Online appendix for the paper

The Third Open Answer Set Programming Competition

published in Theory and Practice of Logic Programming

FRANCESCO CALIMERI, GIOVAMBATTISTA IANNI, FRANCESCO RICCA

Dipartimento di Matematica
Università della Calabria, Italy (*e-mail:* {calimeri,ianni,ricca}@mat.unical.it)

submitted 24 August 2011; revised 12 January 2012; accepted 4 June 2012

Appendix A ASP-Core Syntax and Semantics overview

In the following, an overview of both the main constructs and the semantics of the ASP-Core language is reported. The full ASP-Core language specification can be found in (Calimeri et al. 2011).

A.1 ASP-Core *Syntax*

For the sake of readability, the language specification is hereafter given according to the traditional mathematical notation. A lexical matching table from the following notation to the actual raw input format prescribed for participants is provided in (Calimeri et al. 2011).

Terms, constants, variables. Terms are either *constants* or *variables*. Constants can be either *symbolic constants* (strings starting with lower case letter), *strings* (quoted sequences of characters), or integers. Variables are denoted as strings starting with an upper case letter. As a syntactic shortcut, the special variable "_" is a placeholder for a fresh variable name in the context at hand.

Atoms and Literals. An atom is of the form $p(X_1, \ldots, X_n)$, where p is a *predicate name*, $X_1, \ldots, X_n$ are *terms*, and n ($n \geq 0$) is the fixed arity associated to p[1]. A classical literal is either of the form a (*positive classical literal*) or $-a$ (*negative classical literal*), for a being an *atom*. A *naf-literal* is either a *positive naf-literal* a or a *negative naf-literal* **not** a, for a being a classical literal.

[1] The atom referring to a predicate p of arity 0, can be stated either in the form $p()$ or p.

Rules. An ASP-Core program P is a a finite set of *rules.* A rule r is of the form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_k, \textbf{not } n_1, ..., \textbf{not } n_m.$$

where $n, k, m \geq 0$, and at least one of n,k is greater than 0; $a_1, \ldots, a_n$, $b_1, \ldots, b_k$, and $n_1, \ldots, n_m$ are *classical literals.* $a_1 \vee \cdots \vee a_n$ constitutes the *head* of r, whereas $b_1, \ldots, b_k, \textbf{not } n_1, ..., \textbf{not } n_m$ is the *body* of r. As usual, whenever $k = m = 0$, we omit the "$\leftarrow$" sign. r is a *fact* if $n = 1, k = m = 0$, while r is a *constraint* if $n = 0$.

Rules written in ASP-Core are assumed to be *safe.* A rule r is safe if all its variables occur in at least one positive naf-literal in the body of r. A program P is safe if all its rules are safe. A program (a rule, a literal, an atom) is said to be *ground* (or *propositional*) if it contains no variables.

Ground Queries. A program P can be coupled with a *ground query* in the form q?, where q is a ground naf-literal.

A.2 ASP-Core *Semantics*

The semantics of ASP-Core, based on (Gelfond and Lifschitz 1991), exploits the traditional notion of the Herbrand interpretation.

Herbrand universe. Given a program P, the *Herbrand universe* of P, denoted by U_P, is the set of all constants occurring in P. The *Herbrand base* of P, denoted by B_P, is the set of all ground naf-literals obtainable from the atoms of P by replacing variables with elements from U_P.

Ground programs and Interpretations. A *ground instance* of a rule r is obtained replacing each variable of r by an element from U_P. Given a program P, we define the *instantiation (grounding)* $grnd(P)$ of P as the set of all ground instances of its rules. Given a ground program P, an *interpretation* I for P is a subset of B_P. A *consistent* interpretation is such that $\{a, -a\} \not\subseteq I$ for any ground atom a. In the following, we only deal with consistent interpretations.

Satisfaction. A positive naf-literal $l = a$ (resp., a naf-literal $l = \textbf{not } \ a$), for predicate atom a, is true with respect to an interpretation I if $a \in I$ (resp., $a \notin I$); it is false otherwise. Given a ground rule r, we say that r is satisfied with respect to an interpretation I if some atom appearing in the head of r is true with respect to I or some naf-literal appearing in the body of r is false with respect to I.

Models. Given a ground program P, we say that a consistent interpretation I is a *model* of P iff all rules in $grnd(P)$ are satisfied w.r.t. I. A model M is *minimal* if there is no model N for P such that $N \subset M$.

Gelfond-Lifschitz reduct and Answer Sets. The *Gelfond-Lifschitz reduct* (Gelfond and Lifschitz 1991) of a program P with respect to an interpretation I is the positive ground program P^I obtained from $grnd(P)$ by: (i) deleting all rules with a negative naf-literal false w.r.t. I; (ii) deleting all negative naf-literals from the remaining rules. $I \subseteq B_P$ is an *answer set* for P iff I is a minimal model for P^I. The set of all answer sets for P is denoted by $AS(P)$.

Semantics of ground queries. Let P be a program and $q?$ be a query, $q?$ is *true* iff for all $A \in AS(P)$ it holds that $q \in A$. Basically, the semantics of queries corresponds to *cautious reasoning*, since a query is true if the corresponding atom is true in all answer sets of P.

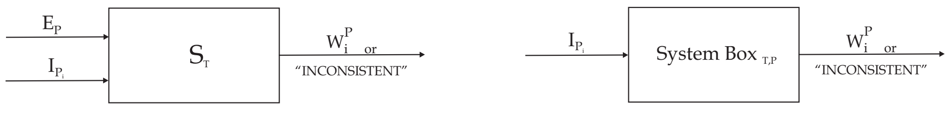
Appendix B Detailed Competition Settings

The competition settings for the two tracks are depicted in Figure B 1. The problems collected into the official problem suite (see Appendix D), were grouped into two different suites, one per each track. The problems belonging to the System Track suite were nearly a proper subset of the ones featured in the M&S Track. In both tracks, for each problem P, a number of instances $I_{P_1}, \ldots, I_{P_N}$ were selected.[2] For any problem P included into the System Track a corresponding fixed declarative specification, written in ASP-Core, E_P was also given.

A team T participating in the System Track had to provide a unique executable system S_T. A team participating in the M&S Track, instead, had to produce a possibly-different execution bundle $SystemBox_{T,P}$ for any problem P in the M&S Track suite. For each problem P, the participants were fed iteratively with all instances I_{P_i} of P (in the case of the System Track each instance was fed together with the corresponding problem encoding E_P).

The submitted executables were challenged to produce either a *witness* solution, denoted by W_i^P, or to report that no solution exists within a predefined amount of allowed time. The expected output format that is determined by the type of the problem (search, query, optimization) is reported in (Calimeri et al. 2011). Participants were made aware, fairly in advance, of fixed encodings (in the case of the System Track), while they were provided only a small set of corresponding training instances. Official instances were kept secret until the actual start of the competition. Scores were awarded according to the competition scoring system (see Section 4 of the original paper and this online Appendix C).

[2] For problems appearing in both tracks, the instances selected for the M&S Track and those selected for the System Track were not necessarily the same.



(a) *System Track* (b) *M&S Track*

Fig. B 1. Competition Setting

Definition of "syntactic special purpose technique". The committee classified as forbidden in the System Track: the switch of internal solver options depending either on command-line filenames, predicate and variable names, and "signature" techniques aimed at recognizing a particular benchmark problem, such as counting the number of rules, constraints, predicates and atoms in a given encoding. In order to discourage the adoption of forbidden techniques, the organizing committee reserved the right to introduce syntactic means for scrambling program encodings, such as file, predicate and variable random renaming. Furthermore, the committee reserved the right to replace official program encodings arbitrarily with equivalent syntactically-changed versions.

