TPLP **20** (2): 176–204, 2020. © Cambridge University Press 2019
doi:10.1017/S1471068419000061 First published online 31 May 2019

The Seventh Answer Set Programming Competition: Design and Results∗

MARTIN GEBSER
Institute for Computer Science, University of Potsdam, Potsdam, Germany
(*e-mail:* gebser@cs.uni-potsdam.de)

MARCO MARATEA
DIBRIS, University of Genova, Genova, Italy
(*e-mail:* marco@dibris.unige.it)

FRANCESCO RICCA
Dipartimento di Matematica e Informatica, Università della Calabria, Rende, Italy
(*e-mail:* ricca@mat.unical.it)

submitted 28 April 2018; revised 25 April 2019; accepted 5 May 2019

Abstract

Answer Set Programming (ASP) is a prominent knowledge representation language with roots in logic programming and non-monotonic reasoning. Biennial ASP competitions are organized in order to furnish challenging benchmark collections and assess the advancement of the state of the art in ASP solving. In this paper, we report on the design and results of the Seventh ASP Competition, jointly organized by the University of Calabria (Italy), the University of Genova (Italy), and the University of Potsdam (Germany), in affiliation with the 14th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR 2017).

KEYWORDS: Answer Set Programming, competition

1 Introduction

Answer Set Programming (ASP) is a prominent knowledge representation language with roots in logic programming and non-monotonic reasoning (Baral 2003; Brewka *et al.* 2011; Eiter *et al.* 2009; Gelfond and Leone 2002; Lifschitz 2002; Marek and Truszczyński 1999; Niemelä 1999). The goal of the ASP Competition series is to promote advancements in ASP methods, collect challenging benchmarks, and assess the state of the art in ASP solving (see, e.g., Alviano *et al.* 2015, 2017; Bruynooghe *et al.* 2015; Gebser *et al.* 2015; Lefèvre *et al.* 2017; Maratea *et al.* 2015; Marple and Gupta 2014; Calimeri *et al.* 2017 for recent ASP systems, and Gebser *et al.* 2018 for a recent survey). Following a nowadays

∗ The organizers of the Seventh ASP Competition would like to thank the LPNMR 2017 officials for the co-location of the event. We also acknowledge the Department of Mathematics and Computer Science at the University of Calabria for supplying the computational resources to run the competition. Finally, we thank all solver and benchmark contributors, and participants, who worked hard to make this competition possible.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

customary practice of publishing results of AI-based competitions in archival journals, where they are expected to remain available and can be used as references, the results of ASP competitions have been hosted in prominent journals of the area (see Calimeri *et al.* 2014, 2016; Gebser *et al.* 2017b). Continuing the tradition, this paper reports on the design and results of the Seventh ASP Competition,[1] which was jointly organized by the University of Calabria (Italy), the University of Genova (Italy), and the University of Potsdam (Germany), in affiliation with the 14th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR 2017).[2]

The Seventh ASP Competition is conceived along the lines of the System track of previous competition editions (Calimeri *et al.* 2016; Lierler *et al.* 2016; Gebser *et al.* 2016, 2017b), with the following characteristics: (i) benchmarks adhere to the ASP-Core-2 standard modeling language,[3] (ii) sub-tracks are based on language features utilized in problem encodings (e.g., aggregates, choice or disjunctive rules, queries, and weak constraints), and (iii) problem instances are classified and selected according to their expected hardness. Both single and multi-processor categories are available in the competition, where solvers in the first category run on a single CPU (core), while they can take advantage of multiple processors (cores) in the second category. In addition to the basic competition design, which has also been addressed in a preliminary version of this report (Gebser *et al.* 2017a), we detail the revised benchmark selection process as well as the results of the event, which were orally presented during LPNMR 2017 in Hanasaari, Espoo, Finland.

The rest of this paper is organized as follows. Section 2 introduces the format of the Seventh ASP Competition. In Section 3, we describe new problem domains contributed to this competition edition as well as the revised benchmark selection process for picking instances to run in the competition. The participant systems of the competition are then surveyed in Section 4. In Section 5, we then present the results of the Seventh ASP Competition along with the winning systems of competition categories. Section 6 concludes the paper with final remarks.

2 Competition format

This section gives an overview of competition categories, sub-tracks, and scoring scheme(s), which are similar to the previous ASP Competition edition. One addition though concerns the system ranking of Optimization problems, where a ranking by the number of instances solved "optimally" complements the relative scoring scheme based on solution quality used previously.

Categories. The competition includes two categories, depending on the computational resources provided to participant systems: *SP*, where one processor (core) is available, and *MP*, where multiple processors (cores) can be utilized. While the *SP* category aims at sequential solving systems, *MP* allows for exploiting parallelism.

[1] http://aspcomp2017.dibris.unige.it.
[2] http://lpnmr2017.aalto.fi.
[3] http://www.mat.unical.it/aspcomp2013/ASPStandardization/.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

Sub-tracks. Both categories are structured into the following four sub-tracks, based on the ASP-Core-2 language features utilized in problem encodings:

- **Sub-track #1** (*Basic Decision*): Encodings consisting of non-disjunctive and non-choice rules (also called normal rules) with classical and built-in atoms only.
- **Sub-track #2** (*Advanced Decision*): Encodings exploiting the language fragment allowing for aggregates, choice as well as disjunctive rules, and queries, yet excepting weak constraints and non-head-cycle-free (non-HCF) disjunction.
- **Sub-track #3** (*Optimization*): Encodings extending the aforementioned language fragment by weak constraints while still excepting non-HCF disjunction.
- **Sub-track #4** (*Unrestricted*): Encodings exploiting the full language and, in particular, non-HCF disjunction.

A problem domain, that is, an encoding together with a collection of instances, belongs to the first sub-track its problem encoding is compatible with.

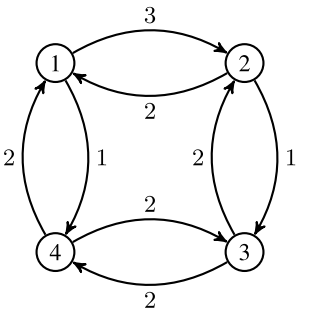
Example 1
To illustrate the sub-tracks and respective language features, consider the directed graph displayed in Figure 1(a) and the corresponding fact representation given in Figure 1(b). Facts over the predicate `node`/1 specify the nodes of the graph, those over `edge`/2 provide the edges, and `cost`/3 associates each edge with its cost. The idea in the following is to encode the well-known Traveling Salesperson problem, which is about finding a Hamiltonian cycle, that is, a round trip visiting each node exactly once, such that the sum of edge costs is minimal. Note that the example graph in Figure 1 includes precisely two outgoing edges per node, and for simplicity the encodings in Figure 1(c)–(e) build on this property, while accommodating an arbitrary number of outgoing edges would also be possible with appropriate modifications.

The first encoding in Figure 1(d) complies with the language fragment of Sub-track #1, as it does not make use of aggregates, choice or disjunctive rules, queries, and weak constraints. Note that terms starting with an uppercase letter, such as `X`, `Y`, and `Z`, stand for universally quantified first-order variables, `Y != Z` is a built-in atom, and `not` denotes the (default) negation connective. Given this, the rule in line 1 expresses that exactly one of the two outgoing edges per node must belong to a Hamiltonian cycle, represented by atoms over the predicate `cycle`/2 within a stable model (Lifschitz 2008). Starting from the distinguished node `1`, the least fixpoint of the rules in lines 2 and 3 provides the nodes reachable from `1` via the edges of a putative Hamiltonian cycle. The so-called integrity constraint, that is, a rule with an empty head that is interpreted as false, in line 4 then asserts that all nodes must be reachable from the starting node `1`, which guarantees that stable models coincide with Hamiltonian cycles. While edge costs are not considered so far, the encoding in Figure 1(c) can be used to decide whether a Hamiltonian cycle exists for a given graph (with precisely two outgoing edges per node).

The second encoding in Figure 1(d) includes a choice rule in line 1, thus making use of language features permitted in Sub-track #2, but incompatible with Sub-track #1. The instance of this choice rule obtained for the node `1`, `{cycle(1,2); cycle(1,4)} = 1.`, again expresses that exactly one outgoing edge of node `1` must be included in a Hamiltonian cycle, and respective rule instances apply to the other nodes of the example graph in Figure 1(a). Notably, the choice rule adapts to an arbitrary number of outgoing edges,

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

(a)



(b)

```
1 node(1). edge(1,2). cost(1,2,3).
2          edge(1,4). cost(1,4,1).
3 node(2). edge(2,1). cost(2,1,2).
4          edge(2,3). cost(2,3,1).
5 node(3). edge(3,2). cost(3,2,2).
6          edge(3,4). cost(3,4,2).
7 node(4). edge(4,1). cost(4,1,2).
8          edge(4,3). cost(4,3,2).
```

(c)

```
1 cycle(X,Y) :- edge(X,Y), edge(X,Z), Y != Z, not cycle(X,Z).
2 reach(Y) :- cycle(1,Y).
3 reach(Y) :- cycle(X,Y), reach(X).
4 :- node(Y), not reach(Y).
```

(d)

```
1 {cycle(X,Y) : edge(X,Y)} = 1 :- node(X).
2 reach(Y) :- cycle(1,Y).
3 reach(Y) :- cycle(X,Y), reach(X).
4 :- node(Y), not reach(Y).
```

(e)

```
1 cycle(1,Y) | cycle(1,Z) :- edge(1,Y), edge(1,Z), Y != Z.
2 cycle(X,Y) | cycle(X,Z) :- edge(X,Y), edge(X,Z), Y != Z,
                                          reach(X), X != 1.
3 reach(Y) :- cycle(X,Y).
4 :- node(Y), not reach(Y).
```

(f)

```
1 :~ cycle(X,Y), cost(X,Y,C). [C,X,Y]
```

Fig. 1. An example graph with edge costs, its fact representation, and corresponding encodings. (a) A directed graph with edge costs. (b) Fact representation of the graph in (a). (c) Basic Decision encoding of Hamiltonian cycles. (d) Advanced Decision encoding of Hamiltonian cycles. (e) Unrestricted encoding of Hamiltonian cycles. (f) Weak constraint for Hamiltonian cycle Optimization.

and the assumption that there are precisely two per node could be dropped when using the encoding in Figure 1(d).

The rules in lines 1 and 2 of the third encoding in Figure 1(e) are disjunctive, and rule instances as follows are obtained together with line 3:

```
cycle(1,2) | cycle(1,4).
cycle(2,1) | cycle(2,3) :- reach(2).
cycle(3,2) | cycle(3,4) :- reach(3).
cycle(4,1) | cycle(4,3) :- reach(4).
```

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

```
reach(1) :- cycle(2,1).     reach(3) :- cycle(2,3).
reach(1) :- cycle(4,1).     reach(3) :- cycle(4,3).
reach(2) :- cycle(1,2).     reach(2) :- cycle(3,2).
reach(4) :- cycle(1,4).     reach(4) :- cycle(3,4).
```

Observe that `reach(3)` occurs in the body of a disjunctive rule with `cycle(3,2)` and `cycle(3,4)` in the head. These atoms further imply `reach(2)` or `reach(4)`, respectively, which lead on two disjunctive rules, one containing `cycle(2,3)` in the head and the other `cycle(4,3)`. As the latter two atoms also occur in the body of rules with `reach(3)` in the head, we have that all of the mentioned atoms recursively depend on each other. `cycle(3,2)` and `cycle(3,4)` jointly constitute the head of a disjunctive rule; this means that rule instances obtained from the encoding in Figure 1(e) are non-HCF (Ben-Eliyahu and Dechter 1994) and thus fall into a syntactic class of logic programs able to express problems at the second level of the polynomial hierarchy (Eiter and Gottlob 1995). Hence, the encoding in Figure 1(e) makes use of a language feature permitted in Sub-track #4 only.

Given that either of the encodings in Figure 1(c)–(e) yields stable models corresponding to Hamiltonian cycles, the weak constraint in Figure 1(f) can be added to each of them to express the objective of finding a Hamiltonian cycle whose sum of edge costs is minimal. In case of the encodings in Figure 1(c) and (d), the addition of the weak constraint leads to a reclassification into Sub-track #3, since the focus is shifted from a Decision to an Optimization problem. For the encoding in Figure 1(e), Sub-track #4 still matches when adding the weak constraint, as non-HCF disjunction is excluded in the other sub-tracks. ■

Scoring scheme. The applied scoring schemes are based on 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 are solved correctly.

