The First Answer Set Programming System Competition

Martin Gebser[1], Lengning Liu[2], Gayathri Namasivayam[2], André Neumann[1], Torsten Schaub[1,⋆], and Mirosław Truszczyński[2]

[1] Institut für Informatik, Universität Potsdam,
August-Bebel-Str. 89, D-14482 Potsdam, Germany
{gebser,aneumann,torsten}@cs.uni-potsdam.de
[2] Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046, USA
{gayathri,lliu1,mirek}@cs.uky.edu

Abstract. This paper gives a summary of the *First Answer Set Programming System Competition* that was held in conjunction with the *Ninth International Conference on Logic Programming and Nonmonotonic Reasoning*. The aims of the competition were twofold: first, to collect challenging benchmark problems, and second, to provide a platform to assess a broad variety of Answer Set Programming systems. The competition was inspired by similar events in neighboring fields, where regular benchmarking has been a major factor behind improvements in the developed systems and their ability to address practical applications.

1 Introduction

Answer Set Programming (ASP) is an area of knowledge representation concerned with logic-based languages for modeling computational problems in terms of constraints [1,2,3,4]. Its origins are in logic programming [5,6] and nonmonotonic reasoning [7,8]. The two areas merged when Gelfond and Lifschitz proposed the *answer set semantics* for logic programs (also known as the *stable model semantics*) [9,10]. On the one hand, the answer set semantics provided what is now commonly viewed to be the correct treatment of the negation connective in logic programs. On the other hand, with the answer set semantics, logic programming turned out to be a special case of Reiter's default logic [8], with answer sets corresponding to default extensions [11,12].

Answer Set Programming was born when researchers proposed a new paradigm for modeling application domains and problems with logic programs under the answer set semantics: a problem is modeled by a program so that answer sets of the program directly correspond to solutions of the problem [1,2]. At about the same time, first software systems to compute answer sets of logic programs were developed: *dlv* [13] and *lparse/smodels* [14]. They demonstrated that the answer set programming paradigm has a potential to be the basis for practical declarative computing.

These two software systems, their descendants, and essentially all other ASP systems that have been developed and implemented so far contain two major components. The first of them, a *grounder*, grounds an input program, that is, produces its propositional

⋆ Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada, and IIIS at Griffith University, Brisbane, Australia.

C. Baral, G. Brewka, and J. Schlipf (Eds.): LPNMR 2007, LNAI 4483, pp. 3–17, 2007.
© Springer-Verlag Berlin Heidelberg 2007

equivalent. The second one, a *solver*, accepts the ground program and actually computes its answer sets (which happen to be the answer sets of the original program).

The emergence of practical software for computing answer sets has been a major impetus behind the rapid growth of ASP in the past decade. Believing that the ultimate success of ASP depends on the continued advances in the performance of ASP software, the organizers of the *Ninth International Conference on Logic Programming and Nonmonotonic Reasoning* (LPNMR'07) asked us to design and run a contest for ASP software systems. It was more than fitting, given that the first two ASP systems, *dlv* and *lparse/smodels*, were introduced exactly a decade ago at the *Fourth International Conference on Logic Programming and Nonmonotonic Reasoning* (LPNMR'97). We agreed, of course, convinced that as in the case of propositional SATisfiability, where solver competitions have been run for years in conjunction with SAT conferences, this initiative will stimulate research on and development of ASP software, and will bring about dramatic improvements in its capabilities.

In this paper, we report on the project — the *First Answer Set Programming System Competition* — conducted as part of LPNMR'07. When designing it, we built on our experiences from running preliminary versions of this competition at two Dagstuhl meetings on ASP in 2002 and 2005 [15]. We were also inspired by the approach of and the framework developed for SAT competitions [16], along with the related competitions in solving Quantified Boolean Formulas and Pseudo-Boolean constraints.

The First Answer Set Programming System Competition was run *prior* to the LPNMR'07 conference. The results are summarized in Section 6 and can be found in full detail at [17]. The competition was run on the *Asparagus* platform [18], relying on benchmarks stored there before the competition as well as on many new ones submitted by the members of the ASP community (cf. Section 4).

The paper is organized as follows. In the next section, we explain the format of the competition. Section 3 provides a brief overview of the Asparagus platform. In Section 4 and 5, we survey the benchmark problems and the competitors that took part in the competition. The competition results are announced in Section 6. Finally, we discuss our experiences and outline potential future improvements.

2 Format

The competition was run in three different categories:

MGS (Modeling, Grounding, Solving) In this category, benchmarks consist of a problem statement, a set of instances (specified in terms of ground facts), and the names of the predicates and their arguments to be used by programmers to encode solutions. The overall performance of software (including both the grounding of input programs and the solving of their ground instantiations) is measured. Success in this category depends on the quality of the input program modeling a problem (the problem encoding), the efficiency of a grounder, and the speed of a solver.

SCore (Solver, Core language) Benchmarks in this category are ground programs in the format common to *dlv* [19] and *lparse* [20]. In particular, aggregates are not

allowed. Instances are classified further into two subgroups: *normal* (SCore) and *disjunctive* (SCore$^\vee$). The time needed by *solvers* to compute answer sets of ground programs is measured. Thus, this category is concerned *only* with the performance of solvers on ground programs in the basic logic programming syntax.

SLparse (Solver, Lparse language) Benchmarks in this category are ground programs in the lparse format *with* aggregates allowed. The performance of solvers on ground programs in lparse format is measured. This category is similar to SCore in that it focuses entirely on solvers. Unlike SCore, though, it assesses the ability of solvers to take advantage of and efficiently process aggregates.

Only decision problems were considered in this first edition of the competition. Thus, every solver had to indicate whether a benchmark instance has an answer set (SAT) or not (UNSAT). Moreover, for instances that are SAT, solvers were required to output a certificate of the existence of an answer set (that is, an answer set itself). This certificate was used to check the correctness of a solution.