It is worth noting that, on the other hand, the semantic recognition of the program structure was allowed, and even encouraged. Allowed semantic recognition techniques explicitly included: *(i)* recognition of the class the problem encoding belongs to (e.g., stratified, positive, etc.), with possible consequent switch-on of on-purpose evaluation techniques; *(ii)* recognition of general rule and program structures (e.g., common un-stratified even and odd-cycles, common join patterns within a rule body, etc.), provided that these techniques were general and not specific of a given problem selected for the competition.

Detailed Software and Hardware Settings. The Competition took place on a battery of four servers, featuring a 4-core Intel Xeon CPU X3430 running at 2.4 Ghz, with 4 GiB of physical RAM and PAE enabled.

The operating system of choice was Linux Debian Lenny (32bit), equipped with the C/C++ compiler GCC 4.3 and common scripting/development tools. Competitors were allowed to install their own compilers/libraries in local home directories, and to prepare system binaries for the specific Competition hardware settings. All the systems were benchmarked with just one out of four processors enabled, with the exception of the parallel solver `clasp-mt` that could exploit all the available core/processors. Each process spawned by a participant system had access to the usual Linux process memory space (slightly less than 3GiB user space + 1GiB kernel space). The total memory allocated by all the child processes created was however constrained to a total of 3 GiB (1 GiB = 2^{30} bytes). The memory footprint of participant systems was controlled by using the Benchmark Tool Run.[3] This tool is not able to detect short memory spikes (within 100 milliseconds) or, in some corner cases, memory overflow is detected with short delay: however, we pragmatically assumed the tool as the official reference.

Detection of Incorrect Answers. Each benchmark domain P was equipped with a checker program C_P taking as input values a witness A and an instance I, and answering "true" in case A is a valid witness for I w.r.t problem P. The collection of checkers underwent thorough assessment and then was pragmatically assumed to be correct.

[3] `http://fmv.jku.at/run/`.

Suppose that a system $\mathcal{S}$ is faulty for instance I of problem P; then, there were two possible scenarios in which incorrect answers needed detection and subsequent disqualification for a given system:

- $\mathcal{S}$ produced an answer A, and A was not a correct solution (either because I was actually unsatisfiable or A was wrong at all). This scenario was detected by checking the output of $C_P(A, I)$;
- $\mathcal{S}$ answered that the instance was not satisfiable, but actually I had some witness. In this case, we checked whether a second system $\mathcal{S}'$ produced a solution A' for which $C_P(A', I)$ was true.

Concerning optimization problems, checkers produced also the cost C of the given witness. This latter value was considered when computing scores and for assessing answers of systems. Note that cases of general failure (e.g. out of memory, other abrupt system failures) were not subject of disqualification on a given benchmark. As a last remark, note that in the setting of the System Track, where problem encodings were fixed, a single stability checker for answer sets could replace our collection of checkers. We preferred to exploit already available checker modules, which were also used for assessing the correctness of fixed official encodings set for the System Track. This enabled us to detect some early errors in fixed encodings: however, our lesson learned suggests that a general stability checker should be placed side-by-side to specific benchmark checkers.

Other settings. The committee kept its neutral position and did not disclose any material submitted by participants until the end of the competition: however, participants were allowed to share their own work willingly at any moment. The above choice was taken in order to prefer scientific collaboration between teams over a strict competitive setting. All participants were asked to agree that any kind of submitted material (system binaries, scripts, problems encodings, etc.) was to be made public after the competition, so to guarantee transparency and reproducibility. None of the members of the organizing committee submitted a system to the Competition, in order to play the role of neutral referee properly and guarantee an unbiased benchmark selection and rule definition process.

Appendix C Detailed scoring regulations

C.1 Principles

The main factors that were taken into account in the scoring framework are illustrated next.

1. Benchmarks with many instances should not dominate the overall score of a category. Thus, the overall score for a given problem P was normalized with respect to the number N of selected instances for P.
2. Nonsound solvers and encodings were strongly discouraged. Thus, if system S produced an incorrect answer for an instance of a problem P then S is disqualified from P and the *overall score* achieved by S for problem P is invalidated (i.e., is set to zero).

3. A system managing to solve a given problem instance sets a clear gap over all systems not able to do so. Thus, a flat reward for each instance I of a problem P was given to a system S that correctly solved I within the allotted time.
4. Concerning time performance, human beings are generally more receptive to the logarithm of the changes of a value, rather than to the changes themselves; this is especially the case when considering evaluation times. Indeed, different systems with time performances being in the same order of magnitude are perceived as comparatively similar, in terms of both raw time performance and quality; furthermore, a system is generally perceived as clearly *fast*, when its solving times are orders of magnitude below the maximum allowed time. Keeping this in mind, and analogously to what has been done in SAT competitions,[4] a logarithmically weighted bonus was awarded to faster systems depending on the time needed for solving each instance.
5. In the case of optimization problems, scoring should depend also on the quality of the provided solution. Thus, bonus points were rewarded to systems able to find better solutions. Also, we wanted to take into account the fact that small improvements in the quality of a solution are usually obtained at the price of much stronger computational efforts: thus the bonus for a better quality solution has been given on an exponential weighting basis.

C.2 Scoring Rules

The final score obtained by a system $\mathcal{S}$ in a track $\mathcal{T}$ consisted of the sum over the scores obtained by $\mathcal{S}$ in all benchmarks selected for $\mathcal{T}$. In particular, a system could get a maximum of 100 points for each given benchmark problem P considered for $\mathcal{T}$. The overall score of a system on a problem P counting N instances, hereafter denoted by $S(P)$, was computed according to the following formulas that depend on whether P is a search, query or optimization problem.

[4] See, for instance, the log based scoring formulas at `http://www.satcompetition.org/2009/spec2009.html`.

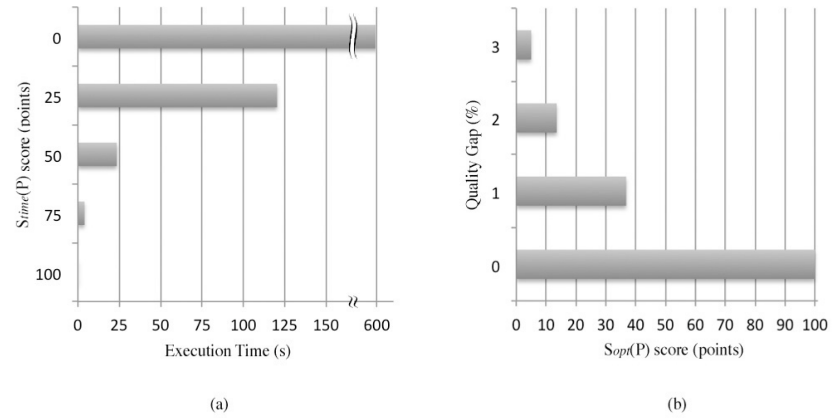


Fig. C1. Scoring Functions Exemplified (one instance, 100 pts max, $t_{out} = 600$).

Wrong Answers. In the case where $\mathcal{S}$ produced an output detected as incorrect[5] for at least one instance of P, then $\mathcal{S}$ was disqualified from P and $S(P)$ was set to zero (i.e., $S(P) = 0$ in case of incorrect output); otherwise, the following formulas were applied for computing $S(P)$.

