements of the current systems amount to 35 more solved instances for LP2NORMAL+CLASP and 171 more for WASP+DLV. The additional margin of the new entrant ME-ASP, which solved 11 instances more than WASP+DLV, further outlines the benefit of a portfolio approach along with well-configured algorithm selection for tackling the variety of competition benchmarks. In summary, these performance results exhibit significant advancements of the state of the art and growing maturity of ASP systems, despite of only one year development time since the previous ASP Competition edition.

7.1.3 IMPACT OF MODELING ON NEW DOMAINS

As described in Section 5.1, alternative encodings were devised for almost all of the domains available in 2014, and corresponding performance results served as basis for selecting the encodings to use in this ASP Competition edition. Similarly, we furnished alternative encodings for five of the six new domains introduced in Section 5.2, and below contrast the outcomes of running the three top-performing systems, LP2NORMAL+CLASP, ME-ASP, and WASP+DLV, on the alternative encodings and those provided by benchmark authors, which were used to rank participant systems. This gives insights into the effectiveness of modeling approaches as well as indications for customizing the benchmark suite in future ASP Competition editions.

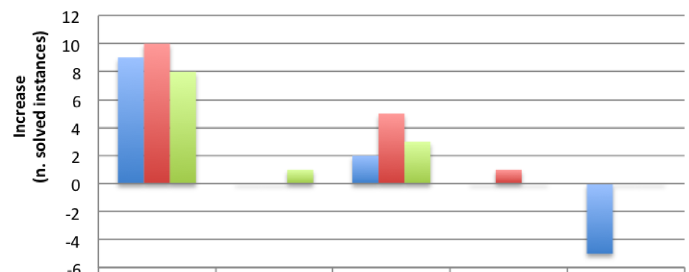
Figures 7(a)–(c) view the performance differences due to the use of alternative encodings from several perspectives, concerning the number of successfully completed runs (confirmed optima in case of Optimization problems), the number of timeouts, and the number of instances for which some solution was found. For the Decision problem in the *Combined Configuration* domain, these three measures coincide, and we observe significant improvements thanks to the alternative encoding: 9, 10, or 8 more instances, respectively, solved by LP2NORMAL+CLASP, ME-ASP, and WASP+DLV. As detailed in Table 7 in the appendix, LP2NORMAL+CLASP and ME-ASP each complete 16 out of 20 instances with the alternative encoding, but the relative increase is highest for WASP+DLV, which solves 9 instances instead of one only with the original encoding. Such performance improvements are owed to compactly formulated symmetry breaking in the alternative encoding (cf. Section 5.2), in order to avoid a combinatorial explosion due to redundant representations of solution candidates.

The other four domains address Optimization problems, where an increase or decrease of the number of solutions found in Figure 7(c) indicates additional or fewer instances, respectively, on which a non-zero score could be achieved. In *MaxSAT*, however, we do not observe such effects, but merely one more instance solved with a confirmed optimum solution by WASP+DLV when run on the alternative encoding. This fortifies the position of WASP+DLV as the system that performs best on *MaxSAT*, now providing confirmed optima for all 20 instances. However, given that both the original and alternative encoding for *MaxSAT* are rather straightforward, where modifications amount to minor syntactic simplifications, such a performance difference is modest as well.

In the *Steiner Tree* domain, our alternative encoding aims to reduce the large ground instantiations obtained with the original encoding, which has a significant impact on solving performance. Most notably, the use of the alternative encoding abolishes seven memory outs of WASP+DLV, as indicated in Figure 7(c) by the increase of solutions found. In fact, the four additional timeouts of WASP+DLV in Figure 7(b) relieve previous memory outs and do not constitute a performance decline, given that some, yet not necessarily optimal, solution is found on each instance. Similarly, LP2NORMAL+CLASP and ME-ASP produce solutions for all instances, where one run of the former failed with the original encoding. In fact, performance improvements in terms of confirmed optimum solutions, shown in Figure 7(a), apply to all three systems: 2, 5, or 3 more instances, respectively, solved by LP2NORMAL+CLASP, ME-ASP, and WASP+DLV. Interestingly, ME-ASP overtakes LP2NORMAL+CLASP and completes six instances with a confirmed optimum solution, one more than LP2NORMAL+CLASP and two more than WASP+DLV, when switching from the original to our alternative encoding. These performance results further indicate an increased empirical hardness of instances in the *Steiner Tree* domain in comparison to *MaxSAT*.

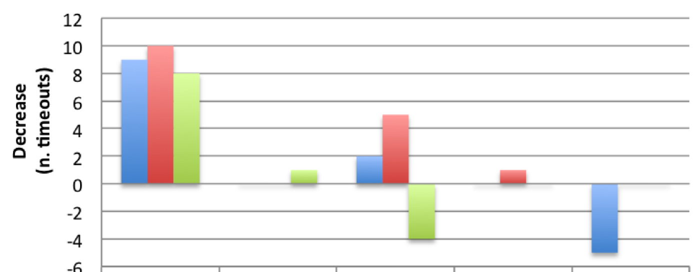
As described in Section 5.2, our alternative encoding for *System Synthesis* boils down to a syntactic reformulation of the three lexicographically ordered objectives in this domain. This leads to rather modest performance differences, i.e., one more instance solved with a confirmed optimum solution by ME-ASP, which increases its number of completed runs from 7 to 8. The fact that penalties in the three objectives amount to zero for all solved instances, while LP2NORMAL+CLASP and WASP+DLV could not provide confirmed optima with either encoding, is particularly striking. We checked that ME-ASP picked the CLASP version of 2014 for solving, whose "aggressive" model-guided optimization strategy turns out to perform well on instances admitting unpenalized solutions.

While the effect of alternative encodings was positive or neutral in the domains considered so far, we observe a clear performance decline of all three systems with the alternative encoding for *Video Streaming*, given that the numbers of instances for which some solution is found decrease in Figure 7(c). Moreover, LP2NORMAL+CLASP, which produces the most, i.e., 15, confirmed optimum



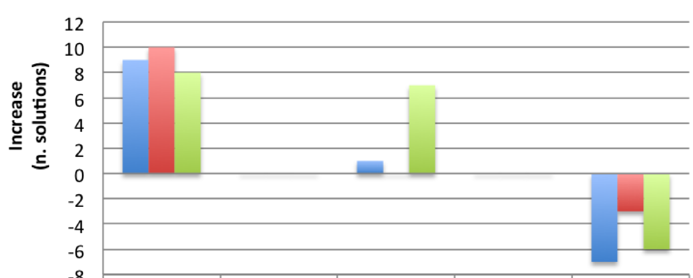
	Combined Config.	MaxSAT	SteinerTree	System Synthesis	Video Streaming
lp2normal+clasp	9	0	2	0	-5
me-asp	10	0	5	1	0
wasp+dlv	8	1	3	0	0

(a) Differences in terms of solved instances (confirmed optima)



	Combined Config.	MaxSAT	SteinerTree	System Synthesis	Video Streaming
lp2normal+clasp	9	0	2	0	-5
me-asp	10	0	5	1	0
wasp+dlv	8	1	-4	0	0

(b) Differences in terms of timeouts



	Combined Config.	MaxSAT	SteinerTree	System Synthesis	Video Streaming
lp2normal+clasp	9	0	1	0	-7
me-asp	10	0	0	0	-3
wasp+dlv	8	0	7	0	-6

(c) Differences in terms of solutions found

Figure 7: Comparison between original and alternative encodings for new domains

solutions when run on the original encoding, falls back to 10 confirmed optima (one more than ME-ASP and WASP+DLV) with our alternative encoding. We conclude that indirection introduced in the alternative encoding to establish a differential penalization scheme as well as hard constraints suppressing redundant solutions (cf. Section 5.2) is counterproductive for the optimization approaches of participant systems. This reminds that, beyond apparent effects regarding the (deterministic) behavior of grounders, the impact of modeling on solving performance remains difficult to predict.

In summary, clear performance improvements due to modeling could be achieved on the Decision problem in the *Combined Configuration* domain and the Optimization problem in the *Steiner Tree* domain. These advantages are owed to symmetry breaking, cutting down a vast number of redundant representations of solution candidates, as well as compact formulations, reducing the size of ground instantiations by some order of magnitude. On the other hand, the modest performance differences observed in the *MaxSAT* and *System Synthesis* domains, where our alternative encodings constitute minor reformulations of the original encodings provided by benchmark authors, indicate a certain robustness of ASP systems towards syntactic perturbations. Finally, the performance decline due to the alternative encoding for *Video Streaming* emphasizes that the impact of particular modeling approaches remains problem-specific and requires empirical investigation, considering that corresponding techniques were found to work well in other domains (cf. Gebser et al., 2012).

7.1.4 MARATHON TRACK

In the Marathon track, the three top-performing systems of the Regular track, namely LP2NORMAL+CLASP, ME-ASP, and WASP+DLV, are granted more time, i.e., 3 hours rather than 20 minutes only, in order to assess their performance on challenging instances in the long run. Since runs of these systems are reproducible, instances completed within less time given in the Regular track (confirmed optima in case of Optimization problems) are solved in the Marathon track as well, so that the focus is on the remaining instances that could not be completed in the Regular track.

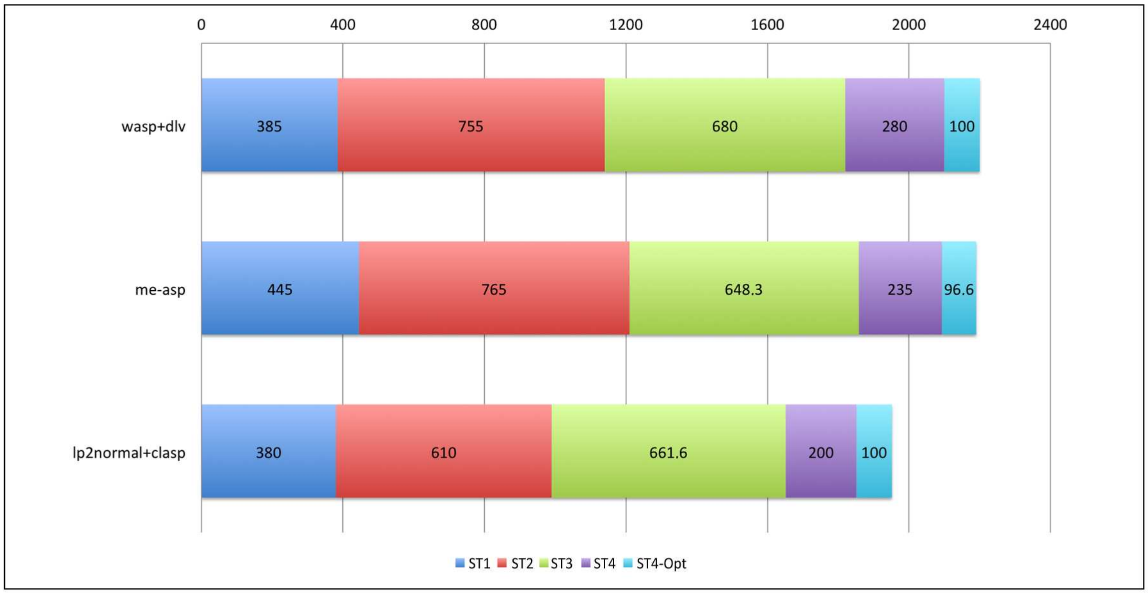
Similar to Figure 3 above, Figures 8(a) and 8(b) show the scores as well as the numbers of instances solved by the three participant systems, listed in the order of their sums of scores over all sub-tracks. This yields the following ranking in the Marathon track:

1. WASP+DLV, by the Wasp team, with 2200 points;

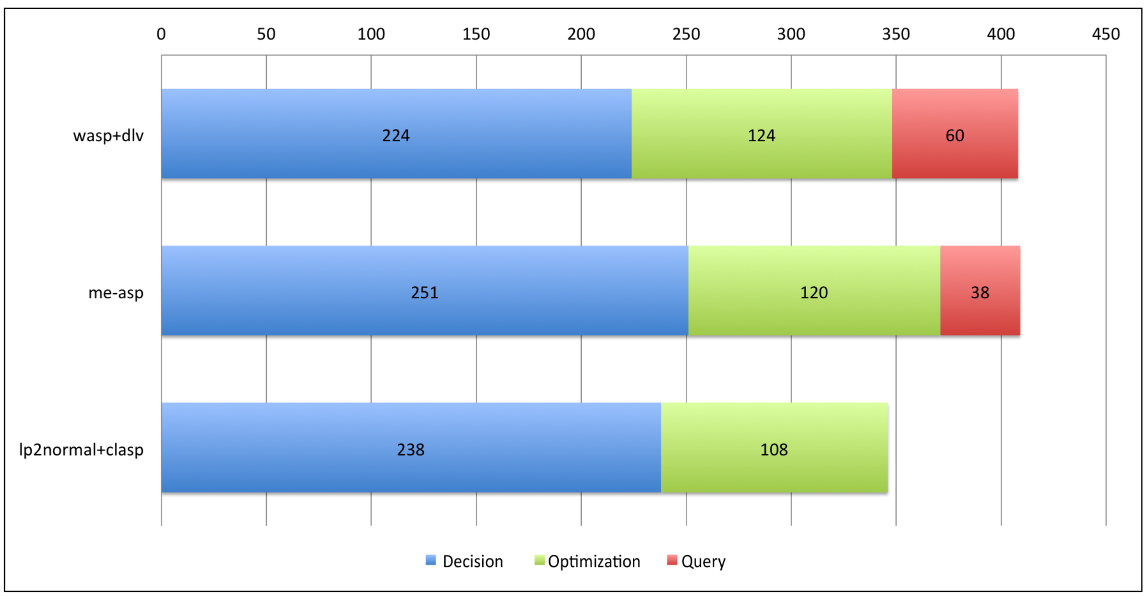
2. ME-ASP, by the ME-ASP team, with 2190 points;

3. LP2NORMAL+CLASP, by the Aalto team, with 1952 points.

In comparison to the Regular track, ME-ASP and WASP+DLV switch their positions on the first two places, owed to the greater improvement of WASP+DLV on Optimization problems in sub-track #3. The combination of model- and core-guided optimization approaches implemented by WASP+DLV is indeed not geared for producing many solutions quickly, but aims at confirmed optima or good-quality approximations of an optimum solution. Moreover, note that the algorithm selection method of ME-ASP was trained relative to the timeout of the Regular track, while the allotted time in the Marathon track is increased by a factor of nine. The margin between the two systems, however, remains virtually imperceptible, and Figure 8(b) even yields one more run completed by ME-ASP. We also note that the gap to LP2NORMAL+CLASP is primarily due to its lacking support for Query problems, while regarding solved instances all three systems benefit near uniformly from the additional runtime in comparison to the Regular track: both LP2NORMAL+CLASP and ME-ASP complete 55 more



(a) Score acquisition per sub-track



(b) Number of solved instances (confirmed optima) per computational task

Figure 8: Overall results of the Marathon track

runs, and WASP+DLV improves by 63. In particular, the detailed results provided in Table 6 in the appendix exhibit increased numbers of instances solved by the respective top-performing system in 19 out of the 28 domains, where exceptions are twofold. On the one hand, some system completes all instances in *Qualitative Spatial Reasoning*, *Reachability*, *Abstract Dialectical Frameworks*, *Minimal Diagnosis*, and *Strategic Companies* in the Regular track already, which does not leave any

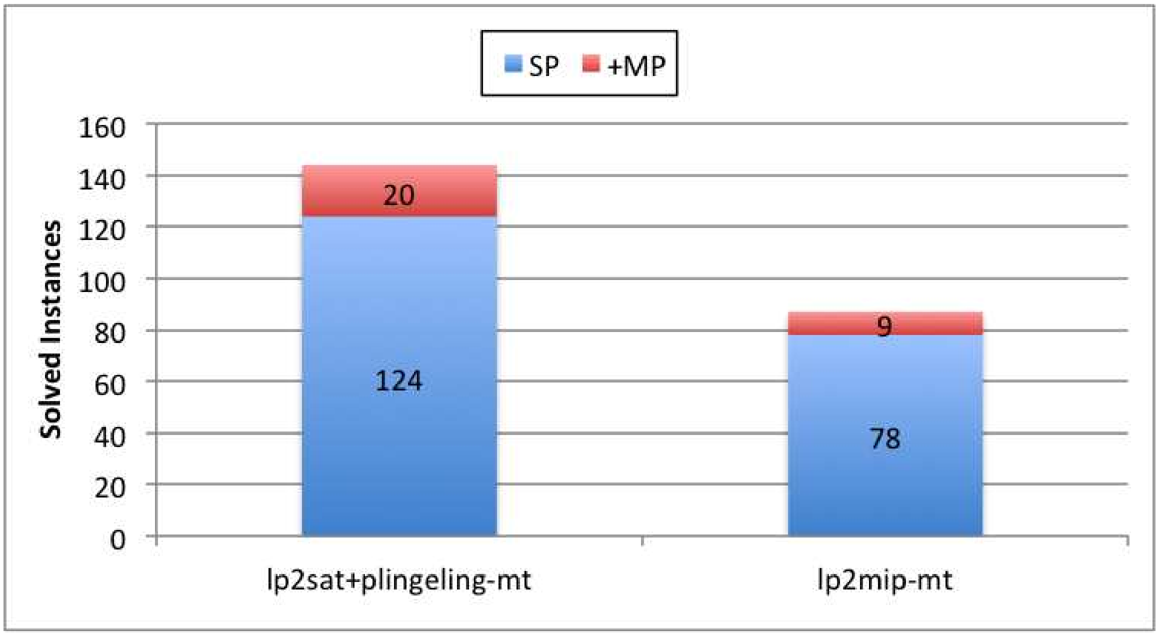


Figure 9: Improvement in terms of solved instances (confirmed optima)

room for improvement. On the other hand, in *Knight Tour with Holes*, *Incremental Scheduling*, *Sokoban*, and *Video Streaming*, instances not solved in the Regular track remain too hard in the Marathon track as well. In general, the results for the top-performing participant systems indicate that performance trends are almost independent from time limits.

7.2 Results in the MP Category

The **MP** category, granting eight computing cores instead of a single one per run, features two translation-based systems, LP2MIP-MT and LP2SAT+PLINGELING-MT, exploiting multi-threaded versions of their back-end solvers CPLEX or LINGELING, respectively. Both systems utilize six parallel threads and participate on Decision problems in sub-tracks #1 and #2, while LP2MIP-MT is the only participant on Optimization problems in sub-track #3 of the **MP** category. Hence, the comparison between multi-threaded versions and their single-threaded reference systems in Figure 9 includes eight more domains for LP2MIP-MT, whose total score of 560 is still lower than the 720 points earned by LP2SAT+PLINGELING-MT. Regarding the number of completed runs, we observe that switching from single- to multi-threading leads to 20 and 9 more instances solved by LP2SAT+PLINGELING-MT or LP2MIP-MT, respectively, so that LP2SAT+PLINGELING-MT maintains the edge of its single-threaded counterpart LP2SAT+LINGELING over LP2MIP (also in terms of total scores). That is, the multi-threaded versions of sequential systems achieve improvements but do not change the overall picture, and parallel portfolios, extending the idea of ME-ASP, may probably be even more effective.

8. Related Competitions

A detailed comparison between past editions of the ASP Competition and various competitions in neighboring areas has been provided by Calimeri et al. (2016). In the following, we focus on the novelties introduced in this ASP Competition edition.

As mentioned in Section 6, the instance selection strategy of the Sixth ASP Competition is inspired by the 2014 SAT Competition (SAT-Comp, 2014). In comparison, our classification includes two more hardness categories: “non-groundable” instances, which have no counterpart in SAT in view of its propositional input format, and “very easy” instances, distinguished from “easy” ones in order to make a fine-grained instance selection covering a range of scalability. The instance classification of the 2014 SAT Competition was based on running five solvers that participated before in the 2012 edition. Then, the selection aimed at a 50-50 ratio between benchmarks categorized as “medium” or “hard”, while disregarding “easy” instances and allowing “too hard” ones within a 20% pool of freely picked instances. In contrast to this approach, we decided to drop “very easy” instances only and to include “too hard” ones as a regular class in balanced instance selection, in order to present challenging instances to participants advancing over earlier reference systems and to leave room for improvement in the Marathon track. The latter has been inspired by the 2006 QBF Competition (QBF-Comp, 2006), accounts for instances that are “too hard” in the Regular track, and also intends to assess the impact of time limits on performance results.

The 2014 edition of the International Planning Competition (IPC-Comp, 2014; Vallati, Chrpa, Grzes, McCluskey, Roberts, & Sanner, 2015) employed an instance selection strategy with similar goals as ours, i.e., picking instances that are neither too easy nor overly hard. The main difference to our approach is the usage of actual participant systems for evaluating the empirical hardness of instances and selecting 20 suitable ones per domain. Building on current participants for selecting the benchmarks used for ranking the same systems is somewhat problematic, as at least in theory it biases the selection towards systems that dominate in particular domains, while less specialized systems performing well in general may be degraded and appear worse than deserved.