The output of solvers had to conform to the following formats (in `typewriter` font):

SAT: `Answer Set: atom1 atom2 ... atomN`
The output is one line containing the keywords '`Answer Set:`' and the names of the atoms in the answer set. Each atom's name is preceded by a single space. Spaces must not occur within atom names.

UNSAT: `No Answer Set`
The output is one line containing the keywords '`No Answer Set`'.

The competition imposed on each software system (grounder plus solver) the same time and space limits. The number of instances solved within the allocated time and space was used as the **primary** measure of the performance of a software system. Average running time is only used as a tie breaker.

3 Platform

The competition was run on the *Asparagus* platform [18], providing a web-based benchmarking environment for ASP. The principal goals of Asparagus are (1) to provide an infrastructure for accumulating challenging benchmarks and (2) to facilitate executing ASP software systems under comparable conditions, guaranteeing reproducible and reliable performance results. Asparagus is a continuously running benchmarking environment that combines several internal machines running the benchmarks with an external server for the remaining functionalities including interaction and storage. The internal machines are clones having a modified Linux kernel to guarantee a strict limitation of time and memory resources. This is important in view of controlling heterogeneous ASP solvers that run in multiple processes (e.g., by invoking a stand-alone SAT solver). A more detailed description of the Asparagus platform can be found in [15].

4 Benchmarks

The benchmarks for the competition were collected on the Asparagus platform. For all competition categories (MGS, SCore, and SLparse), we asked for submissions of non-ground problem encodings and ground problem instances in separate files.

To add a problem to the MGS category, the author of the problem had to provide

- a textual problem description that also specified the names and arguments of input and output predicates and
- a set of problem instances in terms of ground facts (using only input predicates).

The submission of a problem encoding was optional for benchmark problems submitted to the MGS category. In most cases, however, the authors provided it, too. In all remaining cases, the competition team provided an encoding. From all benchmark classes already stored on Asparagus or submitted for the competition, we selected several

Table 1. Benchmarks submitted by the ASP community

Benchmark Class	#Instances	Contributors	M	C	L
15-Puzzle	15	Lengning Liu and Mirosław Truszczyński	×	–	×
15-Puzzle	11	Asparagus	–	×	–
Blocked N-Queens	40	Gayathri Namasivayam and Mirosław Truszczyński	×	–	×
Blocked N-Queens	400	Asparagus	–	×	–
Bounded Spanning Tree	30	Gayathri Namasivayam and Mirosław Truszczyński	×	–	×
Car Sequencing	54	Marco Cadoli	×	–	×
Disjunctive Loops	9	Marco Maratea	–	×	–
EqTest	10	Asparagus	–	×	–
Factoring	10	Asparagus	–	×	×
Fast Food	221	Wolfgang Faber	×	–	×
Gebser Suite	202	Martin Gebser	–	×	×
Grammar-Based Information Extraction	102	Marco Manna	–	×	–
Hamiltonian Cycle	30	Lengning Liu and Mirosław Truszczyński	×	–	×
Hamiltonian Path	58	Asparagus	–	×	×
Hashiwokakero	211	Martin Brain	–	–	×
Hitori	211	Martin Brain	–	–	×
Knight's Tour	165	Martin Brain	–	–	×
Mutex	7	Marco Maratea	–	×	–
Random Non-Tight	120	Enno Schultz, Martin Gebser	–	×	×
Random Quantified Boolean Formulas	40	Marco Maratea	–	×	–
Reachability	53	Giorgio Terracina	×	×	×
RLP	573	Yuting Zhao and Fangzhen Lin	–	×	×
Schur Numbers	33	Lengning Liu and Mirosław Truszczyński	×	–	×
Schur Numbers	5	Asparagus	–	×	–
Social Golfer	175	Marco Cadoli	×	–	×
Sokoban	131	Wolfgang Faber	×	–	–
Solitaire Backward	36	Martin Brain	–	–	×
Solitaire Backward (2)	10	Lengning Liu and Mirosław Truszczyński	–	–	×
Solitaire Forward	41	Martin Brain	–	–	×
Strategic Companies	35	Nicola Leone	–	×	–
Su-Doku	8	Martin Brain	–	–	×
TOAST	54	Martin Brain	–	–	×
Towers of Hanoi	29	Gayathri Namasivayam and Mirosław Truszczyński	×	–	×
Traveling Salesperson	30	Lengning Liu and Mirosław Truszczyński	×	–	×
Weight-Bounded Dominating Set	30	Lengning Liu and Mirosław Truszczyński	×	–	×
Weighted Latin Square	35	Gayathri Namasivayam and Mirosław Truszczyński	×	–	×
Weighted Spanning Tree	30	Gayathri Namasivayam and Mirosław Truszczyński	×	–	×
Word Design DNA	5	Marco Cadoli	×	–	×

Algorithm 1. Semiautomatic Benchmarking Procedure

Input : $Classes$ — set of benchmark classes
$Used$ — set of benchmark instances run already
$Fresh$ — set of benchmark instances not run so far
max — maximum number of suitable benchmark instances per benchmark class

```
1 repeat
2     ToRun ← ∅                                          /* no runs scheduled yet */
3     foreach C in Classes do
4         done ← |SUITABLE(Used[C])|
5         if done < max then ToRun ← ToRun ∪ SELECT(max−done, Fresh[C])
6     RUN(ToRun)                                         /* execute the scheduled runs */
7     Used ← Used ∪ ToRun
8     Fresh ← Fresh \ ToRun
9 until ToRun = ∅
```

instances for use in the competition (we describe our selection criteria below). For SCore and SLparse, we relied on the availability of encodings to produce ground instances according to the input format of the respective category.

It is important to note that competitors in the MGS category did not have to use the default Asparagus encodings. Instead, they had the option to provide their own problem encodings, presumably better than the default ones, as the MGS category was also about assessing the modeling capabilities of grounders and solvers.

The collected benchmarks constitute the result of efforts of the broad ASP community. Table 1 gives an overview of all benchmarks gathered for the competition on the Asparagus platform, listing problems forming benchmark classes, the accompanying number of instances, the names of the contributors, and the associated competition categories (M stands for MGS, C for SCore, and L for SLparse, respectively).