Search and Query Problems. In case of both search and query problems the score $S(P)$ was computed by the sum

$$S(P) = S_{solve}(P) + S_{time}(P)$$

where S_{solve} and $S_{time}(P)$ take into account the number of instances solved by $\mathcal{S}$ in P and the corresponding running times, respectively; in particular

$$S_{solve}(P) = \alpha \frac{N_S}{N} \ ; \quad S_{time}(P) = \frac{100-\alpha}{N} \sum_{i=1}^{N} \left(1 - \left(\frac{\log(t_i+1)}{\log(t_{out}+1)}\right)\right)$$

for N_S being the number of instances solved by P within the time limit, t_{out} is the maximum allowed time, t_i the time spent by S while solving instance i, and α a percentage factor balancing the impact of $S_{solve}(P)$ and $S_{time}(P)$ on the overall score. Both $S_{solve}(P)$ and $S_{time}(P)$ were rounded to the nearest integer.

Note that $S_{time}(P)$ was specified in order to take into account the "perceived" performance of a system (as discussed in C.1). Figure C 1(a) gives an intuitive idea about how S_{time} distributes a maximum score of 100 points considering a single instance and $t_{out} = 600$. Note that half of the maximum score (50 points) is given to performance below 24 seconds about, and significant differences in scoring correspond to differences of orders of magnitude in time performance.

Optimization Problems. As in the previous edition, the score of a system $\mathcal{S}$ in the case of optimization problems depends on whether $\mathcal{S}$ was able to find a solution or not, and in the former case, the score depends on the quality of the given solutions. In addition, as in the case of decision problems, time performance is taken into account. We assumed the cost function associated with optimization problems must be minimized (the lower, the better), and it had 0 as its lowest bound.

The overall score of a system for an optimization problem P is given by the sum

$$S(P) = S_{opt}(P) + S_{time}(P)$$

where $S_{time}(P)$ is defined as for search problems, and $S_{opt}(P)$ takes into account the quality of the solution found. In particular, for each problem P, system $\mathcal{S}$ is rewarded of a number of points defined as

$$S_{opt}(P) = \alpha \cdot \sum_{i=1}^{N} S_{opt}^{i}$$

where, as before, α is a percentage factor balancing the impact of $S_{opt}(P)$ and $S_{time}(P)$ on the overall score, and S_{opt}^{i} is computed by properly summing, for each instance i of P, one or more of these rewards:

[5] Incorrect answers were determined as specified in Appendix B

1. $\frac{1}{N}$ points, if the system correctly recognizes an unsatisfiable instance;
2. $\frac{1}{4N}$ points, if the system produces a correct witness;
3. $\frac{1}{4N}$ points, if the system correctly recognizes an optimum solution and outputs it;
4. $\frac{1}{2N} \cdot e^{M-Q}$ points, where Q denotes the quality of the solution produced by the system and M denotes the quality of the best answer produced by any system for the current instance, for M conventionally set to 100, and Q normalized accordingly.

Taking into account that an incorrect answer causes the whole benchmark to pay no points, three scenarios may come out: timeout, unsatisfiable instance, or solution produced. Note thus that points of groups (1), (2) and (3-4-5) cannot be rewarded for the same instance.

The intuitive impact of the above "quality" score $S_{opt}(P)$ can be seen in Figure C 1(b), in which the quality of a given solution, expressed in percentage distance from the optimal solution, is associated with the corresponding value of S_{opt} (suppose a maximum of 100 points, $\alpha = 100$, and one single instance per benchmark). Note that a system producing a solution with a quality gap of 1% with respect to the best solution gets only 35 points (over 100) and the quality score quota rapidly decreases (it is basically 0 for quality gap $> 4\%$), so that small differences in the quality of a solution determine a strong difference in scoring according to considerations made in C.1.

In the present competition, for each problem domain we set $t_{out} = 600$ seconds and $\alpha = 50$; N has been set to 10 for the System Track, while it varied from problem to problem for the M&S Track, reaching up to 15 instances per single benchmark problem.

Appendix D Benchmark Suite

Benchmark problems were collected, selected and refined during the *Call for Problems* stage. The whole procedure led to the selection of 35 problems, which constituted the M&S Track problem suite. Taking into account what already discussed in Section 2 of the original paper, twenty problems out of the ones constituting the M&S Track were selected for composing the System Track suite: these had a natural and declarative ASP-Core encoding. Benchmark problems were classified according to their type into three categories: *Search problems*, requiring to find a solution (a *witness*) for the problem instance at hand, or to notify the non-existence of a solution; *Query problems*, consisting in checking whether a ground fact is contained in all the witnesses of the problem instance at hand (same as performing cautious reasoning on a given logic program); and, *Optimization problems*, i.e. a search problem in which a cost function associated to witnesses had to be minimized. The System Track did not contain optimization problems.

Problems were further classified according to their computational complexity in three categories:[6] *Polynomial, NP* and *Beyond NP* problems, these latter with the two subcategories composed of Σ_2^P problems and optimization problems. In the following, we break down the benchmark suite according complexity categories and discuss some interesting aspects. The complete list of problems included in the competition benchmark suite, together with detailed problem descriptions, and full benchmark data, is available on the Competition web site (Calimeri et al. 2010).

Polynomial Problems. We classified in this category problems which are known to be solvable in polynomial time in the size of the input data (data complexity). In the Competition suite such problems were usually characterized by the huge size of instance data and, thus, they were a natural test-bench for the impact of memory consumption on performance. It is worth disclaiming that the competition aim was not to compare ASP systems against technologies (database etc.) better tailored to solving this category of problems; nonetheless, several practical real-world applications, which competitors should be able to cope with, fall into this category. Note also that polynomial problems are usually entirely solved by participants' grounder modules, with little or no effort required by subsequent solving stages: indeed, grounders are the technology that mainly underwent assessment while dealing with polynomial problems. There were seven polynomial problems included in the benchmark suite, six of which were selected for the System Track suite.

Four of the above six were specified in a fragment of ASP-Core (i.e., stratified logic programs) with polynomial data complexity; a notable exception was made by the problems StableMarriage and PartnerUnitsPolynomial –which are also known to be solvable in polynomial time (Gusfield and Irving 1989; Falkner et al. 2010)– for which we chose their natural declarative encoding, making usage of disjunction. Note that, in the case of these last two problems, the "combined" ability of grounder and propositional solver modules was tested. The aim was to measure whether, and to what extent, a participant system could be able to converge on a polynomial evaluation strategy when fed with such a natural encoding.

As further remark, note that the polynomial problem Reachability was expressed in terms of a query problem, in which it was asked whether two given nodes were reachable in a given graph: this is a typical setting in which one can test systems on their search space tailoring techniques (such as *magic sets*) (Bancilhon et al. 1986). The polynomial problem participating to the M&S Track only was CompanyControls, given its natural modeling in term of a logic program with aggregates (Faber et al. 2004), these latter not included in the ASP-Core specifications.

NP Problems. We classified in this category NP-complete problems or, more precisely, their corresponding FNP versions. These problems constituted the "core" category, in which to test the attitude of a system in efficiently dealing with problems expressed with the "Guess and Check" methodology (Leone et al. 2006).

[6] The reader can refer to (Papadimitriou 1994) for the definition of basic computational classes herein mentioned.