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

For *Decision* problems and *Query answering* tasks, 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 time and memory limits of 20 min wall-clock time and 12 GB RAM per run.

For *Optimization* problems, we employ two alternative scoring schemes. The first one, which has also been used in the previous competition edition, performs a relative ranking of systems by solution quality, following the approach of the Mancoosi International Solver Competition.[4] 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

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

[4] http://www.mancoosi.org/misc/.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

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 $S_1(D)$ of system S in domain D is then taken as the sum of scores $S(D, I)$ over the N instances I in D.

The second scoring scheme considers the number of instances solved "optimally," that is, a confirmed optimum solution or unsatisfiability is reported. Hence, the score $S_2(D)$ of a system S in a domain D is defined as $S(D)$ above, with N_S being the number of instances solved optimally. This second scoring scheme (inspired by the MaxSAT Competition) gives more importance to solvers that can actually solve instances to the optimum, but it does not consider "non-optimal" solutions. The two measures provide alternative perspectives on the performance of participants solving Optimization problems.

Note that, as with Decision problems and Query answering tasks, $S_1(D)$ and $S_2(D)$ range from 0 to 100 in each domain. $S_1(D)$ focuses on the best solutions found by participant systems, while $S_2(D)$ on completed runs.

In each category and respective sub-tracks, 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 as a tie-breaking criterion.

3 Benchmark suite and selection

The benchmark suite of the Seventh ASP Competition includes 36 domains, where 28 stem from the previous competition edition (Gebser *et al.* 2017b), and 8 domains, as well as additional instances for the *Graph Coloring* problem, were newly submitted. We first describe the eight new domains and then detail the instance selection process based on empirical hardness.

3.1 New domains

The eight new domains of this ASP Competition edition can be roughly characterized as closely related to machine learning (*Bayesian Network Learning*, *Markov Network Learning*, and *Supertree Construction*), personnel scheduling (*Crew Allocation* and *Resource Allocation*), or combinatorial problem solving (*Paracoherent Answer Sets*, *Random Disjunctive ASP*, and *Traveling Salesperson*), respectively. While *Traveling Salesperson* constitutes a classical Optimization problem in computer science, the five domains stemming from machine learning and personnel scheduling are application oriented, and the contribution of such practically relevant benchmarks to the ASP Competition is particularly encouraged (Gebser *et al.* 2017b). Moreover, the *Paracoherent Answer Sets* and *Random Disjunctive ASP* domains contribute to Sub-track #4, which was sparsely populated in recent ASP Competition editions, and beyond theoretical interest these benchmarks are relevant to logic program debugging and industrial solvers development. The following paragraphs provide more detailed background information for each of the new domains.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

Bayesian Network Learning. Bayesian networks are directed acyclic graphs representing (in)dependence relations between variables in multi-variate data analysis. Learning the structure of Bayesian networks, that is, selecting arcs such that the resulting graph fits given data best, is a combinatorial Optimization problem amenable to constraint-based solving methods like the one proposed in Cussens (2011). In fact, data sets from the literature serve as instances in this domain, while a problem encoding in ASP-Core-2 expresses optimal Bayesian networks, given by directed acyclic graphs whose associated cost is minimal.

Crew Allocation. This scheduling problem, which has also been addressed by related constraint-based solving methods (Guerinik and Caneghem 1995), deals with allocating crew members to flights such that the amount of personnel with certain capabilities (e.g., role on board and spoken language) as well as off-times between flights are sufficient. Moreover, instances with different numbers of flights and available personnel restrict the amount of personnel that may be allocated to flights in such a way that no schedule is feasible under the given restrictions.

Markov Network Learning. As with Bayesian networks, the learning problem for Markov networks (Janhunen *et al.* 2017) aims at the Optimization of graphs representing the dependence structure between variables in statistical inference. In this domain, the graphs of interest are undirected and required to be chordal, while associated scores express marginal likelihood with respect to given data. Problem instances of varying hardness are obtained by taking samples of different size and density from literature data sets.

Resource Allocation. This scheduling problem deals with allocating the activities of business processes to human resources such that role requirements and temporal relations between activities are met (Havur *et al.* 2016). Moreover, the total makespan of schedules is subject to an upper bound as well as Optimization. The hardness of instances in this domain varies with respect to the number of activities, temporal relations, available resources, and upper bounds.

Supertree Construction. The goal of the supertree construction problem (Koponen *et al.* 2015) is to combine the leaves of several given phylogenetic subtrees into a single tree fitting the given subtrees as closely as possible. That is, Optimization aims at preserving the structure of subtrees, where the introduction of intermediate nodes between direct neighbors is tolerated, while the avoidance of such intermediate nodes is an Optimization target as well. Instances of varying hardness are obtained by mutating projections of binary trees with different numbers of leaves.

Traveling Salesperson. The well-known traveling salesperson problem (Applegate *et al.* 2007) is to find a round trip through a (directed) graph that is optimal in terms of the accumulated edge costs. Instances in this domain are twofold by stemming from the TSPLIB repository[5] or being randomly generated to increase the variety in the ASP Competition, respectively.

[5] http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

Paracoherent Answer Sets. Given an incoherent logic program P, that is, a program P without answer sets, a paracoherent (or semi-stable) answer set corresponds to a gap-minimal answer set of the epistemic transformation of P (Inoue and Sakama 1996; Amendola *et al.* 2016). The instances in this domain, used in Amendola *et al.* (2017, 2018) to evaluate genuine implementations of paracoherent ASP, are obtained by grounding and transforming incoherent programs from previous editions of the ASP Competition. In particular, weak constraints single out answer sets of a transformed program containing a minimal number of atoms that are actually underivable from the original program.

Random Disjunctive ASP. The disjunctive logic programs in this domain (Amendola *et al.* 2017) express random 2QBF formulas, given as conjunctions of terms in disjunctive normal form, by an extension of the Eiter–Gottlob encoding (Eiter and Gottlob 1995). Parameters controlling the random generation of 2QBF formulas (e.g., number of variables and number of conjunctions) are set so that instances lie close to the phase transition region while having an expected average solving time below the competition timeout of 20 min per run.

3.2 Benchmark selection

Table 1 gives an overview of all problem domains, grouped by their respective sub-tracks, of the Seventh ASP Competition, where the names of new domains are highlighted in boldface. The second column provides the computational task addressed in a domain, distinguishing Decision ("D") and Optimization ("O") problems as well as Query answering ("Q"). Further columns categorize the instances in each domain by their empirical hardness, where hardness classes are based on the performance of the same reference systems, that is, CLASP, LP2NORMAL2+CLASP, and WASP-1.5, as in the previous ASP Competition edition (Gebser *et al.* 2017b):[6]

- **Easy**: Instances completed by at least one reference system in more than 20 s and by all reference systems in less than 2-min solving time.
- **Medium**: Instances completed by at least one reference system in more than 2 min and by all reference systems in less than 20-min (the competition timeout) solving time.
- **Hard**: Instances completed by at least one reference system in less than 40 min, while also at least one (not necessarily the same) reference system did not finish solving in 20 min.
- **Too hard**: Instances such that none of the reference systems finished solving in 40 min.

For each of these hardness classes, numbers of available instances per problem domain are shown within parentheses in Table 1, further distinguishing satisfiable and unsatisfiable instances, whose respective numbers are given first or second, respectively. In

[6] This choice of reference systems allows us to reuse the runtime results for previous domains gathered in exhaustive experiments on all available instances that took about 212 CPU days on the competition platform. Instances that do not belong to any of the listed hardness classes are in the majority of cases "very easy" and the remaining ones "non-groundable," and we exclude such (uninformative regarding the system ranking) instances from the benchmark suite.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

Table 1. *Problem domains of benchmarks for the Seventh ASP Competition, where entries in the* **P** *column indicate* Decision *("*D*"),* Optimization *("*O*"), or* Query answering *("*Q*") tasks. The remaining columns provide numbers of instances per empirical hardness class, distinguishing* satisfiable *and* unsatisfiable *instances classified as* **Easy**, **Medium**, *or* **Hard**, *while* **Too hard** *instances are divided into those known to be* satisfiable *and others whose satisfiability is* unknown. *For each hardness class and satisfiability status, the number in front of parentheses stands for the selected instances out of the respective available instances whose number is given in parentheses.*

Domain	P	Easy				Medium				Hard				Too hard				
Graph Colouring	D	1	(1)	3	(5)	2	(2)	4	(21)	2	(3)	5	(16)	0	(0)	3	(3)	Sub-track #1
Knight Tour with Holes	D	2	(5)	3	(4)	4	(4)	0	(0)	4	(9)	0	(0)	0	(0)	7	(302)	
Labyrinth	D	4	(45)	0	(0)	5	(72)	0	(0)	7	(83)	0	(0)	0	(0)	4	(8)	
Stable Marriage	D	0	(0)	0	(0)	3	(3)	0	(0)	6	(15)	1	(1)	0	(0)	10	(55)	
Visit-all	D	8	(14)	0	(0)	5	(5)	0	(0)	7	(40)	0	(0)	0	(0)	0	(0)	
Combined Configuration	D	1	(1)	0	(0)	1	(1)	0	(0)	12	(44)	0	(0)	0	(0)	6	(34)	Sub-track #2
Consistent Query Answering	Q	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	20	(120)	
Crew Allocation	D	0	(0)	4	(10)	0	(0)	6	(11)	0	(0)	6	(10)	0	(0)	4	(6)	
Graceful Graphs	D	3	(3)	0	(0)	4	(4)	1	(1)	4	(28)	2	(2)	0	(0)	6	(21)	
Incremental Scheduling	D	2	(11)	2	(6)	3	(47)	2	(11)	3	(37)	2	(10)	0	(0)	6	(76)	
Nomystery	D	4	(4)	0	(0)	4	(5)	0	(0)	4	(10)	0	(0)	0	(0)	8	(32)	
Partner Units	D	3	(9)	1	(1)	4	(34)	0	(0)	3	(15)	1	(1)	0	(0)	8	(32)	
Permutation Pattern Matching	D	2	(16)	2	(32)	2	(14)	2	(58)	0	(0)	5	(14)	0	(0)	7	(20)	
Qualitative Spatial Reasoning	D	5	(35)	4	(35)	4	(34)	2	(19)	3	(7)	2	(2)	0	(0)	0	(0)	
Reachability	Q	0	(0)	0	(0)	10	(30)	10	(30)	0	(0)	0	(0)	0	(0)	0	(0)	
Ricochet Robots	D	2	(2)	0	(0)	7	(18)	0	(0)	4	(181)	0	(0)	0	(0)	7	(38)	
Sokoban	D	2	(77)	2	(10)	2	(84)	2	(8)	5	(114)	2	(12)	0	(0)	5	(620)	
Bayesian Network Learning	O	4	(4)	0	(0)	4	(8)	0	(0)	8	(19)	0	(0)	4	(20)	0	(6)	Sub-track #3
Connected Still Life	O	0	(0)	0	(0)	5	(5)	0	(0)	10	(70)	0	(0)	5	(45)	0	(0)	
Crossing Minimization	O	1	(1)	0	(0)	1	(1)	0	(0)	17	(80)	0	(0)	1	(1)	0	(0)	
Markov Network Learning	O	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	10	(10)	10	(50)	
Maximal Clique	O	0	(0)	0	(0)	0	(0)	0	(0)	10	(41)	0	(0)	10	(94)	0	(1)	