For each competition category, benchmarks were selected by a fixed yet random scheme, shown in Algorithm 1. The available benchmark classes are predefined in $Classes$. Their instances that have been run already are in $Used$, the ones not run so far are in $Fresh$. As an invariant, we impose $Used \cap Fresh = \emptyset$. For a benchmark class C in $Classes$, we access used and fresh instances via $Used[C]$ and $Fresh[C]$, respectively. The maximum number max of instances per class aims at having approximately 100 benchmark instances overall in the evaluation, that is, $|Classes| * max \approx 100$ (if feasible). A benchmark instance in $Used[C]$ is considered suitable, that is, it is in SUITABLE($Used[C]$), if at least one call script (see Section 5) was able to solve it and at most three call scripts solved it in less than one second (in other words, some system can solve it, yet it is not too easy). Function SELECT($n, Fresh[C]$) randomly determines n fresh instances from class C if available (if $n \leq |Fresh[C]|$) in order to eventually obtain max suitable instances of class C. Procedure RUN($ToRun$) runs all call scripts on the benchmark instances scheduled in $ToRun$. When $ToRun$ runs empty, no fresh benchmark instances are selectable. If a benchmark class yields less than max suitable instances, Algorithm 1 needs to be re-invoked with an increased max value for the other benchmark classes; thus, Algorithm 1 is "only" semiautomatic.

5 Competitors

As with benchmarks, all executable programs (grounders and solvers) were installed and run on Asparagus. To this end, participants had to obtain Asparagus accounts, unless they already had one, and to register for the respective competition categories.

Different variants of a system were allowed to run in the competition by using different *call scripts*; per competitor, up to three of them could be registered for each competition category. The list of participating solvers and corresponding call scripts can be found in Table 2.

Table 2. Participating solvers and corresponding call scripts ($\cdot^\star$ used in both SCore and SCore$^\vee$; $\cdot^\vee$ used in SCore$^\vee$ only)

Solver	Affiliation	MGS	SCore	SLparse
asper	Angers		ASPeR-call-script ASPeRS20-call-script ASPeRS30-call-script	
assat	Hongkong	script.assat.normal		script.assat.lparse-output
clasp	Potsdam	clasp_cmp_score clasp_cmp_score2 clasp_score_def	clasp_cmp_score_glp clasp_cmp_score_glp2 clasp_score_glp_def	clasp_cmp_slparse clasp_cmp_slparse2 clasp_slparse_def
cmodels	Texas	default scriptAtomreasonLp scriptEloopLp	defaultGlparse.sh scriptAtomreasonGlparse scriptEloopGlparse disjGlparseDefault$^\vee$ disjGparseEloop$^\vee$ disjGparseVerMin$^\vee$	groundedDefault scriptAtomreasonGr scriptEloopGr
dlv	Vienna/ Calabria	dlv-contest-special dlv-contest	dlv-contest-special* dlv-contest*	
gnt	Helsinki	gnt dencode+gnt dencode_bc+gnt	gnt_score* dencode+gnt_score* dencode_bc+gnt_score*	gnt_slparse dencode+gnt_slparse dencode_bc+gnt_slparse
lp2sat	Helsinki		lp2sat+minisat wf+lp2sat+minisat lp2sat+siege	
nomore	Potsdam	nomore-default nomore-localprop nomore-D	nomore-default-SCore nomore-localprop-SCore nomore-D-SCore	nomore-default-slparse nomore-localprop-slparse nomore-D-slparse
pbmodels	Kentucky	pbmodels-minisat+-MGS pbmodels-pueblo-MGS pbmodels-wsatcc-MGS	pbmodels-minisat+-SCore pbmodels-pueblo-SCore pbmodels-wsatcc-SCore	pbmodels-minisat+-SLparse pbmodels-pueble-SLparse pbmodels-wsatcc-SLparse
smodels	Helsinki	smodels smodels_rs smodels_rsn	smodels_score smodels_rs_score smodels_rsn_score	smodels_slparse smodels_rs_slparse smodels_rsn_slparse

6 Results

This section presents the results of the *First Answer Set Programming System Competition*. The placement of systems was determined according to the number of instances solved within the allocated time and space as the primary measure of performance. Run times were used as tie breakers. Given that each system was allowed to participate in each competition category in three variants, we decided to allocate only one place to each system. The place of a system is determined by its best performing variant, represented by a call script.

Each single run was limited to 600 seconds execution time and 448 MB RAM memory usage. For each competition category, we give below tables providing a complete placement of all participating call scripts. We report for each call script the absolute and relative number of solved instances ('Solved' and '%'), its minimum, maximum, and average run times, where 'avg' gives the average run time on all solved instances and 'avg^t' gives the average run time on all instances with a timeout taken as 600 seconds. The last column gives the Euclidean distance between the vector of all run times of a call script and the virtually best solver taken to be the vector of all minimum run times.

For each competition category, we also provide similar data for the used benchmark classes. These tables give the number of instances selected from each class ('#'), the absolute and relative number of successfully completed runs ('Solved' and '%') aggregated over all instances and call scripts, the same separately for satisfiable ('SAT') and unsatisfiable ('UNSAT') instances, and finally, the minimum, maximum, and average times over all successfully completed runs on the instances of a class.

Finally, we chart the numbers of solved instances and the solving times of participating call scripts for each competition category. See Section 6.1 for an exemplary description of these graphics.

More statistics and details are available at [17].