Among the selected NP problems there were ten puzzle problems, six of which inspired by or taken from planning domains; two classical graph problems; six, both temporal and spatial, resource allocation problems; and, three problems related to applicative and academic settings, namely: WEIGHT-ASSIGNMENTTREE (Garcia-Molina et al. 2000) which was concerned with the problem of finding the best join ordering in a conjunctive query; REVERSEFOLDING which was aimed at mimicking the protein folding problem in a simplified setting (Dovier 2011); and, MULTICONTEXTSYSTEMQUERYING, the unique problem considered in the System Track only, which was a query problem originating from reasoning tasks in Multi-Context Systems (Dao-Tran et al. 2010). Notably, this latter problem had an ASP-Core encoding producing several logic submodules, each of which with independent answer sets. The ability to handle both cross-products of answer sets and early constraint firing efficiently were herein assessed.

Beyond NP/Σ_2^P. The category consisted of problems whose decision version was Σ_2^P-complete. Since a significant fraction of current ASP systems cannot properly handle this class of problems, only two benchmarks were selected, namely STRATEGICCOMPANIES and MINIMALDIAGNOSIS. The former is a traditional Σ_2^P problem coming from (Cadoli et al. 1997), while the latter originates from an application in molecular biology (Gebser et al. 2011). As far as the System Track is concerned, Σ_2^P problems have an ASP encoding making unrestricted usage of disjunction in rule heads.

Beyond NP/*Optimization.* These are all the problems with an explicit formulation given in terms of a cost function with respect to each witness has to be minimized. The above categorization does not imply a given problem stays outside $(F)\Sigma_2^P$, although this has been generally the case for this edition of the competition. The selected problems were of heterogenous provenance, including classic graph problems and sequential optimization planning problems. No benchmark from this category was present in the System Track benchmark suite.

Appendix E System Versions

As described in Section 3 of the original paper, the participants submitted original systems and solution bundles possibly relying on different (sub)systems.

In some cases, systems were provided as custom versions compiled on purpose for the Competition; in some other cases, the executables came from the official release sources, but have been built on the competition machines, and hence might differ from the ones officially distributed. We explicitly report here the exact versions, whenever applicable, if explicitly stated by the participants; it is worth remembering that, for the sake of reproducibility, all systems and solution bundles, together with encodings, instances, scripts, and everything else needed for the actual re-execution of the competition, is available on the competition web site (Calimeri et al. 2010), where more details on systems and teams can be found, as well.

System	**Related Systems/Subsystems**
• `clasp` (v 2.0.0-RC2)	• `Gringo` (v.3.0.3)
• `claspD` (v 1.1.1)	• `Gringo` (v.3.0.3)
• `claspfolio` (v 1.0.0)	• `Gringo` (v.3.0.3)
• IDP (custom)	• `Gringo` (v.3.0.3), MINISATID (v. 2.5.0)
• CMODELS (v 3.81)	• `Gringo` (v.3.0.3), MINISAT v 2.0-beta
• SUP (v 0.4)	• `Gringo` (v.3.0.3)
• LP2GMINISAT, LP2LMINISAT, LP2MINISAT	• `Gringo` (v. 3.0.3), SMODELS (v. 2.34), lpcat (v. 1.18), lp2normal (v. 1.11), igen (v. 1.7), lp2lp2 (v. 1.17), LP2SAT (v 1.15), MINISAT (v. 1.14), interpret (v. 1.7)
• LP2DIFFZ3	• `Gringo` (v. 3.0.3), SMODELS (v. 2.34), lpcat (v. 1.18), l2diff (v. 1.27), z3 (v. 2.11), interpret (v. 1.7)
• SMODELS (v. 2.34)	• `Gringo` (v.3.0.3)

Team	**System/Subsistems exploited**
• `Aclasp`	• `clasp` (custom), `Gringo` (v.3.0.4)
• BPSolver	• B-Prolog (v. 7.1),
• EZCSP	• ezcsp (v. 1.6.20b26), `iClingo` (v. 3.0.3), `clasp`, ASPM, B-Prolog, MKAtoms (v. 2.10), `Gringo` (v. 3.0.3)
• Fast Downward	• Fast Downward (custom)
• IDP	• GIDL v. 1.6.12, MINISATID (v. 2.5.0)
• Potassco	• `clasp` (v 2.0.0-RC2), `claspD` (v 1.1.1), `Gringo` (v.3.0.3), `Clingcon` (v. 0.1.2)

Appendix F Detailed result tables for Section 5 of the original paper

We report here detailed figures of the competition.

All the graphs plot a number representing a number of instances (horizontal axis) against the time (expressed in seconds) needed by each solution bundle to solve them (vertical axis): the slower a line grows, the more efficient the corresponding solution bundle performed. Note that not all the participants solved the same number of instances within the maximum allotted time.

In figure F 3 and F 4 participants are ordered by *Final score* (i.e., $\sum_P S(P)$); for each participant, three rows report for each problem P: (*i*) the *Score*, (*ii*) the *Instance* quota (i.e., $\sum_P S_{solve}(P)$ or $\sum_P S_{opt}(P)$ for optimization problems), (*iii*) the *Time* quota (i.e., $\sum_P S_{time}(P)$).

For each category, the best performance among official participants is reported in bold face. In all tables, an asterisk ('$*$') indicates that the system/team has been disqualified for the corresponding benchmark problem.

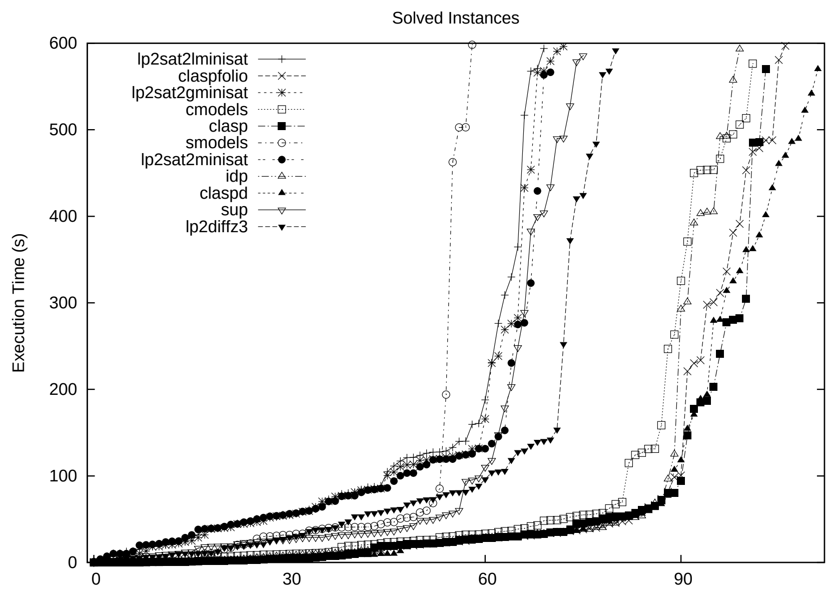


Fig. F 1. *System Track*: Overall Results [Exec. time (y-axis), Solved Instances (x-axis)]