Further competitions featuring an instance selection stage based on empirical hardness include the SMT Competition (SMT-Comp). E.g., SMT solvers submitted in 2011 have been used to classify instances for the 2012 edition (Cok, Griggio, Bruttomesso, & Deters, 2012), and hardness categories contribute different percentages of instances depending on whether a domain is concerned with application, crafted, or random benchmarks. While the ASP Competition does not distinguish the contexts of benchmarks, the underlying domains, more than 40% of which were application-oriented in this edition, give some indications. Moreover, the benchmark collection of the 2016 QBF Competition (QBF-Comp, 2016) was composed of instances considered suitable according to several parameters, such as empirical hardness, syntactic and structural features, and satisfiability, where 10 instances per domain were then picked at random. The Automated Theorem Proving Competition series (ATP-Comp; Sutcliffe, 2016) relies on benchmarks from the TPTP (Thousands of Problems for Theorem Provers) library (Sutcliffe, 2009), which associates instances with numerical ranks, determined by the results of running state-of-the-art systems on them. A benchmark set is then formed from instances with “intermediate” ranks, in order to exclude too easy or overly hard ones, while also aiming to balance the numbers of instances picked per domain and to make sure that at least 50% of the selected instances are new, i.e., have not been run in previous editions. Unlike that, the MaxSAT Evaluation (MaxSAT-Comp) and the 2016 edition of the Pseudo-Boolean Competition (PB-Comp, 2016) pick instances at random among those submitted by benchmark authors, which resembles the approaches of earlier ASP Competition editions.

9. Conclusions and Future Directions

This paper reported about the Sixth ASP Competition, which maintained design decisions made in the previous edition, but also introduced some novelties, including a benchmark selection stage, a Marathon track, and an extended benchmark set featuring six new application-oriented domains. The goals were to (i) assemble a benchmark suite covering a range of scalability, (ii) assess solving performance on challenging instances in the long run, and (iii) put stronger focus on benchmarks arising from applications of practical impact. In the following, we discuss the experiences made and outline potential directions for future ASP Competition editions as well as ASP research in general.

9.1 Portfolio Approaches

In view of the growing maturity of ASP systems, developing a competitive system from scratch is all but an easy task. Hence, portfolio approaches that can take advantage of existing technology constitute a worthwhile alternative, as underpinned by the first place of the newcomer system ME-ASP in the Regular track. Given that algorithm selection is an active research area of its own (see, e.g., Bischl, Kerschke, Kotthoff, Lindauer, Malitsky, Fréchette, Hoos, Hutter, Leyton-Brown, Tierney, & Vanschoren, 2016), there is certainly room for further improvement. Moreover, portfolio approaches may almost seamlessly benefit from parallelization, as demonstrated by the PPFOLIO system in the 2011 SAT Competition (SAT-Comp, 2011). However, no participant system exploited a parallel portfolio in the **MP** category of this ASP Competition edition, so that exploring such techniques is a subject to future research.

9.2 "Minisat hack"-Like Track

Another option to attract junior researchers as well as experts from neighboring areas to the development of ASP solvers may be a track dedicated to modifications of a common reference system, in the spirit of the Minisat hack track of the SAT Competition series (SAT-Comp). This would aim to lower the "entrance barrier" by keeping the effort of a successful participation affordable, especially for one-person teams, and also to provide a test bed for evaluating specific ideas under "laboratory conditions". Somewhat unfortunately, establishing such a track as part of the ASP Competition is not as straightforward as it may seem, given that (to our knowledge) there is no light-weight implementation of a modern ASP solver that would be easy to modify or extend by third parties. In this respect, the range of language features, including aggregates and positive recursion, as well as computational tasks, addressing Decision, Optimization, and Query problems, available in ASP necessitates dedicated functionalities going beyond the basic search procedures of SAT solvers. Hence, coming up with a simple yet general enough reference system is non-trivial and needs initiative of experienced ASP system developers.

9.3 Grounding Techniques

While virtually all ASP systems rely on a grounding step (cf. Section 2.4), merely the DLV grounder and GRINGO support the instantiation of ASP-Core-2 programs. As a matter of fact, both grounders work deterministically (using fixed heuristics), so that the "quality" of encodings at the first-order level is a crucial issue, e.g., in the *System Synthesis* domain investigated in Section 5.2. In order to substitute such manual tuning and handle "unoptimized" encodings more effectively, first-order techniques that go beyond the syntactic analysis of predicate-rule dependencies would need to be applied to non-ground ASP programs. Respective methods are already in use for Query answering, where DLV exploits so-called magic sets (Alviano et al., 2012) to narrow down the scope of a query. Similarly, the "lazy grounding" approach of the IDP system (De Cat, Denecker, Bruynooghe, & Stuckey, 2015) aims to instantiate first-order theories extended with inductive definitions on demand, i.e., driven by search at the ground level. Moreover, IDP is equipped with means to detect functional dependencies among variables (De Cat & Bruynooghe, 2013) and map them to built-in constructs, rather than combinatorially instantiating a problem encoding. Such first-order preprocessing techniques must still be regarded as "exotic", and ASP grounders that provide corresponding options, which would in turn increase the opportunities and potential benefit of grounder selection (Maratea et al., 2013, 2015b), remain a subject to future research.

9.4 Language Features

Beyond the availability of efficient systems, sharing basic search and implementation features with modern (Max)SAT and Pseudo-Boolean solvers, its expressive modeling language is characteristic for ASP. However, instantiation procedures lead to a propositional representation at the ground level, which can become a bottleneck, e.g., when dealing with numerical variables over large domains. To handle such scenarios, Constraint Answer Set Programming (CASP) (Mellarkod, Gelfond, & Zhang, 2008) has been proposed as an extension incorporating numerical variables and corresponding constraints into ASP programs. The additional expressiveness due to such modeling concepts has inspired the development of several systems for (dialects of) CASP, including ADSOLVER (Mellarkod & Gelfond, 2008), CLINGCON (Ostrowski & Schaub, 2012), EZCSP (Balduccini, 2009), EZSMT (Susman & Lierler, 2016), INCA (Drescher & Walsh, 2010), and MINGO (Liu et al., 2012). While tracks dedicated to extensions like CASP would certainly be of interest from the perspective of knowledge representation, the lack of respective standard input formats (possibly extending ASP-Core-2) constitutes an open issue that would need to be settled first.

9.5 Modeling Challenges

From 2009 to 2013, three editions of the ASP Competition featured a so-called Model&Solve track in which the participants had to prepare problem encodings and solving systems domain by domain. On the one hand, the opportunity to use customized encodings allowed more teams to participate, as their systems did not have to comply with any fixed modeling language. On the other hand, the effort of participation was much higher, since encodings had to be devised and systems tuned individually for each domain. Last but not least, the obtained performance results often reflected the "cleverness" put into different problem encodings, rather than indicating the relative strengths and weaknesses of solving systems. Hence, the former Model&Solve track has been replaced by an onsite modeling event in the previous and this edition of the ASP Competition, aiming to emphasize the fun of modeling in ASP but also keep the extra effort low. Maintaining and possibly extending this mode, e.g., by following the Logic/Constraint Programming Contest (LP/CP-Comp, 2016) in supporting a range of solving systems or on-line participation, is certainly one option for future ASP Competition editions. Another consideration would be to reintroduce a lighter version of the former Model&Solve track (each of its editions had included about 30 domains) and restrict challenges for modeling and solving to a few selected domains. In this way, systems relying on distinct input formats or supporting extensions like CASP, which do not pay off on ASP-Core-2 programs, could be showcased, while allowing participants to focus on specific rather than heterogeneous domains.

Acknowledgments

This paper is an extended and revised version of a preliminary report (Gebser, Maratea, & Ricca, 2015a), which presented the design of the event before the competition was indeed run. A brief survey of the Sixth ASP Competition was given by Gebser, Maratea, and Ricca (2016).

The organizers of the Sixth ASP Competition would like to thank the LPNMR 2015 officials for the co-location of the event. Moreover, we are grateful for the local support provided by Mirosław Truszczyński, Victor Marek, and Diane Mier. We further acknowledge the Department of Mathematics and Computer Science at the University of Calabria for supplying the computational resources to run the competition. Our thanks also go to the anonymous reviewers for careful comments that helped to improve this paper. Last but not least, we thank all contributors and participants, who worked hard to make this competition possible (and successful)!

Appendix A.

This appendix provides performance results per domain for systems in the **SP** category (gathering 11 participant systems out of a total of 13). In particular, Tables 4 and 5 refer to the Regular track, Table 6 lists the outcomes of running the three top-performing systems in the Marathon track, and Table 7 compares their performance relative to the original and alternative encodings for new domains. (Recall that LP2ACYCPB+CLASP was disqualified in *Valves Location* and LP2ACYCSAT+GLUCOSE in *Video Streaming*, as they produced incorrect optimum answers or solution candidates, respectively.[4])

Table 4: Detailed results of the Sixth ASP Competition for the **Regular track** (1/2).
The benchmark domains are subdivided by sub-tracks, and "Decision", "Optimization", and "Query" entries in the **Problem** column denote computational tasks. The top-performing participant system is reported for each domain, where an asterisk indicates that no other system obtained the same score. Further columns provide the scores, cumulative CPU times for runs rewarded with positive scores, numbers of time and memory outs, and average memory consumption (in megabytes) for the systems JWASP, LP2ACYCASP+CLASP, LP2ACYCPB+CLASP, LP2ACYCSAT+CLASP, LP2ACYCSAT+GLUCOSE, and LP2MIP.