6.1 Results of the MGS Competition

The winners of the MGS competition are:

FIRST PLACE WINNER	*dlv*
SECOND PLACE WINNER	*pbmodels*
THIRD PLACE WINNER	*clasp*

The detailed placement of call scripts is given in Table 3. Table 4 gives statistics about the benchmark classes used in the MGS competition. The performance of all participat-

Table 3. Placing of call scripts in the MGS competition

Place	Call Script	Solved	%	min	max	avg	avg^t	EuclDist
1	dlv-contest-special	76/119	63.87	0.07	565.38	54.31	251.49	3542.79
2	dlv-contest	66/119	55.46	0.06	579.09	49.73	294.81	3948.3
3	pbmodels-minisat+-MGS	65/111	58.56	0.52	563.39	83.27	297.41	4142.88
4	clasp_cmp_score2	64/111	57.66	0.87	579.14	115.35	320.56	4285.59
5	clasp_score_def	60/111	54.05	0.91	542.64	80.09	318.97	4277.69
6	clasp_cmp_score	58/111	52.25	0.83	469.46	87.40	332.16	4280.92
7	pbmodels-pueble-MGS	54/111	48.65	0.34	584.31	80.94	347.49	4491.85
8	default	51/119	42.86	0.23	453.88	64.96	370.7	4543.12
9	smodels_rs	34/118	28.81	0.30	539.33	153.86	471.45	5349.11
10	smodels	34/104	32.69	1.14	584.06	173.60	460.6	4974.14
11	pbmodels-wsatcc-MGS	23/111	20.72	1.05	563.52	136.97	504.06	5409.17
12	smodels_rsn	22/111	19.82	0.12	579.10	163.14	513.42	5471.81
13	nomore-default	13/111	11.71	22.04	521.11	315.78	566.71	5714.67
14	scriptAtomreasonLp	12/24	50.00	3.59	259.88	91.18	345.59	2121.13
15	nomore-localprop	10/111	9.01	19.54	521.03	324.83	575.21	5787.71
16	nomore-D	9/111	8.11	48.50	473.89	161.99	564.49	5763.96
17	scriptEloopLp	4/16	25.00	49.92	223.03	106.09	476.52	2088.98
18	gnt	0/8	0.00				600	1696.31
19	dencode+gnt	0/8	0.00				600	1696.31
20	dencode_bc+gnt	0/8	0.00				600	1696.31
21	script.assat.normal	0/50	0.00				600	4101.66

Table 4. Benchmarks used in the MGS competition

Benchmark Class	#	Solved	%	SAT	%	UNSAT	%	min	max	avg
Sokoban	8	16/16	100.00	6/6	100.00	10/10	100.00	12.66	109.52	54.39
Weighted Spanning Tree	8	89/127	70.08	89/127	70.08	0/0		0.06	579.10	115.27
Social Golfer	8	75/120	62.50	59/75	78.67	16/45	35.56	0.23	579.09	40.34
Bounded Spanning Tree	8	73/127	57.48	73/127	57.48	0/0		0.23	519.09	65.47
Towers of Hanoi	8	71/136	52.21	71/136	52.21	0/0		1.74	584.31	101.10
Blocked N-Queens	8	60/120	50.00	44/75	58.67	16/45	35.56	6.50	542.64	181.69
Hamiltonian Cycle	8	64/150	42.67	64/150	42.67	0/0		0.88	453.88	57.46
Weighted Latin Square	8	41/120	34.17	22/45	48.89	19/75	25.33	0.12	477.67	93.04
Weight-Bounded Dominating Set	8	42/127	33.07	42/127	33.07	0/0		0.83	563.39	127.87
Schur Numbers	8	32/120	26.67	18/90	20.00	14/30	46.67	3.27	468.79	120.96
15-Puzzle	7	24/105	22.86	24/105	22.86	0/0		52.91	579.14	291.20
Car Sequencing	8	25/120	20.83	24/105	22.86	1/15	6.67	0.69	563.52	106.08
Fast Food	8	19/127	14.96	8/64	12.50	11/63	17.46	5.30	352.11	113.08
Traveling Salesperson	8	16/128	12.50	16/128	12.50	0/0		0.72	11.18	2.17
Reachability	8	8/160	5.00	6/120	5.00	2/40	5.00	0.26	169.90	49.48

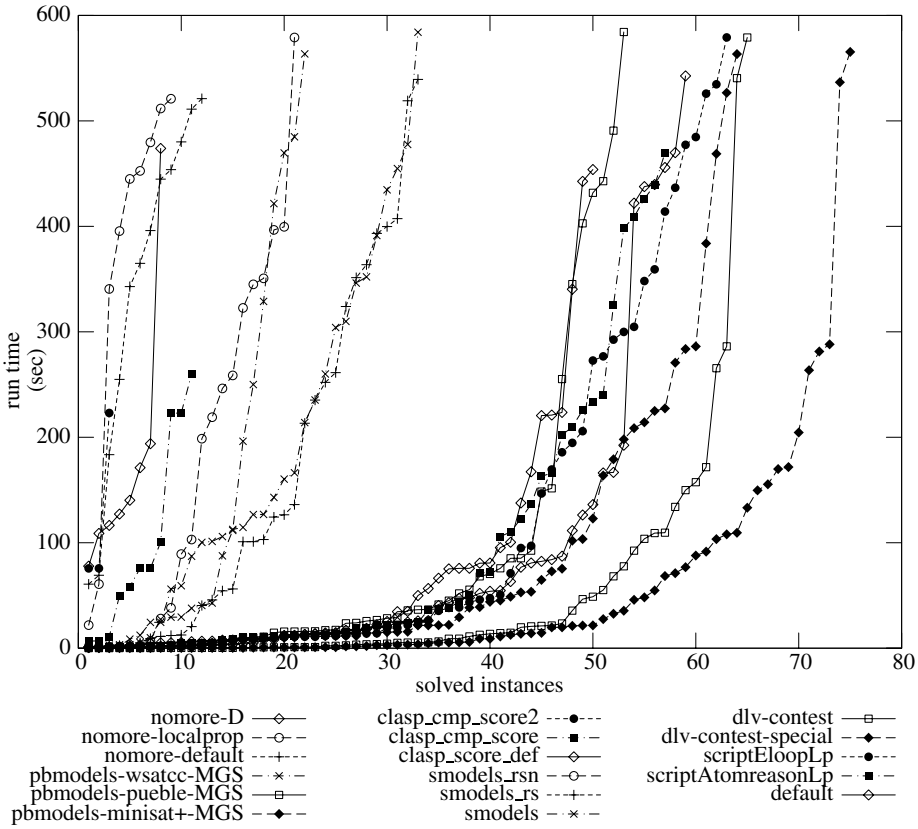

Fig. 1. Chart of the MGS competition

ing call scripts is also given in graphical form in Figure 1. Thereby, the x-axis represents the number of (solved) benchmark instances, and the y-axis represents time. For a given call script, the solved instances are ordered by run times, and a point (x, y) in the chart

expresses that the xth instance was solved in y seconds. The more to the right the curve of a call script ends, the more benchmark instances were solved within the allocated time and space. Since the number of solved instances is our primary measure of performance, the rightmost call script is the winner of the MGS competition.