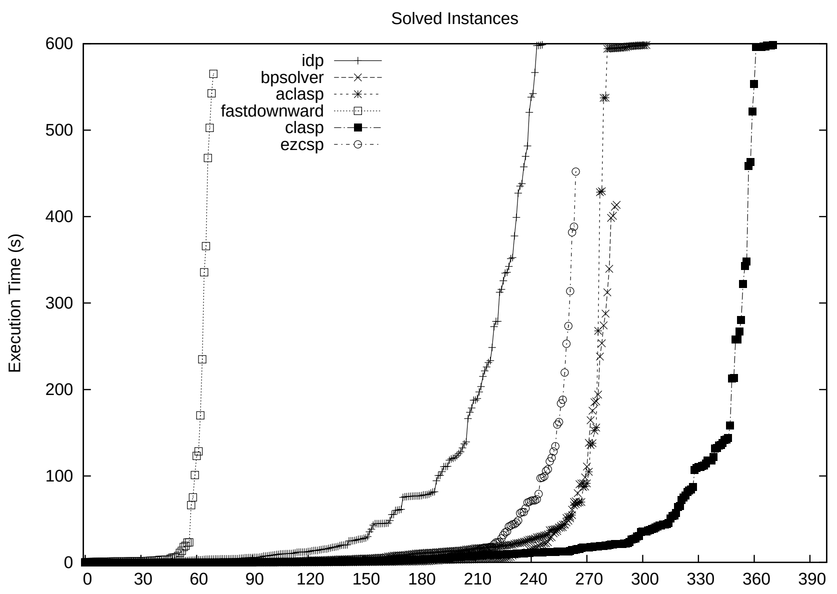


Fig. F 2. *M&S Track*: Overall Results [Exec. time (y-axis), Solved Instances (x-axis)]

			P						NP												Bnd-NP	
System		FINAL SCORE	Reachability	Grammar-Based IE	HydraulicLeaking	HydraulicPlanning	StableMarriage	PartnerUnitsPolynomial	SokobanDecision	KnightTour	DisjunctiveScheduling	PackingProblem	Labyrinth	MCS Querying	Numberlink	HanoiTower	GraphColouring	Solitaire	Weight-AssignmentTree	MazeGeneration	StrategicCompanies	MinimalDiagnosis
claspd	**Score**	**861**	**36**	**72**	**10**	**75**	-	**13**	**68**	**68**	**30**	-	**65**	**75**	**69**	**31**	**19**	**11**	**20**	**96**	**12**	**91**
	Inst	560	25	50	10	50	-	10	45	40	25	-	45	50	40	25	10	10	15	50	10	50
	Time	301	11	22	0	25	-	3	23	28	5	-	20	25	29	6	9	1	5	46	2	41
claspfolio	**Score**	**818**	**35**	**72**	**10**	**74**	**5**	**13**	**66**	**65**	**37**	-	**63**	**75**	**64**	**47**	**55**	**21**	**21**	**95**	-	-
	Inst	535	25	50	10	50	5	10	45	35	25	-	40	50	35	35	40	15	15	50	-	-
	Time	283	10	22	0	24	0	3	21	30	12	-	23	25	29	12	15	6	6	45	-	-
clasp	**Score**	**810**	**36**	**72**	**10**	**75**	**6**	**14**	**78**	**63**	**38**	-	**78**	**75**	**65**	**39**	**23**	**21**	**21**	**96**	-	-
	Inst	520	25	50	10	50	5	10	50	35	25	-	50	50	35	30	15	15	15	50	-	-
	Time	290	11	22	0	25	1	4	28	28	13	-	28	25	30	9	8	6	6	46	-	-
idp	**Score**	**781**	**29**	**71**	**10**	**74**	-	-	**64**	**74**	**38**	-	**52**	**75**	**70**	**65**	**18**	**38**	**8**	**95**	-	-
	Inst	500	20	50	10	50	-	-	45	45	25	-	30	50	40	45	10	25	5	50	-	-
	Time	281	9	21	0	24	-	-	19	29	13	-	22	25	30	20	8	13	3	45	-	-
aclasp	**Score**	**780**	**30**	**72**	-	**69**	**6**	**14**	**75**	**63**	**38**	-	**73**	**75**	**69**	**37**	**25**	**19**	**21**	**94**	-	-
	Inst	510	20	50	-	50	5	10	50	35	25	-	50	50	40	30	15	15	15	50	-	-
	Time	270	10	22	-	19	1	4	25	28	13	-	23	25	29	7	10	4	6	44	-	-
claspmt	**Score**	**766**	**36**	**72**	**10**	*	**6**	**13**	**73**	**90**	**37**	-	**74**	**75**	**73**	**51**	**31**	**31**	*	**94**	-	-
	Inst	485	25	50	10	*	5	10	50	50	25	-	45	50	40	35	20	20	*	50	-	-
	Time	281	11	22	0	*	1	3	23	40	12	-	29	25	33	16	11	11	*	44	-	-
cmodels	**Score**	**766**	**29**	**71**	**10**	**74**	-	-	**67**	**56**	**21**	-	**62**	**75**	**30**	**51**	**29**	**18**	**6**	**95**	-	**72**
	Inst	510	20	50	10	50	-	-	45	30	20	-	45	50	20	35	20	15	5	50	-	45
	Time	256	9	21	0	24	-	-	22	26	1	-	17	25	10	16	9	3	1	45	-	27
lp2diffz3	**Score**	**572**	**35**	**66**	**10**	**67**	-	-	**42**	**55**	-	-	-	**70**	**45**	**47**	**27**	**25**	-	**83**	-	-
	Inst	405	25	50	10	50	-	-	30	35	-	-	-	50	30	35	20	20	-	50	-	-
	Time	167	10	16	0	17	-	-	12	20	-	-	-	20	15	12	7	5	-	33	-	-
sup	**Score**	**541**	**29**	**71**	**10**	**74**	-	**11**	**52**	**40**	**37**	-	**58**	**72**	-	**31**	**16**	**15**	**25**	-	-	-
	Inst	380	20	50	10	50	-	10	35	25	25	-	40	50	-	25	10	10	20	-	-	-
	Time	161	9	21	0	24	-	1	17	15	12	-	18	22	-	6	6	5	5	-	-	-
lp2sat2gmsat	**Score**	**495**	**30**	**66**	**10**	**68**	-	**11**	**36**	**10**	**32**	-	**46**	**71**	**22**	**47**	**17**	**29**	*	-	-	-
	Inst	365	20	50	10	50	-	10	30	5	25	-	35	50	15	35	10	20	*	-	-	-
	Time	130	10	16	0	18	-	1	6	5	7	-	11	21	7	12	7	9	*	-	-	-
lp2diffgz3	**Score**	**495**	**35**	**66**	**10**	**67**	-	-	**40**	**49**	-	-	-	**71**	**32**	**47**	**27**	**25**	-	**26**	-	-
	Inst	360	25	50	10	50	-	-	30	30	-	-	-	50	20	35	20	20	-	20	-	-
	Time	135	10	16	0	17	-	-	10	19	-	-	-	21	12	12	7	5	-	6	-	-
lp2sat2msat	**Score**	**481**	**30**	**66**	**10**	**68**	-	**5**	**39**	-	**32**	-	**52**	**71**	**15**	**47**	**17**	**29**	*	-	-	-
	Inst	355	20	50	10	50	-	5	30	-	25	-	40	50	10	35	10	20	*	-	-	-
	Time	126	10	16	0	18	-	0	9	-	7	-	12	21	5	12	7	9	*	-	-	-
lp2sat2lmsat	**Score**	**472**	**28**	**66**	**10**	**67**	-	-	**35**	-	**32**	-	**53**	**71**	**17**	**47**	**17**	**29**	*	-	-	-
	Inst	350	20	50	10	50	-	-	30	-	25	-	40	50	10	35	10	20	*	-	-	-
	Time	122	8	16	0	17	-	-	5	-	7	-	13	21	7	12	7	9	*	-	-	-
lp2difflz3	**Score**	**471**	**35**	**66**	**10**	**68**	-	-	**41**	**44**	-	-	-	**71**	**37**	**47**	**27**	**25**	-	-	-	-
	Inst	345	25	50	10	50	-	-	30	30	-	-	-	50	25	35	20	20	-	-	-	-
	Time	126	10	16	0	18	-	-	11	14	-	-	-	21	12	12	7	5	-	-	-	-
lp2difflgz3	**Score**	**470**	**35**	**66**	**10**	**67**	-	-	**40**	**44**	-	-	-	**71**	**38**	**47**	**27**	**25**	-	-	-	-
	Inst	345	25	50	10	50	-	-	30	30	-	-	-	50	25	35	20	20	-	-	-	-
	Time	125	10	16	0	17	-	-	10	14	-	-	-	21	13	12	7	5	-	-	-	-
smodels	**Score**	**449**	**28**	**70**	**10**	**72**	-	-	-	**55**	**36**	-	**9**	**53**	**27**	-	-	-	-	**89**	-	-
	Inst	295	20	50	10	50	-	-	-	30	25	-	5	35	20	-	-	-	-	50	-	-
	Time	154	8	20	0	22	-	-	-	25	11	-	4	18	7	-	-	-	-	39	-	-