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

Table 1. *(Continued)*

Domain	P	Easy				Medium				Hard				Too hard			
MaxSAT	O	0	(0)	0	(0)	4	(4)	0	(0)	0	(0)	0	(0)	0	(0)	16	(50)	
Resource Allocation	O	–	(3)	–	(0)	–	(3)	–	(0)	–	(0)	–	(0)	–	(0)	–	(0)	
Steiner Tree	O	0	(0)	0	(0)	0	(0)	0	(0)	1	(1)	0	(0)	16	(45)	3	(3)	
Supertree Construction	O	0	(0)	0	(0)	0	(0)	0	(0)	6	(30)	0	(0)	14	(30)	0	(0)	
System Synthesis	O	0	(0)	0	(0)	0	(0)	0	(0)	8	(16)	0	(0)	8	(80)	4	(4)	
Traveling Salesperson	O	0	(0)	0	(0)	2	(2)	0	(0)	3	(3)	0	(0)	12	(60)	3	(3)	
Valves Location	O	6	(10)	0	(0)	2	(2)	0	(0)	7	(29)	0	(0)	5	(244)	0	(23)	
Video Streaming	O	11	(16)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	8	(22)	1	(1)	
Abstract Dialectical Frameworks	O	4	(18)	0	(0)	8	(20)	0	(0)	6	(122)	0	(0)	2	(2)	0	(0)	Sub-track #4
Complex Optimization	D	0	(0)	0	(0)	0	(0)	0	(0)	20	(78)	0	(0)	0	(0)	0	(0)	
Minimal Diagnosis	D	7	(158)	2	(55)	3	(9)	2	(8)	4	(4)	1	(1)	0	(0)	0	(0)	
Paracoherent Answer Sets	O	0	(0)	0	(0)	1	(1)	0	(0)	12	(112)	0	(0)	0	(0)	7	(43)	
Random Disjunctive ASP	D	0	(0)	0	(0)	0	(0)	0	(0)	5	(48)	13	(73)	0	(0)	2	(2)	
Strategic Companies	Q	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	0	(0)	20	(37)	

case of instances classified as "too hard," however, no reference system could report unsatisfiability, and thus the numbers of instances listed second refer to an unknown satisfiability status. Note that there are likewise no "too hard" instances of Decision problems or Query answering domains known as satisfiable, so that the respective numbers are zero. For example, the *Sokoban* domain features satisfiable as well as unsatisfiable instances for each hardness class apart from the "too hard" one, where 0 instances are known as satisfiable and 620 have an unknown satisfiability status. Unlike that, "too hard" instances of Optimization problems are frequently known to be satisfiable, in which case none of the reference systems was able to confirm an optimum solution within 40 min. Moreover, we discarded any instance of an Optimization problem that was reported to be unsatisfiable, so that the respective numbers given second are zero for the first three hardness classes. This applies, for example, to instances in the *Bayesian Network Learning* domain, including 4, 8, 19, and 20 satisfiable instances that are "easy," "medium," "hard," or "too hard," respectively, while the satisfiability status of further 6 "too hard" instances is unknown. Finally, the numbers in front of parentheses in Table 1 report how many instances were (randomly) selected per hardness class and satisfiability status, and the selection process is described in the rest of this section.

Given the numbers of satisfiable, unsatisfiable, or unknown in case of "too hard" instances per hardness class, our benchmark selection process aims at picking 20 instances in each problem domain such that the four hardness classes are balanced, while another share of instances is added freely. 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 20% of the instances in a domain. Since in most domains the instances are not evenly distributed, it is not possible though to insist on at least four instances per hardness class, and rather we have to compensate for underpopulated classes at which the number of available instances is smaller.

The input to our balancing scheme includes a collection C of classes, where each class is identified with the set of its contained instances. The first step of balancing then determines the non-empty classes from which instances can be picked:

$$\mathit{classes} \;=\; \{x \in C \mid x \neq \emptyset\}.$$

The number of non-empty classes is used to calculate how many instances should ideally be picked per class, where the calculation makes use of the parameters $n = 20$ and $m = 1$, standing for the total number of instances to select per domain and the fraction of instances to pick freely, respectively:

$$\mathit{target} \;=\; \lfloor n/(|\mathit{classes}| + m) \rfloor. \tag{1}$$

To account for underpopulated classes, in the next step we calculate the gap between the intended number of and the available instances in each class:

$$\mathit{gap}(x) \;=\; \begin{cases} 0 & \text{if } \; x \in C \setminus \mathit{classes}, \\ \mathit{target} - |x| & \text{if } \; x \in \mathit{classes}. \end{cases}$$

Example 2

In the *Graceful Graphs* domain, the "easy," "medium," "hard," and "too hard" classes contain 3, 5, 30, or 21 instances, respectively, when not (yet) distinguishing between satisfiable and unsatisfiable instances. Since all four hardness classes are non-empty, we

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

obtain $classes = \{$"easy," "medium," "hard," "too hard"$\}$. The calculation of instances to pick per class yields $target = \lfloor 20/(4+1) \rfloor = 4$, so that we aim at four instances per hardness class. Along with the number of instances available in each class, we then get $gap($"easy"$) = 4-3 = 1$, $gap($"medium"$) = 4-5 = -1$, $gap($"hard"$) = 4-30 = -26$, and $gap($"too hard"$) = 4-21 = -17$. Note that a positive number expresses underpopulation of a class relative to the intended number of instances, while negative numbers indicate capacities to compensate for such underpopulation. ■

Our next objective is to compensate for underpopulated classes by increasing the number of instances to pick from other classes in a fair way. Regarding hardness classes, our compensation scheme relies on the direct successor relation $\prec$ given by "easy" $\prec$ "medium," "medium" $\prec$ "hard," and "hard" $\prec$ "too hard." We denote the strict total order obtained as the transitive closure of $\prec$ by $<$, and its inverse relation by $>$. Moreover, we let $\circ$ below stand for either $<$ or $>$ to specify calculations that are performed symmetrically, such as determining the number of easier or harder instances available to compensate for the (potential) underpopulation of a class:

$$available(x)^\circ \;=\; \sum\nolimits_{x' \circ x} gap(x').$$

The possibility of compensation in favor of easier or harder instances is then determined as follows:

$$compensate(x)^\circ \;=\; \min\{(|gap(x)| + gap(x))/2, (|available(x)^\circ| - available(x)^\circ)/2\}.$$

The calculation is such that a positive gap, standing for the underpopulation of a class, is a prerequisite for obtaining a non-zero outcome, and the availability of easier or harder instances to compensate with is required in addition. Given the compensation possibilities, the following calculations decide about how many easier or harder instances, respectively, are to be picked to resolve an underpopulation, where the distribution should preferably be even and tie-breaking in favor of harder instances is used as secondary criterion if the number of instances to compensate for is odd[7]:

$$\begin{aligned} distribute(x)^> \;&=\; \min\{compensate(x)^<, \max\{|gap(x)| - compensate(x)^>, \lfloor |gap(x)|/2 \rfloor\}\},\\ distribute(x)^< \;&=\; \min\{compensate(x)^>, |gap(x)| - distribute(x)^>\}. \end{aligned}$$

It remains to choose classes whose numbers of instances are to be increased for compensation, where we aim to distribute instances to closest classes with compensation capacities. The following inductive calculation scheme accumulates instances to distribute according to this objective:

$$\begin{aligned} accumulate(x)^\circ \;&=\; \begin{cases} 0 \;\text{ if }\; \{x' \in C \mid x' \circ x\} = \emptyset, \\ accumulate(x')^\circ + distribute(x')^\circ - increase(x')^\circ \;\text{ if }\; x' \circ x \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{and}\;\; x' \prec x \;\; \text{or} \;\; x \prec x', \end{cases}\\ increase(x)^< \;&=\; \min\{accumulate(x)^<, (|gap(x)| - gap(x))/2\},\\ increase(x)^> \;&=\; \min\{accumulate(x)^>, (|gap(x)| - gap(x))/2 - increase(x)^<\}. \end{aligned}$$

[7] Given that $distribute(x)^>$ (or $distribute(x)^<$) is limited by $compensate(x)^<$ (or $compensate(x)^>$), the superscripts ">" and "<" refer to easier or harder instances, respectively, to be picked in addition. This reading is chosen for a convenient notation in the specification of classes whose numbers of instances are to be increased for compensation.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

In a nutshell, $accumulate(x)^<$ and $accumulate(x)^>$ express how many easier or harder instances, respectively, ought to be distributed up to a class x, and $increase(x)^<$ and $increase(x)^>$ stand for corresponding increases of the number of instances to be picked from x. The instances to increase with are then added to the original number of instances to pick from a class as follows:

$$select(x) = |x| - (|gap(x)| - gap(x))/2 + increase(x)^< + increase(x)^>.$$

Example 3
Given $gap(\text{“easy”}) = 1$, $gap(\text{“medium”}) = -1$, $gap(\text{“hard”}) = -26$, and $gap(\text{“too hard”}) = -17$ from Example 2 for the *Graceful Graphs* domain, we obtain the following numbers indicating the availability of easier instances: $available(\text{“easy”})^< = 0$, $available(\text{“medium”})^< = 1$, $available(\text{“hard”})^< = 1 + (-1) = 0$, and $available(\text{“too hard”})^< = 1 + (-1) + (-26) = -26$. Likewise, the available harder instances are expressed by $available(\text{“too hard”})^> = 0$, $available(\text{“hard”})^> = -17$, $available(\text{“medium”})^> = (-17) + (-26) = -43$, and $available(\text{“easy”})^> = (-17) + (-26) + (-1) = -44$. Again note that positive numbers like $available(\text{“medium”})^< = 1$ represent a (cumulative) underpopulation, while negative numbers such as $available(\text{“easy”})^> = -44$ indicate compensation capacities.

Considering “easy” instances, we further calculate $compensate(\text{“easy”})^< = \min\{(|1| + 1)/2, (|0| - 0)/2\} = 0$ and $compensate(\text{“easy”})^> = \min\{(|1| + 1)/2, (|-44| - (-44))/2\} = 1$. This tells us that we can add one harder instance to compensate for the underpopulation of the “easy” class, while $compensate(x)^\circ = compensate(\text{“easy”})^< = 0$ for the other classes $x \in \{\text{“medium,” “hard,” “too hard”}\}$ and $\circ \in \{<, >\}$. Given that instances to distribute are limited by compensation possibilities, which are non-zero at underpopulated classes only, it is sufficient to concentrate on “easy” instances in the *Graceful Graphs* domain. This yields $distribute(\text{“easy”})^> = \min\{0, \max\{1 - 1, 0\}\} = 0$ and $distribute(\text{“easy”})^< = \min\{1, 1 - 0\} = 1$, so that one harder instance is to be picked more.

The calculation of instance number increases to compensate for underpopulated classes then starts with $accumulate(\text{“easy”})^< = 0$, $increase(\text{“easy”})^< = \min\{0, (|1| - 1)/2\} = 0$, $accumulate(\text{“medium”})^< = 0 + 1 - 0 = 1$, $increase(\text{“medium”})^< = \min\{1, (|-1| - (-1))/2\} = 1$, and $accumulate(\text{“hard”})^< = 1 + 0 - 1 = 0$. That is, the instance to distribute from the underpopulated “easy” class to some harder class increases the number of “medium” instances while we obtain $increase(\text{“hard”})^< = accumulate(\text{“too hard”})^< = increase(\text{“too hard”})^< = 0$ as well as $accumulate(x)^> = increase(x)^> = 0$ for all $x \in \{\text{“easy,” “medium,” “hard,” “too hard”}\}$. The final numbers of instances to pick per hardness class in the *Graceful Graphs* domain are thus determined by $select(\text{“easy”}) = 3 - (|1| - 1)/2 + 0 + 0 = 3$, $select(\text{“medium”}) = 5 - (|-1| - (-1))/2 + 1 + 0 = 5$, $select(\text{“hard”}) = 30 - (|-26| - (-26))/2 + 0 + 0 = 4$, and $select(\text{“too hard”}) = 21 - (|-17| - (-17))/2 + 0 + 0 = 4$. Note that 16 instances are to be selected from particular hardness classes in total, sparing the four instances to be picked freely, and also that our balancing scheme takes care of exchanging an “easy” for a “medium” instance. ■

After determining the numbers of instances to pick per hardness class, we also aim to balance between satisfiable and unsatisfiable instances within the same class. In fact, the above balancing scheme is general enough to be reused for this purpose by letting $C = \{\text{satisfiable}(x), \text{unsatisfiable}(x)\}$ consist of the subclasses of satisfiable or unsatisfiable

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

instances, respectively, in a hardness class x that includes at least one instance known to be satisfiable or unsatisfiable.[8] Moreover, the parameters n and m used in (1) are fixed to $n = \mathit{select}(x)$ and $m = 0$, which reflect that the satisfiability status should be balanced among all instances to be picked from x without allocating an additional share of instances to pick freely. For the strict total order on the subclasses in C, we use satisfiable(x) $\prec$ unsatisfiable(x), let $<$ denote the transitive closure of $\prec$, and $>$ its inverse relation.