			jwasp					**lp2acycasp+clasp**					**lp2acycpb+clasp**				
Domain	**Problem**	**Top Performer**	Score	Time	TO	MO	Mem	Score	Time	TO	MO	Mem	Score	Time	TO	MO	Mem
Graph Colouring	Decision	lp2acycsat+clasp	60.0	4649.7	8	0	139.1	85.0	2555.9	3	0	11.4	80.0	4743.6	4	0	21.3
Knight Tour with Holes	Decision	lp2acycsat+clasp	40.0	689.7	12	0	2010.6	70.0	1357.1	6	0	303.4	70.0	1916.7	6	0	682.3
Labyrinth	Decision	me-asp	40.0	1171.6	12	0	5897.5	50.0	951.1	10	0	162.1	45.0	1625.5	11	0	870.8
Stable Marriage	Decision	wasp+dlv*	5.0	869.9	9	10	11089.6	30.0	3714.8	0	14	12237.8	30.0	5395.8	0	14	12236.2
Visit-all	Decision	lp2sat+lingeling*	40.0	1145.3	12	0	1106.9	80.0	6641.8	4	0	179.3	40.0	954.2	12	0	562.7
Combined Configuration	Decision	lp2normal+clasp*	5.0	3.2	19	0	3832.0	20.0	261.6	16	0	255.5	10.0	347.6	18	0	1110.5
Consistent Query Answering	Query	wasp	0.0	–	11	9	10723.5	–	–	–	–	–	–	–	–	–	–
Graceful Graphs	Decision	lp2acycsat+clasp*	35.0	1339.7	13	0	258.4	60.0	2727.2	8	0	58.0	55.0	3102.9	9	0	60.2
Incremental Scheduling	Decision	lp2normal+clasp*	25.0	2192.2	11	4	7761.2	70.0	2134.4	5	1	3186.0	55.0	1927.7	8	1	3184.0
Nomystery	Decision	lp2sat+lingeling*	10.0	76.8	17	1	3982.8	45.0	300.8	11	0	458.2	35.0	2021.1	12	1	3291.2
Partner Units	Decision	lp2acycsat+clasp*	60.0	3164.3	8	0	1097.3	70.0	817.9	6	0	113.2	70.0	1311.0	6	0	366.0
Permutation Pattern Matching	Decision	wasp+dlv	60.0	1805.9	0	8	8178.4	40.0	1344.1	0	12	8473.2	40.0	965.7	0	12	8461.4
Qualitative Spatial Reasoning	Decision	me-asp*	35.0	529.4	0	13	10439.3	45.0	3063.5	11	0	2728.1	35.0	1330.7	13	0	5045.3
Reachability	Query	wasp+dlv*	0.0	–	0	20	12520.2	–	–	–	–	–	–	–	–	–	–
Ricochet Robots	Decision	lp2acycasp+clasp*	25.0	2221.6	15	0	441.5	85.0	5369.2	3	0	53.4	55.0	3687.8	9	0	189.8
Sokoban	Decision	lp2sat+lingeling	40.0	2089.3	12	0	1222.5	60.0	1807.0	8	0	120.2	40.0	2282.8	12	0	379.3
Connected Still Life	Optimization	lp2acycsat+glucose*	50.0	1312.0	10	0	151.0	70.5	18919.6	15	0	22.3	64.5	19523.0	15	0	35.4
Crossing Minimization	Optimization	lp2mip*	95.0	311.8	1	0	121.7	46.5	16894.2	14	0	24.1	51.5	15790.8	13	0	28.4
Maximal Clique	Optimization	lp2acycsat+glucose*	0.0	–	20	0	5978.8	66.5	24000.0	20	0	537.7	63.5	24000.0	20	0	553.8
MaxSAT	Optimization	wasp+dlv*	65.0	5654.3	7	0	5480.1	65.0	14026.5	11	0	455.4	48.5	17457.7	13	0	592.5
Steiner Tree	Optimization	lp2normal+clasp*	5.0	15.5	0	19	11896.1	5.0	103.8	1	18	11675.6	5.0	101.8	1	18	11702.9
System Synthesis	Optimization	wasp/wasp+dlv	0.0	–	20	0	5476.0	61.0	24000.0	20	0	618.7	0.0	–	9	11	11253.8
Valves Location	Optimization	lp2normal+clasp*	20.0	1688.8	14	2	5210.5	96.0	6320.4	5	0	462.0	disq.	disq.	disq.	disq.	disq.
Video Streaming	Optimization	lp2acycasp+clasp*	45.0	308.0	11	0	5081.5	93.0	6890.2	5	0	24.4	82.0	12601.8	10	0	49.7
Abstract Dialectical Frameworks	Optimization	lp2normal+clasp*	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Complex Optimization	Decision	lp2normal+clasp*	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Minimal Diagnosis	Decision	me-asp	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Strategic Companies	Query	wasp+dlv*	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
		Overall	**760**	**31239**	**242**	**86**	**5004**	**1314**	**144201**	**182**	**45**	**1916**	**975**	**121088**	**201**	**57**	**2889**

			lp2acycsat+clasp					**lp2acycsat+glucose**					**lp2mip**				
Domain	**Problem**	**Top Performer**	Score	Time	TO	MO	Mem	Score	Time	TO	MO	Mem	Score	Time	TO	MO	Mem
Graph Colouring	Decision	lp2acycsat+clasp	85.0	2350.5	3	0	16.0	70.0	4455.9	6	0	64.2	10.0	1819.0	18	0	63.3
Knight Tour with Holes	Decision	lp2acycsat+clasp	70.0	856.2	6	0	499.4	50.0	896.5	10	0	459.7	40.0	330.0	12	0	820.2
Labyrinth	Decision	me-asp	45.0	928.9	11	0	185.7	55.0	1660.2	9	0	233.6	0.0	–	20	0	1654.5
Stable Marriage	Decision	wasp+dlv*	30.0	5248.1	0	14	12241.0	25.0	4858.3	1	14	12232.0	0.0	–	0	20	12352.2
Visit-all	Decision	lp2sat+lingeling*	60.0	4383.1	8	0	472.3	40.0	830.5	12	0	505.8	75.0	3430.6	5	0	718.3
Combined Configuration	Decision	lp2normal+clasp*	15.0	847.5	17	0	1755.1	15.0	804.7	17	0	723.0	0.0	–	19	1	3209.3
Consistent Query Answering	Query	wasp	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Graceful Graphs	Decision	lp2acycsat+clasp*	65.0	3490.2	7	0	67.0	35.0	834.5	13	0	148.6	15.0	2229.2	17	0	716.2
Incremental Scheduling	Decision	lp2normal+clasp*	0.0	–	0	20	12410.4	0.0	–	0	20	12339.9	0.0	–	0	20	12381.5
Nomystery	Decision	lp2sat+lingeling*	50.0	834.2	10	0	690.1	40.0	944.5	12	0	987.6	0.0	–	18	2	3839.2
Partner Units	Decision	lp2acycsat+clasp*	75.0	2588.2	5	0	139.0	35.0	2796.1	13	0	229.9	0.0	–	20	0	1353.3
Permutation Pattern Matching	Decision	wasp+dlv	40.0	1254.6	0	12	8460.5	40.0	930.9	0	12	8455.9	25.0	1338.6	3	12	8645.5
Qualitative Spatial Reasoning	Decision	me-asp*	50.0	4224.6	10	0	3471.0	45.0	2940.5	11	0	4622.2	0.0	–	20	0	6873.5
Reachability	Query	wasp+dlv*	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Ricochet Robots	Decision	lp2acycasp+clasp*	75.0	4359.2	5	0	59.0	65.0	4141.0	7	0	170.7	0.0	–	20	0	651.8
Sokoban	Decision	lp2sat+lingeling	60.0	2265.5	8	0	267.7	55.0	2326.5	9	0	360.5	0.0	–	20	0	651.0
Connected Still Life	Optimization	lp2acycsat+glucose*	64.5	19599.6	15	0	43.0	100.0	6764.3	1	0	48.3	3.0	7200.0	20	0	140.0
Crossing Minimization	Optimization	lp2mip*	50.5	16220.8	13	0	28.5	93.0	6447.3	3	0	46.1	100.0	4881.7	1	0	82.2
Maximal Clique	Optimization	lp2acycsat+glucose*	63.5	24000.0	20	0	553.6	98.5	13320.4	6	0	371.7	68.5	11763.2	9	0	596.2
MaxSAT	Optimization	wasp+dlv*	68.5	14496.3	11	0	542.0	77.5	9673.8	2	5	3578.8	64.0	7849.7	8	0	635.2
Steiner Tree	Optimization	lp2normal+clasp*	5.0	72.2	2	17	11431.1	5.0	61.4	2	17	11438.6	5.0	369.7	2	17	11535.1
System Synthesis	Optimization	wasp/wasp+dlv	23.0	10800.0	20	0	2659.9	0.0	–	20	0	2824.9	0.0	–	12	8	10035.8
Valves Location	Optimization	lp2normal+clasp*	30.5	5919.1	4	12	9345.3	33.0	5325.5	3	12	9718.5	0.0	–	0	20	12439.1
Video Streaming	Optimization	lp2acycasp+clasp*	78.5	12883.4	10	0	246.0	disq.	disq.	disq.	disq.	disq.	14.5	10036.8	18	0	2304.4
Abstract Dialectical Frameworks	Optimization	lp2normal+clasp*	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Complex Optimization	Decision	lp2normal+clasp*	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Minimal Diagnosis	Decision	me-asp	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Strategic Companies	Query	wasp+dlv*	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
		Overall	**1104**	**137622**	**185**	**75**	**2981**	**977**	**70013**	**157**	**80**	**3312**	**420**	**51249**	**262**	**100**	**4168**

Table 5: Detailed results of the Sixth ASP Competition for the **Regular track** (2/2).
The benchmark domains are subdivided by sub-tracks, and "Decision", "Optimization", and "Query" entries in the **Problem** column denote computational tasks. The top-performing participant system is reported for each domain, where an asterisk indicates that no other system obtained the same score. Further columns provide the scores, cumulative CPU times for runs rewarded with positive scores, numbers of time and memory outs, and average memory consumption (in megabytes) for the systems LP2NORMAL+CLASP, LP2SAT+LINGELING, ME-ASP, WASP, and WASP+DLV.

			lp2normal+clasp					lp2sat+lingeling					me-asp				
Domain	**Problem**	**Top Performer**	Score	Time	TO	MO	Mem	Score	Time	TO	MO	Mem	Score	Time	TO	MO	Mem
Graph Colouring	Decision	lp2acycsat+clasp	85.0	3426.3	3	0	16.7	80.0	4832.5	4	0	39.0	85.0	3505.1	3	0	25.4
Knight Tour with Holes	Decision	lp2acycsat+clasp	55.0	1103.7	9	0	344.1	10.0	90.6	18	0	712.9	55.0	990.7	9	0	352.0
Labyrinth	Decision	me-asp	80.0	5683.9	4	0	287.5	45.0	2014.4	11	0	352.4	80.0	5534.7	4	0	294.9
Stable Marriage	Decision	wasp+dlv*	20.0	3339.8	2	14	12229.3	30.0	4744.2	0	14	12233.7	50.0	5522.6	10	0	1615.1
Visit-all	Decision	lp2sat+lingeling*	85.0	9400.2	3	0	177.7	100.0	9593.5	0	0	253.4	70.0	6440.6	6	0	125.4
Combined Configuration	Decision	lp2normal+clasp*	35.0	1347.9	13	0	369.0	10.0	110.1	18	0	698.7	30.0	2756.3	14	0	1216.5
Consistent Query Answering	Query	wasp	–	–	–	–	–	–	–	–	–	–	0.0	–	16	4	2218.7
Graceful Graphs	Decision	lp2acycsat+clasp*	55.0	1147.8	9	0	49.7	45.0	765.2	11	0	105.4	55.0	2257.3	9	0	102.8
Incremental Scheduling	Decision	lp2normal+clasp*	75.0	3337.4	4	1	3179.8	0.0	–	0	20	12409.6	65.0	1525.8	7	0	3029.0
Nomystery	Decision	lp2sat+lingeling*	40.0	565.2	12	0	1640.9	60.0	3616.4	8	0	741.5	45.0	1003.0	11	0	1383.6
Partner Units	Decision	lp2acycsat+clasp*	70.0	1582.0	6	0	152.9	20.0	628.6	16	0	250.2	70.0	461.5	6	0	179.0
Permutation Pattern Matching	Decision	wasp+dlv	40.0	997.5	0	12	8458.1	40.0	1109.4	0	12	8452.6	100.0	2779.5	0	0	1191.2
Qualitative Spatial Reasoning	Decision	me-asp*	55.0	5132.2	9	0	2379.8	35.0	1287.3	13	0	2297.9	100.0	2544.5	0	0	811.9
Reachability	Query	wasp+dlv*	–	–	–	–	–	–	–	–	–	–	90.0	2713.4	0	2	3022.4
Ricochet Robots	Decision	lp2acycasp+clasp*	45.0	704.1	11	0	129.4	80.0	4930.7	4	0	84.3	55.0	2342.8	9	0	73.8
Sokoban	Decision	lp2sat+lingeling	40.0	643.0	12	0	314.1	65.0	2082.3	7	0	221.6	65.0	3236.2	7	0	742.0
Connected Still Life	Optimization	lp2acycsat+glucose*	96.5	10091.0	7	0	26.0	–	–	–	–	–	80.0	11159.7	8	0	27.6
Crossing Minimization	Optimization	lp2mip*	85.5	8160.5	6	0	22.0	–	–	–	–	–	97.0	1207.8	1	0	7.9
Maximal Clique	Optimization	lp2acycsat+glucose*	81.5	20579.5	15	0	365.6	–	–	–	–	–	87.0	11289.7	5	0	204.1
MaxSAT	Optimization	wasp+dlv*	70.0	13258.1	10	0	439.2	–	–	–	–	–	81.0	6318.2	5	0	438.4
Steiner Tree	Optimization	lp2normal+clasp*	93.5	19857.6	17	0	3156.1	–	–	–	–	–	80.5	22801.8	19	0	6016.5
System Synthesis	Optimization	wasp/wasp+dlv	71.5	24000.0	20	0	202.4	–	–	–	–	–	81.5	18532.6	13	0	186.1
Valves Location	Optimization	lp2normal+clasp*	97.5	5459.3	4	0	553.7	–	–	–	–	–	80.5	12505.5	9	0	399.4
Video Streaming	Optimization	lp2acycasp+clasp*	89.0	7773.3	5	0	31.3	–	–	–	–	–	55.5	13200.0	11	0	317.4
Abstract Dialectical Frameworks	Optimization	lp2normal+clasp*	100.0	140.9	0	0	16.6	–	–	–	–	–	97.5	2112.4	1	0	28.6
Complex Optimization	Decision	lp2normal+clasp*	95.0	2492.2	1	0	476.1	–	–	–	–	–	35.0	459.4	13	0	303.7
Minimal Diagnosis	Decision	me-asp	100.0	281.8	0	0	264.8	–	–	–	–	–	100.0	188.7	0	0	278.3
Strategic Companies	Query	wasp+dlv*	–	–	–	–	–	–	–	–	–	–	80.0	2939.0	4	0	24.0
		Overall	**1760**	**150505**	**182**	**27**	**1411**	**620**	**35805**	**110**	**46**	**2775**	**1971**	**146329**	**200**	**6**	**879**