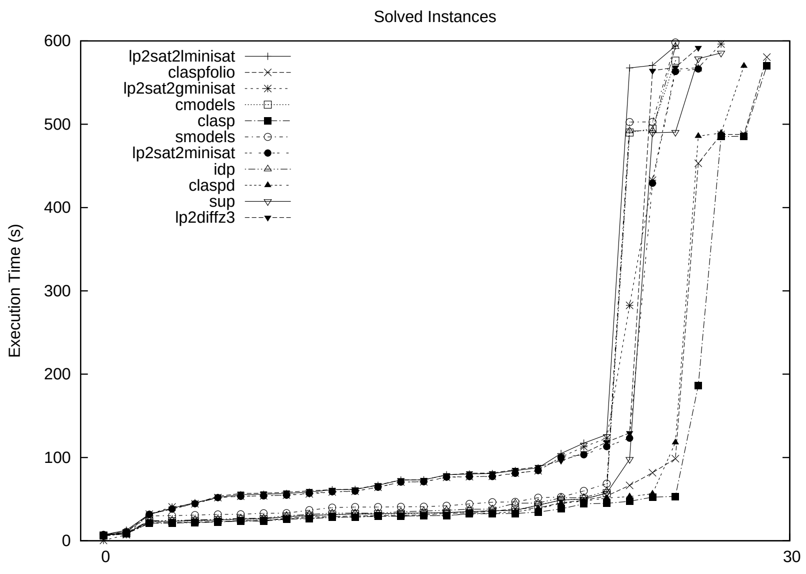
Fig. F 3. System Track - Overall Results

System		FINAL SCORE	P						
			Reachability	CompanyControls	Grammar-Based IE	HydraulicLeaking	HydraulicPlanning	StableMarriage	PartnerUnitsPolynomial
potassco	**Score**	**2413**	**87**	**89**	**80**	**78**	**100**	**25**	**38**
	Inst / Opt	1432	50	50	50	50	50	17	23
	Time	981	37	39	30	28	50	8	15
bpsolver	**Score**	**1970***	**90**	**68**	**96**	**100**	**100**	**5**	**-**
	Inst / Opt	1111*	50	50	50	50	50	3	-
	Time	859*	40	18	46	50	50	2	-
aclasp	**Score**	**1935**	**-**	**84**	**80**	**76**	**100**	**25**	**39**
	Inst / Opt	1140	-	50	50	50	50	17	23
	Time	795	-	34	30	26	50	8	16
ezcsp	**Score**	**1760**	**-**	**-**	**-**	**99**	**99**	**87**	**35**
	Inst / Opt	993	-	-	-	50	50	50	23
	Time	767	-	-	-	49	49	37	12
idp	**Score**	**1436**	**76**	**-**	**-**	**-**	**-**	**72**	**27**
	Inst / Opt	918	50	-	-	-	-	50	17
	Time	518	26	-	-	-	-	22	10
fastdownward	**Score**	**432***	**-**	**-**	**-**	**87**	**94**	**-**	**-**
	Inst / Opt	254*	-	-	-	50	50	-	-
	Time	178*	-	-	-	37	44	-	-

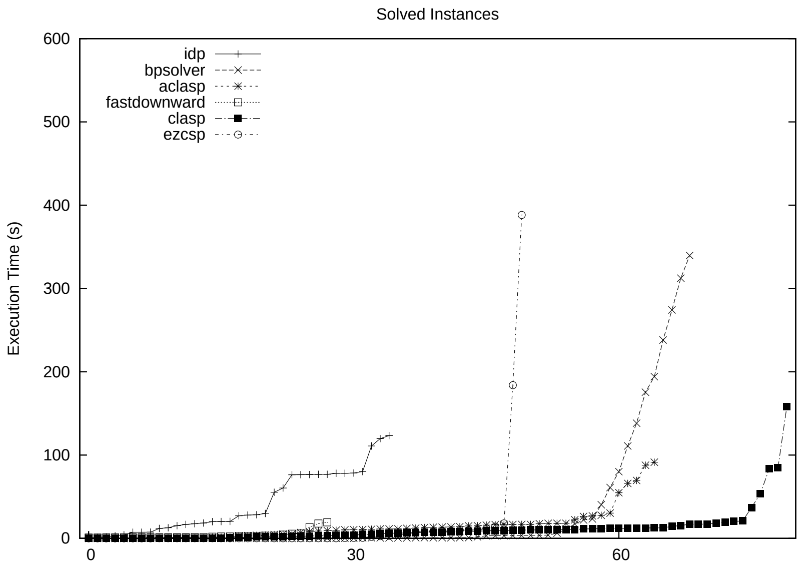
System		NP																		
		GeneralizedSlitherlink	FastfoodOptimalityCheck	SokobanDecision	KnightTour	DisjunctiveScheduling	PackingProblem	Labyrinth	Numberlink	ReverseFolding	HanoiTower	MagicSquareSets	AirportPickup	GraphColouring	Solitaire	PartnerUnits	Weight-AssignmentTree	MazeGeneration	IncrementalScheduling	Tangram
potassco	**Score**	**97**	**97**	**85**	**97**	**67**	**81**	**71**	**72**	**77**	**81**	**98**	**-**	**51**	**73**	**79**	**56**	**97**	**85**	**99**
	Inst / Opt	50	50	50	50	50	50	47	37	47	50	50	-	37	40	50	43	50	47	50
	Time	47	47	35	47	17	31	24	35	30	31	48	-	14	33	29	13	47	38	49
bpsolver	**Score**	*****	**92**	**60**	**97**	**92**	**82**	**72**	**73**	**-**	**94***	**100**	**93**	**15**	**28**	**-**	**97**	**49**	**76**	**98**
	Inst / Opt	*	50	30	50	50	43	40	37	-	47*	50	50	10	20	-	50	37	40	50
	Time	*	42	30	47	42	39	32	36	-	47*	50	43	5	8	-	47	12	36	48
aclasp	**Score**	**96**	**97**	**83**	**97**	**-**	**83**	**68**	**72**	**63**	**77**	**98**	**-**	**28**	**73**	**77**	**-**	**95**	**-**	**99**
	Inst / Opt	50	50	50	50	-	50	43	37	40	50	50	-	20	40	50	-	50	-	50
	Time	46	47	33	47	-	33	25	35	23	27	48	-	8	33	27	-	45	-	49
ezcsp	**Score**	**94**	**97**	**81**	**75**	**26**	**99**	**69**	**70**	**86**	**27**	**99**	**81**	**23**	**50**	**65**	**96**	**96**	**77**	**95**
	Inst / Opt	50	50	50	40	13	50	43	37	47	23	50	50	13	30	50	50	50	40	50
	Time	44	47	31	35	13	49	26	33	39	4	49	31	10	20	15	46	46	37	45
idp	**Score**	**91**	**76**	**69**	**95**	**15**	**17**	**42**	**99**	**-**	**74**	**93**	**-**	**66**	**71**	**33**	**83**	**73**	**30**	**94**
	Inst / Opt	50	50	43	50	13	13	27	50	-	50	47	-	47	40	30	50	50	20	50
	Time	41	26	26	45	2	4	15	49	-	24	46	-	19	31	3	33	23	10	44
fastdownward	**Score**	**-**	**-**	**65***	**5**	**-**	**-**	**-**	**-**	**-**	**27**	**-**	**10**	**-**	**63**	**-**	**-**	**-**	**-**	**-**
	Inst / Opt	-	-	37*	5	-	-	-	-	-	23	-	7	-	35	-	-	-	-	-
	Time	-	-	28*	0	-	-	-	-	-	4	-	3	-	28	-	-	-	-	-