Example 4
Reconsidering the *Graceful Graphs* domain, we obtain the following number of instances to pick based on their satisfiability status: *select*(satisfiable("easy")) = 3, *select*(unsatisfiable("easy")) = 0, *select*(satisfiable("medium")) = 3, *select*(unsatisfiable ("medium")) = 1, *select*(satisfiable("hard")) = 2, and *select*(unsatisfiable("hard")) = 2. Note that *select*(satisfiable(x)) + *select*(unsatisfiable(x)) = *select*(x) for $x \in \{$"easy," "hard"$\}$, while *select*(satisfiable("medium"))+*select*(unsatisfiable("medium")) = 3+1 = $4 < 5 =$ *select*("medium"). The latter is due to rounding in *target* = $\lfloor 5/2 \rfloor = 2$ and then compensating for the underpopulated unsatisfiable instances by increasing the number of satisfiable "medium" instances to pick by one. ■

For instances of Decision problems or Query answering domains, we have that secondary balancing based on the satisfiability status is generally void for "too hard" instances, of which none are known to be satisfiable or unsatisfiable. In case of Optimization problems, where we discard instances known as unsatisfiable, *select*(satisfiable(x)) = *select*(x) holds for $x \in \{$"easy," "medium," "hard"$\}$, while our balancing scheme favors "too hard" instances known as satisfiable over those with an unknown satisfiability status. This approach makes sure that "too hard" instances to be picked possess solutions, yet confirming an optimum is hard, and instances with an unknown satisfiability status can still be contained among those that are picked freely.

Example 5
Regarding the Optimization problem in the *Valves Location* domain, we obtain *select* (satisfiable("too hard")) = *select*("too hard") = 4, given that the 23 instances whose satisfiability status is unknown are not considered for balancing. ■

The described twofold balancing scheme, first considering the hardness of instances and then the satisfiability status of instances of similar hardness, is implemented by an ASP-Core-2 encoding that consists of two parts: a deterministic program part (having a unique answer set) takes care of determining the numbers *select*(x) from the runtime results of reference systems and a non-deterministic part similar to the selection program used in the previous ASP Competition edition. Gebser *et al.* (2017b) encodes the choice of 20 instances per domain such that lower bounds given by the calculated numbers *select*(x) are met. In comparison to the previous competition edition, we updated the deterministic part of the benchmark selection encoding by implementing the balancing scheme described above, which is more general than before and not fixed to a particular number of classes (regarding hardness or satisfiability status) to balance. The instance selection was then performed by running the ASP solver CLASP with the options `--rand-freq`, `--sign-def`, and `--seed` for guaranteeing reproducible randomization,

[8] Otherwise, all subclasses to pick instances from are empty, which would lead to division by zero in (1).

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

using the concatenation of winning numbers in the EuroMillions lottery of May 2, 2017, as the random seed. This process led to the numbers of instances picked per domain, hardness class, and satisfiability status listed in Table 1.

As a final remark, we note that we had to exclude the *Resource Allocation* domain from the main competition in view of an insufficient number of instances belonging to the hardness classes under consideration. In fact, the majority of instances turned out to be "very easy" relative to an optimized encoding devised in the phase of checking/establishing the ASP-Core-2 compliance of initial submissions by benchmark authors. This does not mean that the problem of *Resource Allocation* as such would be trivial or uninteresting, but rather time constraints on running the main competition did unfortunately not permit to extend and then reassess the collection of instances.

4 Participant systems

Fourteen systems, registered by three teams, participate in the System track of the Seventh ASP Competition. The majority of systems runs in the *SP* category, while two (indicated by the suffix "-MT" below) exploit parallelism in the *MP* category. In the following, we survey the registered teams and systems.

Aalto. The team from Aalto University submitted nine systems that utilize normalization (Bomanson *et al.* 2014; 2016) and translation (Bogaerts *et al.* 2016; Bomanson *et al.* 2016; Gebser *et al.* 2014; Janhunen and Niemelä 2011; Liu *et al.* 2012) means. Two systems, LP2SAT+LINGELING and LP2SAT+PLINGELING-MT, perform translation to SAT and use LINGELING or PLINGELING, respectively, as back-end solver. Similarly, LP2MIP and LP2MIP-MT rely on translation to Mixed Integer Programming along with a single- or multi-threaded variant of CPLEX for solving. The LP2ACYCASP, LP2ACYCPB, and LP2ACYCSAT systems incorporate translations based on acyclicity checking, supported by CLASP run as ASP, Pseudo-Boolean, or SAT solver, as well as the GRAPHSAT solver in case of SAT with acyclicity checking. Moreover, LP2NORMAL+LP2STS takes advantage of the SAT-TO-SAT framework to decompose complex computations into several SAT solving tasks. Unlike that, LP2NORMAL+CLASP confines preprocessing to the (selective) normalization of aggregates and weak constraints before running CLASP as ASP solver. Beyond syntactic differences between target formalisms, the main particularities of the available translations concern space complexity and the supported language features. Regarding space, the translation to SAT utilized by LP2SAT+LINGELING and LP2SAT+PLINGELING-MT comes along with a logarithmic overhead in case of non-tight logic programs that involve positive recursion (Fages 1994), while the other translations are guaranteed to remain linear. Considering language features, the systems by the Aalto team do not support queries, and the back-end solver CLASP of LP2ACYCASP, LP2ACYCPB, and LP2NORMAL+CLASP provides a native implementation of aggregates, which the other systems treat by normalization within preprocessing. Optimization problems are supported by all systems but LP2SAT+LINGELING, LP2SAT+PLINGELING-MT, and LP2NORMAL+LP2STS, while only LP2NORMAL+LP2STS and LP2NORMAL+CLASP are capable of handling non-HCF disjunction.

ME-ASP. The ME-ASP team from the University of Genova, the University of Sassari, and the University of Calabria submitted the multi-engine ASP system ME-ASP2, which is an updated version of ME-ASP (Maratea *et al.* 2012; 2014; 2015), the winner system in

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

the Regular track of the Sixth ASP Competition. Like its predecessor version, ME-ASP2 investigates features of an input program to select its back end among a pool of ASP grounders and solvers. Basically, ME-ASP2 applies algorithm selection techniques before each stage of the answer set computation, with the goal of selecting the most promising computation strategy overall. As regards grounders, ME-ASP2 can pick either DLV or GRINGO, while the available solvers include a selection of those submitted to the Sixth ASP Competition as well as the latest version of CLASP. The first selection (basically corresponding to the selection of the grounder) is based on features of non-ground programs and was obtained by implementing the result of the application of the PART Decision list algorithm, whereas the choice of a solver is based on the multi-nomial classification algorithm k-Nearest Neighbors, used to train a model on features of ground programs extracted (whenever required) from the output generated by the grounder (for more details, see Maratea *et al.* 2015).

UNICAL. The team from the University of Calabria submitted four systems utilizing the recent IDLV grounder (Calimeri *et al.* 2017), developed as a redesign of (the grounder component of) DLV going along with the addition of new features. Moreover, back ends for solving are selected from a variety of existing ASP solvers. In particular, IDLV-CLASP-DLV makes use of DLV (Leone *et al.* 2006; Maratea *et al.* 2008) for instances featuring a ground query; otherwise, it consists of the combination of the grounder IDLV with CLASP executed with the option `--configuration=trendy`. The IDLV+-CLASP-DLV system is a variant of the previous system that uses a heuristic-guided rewriting technique (Calimeri *et al.* 2018) relying on hyper-tree decomposition, which aims to automatically replace long rules with sets of smaller ones that are possibly evaluated more efficiently. IDLV+-WASP-DLV is obtained by using WASP in place of CLASP. In more detail, WASP is executed with the options `--shrinking-strategy=progression --shrinking-budget=10 --trim-core --enable-disjcores`, which configure WASP to use two techniques tailored for Optimization problems. Inspired by ME-ASP, IDLV+S (Fuscà *et al.* 2017) integrates IDLV+ with an automatic selector to choose between WASP and CLASP on a per instance basis. To this end, IDLV+S implements classification by means of the well-known support vector machine technique. A more detailed description of the IDLV+S system is provided in Calimeri *et al.* (2019).

5 Results

This section presents the results of the Seventh ASP Competition. We first announce the winners in the *SP* category and analyze their performance, and then proceed overviewing results in the *MP* category. Finally, we analyze the results more in details outlining some of the outcomes.

5.1 Results in the SP category

Figures 2 and 3 summarize the results of the *SP* category, by showing the scores of the various systems, where Figure 2 utilizes function S_1 for computing the score of Optimization problems, while Figure 3 utilizes function S_2. To sum up, considering Figure 2, the first three places go to the systems:

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

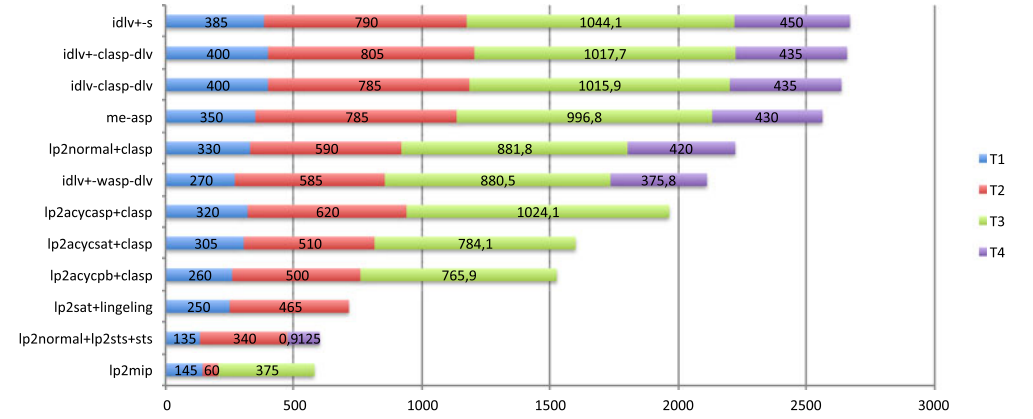


Fig. 2. Results of the *SP* category with computation S_1 for Optimization problems.

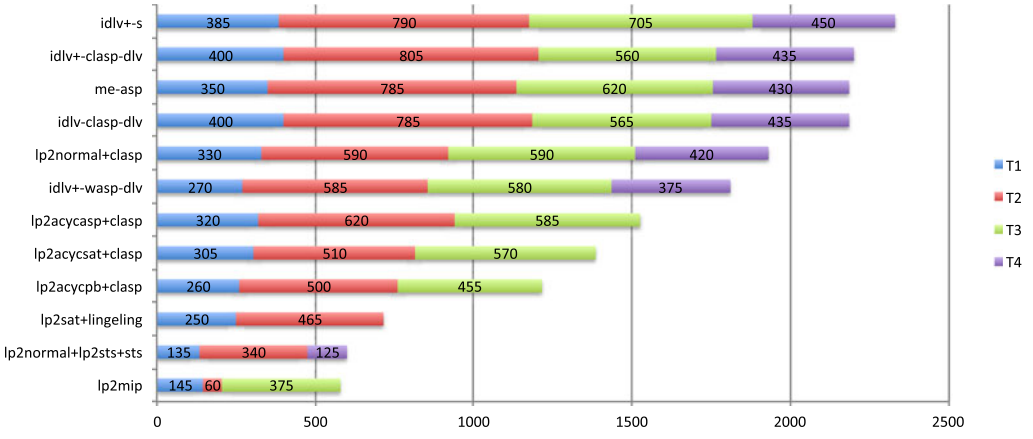


Fig. 3. Results of the *SP* category with computation S_2 for Optimization problems.

1. IDLV+S, by the UNICAL team, with 2665 points;
2. IDLV+-CLASP-DLV, by the UNICAL team, with 2655.9 points;
3. IDLV-CLASP-DLV, by the UNICAL team, with 2634 points.

Also, ME-ASP is quite close in performance, earning 2560 points in total.

Thus, the first three places are taken by versions of the IDLV system pursuing the approaches outlined in Section 4. The fourth place, with very positive results, is instead taken by ME-ASP which pursues a portfolio approach and was the winner of the last competition.

Going into the details of sub-tracks, the three top-performing systems overall take the first places as well:

- Sub-track #1 (Basic Decision): IDLV-CLASP-DLV and IDLV+-CLASP-DLV with 400 points;
- Sub-track #2 (Advanced Decision): IDLV+-CLASP-DLV with 805 points;
- Sub-track #3 (Optimization): IDLV+-CLASP-DLV with 1015.9 points;
- Sub-track #4 (Unrestricted): IDLV+S with 450 points.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

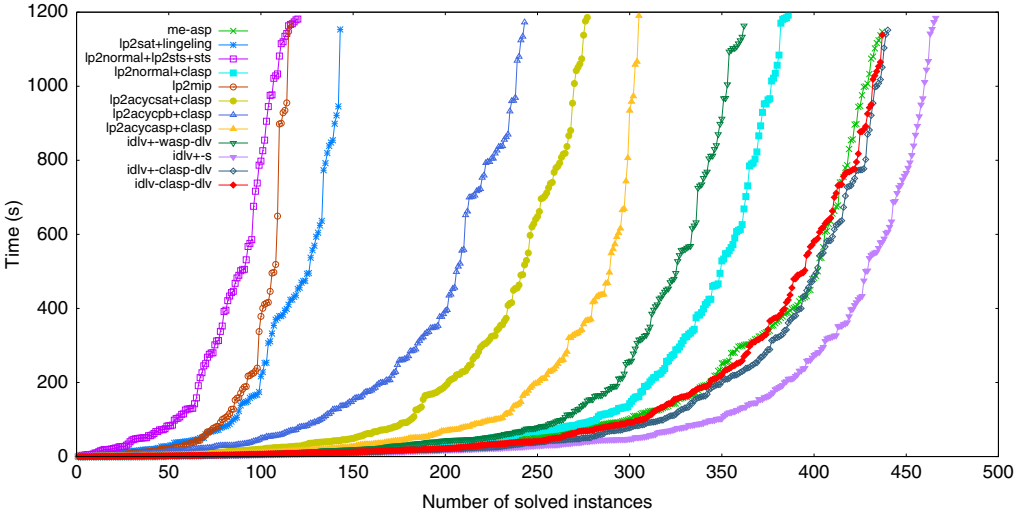


Fig. 4. Cactus plot of solver performances in the *SP* category.

Considering Figure 3, which employs function S_2 for computing scores of Optimization problems, the situation is slightly different, that is, ME-ASP now gets the third place. To sum up, the first three places go to the systems:

1. IDLV+S, by the UNICAL team, with 2330 points;
2. IDLV+-CLASP-DLV, by the UNICAL team, with 2200 points;
3. ME-ASP, by the ME-ASP team, with 2185 points.

IDLV+-CLASP-DLV is now fourth with the same score of 2185 points but higher cumulative CPU time: the difference is in Sub-track #3, where relative results are different with respect to using S_1, and with the new score computation ME-ASP earns 55 points more than IDLV+-CLASP-DLV. In general, employing S_2 function for computing scores of Optimization problems leads to lower scores: indeed, S_2 is more restrictive than S_1 given that only optimal results are considered.

An overall view of the performances of all participant systems on all benchmarks is shown in the cactus plot of Figure 4. Detailed results are reported in the Appendix.

Official results in Figures 2 and 3 are complemented by the data shown in Figures 5 and 6. Figure 5 contains, for each solver, the number of (optimally) solved instances in each reasoning problem of the competition, that is, Decision, Optimization, and Query (denoted Dec, Opt, and Query in the figure, respectively). From the figure, we can see that IDLV+-CLASP-DLV and IDLV-CLASP-DLV are the solvers that perform best on Decision problems, while IDLV+-S is the best on Optimization problems. For what concern Query answering, the first four solvers perform equally well on them. Figure 6, instead, reports, for each solver, the percentage of solved instances (resp. score) in the various sub-tracks out of the total number of (optimally) solved instances (resp. global score), that is, what is the "contribution" of tasks in each sub-track to the results of the respective system.

5.2 Results in the MP category

Figure 7 shows results about the *MP* category. The bottom part of the figure reports the scores acquired by the two participant systems, which cumulatively are: for

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

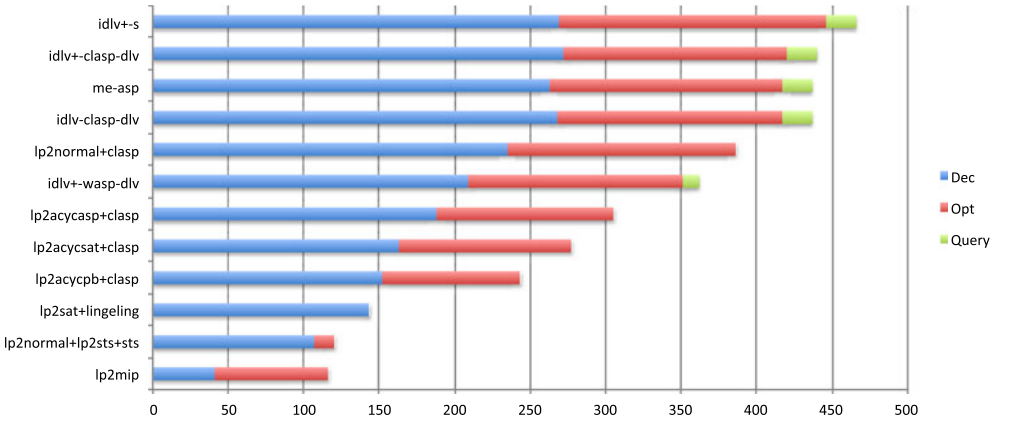


Fig. 5. Number of (optimally) solved instances in the *SP* category by task.

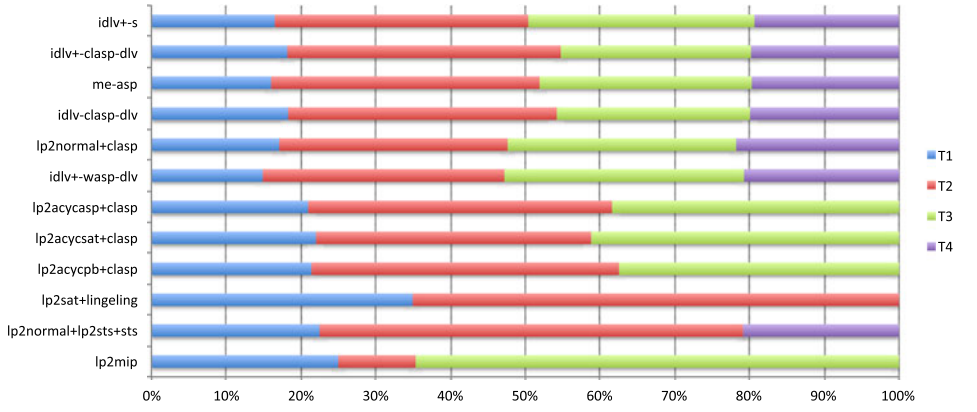


Fig. 6. Percentage of solved instances in the *SP* category.

1. LP2SAT+PLINGELING-MT, by the Aalto team, 715 points and
2. LP2MIP-MT, by the Aalto team, 635 points.

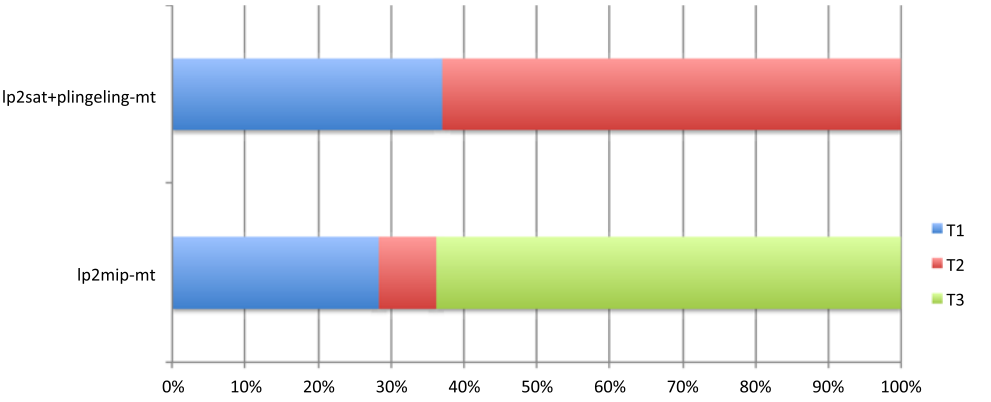
Looking into details of the sub-tracks, we can note that LP2SAT+PLINGELING-MT is better than LP2MIP-MT on Sub-track #1 and much better on Sub-track #2, while on Sub-track #3, where LP2SAT+PLINGELING-MT does not compete, LP2MIP-MT earns a consistent number of points, but not enough to globally reach the score of LP2SAT+PLINGELING-MT in the first two sub-tracks.

The top part of Figure 7, instead, complements the results by showing the "contribution" of solved instances in each sub-track out of the score of the respective system.

5.3 Analysis of the results

There are some general observations that can be drawn out of the results presented in this section. First, the best overall solver implements algorithm selection techniques and continues the "tradition" of the efficiency of portfolio-based solvers in ASP competitions, given that CLASPFOLIO (Gebser *et al.* 2011) and ME-ASP (Maratea *et al.* 2014) were the overall winners of the 2011 and 2015 competitions, respectively. At the same time, the result outlines the importance of the introduction of new evaluation techniques and

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press



	lp2mip-mt	lp2sat+plingeling-mt
T1	180	265
T2	50	450
T3	405	

Fig. 7. Results of the *MP* category.

implementations. Indeed, although IDLV+-S applies a strategy similar to the one of ME-ASP, IDLV+-S exploited a new component (i.e., the grounder Calimeri *et al.* 2016) that was not present in ME-ASP (which is based on solvers from the previous competition). Second, the approach implemented by LP2NORMAL using CLASP confirms its very good behavior in all sub-tracks, and thus overall. Third, specific istantiations of the translation-based approach perform particularly well in some sub-tracks: this is the case for the LP2ACYCASP solver using CLASP in Sub-track #3, especially when considering scoring scheme S_1, but also for LP2MIP, that compiles to a general purpose solver, in the same sub-track, especially when considering scoring scheme S_2 (even if to a less extent). As far as the comparison between solvers in the *MP* category and their counter-part in the *SP* category is concerned, we can see that globally the score of LP2SAT+PLINGELING-MT and LP2SAT+PLINGELING is the same, with the small advantage of LP2SAT+PLINGELING-MT in Sub-track #1 being compensated in Sub-track #2. Instead, LP2MIP+MT earns a consistent number of points more than LP2MIP, especially in Sub-tracks #1 and #3. In general, more specific research is probably needed on ASP solvers exploiting multi-threading to take real advantage from this setting.

6 Conclusion and final remarks

We have presented design and results of the Seventh ASP Competition, with particular focus on new problem domains, revised benchmark selection process, systems registered for the event, and results.

In the following, we draw some recommendations for future editions. These resemble the ones of the past event: for some of them some steps have been already made in this seventh's event, but they may be considered, with the aim of widening the number of participant systems and application domains that can be analyzed, starting from the next (Eighth) ASP competition that will take place in 2019 in affiliation with the

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

15th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR 2019), in Philadelphia, USA:

- We also tried to re-introduce a Model&Solve track at the competition. But, given the short call for contributions and the low number of expressions of interest received, we decided not to run the track. Despite this, we still think that a (restricted form of a) Model&Solve track should be re-introduced in the ASP competition series.
- Our aim with the re-introduction of a Model&Solve track was at solving domains involving, for example, discrete as well as continuous dynamics (Balduccini *et al.* 2017), so that extensions like Constraint ASP (Mellarkod *et al.* 2008) and incremental ASP solving (Gebser *et al.* 2008) may be exploited. The mentioned extensions could be added as tracks of the competition, but for CASP the first step that would be needed is a standardization of its language.
- Given that still basically all participant systems rely on grounding, the availability of more grounders is crucial. In this event the I-DLV grounder came into play, but there is also the need for more diverse techniques. This may also help improving portfolio solvers, by exploiting machine learning techniques at non-ground level (for a preliminary investigation, see Maratea *et al.* 2013, 2015).
- Portfolio solvers showed good performance in the editions where they participated. However, no such system in the various editions has exploited a parallel portfolio approach. Exploring such techniques in conjunction could be an interesting topic of future research for further improving the efficiency.
- Another option for attracting (young) researchers 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. This would lower the entrance barrier by keeping the effort of a participation affordable, even for small teams.

References

Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P. and Zangari, J. 2017. The ASP system DLV2. In *Proceedings of the Fourteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, M. Balduccini and T. Janhunen, Eds. Lecture Notes in AI (LNAI), vol. 10377. Springer-Verlag, 215–221.

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

Amendola, G., Dodaro, C., Faber, W., Leone, N. and Ricca, F. 2017. On the computation of paracoherent answer sets. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17)*, S. P. Singh and S. Markovitch, Eds. AAAI Press, 1034–1040.

Amendola, G., Dodaro, C., Faber, W. and Ricca, F. 2018. Externally supported models for efficient computation of paracoherent answer sets. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI'18)*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 1720–1727.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

Amendola, G., Eiter, T., Fink, M., Leone, N. and Moura, J. 2016. Semi-equilibrium models for paracoherent answer set programs. *Artificial Intelligence 234*, 219–271.

Amendola, G., Ricca, F. and Truszczyński, M. 2017. Generating hard random Boolean formulas and disjunctive logic programs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI'17)*, C. Sierra, Ed. ijcai.org, 532–538.

Applegate, D., Bixby, R., Chvátal, V. and Cook, W. 2007. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Balduccini, M., Magazzeni, D., Maratea, M. and Leblanc, E. 2017. CASP solutions for planning in hybrid domains. *Theory and Practice of Logic Programming 17,* 4, 591–633.

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

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

Bogaerts, B., Janhunen, T. and Tasharrofi, S. 2016. Stable-unstable semantics: Beyond NP with normal logic programs. *Theory and Practice of Logic Programming 16,* 5–6, 570–586.

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

Bomanson, J., Gebser, M. and Janhunen, T. 2016. Rewriting optimization statements in answer-set programs. In *Technical Communications of the Thirty-Second International Conference on Logic Programming (ICLP'16)*, M. Carro and A. King, Eds. Open Access Series in Informatics, vol. 52. Schloss Dagstuhl, 5:1–5:15.

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

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

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

Calimeri, F., Dodaro, C., Fuscà, D., Perri, S. and Zangari, J. 2019. Efficiently coupling the I-DLV grounder with ASP solvers. *Theory and Practice of Logic Programming. To appear.*

Calimeri, F., Fuscà, D., Perri, S. and Zangari, J. 2016. I-DLV: The new intelligent grounder of DLV. In *Proceedings of AI*IA 2016: Advances in Artificial Intelligence - Fifteenth International Conference of the Italian Association for Artificial Intelligence*, G. Adorni, S. Cagnoni, M. Gori and M. Maratea, Eds. Lecture Notes in Computer Science, vol. 10037. Springer, 192–207.

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

Calimeri, F., Fuscà, D., Perri, S. and Zangari, J. 2018. Optimizing answer set computation via heuristic-based decomposition. In *Proceedings of the Twentieth International Symposium on Practical Aspects of Declarative Languages (PADL'18)*, F. Calimeri, K. W. Hamlen and N. Leone, Eds. Lecture Notes in Computer Science, vol. 10702. Springer, 135–151.

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

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

Cussens, J. 2011. Bayesian network learning with cutting planes. In *Proceedings of the Twenty-Seventh International Conference on Uncertainty in Artificial Intelligence (UAI'11)*, F. Cozman and A. Pfeffer, Eds. AUAI Press, 153–160.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

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

Eiter, T., Ianni, G. and Krennwallner, T. 2009. Answer set programming: A primer. In *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School - Tutorial Lectures*. Brixen-Bressanone, Italy, 40–110.

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

Fuscà, D., Calimeri, F., Zangari, J. and Perri, S. 2017. I-DLV+MS: Preliminary report on an automatic ASP solver selector. In *Proceedings of the Twenty-Fourth RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA'17)*, M. Maratea and I. Serina, Eds. CEUR Workshop Proceedings, vol. 2011. CEUR-WS.org, 26–32.

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

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T. and Thiele, S. 2008. Engineering an incremental ASP solver. In *Proceedings of the Twenty-Fourth International Conference on Logic Programming (ICLP'08)*, M. Garcia de la Banda and E. Pontelli, Eds. Lecture Notes in Computer Science, vol. 5366. Springer-Verlag, 190–205.

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

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

Gebser, M., Leone, N., Maratea, M., Perri, S., Ricca, F. and Schaub, T. 2018. Evaluation techniques and systems for answer set programming: a survey. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI 2018)*, J. Lang, Ed. ijcai.org, 5450–5456.

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

Gebser, M., Maratea, M. and Ricca, F. 2017a. The design of the seventh answer set programming competition. In *Proceedings of the Fourteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, M. Balduccini and T. Janhunen, Eds. Lecture Notes in AI (LNAI), vol. 10377. Springer-Verlag, 3–9.

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

Gelfond, M. and Leone, N. 2002. Logic programming and knowledge representation – the A-prolog perspective. *Artificial Intelligence 138,* 1–2, 3–38.

Guerinik, N. and Caneghem, M. V. 1995. Solving crew scheduling problems by constraint programming. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP'95)*, U. Montanari and F. Rossi, Eds. Lecture Notes in Computer Science, vol. 976. Springer, 481–498.

Havur, G., Cabanillas, C., Mendling, J. and Polleres, A. 2016. Resource allocation with dependencies in business process management systems. In *Proceedings of the Business Process Management Forum (BPM'16)*, M. L. Rosa, P. Loos and O. Pastor, Eds. Lecture Notes in Business Information Processing, vol. 260. Springer, 3–19.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

Inoue, K. and Sakama, C. 1996. A fixpoint characterization of abductive logic programs. *Journal of Logic Programming 27,* 2, 107–136.

Janhunen, T., Gebser, M., Rintanen, J., Nyman, H., Pensar, J. and Corander, J. 2017. Learning discrete decomposable graphical models via constraint optimization. *Statistics and Computing 27,* 1, 115–130.

Janhunen, T. and Niemelä, I. 2011. Compact translations of non-disjunctive answer set programs to propositional clauses. In *Proceedings of the Symposium on Constructive Mathematics and Computer Science in Honour of Michael Gelfonds 65th Anniversary*. Lecture Notes in Computer Science, vol. 6565. Springer, 111–130.

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

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

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

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

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence 138*, 39–54.

Lifschitz, V. 2008. Twelve definitions of a stable model. In *Proceedings of the Twenty-Fourth International Conference on Logic Programming (ICLP'08)*, M. Garcia de la Banda and E. Pontelli, Eds. Lecture Notes in Computer Science, vol. 5366. Springer-Verlag, 37–51.

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

Maratea, M., Pulina, L. and Ricca, F. 2012. The multi-engine ASP solver me-asp. In *Proceedings of the 13th European Conference on Logics in Artificial Intelligence (JELIA 2012)*, L. F. del Cerro, A. Herzig and J. Mengin, Eds. Lecture Notes in Computer Science, vol. 7519. Springer, 484–487.

Maratea, M., Pulina, L. and Ricca, F. 2013. Automated selection of grounding algorithm in answer set programming. In *Advances in Artificial Intelligence - Proceedings of the 13th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2013)*, M. Baldoni, C. Baroglio, G. Boella and R. Micalizio, Eds. Lecture Notes in Computer Science, vol. 8249. Springer, 73–84.

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

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

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

Marek, V. W. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*, K. R. Apt, V. W. Marek, M. Truszczyński and D. S. Warren, Eds. Springer-Verlag, 375–398.

Marple, K. and Gupta, G. 2014. Dynamic consistency checking in goal-directed answer set programming. *Theory and Practice of Logic Programming 14,* 4–5, 415–427.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

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

Niemelä, I. 1999. Logic programming with stable model semantics as constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3–4, 241–273.

Appendix. Detailed results

We report in this appendix the detailed results aggregated by solver. In particular, Figures A1–A4 report for each solver and for each domain the score with computation S_1 for Optimization problems (ScoreASP2015), with computation score S_2 for Optimization problems (ScoreSolved), the sum of the execution times for all instances (Sum(Time)), the average memory usage on solved instances (Avg(Mem)), the number of solved instances (#Sol), the number of timed out executions (#TO), the number of execution terminated because the solver exceeded the memory limit (#MO) and the number of execution with abnormal execution (#OE), this last counting the instances that could not be solved by a solver, thus including output errors, abnormal terminations, give-ups as well as instances that cannot be solved by a solver when it did not participate with a domain. An "*" near to a score of 0 indicates that the solver was disqualified from a domain because it terminated normally but produced a wrong witness in some instance of the domain.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

	idlv-clasp-dlv								idlv+-clasp-dlv								idlv+-s							
Domain	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**
Abstract Dialectical Frameworks	100.00	100	150.64	18.68	20	0	0	0	100.00	100	146.68	19.10	20	0	0	0	100.00	100	173.58	67.75	20	0	0	0
Bayesian Network Learning	97.73	75	6523.23	15.31	15	5	0	0	97.73	75	6521.76	15.46	15	5	0	0	97.73	75	6538.84	66.76	15	5	0	0
Combined Configuration	75.00	75	6745.83	685.71	15	5	0	0	75.00	75	6749.90	682.86	15	5	0	0	75.00	75	6771.63	615.12	15	5	0	0
Complex Optimization	100.00	100	3448.35	103.65	20	0	0	0	100.00	100	3509.55	105.33	20	0	0	0	100.00	100	3420.13	102.04	20	0	0	0
Connected Still Life	78.18	35	17885.11	19.88	7	13	0	0	78.18	35	17854.87	19.88	7	13	0	0	77.73	50	12940.25	128.41	10	10	0	0
Consistent Query Answering	100.00	100	4616.18	8429.10	20	0	0	0	100.00	100	4649.93	8429.04	20	0	0	0	100.00	100	4524.89	8429.14	20	0	0	0
Crew Allocation	85.00	85	6549.39	12.90	17	3	0	0	85.00	85	6556.59	12.94	17	3	0	0	80.00	80	6628.81	65.90	16	4	0	0
Crossing Minimization	70.91	55	11789.21	18.85	11	9	0	0	69.55	55	11786.58	18.88	11	9	0	0	100.00	95	1254.71	83.92	19	1	0	0
Graceful Graphs	60.00	60	11474.31	69.18	12	8	0	0	60.00	60	11470.40	69.03	12	8	0	0	60.00	60	11473.47	91.06	12	8	0	0
Graph Colouring	85.00	85	5725.53	13.89	17	3	0	0	85.00	85	5732.59	13.86	17	3	0	0	85.00	85	5745.79	68.62	17	3	0	0
Incremental Scheduling	75.00	75	6828.66	1021.48	15	5	0	0	70.00	70	8071.44	1033.40	14	6	0	0	70.00	70	8106.03	1035.69	14	6	0	0
Knight Tour with Holes	75.00	75	6477.45	220.69	15	5	0	0	75.00	75	6478.07	221.49	15	5	0	0	75.00	75	6500.11	195.38	15	5	0	0
Labyrinth	55.00	55	12679.72	238.60	11	9	0	0	55.00	55	12654.07	244.27	11	9	0	0	55.00	55	12684.70	243.65	11	9	0	0
Markov Network Learning	98.64	75	8751.09	149.34	15	5	0	0	98.64	75	8744.99	150.71	15	5	0	0	98.64	75	8781.25	153.20	15	5	0	0
Maximal Clique	78.18	0	24010.14	540.94	0	20	0	0	78.18	0	24010.18	540.75	0	20	0	0	67.73	50	14919.49	874.94	10	10	0	0
MaxSAT	76.82	55	11971.99	158.42	11	9	0	0	76.82	55	11963.64	159.57	11	9	0	0	95.91	95	2372.10	392.93	19	1	0	0
Minimal Diagnosis	100.00	100	235.95	311.54	20	0	0	0	100.00	100	233.48	317.03	20	0	0	0	100.00	100	272.42	307.62	20	0	0	0
Nomystery	55.00	55	12146.17	218.60	11	9	0	0	55.00	55	12216.83	218.60	11	9	0	0	55.00	55	12164.57	226.05	11	9	0	0
Partner Units	50.00	50	12162.01	115.58	10	10	0	0	50.00	50	12158.53	115.69	10	10	0	0	50.00	50	12172.27	125.58	10	10	0	0
Permutation Pattern Matching	65.00	65	6170.24	7046.43	13	0	7	0	100.00	100	342.86	371.68	20	0	0	0	100.00	100	471.05	369.53	20	0	0	0
Qualitative Spatial Reasoning	100.00	100	1704.67	793.64	20	0	0	0	100.00	100	1719.76	796.18	20	0	0	0	100.00	100	1728.38	791.25	20	0	0	0
Random Disjunctive ASP	55.00	55	13166.08	17.77	11	9	0	0	55.00	55	13162.58	17.77	11	9	0	0	70.00	70	10736.35	21.88	14	6	0	0
Reachability	0.00	0	2786.01	4503.49	0	0	0	20	0.00	0	2865.54	4503.76	0	0	0	20	0.00	0	2791.61	4504.08	0	0	0	20
Ricochet Robots	60.00	60	12733.63	62.37	12	8	0	0	50.00	50	13672.02	62.13	10	10	0	0	50.00	50	13654.11	77.05	10	10	0	0
Paracoherent Answer Sets	80.00	80	7365.34	1178.61	16	4	0	0	80.00	80	7902.38	3535.91	16	4	0	0	80.00	80	7871.15	3537.62	16	4	0	0
Sokoban	60.00	60	11789.02	129.28	12	8	0	0	60.00	60	11696.23	129.28	12	8	0	0	50.00	50	12936.19	407.97	10	10	0	0
Stable Marriage	90.00	90	13372.56	6047.63	18	2	0	0	90.00	90	13591.68	6059.53	18	2	0	0	70.00	70	16217.84	6053.81	14	6	0	0
Steiner Tree	96.82	70	7475.35	34.81	14	6	0	0	96.82	70	7668.05	34.23	14	6	0	0	96.82	70	7687.10	79.86	14	6	0	0
StrategicCompanies	0.00	0	221.53	16.94	0	0	0	20	0.00	0	221.66	16.94	0	0	0	20	0.00	0	226.33	16.97	0	0	0	20
Supertree Contruction	79.09	40	16296.25	64.41	8	12	0	0	84.55	35	16727.80	71.15	7	13	0	0	82.73	35	16696.63	92.68	7	13	0	0
System Synthesis	71.36	0	24010.43	607.89	0	20	0	0	71.36	0	24010.32	606.45	0	20	0	0	61.36	0	24007.85	638.08	0	20	0	0
Traveling Salesperson	75.00	15	20446.00	145.89	3	17	0	0	75.91	15	20442.50	145.47	3	17	0	0	75.45	15	20447.47	161.00	3	17	0	0
Valves Location	97.27	80	5162.03	198.45	16	4	0	0	94.09	80	5542.93	199.03	16	4	0	0	94.09	80	5584.63	211.69	16	4	0	0
Video Streaming	95.91	65	8617.00	28.69	13	7	0	0	95.91	65	8621.77	28.80	13	7	0	0	95.91	65	8635.12	69.28	13	7	0	0
Visit-all	95.00	95	4954.64	55.61	19	1	0	0	95.00	95	5370.06	54.56	19	1	0	0	100.00	100	5313.43	83.98	20	0	0	0
Total	**2636**	**2185**	**326442.00**	**33294.00**	**437**	**216**	**7**	**40**	**2658**	**2200**	**325068.00**	**29021.00**	**440**	**220**	**0**	**40**	**2669**	**2330**	**292450.00**	**30390.00**	**466**	**194**	**0**	**40**

Fig. A1. Detailed results for IDLV-CLASP-DLV, IDLV+-CLASP-DLV, IDLV+-S.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

	idlv+-wasp-dlv								*lp2acycasp+clasp*								*lp2acycpb+clasp*							
Domain	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**
Abstract Dialectical Frameworks	75.83	75	6348.38	199.395	15	5	0	0	0.00	0	35.71	14.01	0	0	0	20	0.00	0	35.47	13.97	0	0	0	20
Bayesian Network Learning	87.27	80	6213.08	175.9	16	4	0	0	97.73	80	5482.38	20.86	16	4	0	0	94.09	75	6897.89	24.08	15	5	0	0
Combined Configuration	45.00	45	16055.62	2162.72	9	11	0	0	75.00	75	7211.33	794.82	15	5	0	0	35.00	35	14689.82	3321.40	7	9	4	0
Complex Optimization	100.00	100	2600.98	229.895	20	0	0	0	0.00	0	10.87	42.79	0	0	0	20	0.00	0	11.19	42.50	0	0	0	20
Connected Still Life	75.45	50	13355.24	113.715	10	10	0	0	68.64	35	17197.80	22.79	7	13	0	0	58.18	35	18612.10	33.95	7	13	0	0
Consistent Query Answering	55.00	55	3810.26	14506.875	11	0	9	0	0.00	0	261.46	218.06	0	0	0	20	0.00	0	253.84	216.82	0	0	0	20
Crew Allocation	25.00	25	19825.59	86.59	5	15	0	0	85.00	85	5654.98	10.98	17	3	0	0	80.00	80	8868.62	11.67	16	4	0	0
Crossing Minimization	100.00	95	1242.40	22.735	19	1	0	0	71.36	55	12061.69	17.46	11	9	0	0	76.36	60	10582.03	19.83	12	8	0	0
Graceful Graphs	35.00	35	17784.23	299.98	7	13	0	0	65.00	65	10865.79	60.07	13	7	0	0	65.00	65	12036.82	58.14	13	7	0	0
Graph Colouring	40.00	40	16245.06	69.235	8	12	0	0	80.00	80	6783.93	13.08	16	4	0	0	65.00	65	12119.32	22.26	13	7	0	0
Incremental Scheduling	45.00	45	15698.65	1335.05	9	11	0	0	65.00	65	9051.58	2672.03	13	7	0	0	50.00	50	13967.48	2672.00	10	10	0	0
Knight Tour with Holes	65.00	65	9391.45	388.02	13	7	0	0	75.00	75	6426.12	201.40	15	5	0	0	70.00	70	8626.20	579.97	14	6	0	0
Labyrinth	65.00	65	9624.98	448.495	13	7	0	0	60.00	60	11445.82	257.73	12	8	0	0	40.00	40	16435.96	974.71	8	12	0	0
Markov Network Learning	40.00	0	24008.76	1025.15	0	20	0	0	100.00	90	4760.32	577.96	18	2	0	0	87.73	60	12121.20	578.13	12	8	0	0
Maximal Clique	70.00	55	14620.09	1026.815	11	9	0	0	78.18	0	24010.21	599.19	0	20	0	0	70.91	0	24012.70	593.75	0	20	0	0
MaxSAT	96.36	95	2834.70	478.355	19	1	0	0	75.91	55	11419.98	278.95	11	9	0	0	75.00	55	11364.23	303.76	11	9	0	0
Minimal Diagnosis	100.00	100	486.00	1508.84	20	0	0	0	0.00	0	119.46	158.83	0	0	0	20	0.00	0	118.64	159.65	0	0	0	20
Nomystery	50.00	50	14385.78	546.23	10	10	0	0	60.00	60	11628.13	250.94	12	8	0	0	40.00	40	15405.62	2113.06	8	11	1	0
Partner Units	40.00	40	17701.85	506.44	8	12	0	0	55.00	55	11623.11	109.29	11	9	0	0	50.00	50	13314.73	377.57	10	10	0	0
Permutation Pattern Matching	100.00	100	317.65	244.01	20	0	0	0	40.00	40	3047.50	7794.15	8	0	12	0	40.00	40	2619.61	7789.82	8	0	12	0
Qualitative Spatial Reasoning	90.00	90	6069.47	2834.915	18	2	0	0	55.00	55	13002.68	1324.10	11	9	0	0	55.00	55	15177.96	2027.51	11	9	0	0
Random Disjunctive ASP	45.00	45	14581.17	75.805	9	11	0	0	0.00	0	0.00	0.00	0	0	0	20	0.00	0	0.00	0.29	0	0	0	20
Reachability	0.00	0	2848.63	4504.92	0	0	0	20	0.00	0	4942.75	1572.23	0	0	0	20	0.00	0	4655.60	1572.25	0	0	0	20
Ricochet Robots	50.00	50	15772.93	241.05	10	10	0	0	65.00	65	12109.32	58.65	13	7	0	0	40.00	40	18583.48	209.10	8	12	0	0
Paracoherent Answer Sets	55.00	55	11284.05	3534.235	11	9	0	0	0.00	0	2260.51	6297.22	0	0	7	13	0.00	0	2093.76	6301.63	0	0	7	13
Sokoban	50.00	50	12895.89	403.18	10	10	0	0	55.00	55	11756.85	128.03	11	9	0	0	45.00	45	15999.75	401.01	9	11	0	0
Stable Marriage	35.00	35	20502.95	1983.25	7	13	0	0	20.00	20	5457.74	12262.65	4	0	16	0	20.00	20	6239.59	12266.73	4	0	16	0
Steiner Tree	55.91	30	17629.91	415.785	6	14	0	0	95.91	70	7566.50	31.52	14	6	0	0	80.91	60	12085.43	74.79	12	8	0	0
StrategicCompanies	0.00	0	223.11	16.955	0	0	0	20	0.00	0	1.86	6.22	0	0	0	20	0.00	0	1.84	6.82	0	0	0	20
Supertree Contruction	55.45	25	18357.56	236.525	5	15	0	0	87.73	30	16918.72	70.16	6	14	0	0	47.73	25	18163.28	393.65	5	15	0	0
System Synthesis	61.36	0	24010.74	635.615	0	20	0	0	77.27	0	24011.35	622.60	0	20	0	0	5.00	5	6497.15	11824.37	1	5	14	0
Traveling Salesperson	88.18	20	19475.73	1821.365	4	16	0	0	80.00	20	19919.86	559.66	4	16	0	0	82.73	20	19919.23	694.62	4	16	0	0
Valves Location	85.00	75	7499.86	383.66	15	5	0	0	94.55	80	5522.40	557.45	16	4	0	0	0.00	0	3613.13	1677.67	0	2	0	18
Video Streaming	65.45	55	10804.88	247.4	11	9	0	0	96.82	70	7717.60	24.30	14	6	0	0	87.27	60	10188.36	45.12	12	8	0	0
Visit-all	65.00	65	9312.22	481.59	13	7	0	0	85.00	85	5683.92	105.49	17	3	0	0	65.00	65	10528.72	350.76	13	7	0	0
Total	**2111**	**1810**	**403820.00**	**43191.00**	**362**	**289**	**9**	**40**	**1964**	**1525**	**295970.00**	**37736.00**	**305**	**207**	**35**	**153**	**1526**	**1215**	**345841.00**	**57753.00**	**243**	**232**	**54**	**171**

Fig. A2. Detailed results for IDLV-WASP-DLV, LP2ACYCASP+CLASP, LP2ACYCPB+CLASP.

	lp2acycsat+clasp								*lp2mip*								*lp2normal+clasp*							
Domain	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**
Abstract Dialectical Frameworks	0.00	0	35.25	14.16	0	0	0	20	0.00	0	34.82	13.98	0	0	0	20	100.00	100	170.35	18.57	20	0	0	0
Bayesian Network Learning	89.55	80	5876.95	44.85	16	4	0	0	40.00	40	10789.49	537.61	8	8	0	4	90.45	75	6469.01	35.28	15	5	0	0
Combined Configuration	20.00	20	19733.55	1639.00	4	16	0	0	0.00	0	22239.72	4944.04	0	16	4	0	75.00	75	6787.78	614.06	15	5	0	0
Complex Optimization	0.00	0	11.51	42.65	0	0	0	20	0.00	0	11.39	42.77	0	0	0	20	95.00	95	3953.30	71.82	19	1	0	0
Connected Still Life	100.00	90	6173.04	28.67	18	2	0	0	0.00	0	24009.55	131.89	0	20	0	0	99.09	80	8328.41	21.91	16	4	0	0
Consistent Query Answering	0.00	0	253.59	217.54	0	0	0	20	0.00	0	252.31	218.31	0	0	0	20	0.00	0	246.00	217.00	0	0	0	20
Crew Allocation	80.00	80	6809.60	14.55	16	4	0	0	0.00	0	24009.36	90.49	0	20	0	0	75.00	75	6922.37	13.76	15	5	0	0
Crossing Minimization	98.64	95	2083.64	20.92	19	1	0	0	95.00	95	2246.13	47.90	19	1	0	0	100.00	95	2235.66	18.03	19	1	0	0
Graceful Graphs	70.00	70	10180.97	66.28	14	6	0	0	15.00	15	23122.63	895.27	3	17	0	0	65.00	65	9933.41	63.39	13	7	0	0
Graph Colouring	85.00	85	5820.94	16.16	17	3	0	0	15.00	15	23390.93	54.23	3	17	0	0	80.00	80	6793.76	13.08	16	4	0	0
Incremental Scheduling	5.00	5	5496.40	12050.23	1	0	19	0	0.00	0	5828.52	12334.36	0	0	20	0	60.00	60	12553.54	2775.11	12	8	0	0
Knight Tour with Holes	75.00	75	6805.67	434.14	15	5	0	0	40.00	40	14641.07	705.35	8	12	0	0	75.00	75	7605.89	192.10	15	5	0	0
Labyrinth	40.00	40	14831.13	221.59	8	12	0	0	0.00	0	24011.34	1934.57	0	20	0	0	60.00	60	12843.29	230.53	12	8	0	0
Markov Network Learning	73.64	45	15269.70	599.30	9	11	0	0	0.00	0	23142.17	888.66	0	19	0	1	50.91	25	19144.11	580.14	5	15	0	0
Maximal Clique	98.64	45	17666.56	455.65	9	11	0	0	40.00	40	17379.07	766.19	8	12	0	0	95.45	35	20155.70	454.26	7	13	0	0
MaxSAT	85.45	75	7377.19	492.92	15	5	0	0	55.00	55	10281.92	289.57	11	8	0	1	85.91	75	6911.29	367.38	15	5	0	0
Minimal Diagnosis	0.00	0	142.33	159.77	0	0	0	20	0.00	0	141.37	159.29	0	0	0	20	100.00	100	313.67	302.97	20	0	0	0
Nomystery	55.00	55	13341.41	394.13	11	9	0	0	0.00	0	23677.09	2554.49	0	19	1	0	55.00	55	11652.18	258.84	11	9	0	0
Partner Units	55.00	55	13636.90	140.13	11	9	0	0	0.00	0	24010.48	1188.93	0	20	0	0	50.00	50	14883.64	124.16	10	10	0	0
Permutation Pattern Matching	40.00	40	3013.84	7787.12	8	0	12	0	35.00	35	4845.98	7943.82	7	1	12	0	40.00	40	3120.69	7787.70	8	0	12	0
Qualitative Spatial Reasoning	55.00	55	13729.32	1518.15	11	9	0	0	10.00	10	22520.61	4769.73	2	18	0	0	55.00	55	13053.97	1315.66	11	9	0	0
Random Disjunctive ASP	0.00	0	0.00	0.00	0	0	0	20	0.00	0	0.00	0.00	0	0	0	20	60.00	60	12882.62	16.58	12	8	0	0
Reachability	0.00	0	4808.19	1572.25	0	0	0	20	0.00	0	4937.00	1572.27	0	0	0	20	0.00	0	4628.06	1572.21	0	0	0	20
Ricochet Robots	75.00	75	9108.10	57.39	15	5	0	0	0.00	0	24012.79	685.72	0	20	0	0	60.00	60	10444.33	56.79	12	8	0	0
Paracoherent Answer Sets	0.00	0	2237.25	6276.27	0	0	7	13	0.00	0	2976.00	6296.55	0	0	7	13	65.00	65	2234.84	6278.61	13	0	7	0
Sokoban	55.00	55	12520.08	289.52	11	9	0	0	0.00	0	24010.19	806.66	0	20	0	0	55.00	55	12295.68	144.74	11	9	0	0
Stable Marriage	20.00	20	6272.04	12283.16	4	0	16	0	0.00	0	6655.44	12348.16	0	0	20	0	20.00	20	5432.49	12270.81	4	0	16	0
Steiner Tree	70.91	55	11675.02	1005.26	11	8	1	0	65.00	65	8678.71	780.92	13	7	0	0	75.00	60	11115.01	744.33	12	8	0	0
StrategicCompanies	0.00	0	1.84	6.38	0	0	0	20	0.00	0	1.81	6.35	0	0	0	20	0.00	0	1.73	6.02	0	0	0	20
Supertree Contruction	71.36	30	17113.85	111.90	6	14	0	0	0.00	0	24011.07	465.59	0	20	0	0	75.45	30	16944.67	79.73	6	14	0	0
System Synthesis	0.00	0	24013.45	5612.74	0	20	0	0	0.00	0	15337.30	10630.18	0	9	11	0	30.00	0	24011.16	1688.08	0	20	0	0
Traveling Salesperson	37.27	15	19474.71	2607.41	3	16	1	0	65.00	65	7833.69	1373.33	13	6	1	0	45.45	15	20565.15	1437.00	3	17	0	0
Valves Location	15.00	15	11306.25	8482.65	3	7	10	0	0.00	0	6456.81	12365.68	0	0	20	0	97.73	80	5687.56	672.07	16	4	0	0
Video Streaming	43.64	25	21844.18	1346.77	5	15	0	0	15.00	15	20832.34	2290.84	3	17	0	0	36.36	20	23242.67	767.68	4	16	0	0
Visit-all	85.00	85	6397.74	275.88	17	3	0	0	90.00	90	4233.48	424.80	18	2	0	0	95.00	95	5641.21	105.95	19	1	0	0
Total	**1599**	**1385**	**315062.00**	**66285.00**	**277**	**204**	**66**	**153**	**580**	**580**	**450563.00**	**90558.00**	**116**	**329**	**96**	**159**	**2222**	**1930**	**329205.00**	**41316.00**	**386**	**219**	**35**	**60**

Fig. A3. Detailed results for LP2ACYCSAT+CLASP, LP2MIP, LP2NORMAL+CLASP.

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

https://doi.org/10.1017/S1471068419000061 Published online by Cambridge University Press

	lp2normal+lp2sts+sts								*lp2sat+lingeling*								*me-asp*							
Domain	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**	**ScoreASP2015**	**ScoreSolved**	**Sum(Time)**	**Avg(Mem)**	**#Sol**	**#TO**	**#MO**	**#OE**
Abstract Dialectical Frameworks	0.00	0	17072.20	134.64	0	11	0	9	0.00	0	35.14	14.09	0	0	0	20	75.00	75	72.54	22.11	15	0	0	5
Bayesian Network Learning	0.00	0	21187.79	107.57	0	14	0	6	0.00	0	5.28	18.07	0	0	0	20	82.73	75	6826.59	37.06	15	5	0	0
Combined Configuration	0.00	0	24010.70	2110.79	0	20	0	0	10.00	10	21761.86	1815.38	2	18	0	0	75.00	75	7450.97	930.08	15	5	0	0
Complex Optimization	5.00	5	23277.47	129.79	1	19	0	0	0.00	0	11.58	43.18	0	0	0	20	100.00	100	1217.02	59.87	20	0	0	0
Connected Still Life	0.00	0	17172.70	57.87	0	12	0	8	0.00	0	0.27	3.83	0	0	0	20	97.27	75	8290.42	33.17	15	5	0	0
Consistent Query Answering	0.00	0	251.09	216.99	0	0	0	20	0.00	0	249.97	219.27	0	0	0	20	100.00	100	4650.91	9819.33	20	0	0	0
Crew Allocation	80.00	80	6862.65	28.88	16	4	0	0	75.00	75	7334.19	32.64	15	5	0	0	75.00	75	9084.43	34.03	15	5	0	0
Crossing Minimization	0.00	0	12925.64	51.40	0	8	0	12	0.00	0	0.04	0.55	0	0	0	20	95.91	90	3494.66	26.16	18	2	0	0
Graceful Graphs	30.00	30	18836.60	82.98	6	14	0	0	55.00	55	11988.28	89.47	11	9	0	0	65.00	65	10880.61	61.75	13	7	0	0
Graph Colouring	65.00	65	12822.94	37.25	13	7	0	0	80.00	80	8724.25	27.19	16	4	0	0	80.00	80	9030.66	25.41	16	4	0	0
Incremental Scheduling	5.00	5	6132.55	12111.19	1	0	19	0	5.00	5	5546.00	11956.07	1	0	19	0	65.00	65	9745.46	2519.89	13	7	0	0
Knight Tour with Holes	15.00	15	20462.64	371.45	3	17	0	0	15.00	15	20586.42	628.64	3	17	0	0	65.00	65	8652.74	126.96	13	7	0	0
Labyrinth	0.00	0	24009.80	316.19	0	20	0	0	35.00	35	18449.95	405.91	7	13	0	0	80.00	80	8252.64	213.35	16	4	0	0
Markov Network Learning	0,00	0	24009.61	705.73	0	20	0	0	0.00	0	399.47	577.98	0	0	0	20	89.55	65	11189.19	485.07	13	7	0	0
Maximal Clique	0.00	0	24011.96	456.43	0	20	0	0	0.00	0	415.01	456.58	0	0	0	20	86.36	15	22918.98	313.43	3	17	0	0
MaxSAT	0.91	0	2401.27	580.67	0	17	0	3	0.00	0	164.53	346.24	0	0	0	20	75.45	60	9956.60	211.74	12	8	0	0
Minimal Diagnosis	0*	0*	482.40	694.38	0	0	0	20	0.00	0	139.49	160.07	0	0	0	20	100.00	100	142.18	252.43	20	0	0	0
Nomystery	15.00	15	22153.61	456.65	3	17	0	0	55.00	55	12045.16	337.53	11	9	0	0	45.00	45	14273.38	562.83	9	11	0	0
Partner Units	40.00	40	19563.03	183.38	8	12	0	0	35.00	35	16956.14	125.04	7	13	0	0	55.00	55	12076.11	108.23	11	9	0	0
Permutation Pattern Matching	40.00	40	2462.25	7789.92	8	0	12	0	40.00	40	2594.82	7796.08	8	0	12	0	95.00	95	3654.45	1528.07	19	1	0	0
Qualitative Spatial Reasoning	60.00	60	10714.84	5476.65	12	8	0	0	55.00	55	13384.44	1244.11	11	9	0	0	100.00	100	3605.43	999.15	20	0	0	0
Random Disjunctive ASP	55.00	55	13491.48	50.30	11	9	0	0	0.00	0	0.00	0.00	0	0	0	20	80.00	80	10684.65	24.99	16	4	0	0
Reachability	0.00	0	4788.74	1572.26	0	0	0	20	0.00	0	4687.58	1572.24	0	0	0	20	0.00	0	3601.71	5710.41	0	0	0	20
Ricochet Robots	35.00	35	18821.52	179.66	7	13	0	0	70.00	70	10660.58	69.61	14	6	0	0	55.00	55	14190.17	103.87	11	9	0	0
Paracoherent Answer Sets	65.00	65	2099.16	6284.15	13	0	7	0	0.00	0	2031.75	6294.97	0	0	7	13	75.00	75	2883.22	5928.96	15	0	5	0
Sokoban	35.00	35	19647.30	351.71	7	13	0	0	65.00	65	10729.55	203.77	13	7	0	0	55.00	55	11384.26	844.51	11	8	1	0
Stable Marriage	0.00	0	7935.16	12211.35	0	4	16	0	20.00	20	6654.04	12269.54	4	0	16	0	40.00	40	19502.20	1692.68	8	12	0	0
Steiner Tree	0.00	0	19665.24	1315.33	0	16	1	3	0.00	0	514.61	803.12	0	0	1	19	62.27	45	13537.63	37.70	9	11	0	0
StrategicCompanies	0.00	0	1.73	5.78	0	0	0	20	0.00	0	1.74	5.95	0	0	0	20	0.00	0	224.14	24.43	0	0	0	20
Supertree Contruction	0.00	0	23875.08	167.51	0	19	0	1	0.00	0	21.16	40.47	0	0	0	20	92.73	35	16649.44	75.40	7	13	0	0
System Synthesis	0.00	0	8510.10	10507.49	0	6	14	0	0.00	0	1055.48	2876.04	0	0	0	20	90.91	0	24011.57	212.43	0	20	0	0
Traveling Salesperson	0.00	0	20380.14	2795.55	0	16	2	2	0.00	0	604.33	1502.81	0	0	0	20	40.45	15	20457.25	78.28	3	17	0	0
Valves Location	0.00	0	11199.92	10720.81	0	7	12	1	0.00	0	2487.44	7290.58	0	0	10	10	93.64	75	6541.57	524.47	15	5	0	0
Video Streaming	0.00	0	11428.65	2390.19	0	9	0	11	0.00	0	420.06	681.27	0	0	0	20	89.55	70	8371.66	47.60	14	6	0	0
Visit-all	55.00	55	21745.58	306.63	11	9	0	0	100.00	100	3317.46	148.12	20	0	0	0	85.00	85	5914.80	83.92	17	3	0	0
Total	**601**	**600**	**494414.00**	**80959.00**	**120**	**361**	**83**	**136**	**715**	**715**	**183978.00**	**60056.00**	**143**	**110**	**65**	**382**	**2562**	**2185**	**323420.00**	**33760.00**	**437**	**212**	**6**	**45**

Fig. A4. Detailed results for LP2NORMAL+LP2STS+STS, LP2SAT+LINGELING, ME-ASP.