			wasp					wasp+dlv				
Domain	**Problem**	**Top Performer**	Score	Time	TO	MO	Mem	Score	Time	TO	MO	Mem
Graph Colouring	Decision	lp2acycsat+clasp	50.0	2831.2	10	0	83.4	45.0	1818.6	11	0	136.4
Knight Tour with Holes	Decision	lp2acycsat+clasp	50.0	1381.6	10	0	338.6	50.0	1362.4	10	0	522.9
Labyrinth	Decision	me-asp	70.0	3492.3	6	0	462.9	70.0	3420.9	6	0	829.2
Stable Marriage	Decision	wasp+dlv*	75.0	9847.9	5	0	1622.9	80.0	10830.1	4	0	1650.2
Visit-all	Decision	lp2sat+lingeling*	40.0	378.1	12	0	878.5	40.0	354.9	12	0	2147.8
Combined Configuration	Decision	lp2normal+clasp*	5.0	784.7	19	0	563.7	5.0	792.2	19	0	1643.5
Consistent Query Answering	Query	wasp	90.0	5465.7	2	0	5275.3	90.0	5518.0	2	0	5268.9
Graceful Graphs	Decision	lp2acycsat+clasp*	15.0	897.9	17	0	361.3	15.0	881.8	17	0	953.9
Incremental Scheduling	Decision	lp2normal+clasp*	45.0	1394.2	10	1	3638.4	45.0	1440.1	10	1	3638.4
Nomystery	Decision	lp2sat+lingeling*	45.0	994.8	11	0	1081.9	45.0	1039.5	11	0	2512.5
Partner Units	Decision	lp2acycsat+clasp*	70.0	4400.2	6	0	454.6	70.0	4391.6	6	0	1373.3
Permutation Pattern Matching	Decision	wasp+dlv	100.0	2578.1	0	0	1181.2	100.0	2491.8	0	0	1179.8
Qualitative Spatial Reasoning	Decision	me-asp*	90.0	5236.6	2	0	3237.9	90.0	5286.6	2	0	3232.0
Reachability	Query	wasp+dlv*	0.0	–	0	20	12898.4	100.0	3470.2	0	0	6149.6
Ricochet Robots	Decision	lp2acycasp+clasp*	45.0	2675.9	11	0	331.3	45.0	2666.8	11	0	852.7
Sokoban	Decision	lp2sat+lingeling	55.0	2984.7	9	0	544.9	60.0	4082.1	8	0	1683.6
Connected Still Life	Optimization	lp2acycsat+glucose*	65.0	1251.5	7	0	92.8	75.5	10164.7	8	0	100.5
Crossing Minimization	Optimization	lp2mip*	95.0	647.8	1	0	67.2	96.5	1278.1	1	0	37.0
Maximal Clique	Optimization	lp2acycsat+glucose*	50.0	3788.2	10	0	891.7	74.0	14405.6	8	0	891.0
MaxSAT	Optimization	wasp+dlv*	93.5	1830.5	1	0	1074.1	96.0	3125.8	1	0	1059.6
Steiner Tree	Optimization	lp2normal+clasp*	59.5	14463.3	12	7	7282.8	59.5	14464.7	12	7	7308.6
System Synthesis	Optimization	wasp/wasp+dlv	96.5	24000.0	20	0	604.3	96.5	24000.0	20	0	605.2
Valves Location	Optimization	lp2normal+clasp*	93.0	5992.1	5	0	1301.9	91.0	6615.8	5	0	1303.7
Video Streaming	Optimization	lp2acycasp+clasp*	45.0	0.8	11	0	296.7	53.5	12000.7	11	0	308.7
Abstract Dialectical Frameworks	Optimization	lp2normal+clasp*	95.0	628.9	1	0	129.3	95.0	2612.6	2	0	138.3
Complex Optimization	Decision	lp2normal+clasp*	35.0	2234.6	13	0	1594.8	50.0	5378.8	10	0	1594.9
Minimal Diagnosis	Decision	me-asp	100.0	360.2	0	0	1419.7	100.0	372.0	0	0	1421.2
Strategic Companies	Query	wasp+dlv*	0.0	–	0	20	12607.4	100.0	851.5	0	0	19.1
		Overall	**1673**	**100542**	**211**	**48**	**2154**	**1938**	**145118**	**207**	**8**	**1734**

Table 6: Detailed results of the Sixth ASP Competition for the **Marathon track**.
The benchmark domains are subdivided by sub-track, and “Decision”, “Optimization”, and “Query” entries in the **Problem** column denote computational tasks. The top-performing participant system is reported for each domain, where an asterisk indicates that no other system obtained the same score. Further columns provide the scores, cumulative CPU times for runs rewarded with positive scores, numbers of time and memory outs, and average memory consumption (in megabytes) for the systems LP2NORMAL+CLASP, ME-ASP, and WASP+DLV.

			lp2normal+clasp					me-asp					wasp+dlv				
Domain	**Problem**	**Top Performer**	Score	Time	TO	MO	Mem	Score	Time	TO	MO	Mem	Score	Time	TO	MO	Mem
Graph Colouring	Decision	lp2normal+clasp	95.0	11987.0	1	0	16.7	95.0	12858.8	1	0	25.4	75.0	20027.1	5	0	136.4
Knight Tour with Holes	Decision	me-asp	55.0	1103.7	9	0	344.1	55.0	990.7	9	0	352.0	55.0	8014.1	9	0	522.9
Labyrinth	Decision	lp2normal+clasp	100.0	18312.3	0	0	287.5	100.0	18687.4	0	0	294.9	90.0	15690.7	2	0	829.2
Stable Marriage	Decision	wasp+dlv	30.0	8756.8	0	14	12229.3	95.0	23162.5	1	0	1615.1	95.0	16672.2	1	0	1650.2
Visit-all	Decision	lp2normal+clasp	100.0	13409.4	0	0	177.7	100.0	15340.6	0	0	125.4	70.0	40486.7	6	0	2147.8
Combined Configuration	Decision	lp2normal+clasp*	60.0	17974.1	8	0	369.0	45.0	19058.6	11	0	1216.5	10.0	6853.4	18	0	1643.5
Consistent Query Answering	Query	wasp+dlv*	—	—	—	—	—	0.0	—	12	8	2218.7	100.0	8837.0	0	0	5268.9
Graceful Graphs	Decision	me-asp	65.0	6715.8	7	0	49.7	65.0	6660.8	7	0	102.8	30.0	15665.7	14	0	953.9
Incremental Scheduling	Decision	lp2normal+clasp*	75.0	3337.4	4	1	3179.8	70.0	2908.3	6	0	3029.0	55.0	9946.4	8	1	3638.4
Nomystery	Decision	me-asp*	50.0	6210.2	10	0	1640.9	55.0	13946.4	9	0	1383.6	50.0	4616.6	10	0	2512.5
Partner Units	Decision	me-asp	75.0	4021.2	5	0	152.9	75.0	3201.7	5	0	179.0	75.0	6505.8	5	0	1373.3
Permutation Pattern Matching	Decision	wasp+dlv	40.0	997.5	0	12	8458.1	100.0	2779.5	0	0	1191.2	100.0	2491.8	0	0	1179.8
Qualitative Spatial Reasoning	Decision	me-asp	100.0	20875.2	0	0	2379.8	100.0	2544.5	0	0	811.9	100.0	9045.6	0	0	3232.0
Reachability	Query	wasp+dlv*	—	—	—	—	—	90.0	2713.4	0	2	3022.4	100.0	3470.2	0	0	6149.6
Ricochet Robots	Decision	me-asp*	85.0	32802.2	3	0	129.4	100.0	25180.4	0	0	73.8	70.0	23694.0	6	0	852.7
Sokoban	Decision	me-asp	60.0	13948.7	8	0	314.1	65.0	3236.2	7	0	742.0	65.0	10742.9	7	0	1683.6
Connected Still Life	Optimization	wasp+dlv*	93.3	63000.0	5	0	36.8	78.3	79829.6	7	0	34.8	96.7	46193.5	1	0	171.2
Crossing Minimization	Optimization	me-asp	90.0	46859.0	3	0	27.3	100.0	3205.3	0	0	8.3	100.0	8309.0	0	0	64.9
Maximal Clique	Optimization	me-asp*	80.0	118558.7	10	0	508.8	93.3	50023.6	4	0	203.0	86.7	56145.5	4	0	1086.7
MaxSAT	Optimization	wasp+dlv*	73.3	109258.1	10	0	462.9	83.3	54318.2	5	0	603.1	100.0	4583.5	0	0	1059.6
Steiner Tree	Optimization	lp2normal+clasp*	91.7	184257.6	17	0	3443.8	50.0	124359.7	8	11	10027.3	53.3	123660.7	11	8	8011.7
System Synthesis	Optimization	me-asp*	35.0	216000.0	20	0	372.6	85.0	117067.7	8	0	327.5	81.7	208624.5	19	0	864.4
Valves Location	Optimization	lp2normal+clasp*	100.0	26067.5	2	0	553.7	85.0	73475.9	5	0	512.4	93.3	32057.2	2	0	1467.4
Video Streaming	Optimization	lp2normal+clasp*	98.3	55773.3	5	0	53.6	73.3	118800.0	11	0	1035.3	68.3	108000.7	11	0	800.5
Abstract Dialectical Frameworks	Optimization	lp2normal+clasp	100.0	140.9	0	0	16.6	96.7	11712.4	1	0	30.3	100.0	7020.5	0	0	165.6
Complex Optimization	Decision	lp2normal+clasp*	100.0	4027.5	0	0	476.1	35.0	459.4	13	0	303.7	80.0	21194.7	4	0	1594.9
Minimal Diagnosis	Decision	me-asp	100.0	281.8	0	0	264.8	100.0	188.7	0	0	278.3	100.0	372.0	0	0	1421.2
Strategic Companies	Query	wasp+dlv	—	—	—	—	—	100.0	13879.1	0	0	24.0	100.0	851.5	0	0	19.1
		Overall	**1952**	**984676**	**127**	**27**	**1438**	**2190**	**800589**	**130**	**21**	**1063**	**2200**	**819773**	**143**	**9**	**1804**