System		Bnd-NP		Opt					
		StrategicCompanies	MinimalDiagnosis	CrossingMinimization	CompanyControlsOptimize	FastfoodOptimization	SokobanOptimization	Tomography	MaximalClique
potassco	**Score**	**11**	**99**	**37**	**87**	**83**	**87**	**13**	**36**
	Inst / Opt	10	50	35	50	50	50	13	36
	Time	1	49	2	37	33	37	0	0
bpsolver	**Score**	**-**	**86**	**13**	**-**	**48**	**70**	**39**	**37**
	Inst / Opt	-	43	13	-	35	37	39	37
	Time	-	43	0	-	13	33	0	0
aclasp	**Score**	**-**	**-**	**16**	**87**	**83**	**85**	**18**	**36**
	Inst / Opt	-	-	16	50	50	50	18	36
	Time	-	-	0	37	33	35	0	0
ezcsp	**Score**	**-**	**-**	**-**	**-**	**-**	**-**	**34**	**-**
	Inst / Opt	-	-	-	-	-	-	34	-
	Time	-	-	-	-	-	-	0	-
idp	**Score**	**-**	**-**	**13**	**-**	**64**	**21**	**13**	**29**
	Inst / Opt	-	-	13	-	48	18	13	29
	Time	-	-	0	-	16	3	0	0
fastdownward	**Score**	**-**	**-**	**-**	**-**	**-**	**81**	**-**	**-**
	Inst / Opt	-	-	-	-	-	47	-	-
	Time	-	-	-	-	-	34	-	-

Fig. F 4. M&S Track - Overall Results

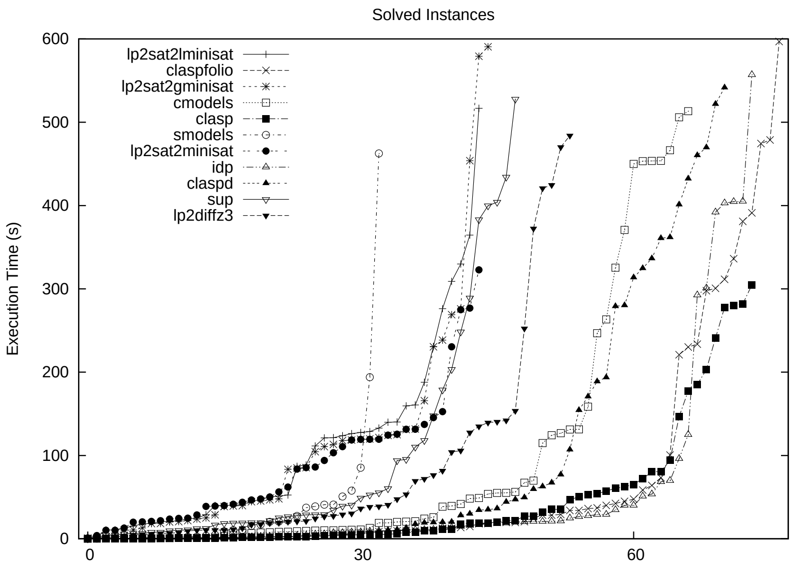


(a) System Track P

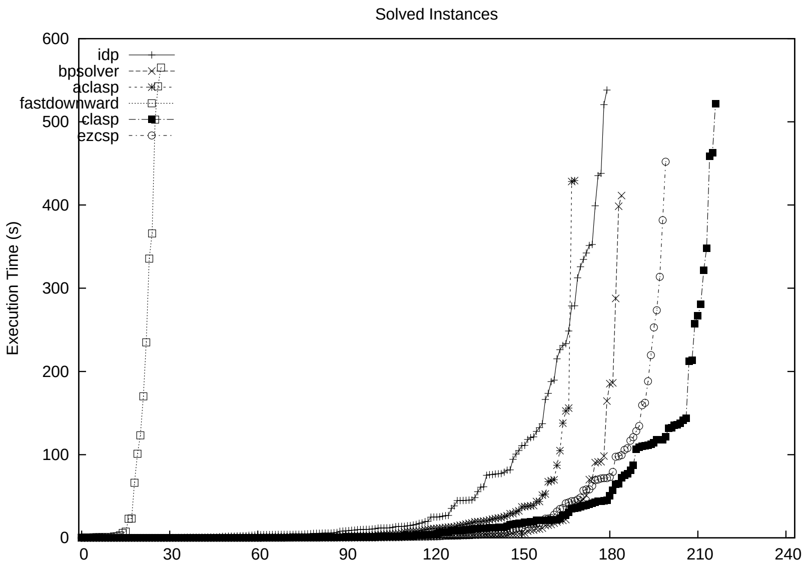


(b) Team Track P

Fig. F 5. Results in Detail: Execution time (y-axis), Solved Instances (x-axis).