6.2 Results of the SCore Competition

The winners of the SCore competition are:

FIRST PLACE WINNER	*clasp*
SECOND PLACE WINNER	*smodels*
THIRD PLACE WINNER	*cmodels*

The detailed placement of call scripts is given in Table 5. Table 6 gives statistics about the benchmark classes used in the SCore competition. The performance of all participating call scripts is charted in Figure 2.

Table 5. Placing of call scripts in the SCore competition

Place	Call Script	Solved	%	min	max	avg	avg^t	EuclDist
1	clasp_cmp_score_glp2	89/95	93.68	0.56	530.21	29.81	65.82	1080.46
2	clasp_cmp_score_glp	89/95	93.68	0.75	504.49	30.36	66.34	1099.14
3	clasp_score_glp_def	86/95	90.53	0.75	431.66	25.20	79.66	1386.63
4	smodels_rs_score	81/95	85.26	1.21	346.36	38.93	121.61	1872.81
5	defaultGlparse.sh	81/95	85.26	1.35	597.97	46.86	128.38	2089.18
6	scriptAtomreasonGlparse	80/95	84.21	1.30	576.80	42.40	130.44	2107.83
7	pbmodels-minisat+-SCore	80/95	84.21	0.72	436.11	57.18	142.89	2170.4
8	pbmodels-pueblo-SCore	78/95	82.11	0.34	452.84	41.00	141.03	2210.39
9	dencode+gnt_score	78/95	82.11	1.27	363.19	42.80	142.51	2162.64
10	smodels_score	77/95	81.05	1.28	352.41	40.40	146.43	2217.61
11	dencode_bc+gnt_score	77/95	81.05	1.27	360.70	42.52	148.15	2228.65
12	gnt_score	77/95	81.05	1.27	359.77	42.56	148.18	2228.83
13	scriptEloopGlparse	75/95	78.95	1.36	598.20	42.86	160.15	2493.41
14	smodels_rsn_score	75/95	78.95	1.21	486.23	63.00	176.05	2503.32
15	lp2sat+minisat	75/95	78.95	1.10	561.06	79.89	189.39	2621.13
16	wf+lp2sat+minisat	73/95	76.84	1.56	587.40	86.42	205.35	2792.51
17	dlv-contest-special	69/95	72.63	0.24	586.62	102.47	238.64	3090.71
18	dlv-contest	68/95	71.58	0.24	587.83	96.69	239.74	3110.36
19	lp2sat+siege	68/95	71.58	1.11	471.36	97.50	240.32	3052.8
20	nomore-localprop-SCore	64/95	67.37	2.45	550.43	103.23	265.34	3316.33
21	nomore-default-SCore	63/95	66.32	2.45	554.76	124.62	284.75	3415.78
22	nomore-D-SCore	62/95	65.26	2.77	559.88	161.15	313.59	3583.85
23	ASPeR-call-script	24/95	25.26	1.47	592.24	98.28	473.25	4906.79
24	ASPeRS30-call-script	21/95	22.11	1.51	561.20	88.99	487.04	4995.78
25	ASPeRS20-call-script	21/95	22.11	1.49	381.33	89.40	487.13	4980.24
26	pbmodels-wsatcc-SCore	6/95	6.32	25.57	529.80	208.15	575.25	5514.97

Table 6. Benchmarks used in the SCore competition

Benchmark Class	#	Solved	%	SAT	%	UNSAT	%	min	max	avg
15-Puzzle	10	236/260	90.77	121/130	93.08	115/130	88.46	0.74	480.13	25.49
Factoring	5	114/130	87.69	46/52	88.46	68/78	87.18	1.21	554.76	50.35
RLP-150	14	306/364	84.07	21/26	80.77	285/338	84.32	0.34	205.03	22.01
RLP-200	14	287/364	78.85	0/0		287/364	78.85	0.39	581.98	75.21
Schur Numbers	5	99/130	76.15	88/104	84.62	11/26	42.31	2.76	561.20	49.82
EqTest	5	93/130	71.54	0/0		93/130	71.54	0.66	592.24	75.02
Hamiltonian Path	14	219/364	60.16	201/338	59.47	18/26	69.23	0.24	559.88	64.74
Random Non-Tight	14	216/364	59.34	38/52	73.08	178/312	57.05	0.57	598.20	121.87
Blocked N-Queens	14	167/364	45.88	55/156	35.26	112/208	53.85	13.70	587.40	110.59