Table 7: Detailed comparison between original and alternative encodings for new domains.
The benchmark domains are subdivided by domains, and “Decision”, “Optimization”, and “Query” entries in the **Problem** column denote computational tasks. Further columns provide the number of solutions found, cumulative CPU times for runs in which some solution was found, and numbers of time and memory outs for the systems LP2NORMAL+CLASP, ME-ASP, and WASP+DLV.

			lp2normal+clasp				me-asp				wasp+dlv			
Domain	**Problem**	**Encoding**	Sol	Time	TO	MO	Sol	Time	TO	MO	Sol	Time	TO	MO
Combined Configuration	Decision	Original	7	1347.9	13	0	6	2756.3	14	0	1	792.2	19	0
Combined Configuration	Decision	Alternative	16	2133.3	4	0	16	1517.6	4	0	9	4615.8	11	0
MaxSAT	Optimization	Original	20	13258.1	10	0	20	6318.2	5	0	20	3125.8	1	0
MaxSAT	Optimization	Alternative	20	12233.3	10	0	20	6139.8	5	0	20	835.2	0	0
Steiner Tree	Optimization	Original	19	19857.6	17	0	20	22801.8	19	0	13	14464.7	12	7
Steiner Tree	Optimization	Alternative	20	18060.4	15	0	20	17341.0	14	0	20	19200.9	16	0
System Synthesis	Optimization	Original	20	24000.0	20	0	20	18532.6	13	0	20	24000.0	20	0
System Synthesis	Optimization	Alternative	20	24000.0	20	0	20	15651.4	12	0	20	24000.0	20	0
Video Streaming	Optimization	Original	20	7773.3	5	0	20	13200.0	11	0	19	12000.7	11	0
Video Streaming	Optimization	Alternative	13	4826.2	10	0	17	9605.4	11	0	13	6008.2	11	0

References

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

Alviano, M., Dodaro, C., & Ricca, F. (2015). Reduct-based stability check using literal assumptions. In Inclezan, D., & Maratea, M. (Eds.), *Proceedings of the Eighth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'15)*. `https://sites.google.com/site/aspocp2015/ASPOCP2015paper8.pdf`.

Alviano, M., Faber, W., Greco, G., & Leone, N. (2012). Magic sets for disjunctive Datalog programs. *Artificial Intelligence*, *187-188*, 156–192.

Alviano, M., Dodaro, C., Leone, N., & Ricca, F. (2015a). Advances in WASP. In Calimeri, F., Ianni, G., & Truszczyński, M. (Eds.), *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15)*, Vol. 9345 of *Lecture Notes in Computer Science*, pp. 40–54. Springer.

Alviano, M., Dodaro, C., Marques-Silva, J., & Ricca, F. (2015b). Optimum stable model search: Algorithms and implementation. *Journal of Logic and Computation*, *Article ID exv061*, doi.org/10.1093/logcom/exv061.

Alviano, M., Dodaro, C., & Ricca, F. (2014). Anytime computation of cautious consequences in answer set programming. *Theory and Practice of Logic Programming*, *14*(4-5), 755–770.

Aschinger, M., Drescher, C., Friedrich, G., Gottlob, G., Jeavons, P., Ryabokon, A., & Thorstensen, E. (2011). Optimization methods for the partner units problem. In Achterberg, T., & Beck, C. (Eds.), *Proceedings of the Eighth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'11)*, Vol. 6697 of *Lecture Notes in Computer Science*, pp. 4–19. Springer.

ASP-Comp (2015). *The Sixth Answer Set Programming Competition*. `http://aspcomp2015.dibris.unige.it`.

ATP-Comp. *The CADE ATP System Competition*. `http://www.cs.miami.edu/~tptp/CASC/`.

Babovich, Y., & Lifschitz, V. (2003). Computing answer sets using program completion. `http://www.cs.utexas.edu/users/tag/cmodels/cmodels-1.ps`.

Balduccini, M. (2011). Industrial-size scheduling with ASP+CP. In Delgrande, J., & Faber, W. (Eds.), *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, Vol. 6645 of *Lecture Notes in Computer Science*, pp. 284–296. Springer.

Balduccini, M. (2009). Representing constraint satisfaction problems in answer set programming. In Faber, W., & Lee, J. (Eds.), *Proceedings of the Second Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'09)*, pp. 16–30. `http://mbal.tk/papers/bal09.pdf`.

Balduccini, M., Gelfond, M., Watson, R., & Nogueira, M. (2001). The USA-Advisor: A case study in answer set planning. In Eiter, T., Faber, W., & Truszczyński, M. (Eds.), *Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, Vol. 2173 of *Lecture Notes in Computer Science*, pp. 439–442. Springer.

Balduccini, M., Pontelli, E., El-Khatib, O., & Le, H. (2005). Issues in parallel execution of non-monotonic reasoning systems. *Parallel Computing*, *31*(6), 608–647.

Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Baral, C., & Gelfond, M. (2000). Reasoning agents in dynamic domains. In Minker, J. (Ed.), *Logic-Based Artificial Intelligence*, pp. 257–279. Kluwer Academic Publishers.

Baral, C., & Uyan, C. (2001). Declarative specification and solution of combinatorial auctions using logic programming. In Eiter, T., Faber, W., & Truszczyński, M. (Eds.), *Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, Vol. 2173 of *Lecture Notes in Computer Science*, pp. 186–199. Springer.

Bardadym, V. (1996). Computer-aided school and university timetabling: The new wave. In Burke, E., & Ross, P. (Eds.), *Proceedings of the First International Conference on Practice and Theory of Automated Timetabling (PATAT'95)*, Vol. 1153 of *Lecture Notes in Computer Science*, pp. 22–45. Springer.

Ben-Eliyahu, R., & Dechter, R. (1994). Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, *12*(1-2), 53–87.

Bertossi, L., Hunter, A., & Schaub, T. (Eds.). (2005). *Inconsistency Tolerance*, Vol. 3300 of *Lecture Notes in Computer Science*. Springer.

Biere, A., & Fröhlich, A. (2015). Evaluating CDCL variable scoring schemes. In Heule, M., & Weaver, S. (Eds.), *Proceedings of the Eighteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, Vol. 9340 of *Lecture Notes in Computer Science*, pp. 405–422. Springer.

Biewer, A., Andres, B., Gladigau, J., Schaub, T., & Haubelt, C. (2015). A symbolic system synthesis approach for hard real-time systems based on coordinated SMT-solving. In Nebel, W., & Atienza, D. (Eds.), *Proceedings of the Eighteenth Conference on Design, Automation and Test in Europe (DATE'15)*, pp. 357–362. ACM Press.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., & Vanschoren, J. (2016). ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, *237*, 41–58.

Bomanson, J., Gebser, M., & Janhunen, T. (2014). Improving the normalization of weight rules in answer set programs. In Fermé, E., & Leite, J. (Eds.), *Proceedings of the Fourteenth European Conference on Logics in Artificial Intelligence (JELIA'14)*, Vol. 8761 of *Lecture Notes in Computer Science*, pp. 166–180. Springer.

Bomanson, J., Gebser, M., Janhunen, T., Kaufmann, B., & Schaub, T. (2016). Answer set programming modulo acyclicity. *Fundamenta Informaticae*, *147*(1), 63–91.

Bravo, L., & Bertossi, L. (2003). Logic programming for consistently querying data integration systems. In Gottlob, G., & Walsh, T. (Eds.), *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pp. 10–15. Morgan Kaufmann.

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

Brewka, G., Niemelä, I., & Syrjänen, T. (2002). Implementing ordered disjunction using answer set solvers for normal programs. In Flesca, S., Greco, S., Ianni, G., & Leone, N. (Eds.), *Proceedings of the Eighth European Conference on Logics in Artificial Intelligence (JELIA'02)*, Vol. 2424 of *Lecture Notes in Computer Science*, pp. 444–455. Springer.

Bruynooghe, M., Blockeel, H., Bogaerts, B., De Cat, B., De Pooter, S., Jansen, J., Labarre, A., Ramon, J., Denecker, M., & 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.

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

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., & Schaub, T. (2012). ASP-Core-2: Input language format. `https://www.mat.unical.it/aspcomp2013/ASPStandardization/`.

Calimeri, F., Faber, W., Leone, N., & Pfeifer, G. (2006). Pruning operators for disjunctive logic programming systems. *Fundamenta Informaticae*, *71*(2-3), 183–214.

Calimeri, F., Ianni, G., & Ricca, F. (2014). The third open answer set programming competition. *Theory and Practice of Logic Programming*, *14*(1), 117–135.

Clark, K. (1978). Negation as failure. In Gallaire, H., & Minker, J. (Eds.), *Logic and Data Bases*, pp. 293–322. Plenum Press.

Cok, D., Griggio, A., Bruttomesso, R., & Deters, M. (2012). The 2012 SMT competition. In Fontaine, P., & Goel, A. (Eds.), *Proceedings of the Tenth International Workshop on Satisfiability Modulo Theories (SMT'12)*, Vol. 20 of *EPiC Series in Computing*, pp. 131–142. EasyChair.

Crawford, J., & Baker, A. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. In Hayes-Roth, B., & Korf, R. (Eds.), *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pp. 1092–1097. AAAI Press.

Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem proving. *Communications of the ACM*, *5*(7), 394–397.

Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, *7*, 201–215.

De Cat, B., & Bruynooghe, M. (2013). Detection and exploitation of functional dependencies for model generation. *Theory and Practice of Logic Programming*, *13*(4-5), 471–485.

De Cat, B., Denecker, M., Bruynooghe, M., & Stuckey, P. (2015). Lazy model expansion: Interleaving grounding with search. *Journal of Artificial Intelligence Research*, *52*, 235–286.

Dovier, A., Formisano, A., Pontelli, E., & Vella, F. (2016). A GPU implementation of the ASP computation. In Gavanelli, M., & Reppy, J. (Eds.), *Proceedings of the Eighteenth International Symposium on Practical Aspects of Declarative Languages (PADL'16)*, Vol. 9585 of *Lecture Notes in Computer Science*, pp. 30–47. Springer.

Drescher, C., & Walsh, T. (2010). A translational approach to constraint answer set solving. *Theory and Practice of Logic Programming*, *10*(4-6), 465–480.

Dworschak, S., Grell, S., Nikiforova, V., Schaub, T., & Selbig, J. (2008). Modeling biological networks by action languages via answer set programming. *Constraints*, *13*(1-2), 21–65.

Eén, N., & Sörensson, N. (2003). Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, *89*(4), 543–560.

Eiter, T., & Gottlob, G. (1995). On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, *15*(3-4), 289–323.

Erdem, E., & Wong, M. (2004). Rectilinear Steiner tree construction using answer set programming. In Demoen, B., & Lifschitz, V. (Eds.), *Proceedings of the Twentieth International Conference on Logic Programming (ICLP'04)*, Vol. 3132 of *Lecture Notes in Computer Science*, pp. 386–399. Springer.

Erdem, E. (2011). Applications of answer set programming in phylogenetic systematics. In Balduccini, M., & Son, T. (Eds.), *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, Vol. 6565 of *Lecture Notes in Computer Science*, pp. 415–431. Springer.

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

Erdem, E., & Lifschitz, V. (2003). Tight logic programs. *Theory and Practice of Logic Programming*, *3*(4-5), 499–518.

Faber, W., Leone, N., & Perri, S. (2012). The intelligent grounder of DLV. In Erdem, E., Lee, J., Lierler, Y., & Pearce, D. (Eds.), *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, Vol. 7265 of *Lecture Notes in Computer Science*, pp. 247–264. Springer.

Faber, W., Leone, N., & Pfeifer, G. (2004). Recursive aggregates in disjunctive logic programs: Semantics and complexity. In Alferes, J., & Leite, J. (Eds.), *Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA'04)*, Vol. 3229 of *Lecture Notes in Computer Science*, pp. 200–212. Springer.

Faber, W., Leone, N., & Pfeifer, G. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, *175*(1), 278–298.

Fages, F. (1994). Consistency of Clark's completion and the existence of stable models. *Journal of Methods of Logic in Computer Science*, *1*, 51–60.

Finkel, R., Marek, V., Moore, N., & Truszczyński, M. (2001). Computing stable models in parallel. In Provetti, A., & Son, T. (Eds.), *Proceedings of the First International Workshop on Answer Set Programming (ASP'01)*. `http://www.cs.nmsu.edu/~tson/ASP2001/18.ps`.

Gange, G., Stuckey, P., & Marriott, K. (2010). Optimal *k*-level planarization and crossing minimization. In Brandes, U., & Cornelsen, S. (Eds.), *Proceedings of the Eighteenth International Symposium on Graph Drawing (GD'10)*, Vol. 6502 of *Lecture Notes in Computer Science*, pp. 238–249. Springer.

Garro, A., Palopoli, L., & Ricca, F. (2006). Exploiting agents in e-learning and skills management context. *AI Communications*, *19*(2), 137–154.

Gavanelli, M., Nonato, M., & Peano, A. (2015). An ASP approach for the valves positioning optimization in a water distribution system. *Journal of Logic and Computation*, *25*(6), 1351–1369.

Gebser, M., Guziolowski, C., Ivanchev, M., Schaub, T., Siegel, A., Thiele, S., & Veber, P. (2010). Repair and prediction (under inconsistency) in large biological networks with answer set programming. In Lin, F., & Sattler, U. (Eds.), *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*, pp. 497–507. AAAI Press.

Gebser, M., Janhunen, T., & Rintanen, J. (2014a). Answer set programming as SAT modulo acyclicity. In Schaub, T., Friedrich, G., & O'Sullivan, B. (Eds.), *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI'14)*, Vol. 263 of *Frontiers in Artificial Intelligence and Applications*, pp. 351–356. IOS Press.

Gebser, M., Janhunen, T., & Rintanen, J. (2014b). SAT modulo graphs: Acyclicity. In Fermé, E., & Leite, J. (Eds.), *Proceedings of the Fourteenth European Conference on Logics in Artificial Intelligence (JELIA'14)*, Vol. 8761 of *Lecture Notes in Computer Science*, pp. 137–151. Springer.

Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., & Schaub, T. (2015). Progress in clasp series 3. In Calimeri, F., Ianni, G., & Truszczyński, M. (Eds.), *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15)*, Vol. 9345 of *Lecture Notes in Computer Science*, pp. 368–383. Springer.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012). *Answer Set Solving in Practice*. Morgan and Claypool Publishers.

Gebser, M., Kaminski, R., König, A., & Schaub, T. (2011a). Advances in gringo series 3. In Delgrande, J., & Faber, W. (Eds.), *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, Vol. 6645 of *Lecture Notes in Computer Science*, pp. 345–351. Springer.

Gebser, M., Kaminski, R., & Schaub, T. (2011b). Complex optimization in answer set programming. *Theory and Practice of Logic Programming*, *11*(4-5), 821–839.

Gebser, M., Kaminski, R., & Schaub, T. (2015). Grounding recursive aggregates: Preliminary report. In Denecker, M., & Janhunen, T. (Eds.), *Proceedings of the Third Workshop on Grounding, Transforming, and Modularizing Theories with Variables (GTTV'15)*. `https://dtai.cs.kuleuven.be/krr/GTTV/GTTV15_submission_5.pdf`.

Gebser, M., Kaufmann, B., & Schaub, T. (2012). Multi-threaded ASP solving with clasp. *Theory and Practice of Logic Programming*, *12*(4-5), 525–545.

Gebser, M., Kaufmann, B., & Schaub, T. (2013). Advanced conflict-driven disjunctive answer set solving. In Rossi, F. (Ed.), *Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13)*, pp. 912–918. IJCAI/AAAI Press.

Gebser, M., Maratea, M., & Ricca, F. (2016). What's hot in the answer set programming competition. In Schuurmans, D., & Wellman, M. (Eds.), *Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16)*, pp. 4327–4329. AAAI Press.

Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M., & Ziller, S. (2011a). A portfolio solver for answer set programming: Preliminary report. In Delgrande, J., & Faber, W. (Eds.), *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, Vol. 6645 of *Lecture Notes in Computer Science*, pp. 352–357. Springer.

Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., & Schnor, B. (2011b). Cluster-based ASP solving with claspar. In Delgrande, J., & Faber, W. (Eds.), *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, Vol. 6645 of *Lecture Notes in Computer Science*, pp. 364–369. Springer.

Gebser, M., Kaufmann, B., & Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, *187-188*, 52–89.

Gebser, M., Maratea, M., & Ricca, F. (2015a). The design of the sixth answer set programming competition: Report. In Calimeri, F., Ianni, G., & Truszczyński, M. (Eds.), *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15)*, Vol. 9345 of *Lecture Notes in Computer Science*, pp. 531–544. Springer.

Gebser, M., Ryabokon, A., & Schenner, G. (2015b). Combining heuristics for configuration problems using answer set programming. In Calimeri, F., Ianni, G., & Truszczyński, M. (Eds.), *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15)*, Vol. 9345 of *Lecture Notes in Computer Science*, pp. 384–397. Springer.

Gebser, M., Schaub, T., & Thiele, S. (2007). Gringo: A new grounder for answer set programming. In Baral, C., Brewka, G., & Schlipf, J. (Eds.), *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, Vol. 4483 of *Lecture Notes in Computer Science*, pp. 266–271. Springer.

Gebser, M., Schaub, T., Thiele, S., & Veber, P. (2011). Detecting inconsistencies in large biological networks with answer set programming. *Theory and Practice of Logic Programming*, *11*(2-3), 323–360.

Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, *9*, 365–385.

Giunchiglia, E., Lierler, Y., & Maratea, M. (2006). Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning*, *36*(4), 345–377.

Gressmann, J., Janhunen, T., Mercer, R., Schaub, T., Thiele, S., & Tichy, R. (2006). On probing and multi-threading in platypus. In Brewka, G., Coradeschi, S., Perini, A., & Traverso, P. (Eds.), *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06)*, Vol. 141 of *Frontiers in Artificial Intelligence and Applications*, pp. 392–396. IOS Press.

Hoos, H., Kaminski, R., Lindauer, M., & Schaub, T. (2015). aspeed: Solver scheduling via answer set programming. *Theory and Practice of Logic Programming*, *15*(1), 117–142.

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

Huang, J. (2007). The effect of restarts on the efficiency of clause learning. In Veloso, M. (Ed.), *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 2318–2323. AAAI/MIT Press.

Ielpa, S., Iiritano, S., Leone, N., & Ricca, F. (2009). An ASP-based system for e-tourism. In Erdem, E., Lin, F., & Schaub, T. (Eds.), *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, Vol. 5753 of *Lecture Notes in Computer Science*, pp. 368–381. Springer.

IPC-Comp (2014). *International Planning Competition 2014*. `https://helios.hud.ac.uk/scommv/IPC-14/`.

Janhunen, T., Niemelä, I., Seipel, D., Simons, P., & You, J. (2006). Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic*, *7*(1), 1–37.

Janhunen, T. (2006). Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics*, *16*(1-2), 35–86.

Janhunen, T., & Niemelä, I. (2011). Compact translations of non-disjunctive answer set programs to propositional clauses. In Balduccini, M., & Son, T. (Eds.), *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, Vol. 6565 of *Lecture Notes in Computer Science*, pp. 111–130. Springer.

Janhunen, T., Niemelä, I., & Sevalnev, M. (2009). Computing stable models via reductions to difference logic. In Erdem, E., Lin, F., & Schaub, T. (Eds.), *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, Vol. 5753 of *Lecture Notes in Computer Science*, pp. 142–154. Springer.

Janota, M., & Marques-Silva, J. (2011). On deciding MUS membership with QBF. In Lee, J. (Ed.), *Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP'11)*, Vol. 6876 of *Lecture Notes in Computer Science*, pp. 414–428. Springer.

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

Koch, C., Leone, N., & Pfeifer, G. (2003). Enhancing disjunctive logic programming systems by SAT checkers. *Artificial Intelligence*, *15*(1-2), 177–212.