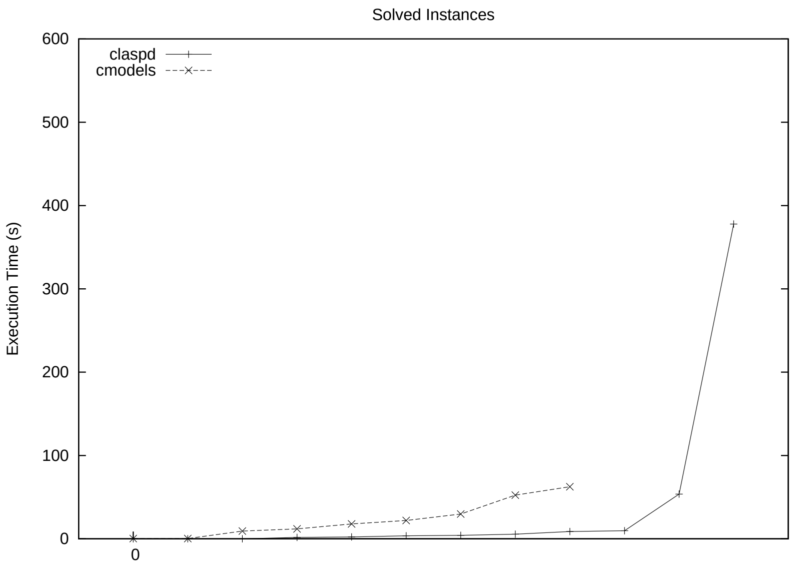


(a) System Track *NP*

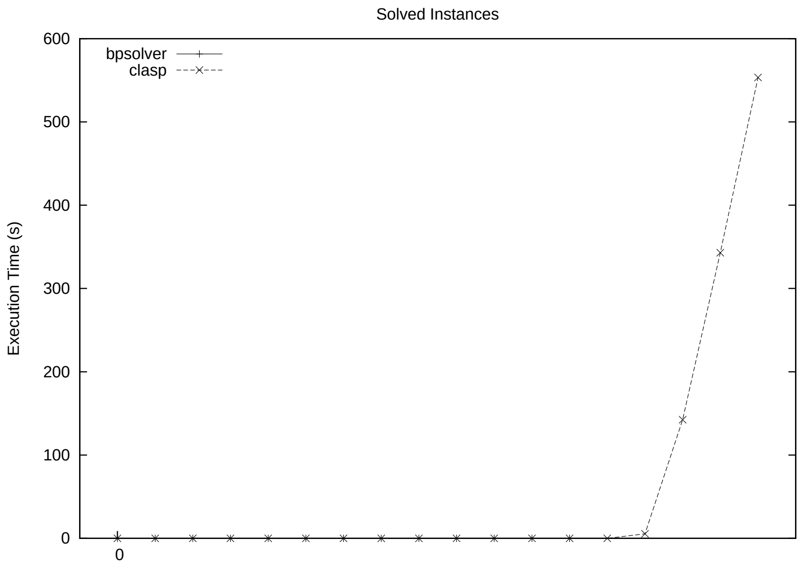


(b) Team Track *NP*

Fig. F 6. Results in Detail: Execution time (y-axis), Solved Instances (x-axis).

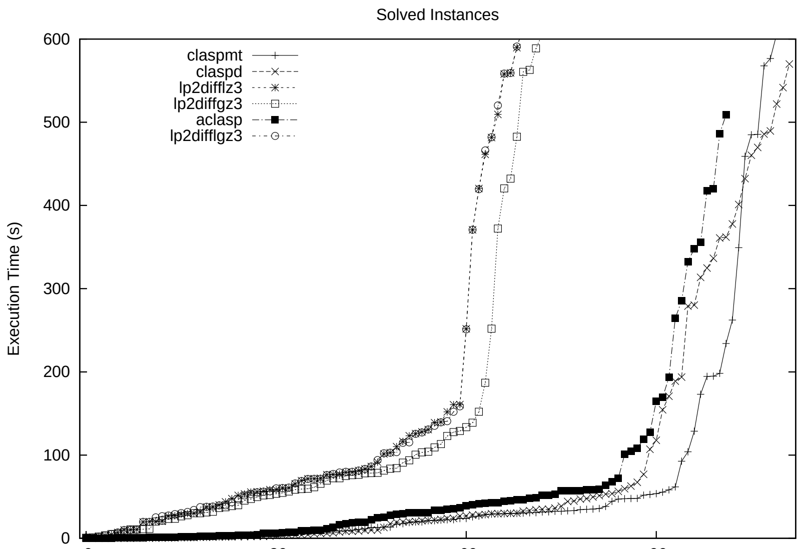


(a) System Track Beyond *NP*

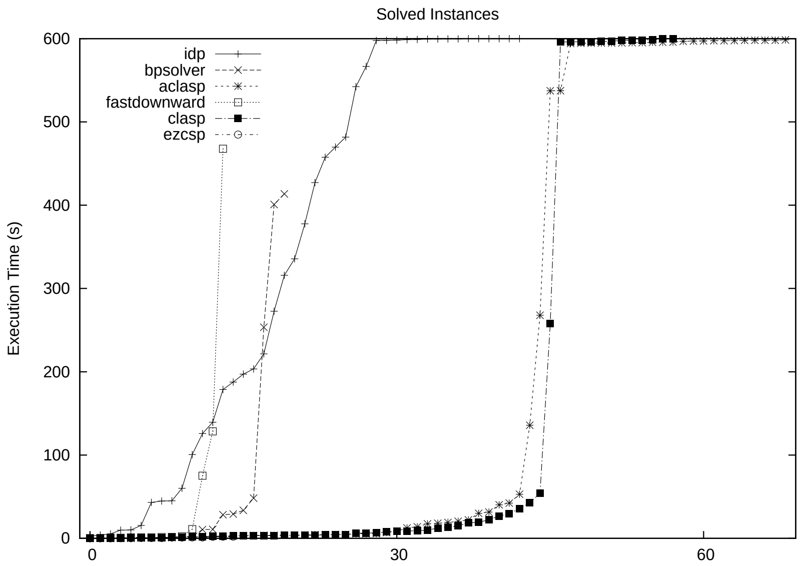


(b) Team Track Beyond *NP*

Fig. F 7. Results in Detail: Execution time (y-axis), Solved Instances (x-axis).



(a) System Track Non-participants



(b) Team Track Optimization

Fig. F 8. Results in Detail: Execution time (y-axis), Solved Instances (x-axis).

References

Bancilhon, F., Maier, D., Sagiv, Y., and Ullman, J. D. 1986. Magic Sets and Other Strange Ways to Implement Logic Programs. In *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS 1986).* Cambridge, Massachusetts, 1–15.

Cadoli, M., Eiter, T., and Gottlob, G. 1997. Default Logic as a Query Language. *IEEE Transactions on Knowledge and Data Engineering 9,* 3 (May/June), 448–463.

Calimeri, F., Ianni, G., and Ricca, F. 2011. Third ASP Competition, File and language formats. `http://www.mat.unical.it/aspcomp2011/files/LanguageSpecifications.pdf`.

Calimeri, F., Ianni, G., Ricca, F., and The Università della Calabria Organizing Committee. 2010. The Third Answer Set Programming Competition homepage. `http://www.mat.unical.it/aspcomp2011/`.

Dao-Tran, M., Eiter, T., Fink, M., and Krennwallner, T. 2010. Distributed Nonmonotonic Multi-Context Systems. In *12th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2010), Toronto, Canada, May 9-13, 2010*, F. Lin and U. Sattler, Eds. AAAI Press, 60–70.

Dovier, A. 2011. Recent constraint/logic programming based advances in the solution of the protein folding problem. *Intelligenza Artificiale 5,* 1, 113–117.

Faber, W., Leone, N., and Pfeifer, G. 2004. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of the 9th European Conference on Artificial Intelligence (JELIA 2004)*, J. J. Alferes and J. Leite, Eds. Lecture Notes in AI (LNAI), vol. 3229. Springer Verlag, 200–212.

Falkner, A., Haselböck, A., and Schenner, G. 2010. Modeling Technical Product Configuration Problems. In *Proceedings of ECAI 2010 Workshop on Configuration.* Lisbon, Portugal, 40–46.

Garcia-Molina, H., Ullman, J. D., and Widom, J. 2000. *Database System Implementation.* Prentice Hall.

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

Gelfond, M. and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing 9*, 365–385.

Gusfield, D. and Irving, R. W. 1989. *The stable marriage problem: structure and algorithms.* MIT Press, Cambridge, MA, USA.

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 (July), 499–562.

Papadimitriou, C. H. 1994. *Computational Complexity.* Addison-Wesley.