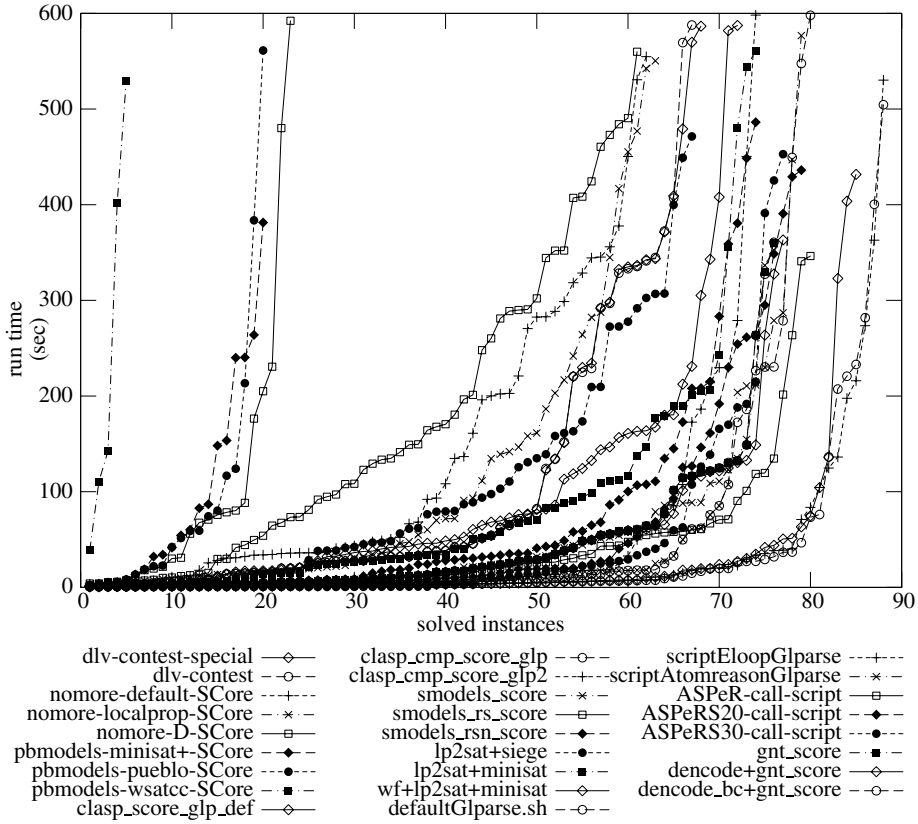

Fig. 2. Chart of the SCore competition

Table 7. Placing of call scripts in the SCore$^\vee$ competition

Place	Call Script	Solved	%	min	max	avg	avg^t	EuclDist
1	dlv-contest-special	54/55	98.18	0.03	258.73	23.59	34.07	279.35
2	dlv-contest	54/55	98.18	0.03	259.97	23.86	34.33	279.44
3	disjGlparseDefault	33/55	60.00	1.06	521.59	54.49	272.69	2631.4
4	dencode+gnt_score	29/55	52.73	2.23	521.51	56.34	313.34	2922.73
5	gnt_score	29/55	52.73	2.21	521.91	56.44	313.4	2922.87
6	dencode_bc+gnt_score	29/55	52.73	2.22	522.63	56.45	313.4	2923.17
7	disjGparseEloop	27/55	49.09	1.21	521.55	33.83	322.06	2978.46
8	disjGparseVerMin	27/55	49.09	1.22	523.40	33.98	322.14	2978.77

The winners of the SCore$^\vee$ competition are:

FIRST PLACE WINNER	*dlv*
SECOND PLACE WINNER	*cmodels*
THIRD PLACE WINNER	*gnt*

Table 8. Benchmarks used in the SCore$^\vee$ competition

Benchmark Class	#	Solved	%	SAT	%	UNSAT	%	min	max	avg
Grammar-Based Information Extraction	15	120/120	100.00	64/64	100.00	56/56	100.00	0.62	7.86	5.03
Disjunctive Loops	3	21/24	87.50	0/0		21/24	87.50	0.44	522.63	95.24
Strategic Companies	15	88/120	73.33	88/120	73.33	0/0		0.35	523.40	71.22
Mutex	7	18/56	32.14	0/0		18/56	32.14	0.03	259.97	37.41
Random Quantified Boolean Formulas	15	35/120	29.17	0/0		35/120	29.17	0.11	290.99	44.41

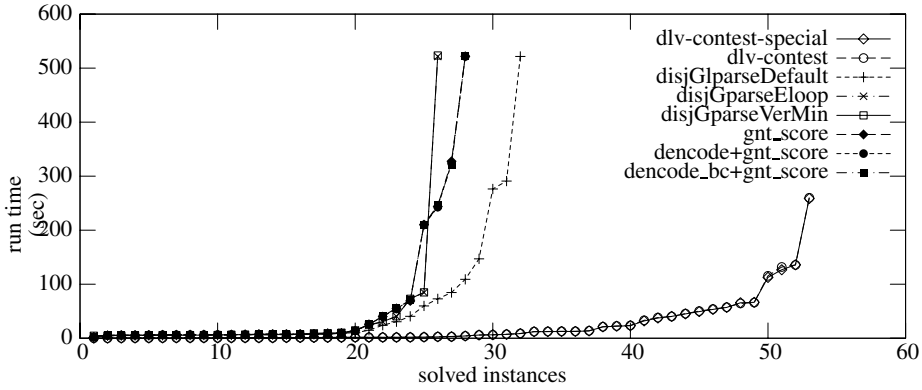

Fig. 3. Chart of the SCore$^\vee$ competition

Table 9. Placing of call scripts in the SLparse competition

Place	Call Script	Solved	%	min	max	avg	avg^t	EuclDist
1	clasp_cmp_slparse2	100/127	78.74	0.38	556.49	75.96	187.37	2791.89
2	clasp_cmp_slparse	94/127	74.02	0.41	502.53	61.37	201.33	2919.46
3	pbmodels-minisat+-SLparse	91/127	71.65	0.49	503.57	76.69	225.03	3241.06
4	clasp_slparse_def	89/127	70.08	0.37	546.50	55.62	218.5	3152.34
5	smodels_rs_slparse	87/127	68.50	0.23	576.28	95.92	254.69	3403.9
6	groundedDefault	81/127	63.78	0.25	407.49	46.20	246.79	3448.34
7	scriptAtomreasonGr	81/127	63.78	0.25	407.46	50.55	249.56	3465.67
8	scriptEloopGr	78/127	61.42	0.24	407.48	46.15	259.84	3598.7
9	smodels_slparse	75/127	59.06	0.26	518.08	102.76	306.35	3958.86
10	smodels_rsn_slparse	74/127	58.27	0.35	596.39	70.52	291.49	3815.31
11	pbmodels-pueblo-SLparse	69/127	54.33	0.25	593.05	87.25	321.42	4189.03
12	nomore-D-slparse	54/127	42.52	1.08	530.39	152.78	409.84	4765.06
13	nomore-localprop-slparse	50/127	39.37	1.08	517.63	120.80	411.34	4846.58
14	nomore-default-slparse	49/127	38.58	1.08	549.80	143.44	423.85	4920.23
15	gnt_slparse	35/127	27.56	2.10	482.13	81.40	457.08	5276.26
16	dencode+gnt_slparse	35/127	27.56	2.18	482.81	81.64	457.15	5276.38
17	dencode_bc+gnt_slparse	35/127	27.56	2.10	485.36	81.70	457.16	5276.53
18	script.assat.lparse-output	30/127	23.62	1.00	225.28	38.64	467.4	5379.18
19	pbmodels-wsatcc-SLparse	25/127	19.69	1.12	272.98	46.56	491.05	5585.82