Koponen, L., Oikarinen, E., Janhunen, T., & Säilä, L. (2015). Optimizing phylogenetic supertrees using answer set programming. *Theory and Practice of Logic Programming*, *15*(4-5), 604–619.

Le Berre, D., & Parrain, A. (2010). The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, *7*(2-3), 59–64.

Lee, J. (2005). A model-theoretic counterpart of loop formulas. In Kaelbling, L., & Saffiotti, A. (Eds.), *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pp. 503–508. Professional Book Center.

Lee, J., & Lifschitz, V. (2003). Loop formulas for disjunctive logic programs. In Palamidessi, C. (Ed.), *Proceedings of the Nineteenth International Conference on Logic Programming (ICLP'03)*, Vol. 2916 of *Lecture Notes in Computer Science*, pp. 451–465. Springer.

Leone, N., Gottlob, G., Rosati, R., Eiter, T., Faber, W., Fink, M., Greco, G., Ianni, G., Kałka, E., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Ruzzi, M., Staniszkis, W., & Terracina, G. (2005). The INFOMIX system for advanced integration of incomplete and inconsistent data. In Özcan, F. (Ed.), *Proceedings of the Twenty-fourth ACM SIGMOD International Conference on Management of Data (SIGMOD'05)*, pp. 915–917. ACM Press.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., & Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, *7*(3), 499–562.

Leone, N., Rullo, P., & Scarcello, F. (1997). Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation*, *135*(2), 69–112.

Li, C., & Manyà, F. (2009). MaxSAT. In Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.), *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications*, pp. 613–631. IOS Press.

Lierler, Y. (2005). Disjunctive answer set programming via satisfiability. In Baral, C., Greco, G., Leone, N., & Terracina, G. (Eds.), *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, Vol. 3662 of *Lecture Notes in Computer Science*, pp. 447–451. Springer.

Lierler, Y., Maratea, M., & Ricca, F. (2016). Systems, engineering environments, and competitions. *AI Magazine*, *37*(3), 45–52.

Lifschitz, V., & Razborov, A. (2006). Why are there so many loop formulas?. *ACM Transactions on Computational Logic*, *7*(2), 261–268.

Lin, F., & Zhao, Y. (2004). ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, *157*(1-2), 115–137.

Liu, G., Janhunen, T., & Niemelä, I. (2012). Answer set programming via mixed integer programming. In Brewka, G., Eiter, T., & McIlraith, S. (Eds.), *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, pp. 32–42. AAAI Press.

LP/CP-Comp (2016). *LP/CP Programming Contest 2016*. `https://sites.google.com/site/prologcontest2016/`.

LPNMR (2015). *13th International Conference on Logic Programming and Non-Monotonic Reasoning*. `http://lpnmr2015.mat.unical.it`.

Manna, M., Ricca, F., & Terracina, G. (2013). Consistent query answering via ASP from different perspectives: Theory and practice. *Theory and Practice of Logic Programming*, *13*(2), 227–252.

Manna, M., Ricca, F., & Terracina, G. (2015). Taming primary key violations to query large inconsistent data via ASP. *Theory and Practice of Logic Programming*, *15*(4-5), 696–710.

Maratea, M., Pulina, L., & Ricca, F. (2013). Automated selection of grounding algorithm in answer set programming. In Baldoni, M., Baroglio, C., Boella, G., & Micalizio, R. (Eds.), *Proceedings of the Thirteenth International Conference on Advances in Artificial Intelligence (AI*IA'13)*, Vol. 8249 of *Lecture Notes in Computer Science*, pp. 73–84. Springer.

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

Maratea, M., Pulina, L., & Ricca, F. (2015a). Multi-engine ASP solving with policy adaptation. *Journal of Logic and Computation*, *25*(6), 1285–1306.

Maratea, M., Pulina, L., & Ricca, F. (2015b). Multi-level algorithm selection for ASP. In Calimeri, F., Ianni, G., & Truszczyński, M. (Eds.), *Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15)*, Vol. 9345 of *Lecture Notes in Computer Science*, pp. 439–445. Springer.

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

Mariën, M., Wittocx, J., Denecker, M., & Bruynooghe, M. (2008). SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In Kleine Büning, H., & Zhao, X. (Eds.), *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, Vol. 4996 of *Lecture Notes in Computer Science*, pp. 211–224. Springer.

Marques-Silva, J., & Sakallah, K. (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, *48*(5), 506–521.

MaxSAT-Comp. *MaxSAT Evaluations*. `http://www.maxsat.udl.cat`.

MaxSAT-Comp (2014). *Ninth Max-SAT Evaluation*. `http://www.maxsat.udl.cat/14/`.

Mellarkod, V., & Gelfond, M. (2008). Integrating answer set reasoning with constraint solving techniques. In Garrigue, J., & Hermenegildo, M. (Eds.), *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, Vol. 4989 of *Lecture Notes in Computer Science*, pp. 15–31. Springer.

Mellarkod, V., Gelfond, M., & Zhang, Y. (2008). Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, *53*(1-4), 251–287.

MISC-Comp. *Mancoosi International Solver Competition*. `http://www.mancoosi.org/misc/`.

Morgado, A., Heras, F., Liffiton, M., Planes, J., & Marques-Silva, J. (2013). Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, *18*(4), 478–534.

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pp. 530–535. ACM Press.

Nguyen, M., Janhunen, T., & Niemelä, I. (2011). Translating answer-set programs into bit-vector logic. In Tompits, H., Abreu, S., Oetsch, J., Pührer, J., Seipel, D., Umeda, M., & Wolf, A. (Eds.), *Proceedings of the Nineteenth International Conference on Applications of Declarative Programming and Knowledge Management (INAP'11) and the Twenty-fifth Workshop on Logic Programming (WLP'11)*, Vol. 7773 of *Lecture Notes in Computer Science*, pp. 95–113. Springer.

Niemelä, I. (2008). Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence*, *53*(1-4), 313–329.

Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., & Barry, M. (2001). An A-Prolog decision support system for the space shuttle. In Ramakrishnan, I. (Ed.), *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, Vol. 1990 of *Lecture Notes in Computer Science*, pp. 169–183. Springer.

Ostrowski, M., & Schaub, T. (2012). ASP modulo CSP: The clingcon system. *Theory and Practice of Logic Programming*, *12*(4-5), 485–503.

PB-Comp (2016). *Pseudo-Boolean Competition 2016*. `http://www.cril.univ-artois.fr/PB16/`.

Perri, S., Ricca, F., & Sirianni, M. (2013). Parallel instantiation of ASP programs: Techniques and experiments. *Theory and Practice of Logic Programming*, *13*(2), 253–278.

Pfeifer, G. (2004). Improving the model generation/checking interplay to enhance the evaluation of disjunctive programs. In Lifschitz, V., & Niemelä, I. (Eds.), *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, Vol. 2923 of *Lecture Notes in Computer Science*, pp. 220–233. Springer.

Pontelli, E., Balduccini, M., & Bermudez, F. (2003). Non-monotonic reasoning on Beowulf platforms. In Dahl, V., & Wadler, P. (Eds.), *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL'03)*, Vol. 2562 of *Lecture Notes in Computer Science*, pp. 37–57. Springer.

Prolog-Comp. *The Prolog Programming Contests*. `https://people.cs.kuleuven.be/~bart.demoen/PrologProgrammingContests/`.

Prosser, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, *9*, 268–299.

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

QBF-Comp (2006). *QBF Evaluation 2006*. `http://www.qbflib.org/event_page.php?year=2006`.

QBF-Comp (2016). *QBF Evaluation 2016*. `http://www.qbflib.org/qbfeval16.php`.

Ricca, F., Faber, W., & Leone, N. (2006). A backjumping technique for disjunctive logic programming. *AI Communications*, *19*(2), 155–172.

Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., & 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. (1976). The algorithm selection problem. *Advances in Computers*, *15*, 65–118.

Roussel, O. (2011). Description of ppfolio. `http://www.cril.univ-artois.fr/~roussel/ppfolio/solver1.pdf`.

SAT-Comp. *The International SAT Competitions Web Page*. `http://www.satcompetition.org`.

SAT-Comp (2011). *SAT Competition 2011*. `http://www.satcompetition.org/2011/`.

SAT-Comp (2014). *SAT Competition 2014*. `http://www.satcompetition.org/2014/`.

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

SMT-Comp. *Annual SMT Competition*. `http://smtcomp.sourceforge.net`.

Susman, B., & Lierler, Y. (2016). SMT-based constraint answer set solver EZSMT. In Carro, M., & King, A. (Eds.), *Technical Communications of the Thirty-second International Conference on Logic Programming (ICLP'16)*, Vol. 52 of *Open Access Series in Informatics*. Schloss Dagstuhl.

Sutcliffe, G. (2009). The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, *43*(4), 337–362.

Sutcliffe, G. (2016). The 8th IJCAR automated theorem proving system competition – CASC-J8. *AI Communications*, *29*(5), 607–619.

Syrjänen, T. (2001). Omega-restricted logic programs. In Eiter, T., Faber, W., & Truszczyński, M. (Eds.), *Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, Vol. 2173 of *Lecture Notes in Computer Science*, pp. 267–279. Springer.

Tamura, N., Taga, A., Kitagawa, S., & Banbara, M. (2009). Compiling finite linear CSP into SAT. *Constraints*, *14*(2), 254–272.

Toni, L., Aparicio-Pardo, R., Simon, G., Blanc, A., & Frossard, P. (2014). Optimal set of video representations in adaptive streaming. In Zimmermann, R. (Ed.), *Proceedings of the 2014 ACM Multimedia Systems Conference (MMSys'14)*, pp. 271–282. ACM Press.

Vallati, M., Chrpa, L., Grzes, M., McCluskey, T., Roberts, M., & Sanner, S. (2015). The 2014 international planning competition: Progress and trends. *AI Magazine*, *36*(3), 90–98.

Van Gelder, A., Ross, K., & Schlipf, J. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, *38*(3), 620–650.

VSL (2014). *Vienna Summer of Logic*. `http://vsl2014.at`.

Ward, J., & Schlipf, J. (2004). Answer set programming with clause learning. In Lifschitz, V., & Niemelä, I. (Eds.), *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, Vol. 2923 of *Lecture Notes in Computer Science*, pp. 302–313. Springer.

Wittocx, J., Mariën, M., & Denecker, M. (2008). GidL: A grounder for FO+. In Pagnucco, M., & Thielscher, M. (Eds.), *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, pp. 189–198. `https://lirias.kuleuven.be/bitstream/123456789/197022/1/nmr08.pdf`.

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

Zhang, H., Bonacina, M., & Hsiang, J. (1996). PSATO: A distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, *21*(4), 543–560.

Zhang, L., Madigan, C., Moskewicz, M., & Malik, S. (2001). Efficient conflict driven learning in a Boolean satisfiability solver. In Ernst, R. (Ed.), *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'01)*, pp. 279–285. ACM Press.