The detailed placement of call scripts is given in Table 7. Table 8 gives statistics about the benchmark classes used in the SCore$^\vee$ competition. The performance of all participating call scripts is charted in Figure 3.

Table 10. Benchmarks used in the SLparse competition

Benchmark Class	#	Solved	%	SAT	%	UNSAT	%	min	max	avg
RLP-200	5	89/95	93.68	18/19	94.74	71/76	93.42	0.25	465.69	60.94
RLP-150	5	87/95	91.58	17/19	89.47	70/76	92.11	0.25	183.29	14.17
Factoring	4	69/76	90.79	36/38	94.74	33/38	86.84	0.63	549.80	64.09
verifyTest-variableSearchSpace (TOAST)	5	81/95	85.26	81/95	85.26	0/0		0.24	303.32	16.43
Random Non-Tight	5	75/95	78.95	41/57	71.93	34/38	89.47	0.41	518.08	123.56
Knight's Tour	5	71/95	74.74	71/95	74.74	0/0		1.04	248.97	28.29
Su-Doku	3	42/57	73.68	42/57	73.68	0/0		18.15	176.69	68.00
searchTest-plain (TOAST)	5	68/95	71.58	18/38	47.37	50/57	87.72	1.54	339.51	45.50
searchTest-verbose (TOAST)	5	63/95	66.32	63/95	66.32	0/0		25.84	485.36	136.07
Hamiltonian Path	5	60/95	63.16	60/95	63.16	0/0		0.37	530.39	58.02
Weighted Spanning Tree	5	58/95	61.05	58/95	61.05	0/0		3.24	596.39	122.04
Solitaire Forward	5	55/95	57.89	55/95	57.89	0/0		1.19	593.05	49.06
Bounded Spanning Tree	5	54/95	56.84	54/95	56.84	0/0		7.43	413.63	70.11
Hamiltonian Cycle	5	51/95	53.68	51/95	53.68	0/0		0.47	464.71	51.30
Solitaire Backward	5	47/95	49.47	33/76	43.42	14/19	73.68	0.30	552.11	71.72
Towers of Hanoi	5	43/95	45.26	43/95	45.26	0/0		6.28	478.30	169.96
Blocked N-Queens	5	40/95	42.11	36/76	47.37	4/19	21.05	1.57	590.04	193.59
Social Golfer	5	37/95	38.95	24/38	63.16	13/57	22.81	0.84	291.48	30.70
Schur Numbers	5	31/95	32.63	11/57	19.30	20/38	52.63	1.23	496.82	104.57
Hashiwokakero	5	26/95	27.37	0/0		26/95	27.37	6.71	377.69	72.36
Weighted Latin Square	5	23/95	24.21	6/19	31.58	17/76	22.37	0.23	576.28	144.13
15-Puzzle	5	17/95	17.89	17/95	17.89	0/0		106.66	502.53	327.56
Weight-Bounded Dominating Set	5	15/95	15.79	15/95	15.79	0/0		1.55	467.51	112.55
Traveling Salesperson	5	12/95	12.63	12/95	12.63	0/0		0.35	212.66	20.24
Solitaire Backward (2)	5	11/95	11.58	11/95	11.58	0/0		5.32	330.89	101.55
Car Sequencing	5	7/95	7.37	7/95	7.37	0/0		7.17	556.49	249.51

6.3 Results of the SLparse Competition

The winners of the SLparse competition are:

FIRST PLACE WINNER	*clasp*
SECOND PLACE WINNER	*pbmodels*
THIRD PLACE WINNER	*smodels*

The detailed placement of call scripts is given in Table 9. Table 10 gives statistics about the benchmark classes used in the SLparse competition. The performance of all participating call scripts is charted in Figure 4.

7 Discussion

This *First Answer Set Programming System Competition* offers many interesting lessons stemming from running diverse solvers on multifaceted benchmark instances. Some of the lessons may have general implications on the future developments in ASP.

First, the experiences gained from the effort to design the competition clearly point out that the lack of well-defined input, intermediate, and output languages is a major problem. In some cases, it forced the competition team to resort to "ad hoc" solutions. Further, there is no standard core ASP language covering programs with aggregates, which makes it difficult to design a single and fair field for all systems to compete. No standard way in which errors are signaled and no consensus on how to deal with incomplete solvers are somewhat less critical but also important issues. Benchmark selection is a major problem. The way benchmarks and their instances are chosen may

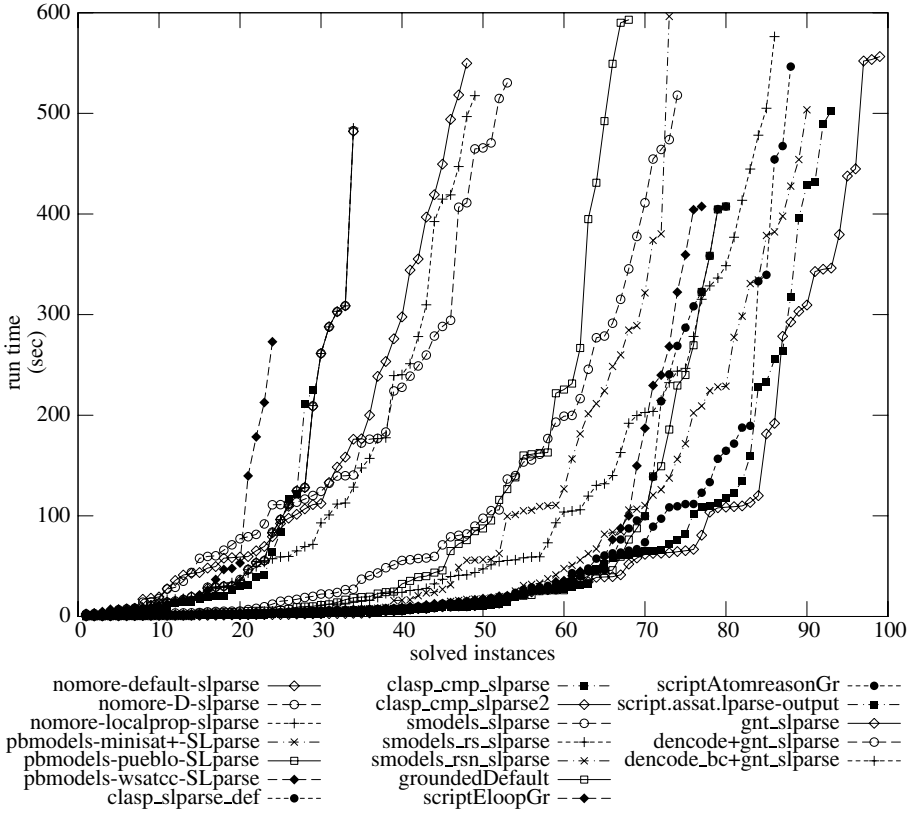

Fig. 4. Chart of the SLparse competition

have a significant impact on the results in view of diverging performances of solvers and different degrees of difficulty among the instances of benchmark classes. Sometimes even grounding problem encodings on problem instances to produce ground programs on which solvers were to compete was a major hurdle (see below).

This first edition of the competition focused on the performance of solvers on ground programs, which is certainly important. However, the roots of the ASP approach are in declarative programming and knowledge representation. For both areas, *modeling* knowledge domains and problems that arise in them is of major concern (this is especially the case for knowledge representation). By developing the MGS category, we tried to create a platform where ASP systems could be differentiated from the perspective of their modeling functionality. However, only one group chose to develop programs specialized to their system (hence, this group and their system are the well-deserved winner). All other groups relied on default encodings. It is critical that a better venue for testing modeling capabilities is provided for future competitions.

Further, not only modeling support and the performance of solvers determine the quality of an ASP system. Grounding is an essential part of the process too and, in some cases, it is precisely where the bottleneck lies. The MGS category was the only category

that took both the grounding time and the solving time into account. It is important to stress more the role of grounding in future competitions.

There will be future competitions building on the experiences of this one. Their success and their impact on the field will depend on continued broad community participation in fine-tuning and expanding the present format. In this respect, the *First Answer Set Programming System Competition* should encourage the further progress in the development of ASP systems and applications, similar to competitions in related areas, such as SATisfiability, Quantified Boolean Formulas, and Pseudo-Boolean constraints.

Acknowledgments

This project would not have been possible without strong and broad support from the whole ASP community. We are grateful to all friends and colleagues who contributed ideas on the contest format, submitted benchmark problems, and provided continued encouragement. Most of all, we want to thank all the competitors. Without you, there would have been no competition.

Mirosław Truszczyński acknowledges the support of NSF grant IIS-0325063 and KSEF grant 1036-RDE-008.

References

1. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3-4) (1999) 241–273
2. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In Apt, K., Marek, W., Truszczyński, M., Warren, D., eds.: The Logic Programming Paradigm: a 25-Year Perspective. Springer (1999) 375–398
3. Gelfond, M., Leone, N.: Logic programming and knowledge representation — the A-prolog perspective. Artificial Intelligence **138**(1-2) (2002) 3–38
4. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
5. Colmerauer, A., Kanoui, H., Pasero, R., Roussel, P.: Un systeme de communication homme-machine en Francais. Technical report, University of Marseille (1973)
6. Kowalski, R.: Predicate logic as a programming language. In Rosenfeld, J., ed.: Proceedings of the Congress of the International Federation for Information Processing, North Holland (1974) 569–574
7. McCarthy, J.: Circumscription — a form of nonmonotonic reasoning. Artificial Intelligence **13**(1-2) (1980) 27–39
8. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13**(1-2) (1980) 81–132
9. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: Proceedings of the International Conference on Logic Programming, MIT Press (1988) 1070–1080
10. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9**(3-4) (1991) 365–385
11. Marek, W., Truszczyński, M.: Stable semantics for logic programs and default theories. In Lusk, E., Overbeek, R., eds.: Proceedings of the North American Conference on Logic Programming, MIT Press (1989) 243–256

12. Bidoit, N., Froidevaux, C.: Negation by default and unstratifiable logic programs. Theoretical Computer Science **78**(1) (1991) 85–112
13. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: A deductive system for nonmonotonic reasoning. In Dix, J., Furbach, U., Nerode, A., eds.: Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning, Springer (1997) 364–375
14. Niemelä, I., Simons, P.: Smodels — an implementation of the stable model and well-founded semantics for normal logic programs. In Dix, J., Furbach, U., Nerode, A., eds.: Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning, Springer (1997) 420–429
15. Borchert, P., Anger, C., Schaub, T., Truszczyński, M.: Towards systematic benchmarking in answer set programming: The Dagstuhl initiative. In Lifschitz, V., Niemelä, I., eds.: Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning, Springer (2004) 3–7
16. Le Berre, D., Simon, L., eds.: Special Volume on the SAT 2005 Competitions and Evaluations. Journal on Satisfiability, Boolean Modeling and Computation **2**(1-4) (2006)
17. (`http://asparagus.cs.uni-potsdam.de/contest`)
18. (`http://asparagus.cs.uni-potsdam.de`)
19. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7**(3) (2006) 499–562
20. Syrjänen, T.: Lparse 1.0 user's manual. (http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz)