Unsatisfiable Core Shrinking for Anytime Answer Set Optimization*

Mario Alviano[1] **and Carmine Dodaro**[2]
[1]DEMACS, University of Calabria, Italy
[2]DIBRIS, University of Genova, Italy
alviano@mat.unical.it, dodaro@dibris.unige.it

Abstract

Efficient algorithms for the computation of optimum stable models are based on unsatisfiable core analysis. However, these algorithms essentially run to completion, providing few or even no suboptimal stable models. This drawback can be circumvented by shrinking unsatisfiable cores. Interestingly, the resulting anytime algorithm can solve more instances than the original algorithm.

1 Introduction

In answer set programming (ASP), programs are associated with *stable models* [Gelfond and Lifschitz, 1991; Niemelä, 1999; Marek *et al.*, 2008; Lifschitz, 2008; Eiter *et al.*, 2009; Brewka *et al.*, 2011], i.e., classical models satisfying a stability condition: only necessary information is included in a model of the input program under the assumptions provided by the model itself for the *unknown knowledge* in the program, where unknown knowledge is encoded by means of *default negation*. Reasoning in presence of unknown knowledge is common for rational agents acting in the real world. It is also common that real world agents cannot meet all their desiderata, and therefore ASP programs may come with *soft literals* for representing numerical preferences over jointly incompatible conditions. Stable models are therefore associated with a cost given by the number of the unsatisfied soft literals, so that stable models of minimum cost are *preferred*.

It is important here to stress the meaning of the word preferred: any stable model describes a plausible scenario for the knowledge represented in the input program, even if it may be only an admissible solution of non optimum cost. In fact, many rational agents would still accept suboptimal solutions, possibly with an estimate on the maximum distance to the optimum cost. This flexibility is also justified by the intrinsic complexity of the problem: the computation of an optimum stable model requires in general at least linearly many calls to a Σ_2^P oracle [Buccafurri *et al.*, 2000], and it is therefore practically unfeasible for the hardest instances.

According to the above observations, a good algorithm for answer set optimization should produce better and better stable models during the computation of an optimum stable model. Algorithms having this property are called *anytime* in the literature [Alviano *et al.*, 2014; Bliem *et al.*, 2016]. However, the most efficient algorithms are not anytime by themselves: they are based on *unsatisfiable core* analysis [Alviano *et al.*, 2015b], which means that they try to satisfy all soft literals, possibly replacing those in the input program with less restricting constraints until an optimum stable model is found.

Anytime variants of these algorithms are obtained thanks to following simple observation [Alviano and Dodaro, 2016b]: Unsatisfiable cores are often non-minimal, and their sizes can be significantly reduced by a few additional oracle calls, where each call may either return a smaller core, or a stable model possibly improving the current overestimate. Within this respect, we implemented two strategies, referred to as *linear* and *reiterated progression based shrinking*.

Interestingly, the overhead introduced by the additional oracle calls is often mitigated by the performance gain obtained thanks to the smaller unsatisfiable cores that the algorithm has to analyze. Indeed, we provide empirical evidence that often the running time of our core based algorithm sensibly decreases when core shrinking is performed (Section 4). The advantage of introducing our strategy for core shrinking is also confirmed by a comparison with CLASP [Gebser *et al.*, 2015a]: even if our solver, WASP [Alviano *et al.*, 2015a; Alviano and Dodaro, 2016a], is in general slower than CLASP at completing stable model searches, its performance is sufficiently improved by core shrinking that the two solvers are almost on par in terms of solved instances, with the crucial difference that WASP provides both overestimates and underestimates during the computation, while ones or the others are produced by CLASP only after running to completion.

2 Background

Let $\mathscr{A}$ be a set of (propositional) *atoms* comprising $\bot$. A *literal* is an atom p preceded by zero or more occurrences of the *default negation* symbol $\sim$. A *rule* r is an implication $H(r) \leftarrow B(r)$, where $H(r)$ is a disjunction of atoms, and $B(r)$ is a conjunction of literals. $H(r)$ and $B(r)$ are called *head* and

*This work is based on a paper presented at the 32nd International Conference on Logic Programming (ICLP 2016) [Alviano and Dodaro, 2016], and was partially supported by the National Group for Scientific Computation (GNCS-INDAM), and by project "Smarter Solutions in the Big Data World (S2BDW)" funded by the Italian Ministry for Economic development (MISE), PON "Imprese e competitività" 2014-2020.

body of r, and abusing of notation also denote the sets of their elements. If $H(r) \subseteq \{\bot\}$, then r is called *integrity constraint*. A *program* Π is a set of rules. Let $At(\Pi)$ denote the set of atoms occurring in Π.

An *interpretation* I is a set of atoms not containing $\bot$. Relation $\models$ is inductively defined as follows: for $p \in \mathscr{A}$, $I \models p$ if $p \in I$; $I \models {\sim}\ell$ if $I \not\models \ell$; for a rule r, $I \models B(r)$ if $I \models \ell$ for all $\ell \in B(r)$, and $I \models r$ if $I \cap H(r) \neq \emptyset$ whenever $I \models B(r)$; for a program Π, $I \models \Pi$ if $I \models r$ for all $r \in \Pi$. I is a *model* of a literal, rule, or program π if $I \models \pi$.

The *reduct* Π^I of a program Π with respect to an interpretation I is obtained from Π by removing any rule r such that $I \not\models B(r)$ is removed, and then by removing any negated literal. An interpretation I is a *stable model* of a program Π if $I \models \Pi$, and there is no $J \subset I$ such that $J \models \Pi^I$. Let $SM(\Pi)$ denote the set of stable models of Π. A program Π is *coherent* if $SM(\Pi) \neq \emptyset$; otherwise, Π is *incoherent*.

In order to simplify the presentation, a program Π may include *count constraints* of the form $\text{COUNT}\{\ell_1, \ldots, \ell_n\} \geq k$, where $\ell_1, \ldots, \ell_n$ $(n \geq 0)$ are literals, and $k \geq 0$, to enforce $|\{i \in [1..n] \mid I \models \ell_i\}| \geq k$ for all $I \in SM(\Pi)$.

For a set S of literals, called *soft*, the *cost* of an interpretation I is $S(I) := |\{\ell \in S \mid I \not\models \ell\}|$, that is, the number of false soft literals. I is an *optimum stable model* of a program Π with respect to S if $I \in SM(\Pi)$, and there is no $J \in SM(\Pi)$ such that $S(J) < S(I)$. Let $OSM(\Pi, S)$ denote the set of optimum stable models of Π with respect to S. *Optimum stable model search* is the following computational problem: Given a (coherent) program Π and a set of soft literals S, compute an optimum stable model $I^* \in OSM(\Pi, S)$.

Example 1 *Let Π_1 be the following program:*

$$a \leftarrow {\sim}{\sim}a \quad b \vee c \leftarrow a \quad b \vee d \leftarrow a \quad b \leftarrow {\sim}d \quad c \leftarrow {\sim}a$$

Its stable models are $I_1 = \{b, c\}$, $I_2 = \{a, b\}$ and $I_3 = \{a, c, d\}$, and the associated reducts are the following:

$$\begin{array}{ll} \Pi_1^{I_1}: & b \leftarrow \quad c \leftarrow \\ \Pi_1^{I_2}: & a \leftarrow \quad b \vee c \leftarrow a \quad b \vee d \leftarrow a \quad b \leftarrow \\ \Pi_1^{I_3}: & a \leftarrow \quad b \vee c \leftarrow a \quad b \vee d \leftarrow a \end{array}$$

If $S = \{{\sim}a, {\sim}b, {\sim}c, {\sim}d\}$ is a set of soft literals, the associated costs are $S(I_1) = S(I_2) = 2$, and $S(I_3) = 3$. Hence, $OSM(\Pi_1, S) = \{I_1, I_2\}$. ■

3 Optimum Stable Model Search via Unsatisfiable Core Analysis

Modern ASP solvers accept as input a set L of literals, called *assumptions*, in addition to the usual logic program Π, and return a stable model I of Π such that $L \subseteq I$, if it exists; otherwise, they return a set $C \subseteq L$ such that $\Pi \cup \{\bot \leftarrow {\sim}p \mid p \in C\}$ is incoherent, which is called *unsatisfiable core*.

Example 2 *Consider program Π_1 from Example 1. If $S = \{{\sim}a, {\sim}b, {\sim}c, {\sim}d\}$ is the set of assumptions, the unsatisfiable cores are $\{{\sim}a, {\sim}b\}$, $\{{\sim}a, {\sim}c\}$, $\{{\sim}b, {\sim}c\}$, $\{{\sim}b, {\sim}d\}$, and their supersets.* ■

The algorithm presented in this paper is ONE, reported as Algorithm 1 (lines 4–10 will be *injected* later to shrink unsatisfiable cores). A stable model containing all soft literals is searched (line 2). If found, it is an optimum stable model. Otherwise, an unsatisfiable core $\{p_0, \ldots, p_n\}$ is returned; since at least one of $p_0, \ldots, p_n$ must be false in any optimum stable model, the lower bound is increased by one, and the problem is relaxed so that the next call to function *solve* has to search for a stable model satisfying at least n literals among $p_0, \ldots, p_n$. *Symmetry breakers* of the form $\bot \leftarrow s_i, {\sim}s_{i+1}$ are also added to Π, so that s_i is true if and only if at least $n - i + 1$ literals among $p_0, \ldots, p_n$ are true.

Example 3 *Consider program Π_1 and soft literals $S = \{{\sim}a, {\sim}b, {\sim}c, {\sim}d\}$ from Example 1. A stable model for the program Π_1 and assumptions S is searched, and an unsatisfiable core is returned. Assume that the returned unsatisfiable core is S itself. The lower bound lb is set to 1, the set S is now equal to $\{s_1, s_2, s_3\}$ and Π_1 is extended with the following rules:*

$$\begin{array}{lll} s_1 \leftarrow {\sim}{\sim}s_1 & s_2 \leftarrow {\sim}{\sim}s_2 & s_3 \leftarrow {\sim}{\sim}s_3 \\ \bot \leftarrow s_1, {\sim}s_2 & \bot \leftarrow s_2, {\sim}s_3 & \\ \multicolumn{3}{l}{\text{COUNT}\{{\sim}a, {\sim}b, {\sim}c, {\sim}d, {\sim}s_1, {\sim}s_2, {\sim}s_3\} \geq 3.} \end{array}$$

A stable model for the assumptions $\{s_1, s_2, s_3\}$ is searched and an unsatisfiable core, say $\{s_1\}$, is returned. The lower bound lb is set to 2, the set S is now equal to $\{s_2, s_3\}$. (Note that the program is extended with the count constraint $\text{COUNT}\{s_1\} \geq 0$, which however is trivially satisfied.) A stable model for the assumptions $\{s_2, s_3\}$ is searched and the answer set $I_1' = \{b, c, s_2, s_3\}$ is found. Thus, the algorithm terminates returning $I_1' \cap \{a, b, c, d\} = \{b, c\} = I_1$. ■

The analyzed unsatisfiable cores significantly influence the execution of the algorithm, as the set of assumptions and the introduced rules are different for different unsatisfiable cores.

Example 4 *Suppose that the first unsatisfiable core returned by function solve for Π_1 and S from Example 1 is $\{{\sim}a, {\sim}b, {\sim}c\}$. Set S becomes $\{{\sim}b, s_1, s_2\}$ and Π_1 is extended with the following rules:*

$$\begin{array}{l} s_1 \leftarrow {\sim}{\sim}s_1 \quad s_2 \leftarrow {\sim}{\sim}s_2 \quad \bot \leftarrow s_1, {\sim}s_2 \\ \text{COUNT}\{{\sim}a, {\sim}b, {\sim}c, {\sim}s_1, {\sim}s_2\} \geq 2. \end{array}$$

The next unsatisfiable core may be $\{{\sim}d, s_1\}$; therefore, S becomes $\{s_2, s_3\}$, and Π_1 is extended with the following rules:

$$s_3 \leftarrow {\sim}{\sim}s_3 \quad \text{COUNT}\{{\sim}d, s_1, {\sim}s_3\} \geq 1.$$

At this point a stable model, say $I_1' = \{b, c, s_2, s_3\}$, is found, and $I_1' \cap \{a, b, c, d\} = \{b, c\} = I_1$ is returned. ■

Note that the algorithm described in this section is completely silent, as it essentially runs to completion without printing any suboptimal stable models. The goal of the next section is to circumvent such a drawback.

3.1 Unsatisfiable Core Shrinking

Unsatisfiable cores returned by function *solve* are not subset minimal in general. The non-minimality of the unsatisfiable core is justified both theoretically and practically: linearly many coherence checks are required in general to verify

Algorithm 1: Unsatisfiable Core Analysis with ONE

Input : A coherent program Π, and a nonempty set of soft literals S.
Output: An optimum stable model $I^* \in OSM(\Pi, S)$.

1 $lb := 0$; $ub := \infty$; $V := At(\Pi)$; `// init bounds and visible atoms`
2 $(res, I, C) := solve(\Pi, \{p \in S\})$;
3 **if** *res* **is** *COHERENT* **then** $I^* := I \cap V$; **return** I^*;
11 Let C be $\{p_0, \ldots, p_n\}$ (for some $n \geq 0$), and $s_1, \ldots, s_n$ be fresh atoms;
12 $\Pi := \Pi \cup \{s_i \leftarrow \sim\sim s_i \mid i \in [1..n]\} \cup \{\bot \leftarrow s_i, \sim s_{i+1} \mid i \in [1..n-1]\} \cup \{\text{COUNT}\{p_0, \ldots, p_n, \sim s_1, \ldots, \sim s_n\} \geq n\}$;
13 $lb := lb + 1$; $S := (S \setminus C) \cup \{s_1, \ldots, s_n\}$; **goto** 2; `// try to solve the relaxed problem`

Algorithm 2: Unsatisfiable Core Shrinking with Reiterated Progression

4 $m := -1$; $pr := 1$;
5 Let C be $\{p_0, \ldots, p_n\}$ (for some $n \geq 0$);
6 $(res, I, C') := solve\_with\_budget(\Pi, \{p_i \mid i \in [0..m + pr]\})$;
7 **if** *res* **is** *INCOHERENT* **then** $C := C'$; `// smaller core found`
8 **if** *res* **is** *COHERENT* **and** $lb + S(I) < ub$ **then** $I^* := I \cap V$; $ub := lb + S(I)$;
9 **if** $m + 2 \cdot pr \geq |C| - 1$ **then** $m := m + pr$; $pr := 1/2$; `// reiterate progression`
10 **if** $m + 2 \cdot pr < |C| - 1$ **then** $pr := 2 \cdot pr$; **goto** 5; `// increase progression`

the minimality of an unsatisfiable core, hence giving a Δ_3^P-complete problem; on the other hand, extracting an unsatisfiable core after a stable model search failure is quite easy and usually implemented by identifying the assumptions involved in the refutation. The non minimality of the analyzed unsatisfiable cores may affect negatively the performance of subsequent calls to function *solve* due to aggregation over large sets. However, it also gives an opportunity to improve Algorithm 1: the size of unsatisfiable cores can be reduced by performing a few stable model searches within a given budget on the running time. In more detail, Algorithm 2 is injected in Algorithm 1. It implements a progression search in the unsatisfiable core $\{p_0, \ldots, p_n\}$: the size of the assumptions passed to function *solve_with_budget* is doubled at each call (line 10), and the progression is reiterated when all assumptions are covered (line 9). If *solve_with_budget* terminates within the given budget, it either returns a smaller unsatisfiable core (line 7), or a stable model that possibly improves the current upper bound (line 8).

Example 5 *Consider again the program from Example 3 and the unsatisfiable core $\{\sim a, \sim b, \sim c, \sim d\}$ returned after the first call to function solve. The shrinking process searches a stable model with assumption $\{\sim a\}$, and $I_1 = \{b, c\}$ may be found within the allotted budget. In any case, a stable model satisfying the assumptions $\{\sim a, \sim b\}$ is searched, and the unsatisfiable core $\{\sim a, \sim b\}$ may be returned if the budget is sufficient. Otherwise, the progression is reiterated, and one more soft literal is added to the assumptions. Hence, $\{\sim a, \sim b, \sim c\}$ may be returned as an unsatisfiable core if the budget is sufficient. Otherwise, the original unsatisfiable core is processed.* ■

As an alternative, the shrinking procedure reported in Algorithm 2 can be modified as follows: variable *pr* is not doubled in line 10, but instead it is incremented by one, i.e., $pr := pr + 1$. The resulting procedure is called *linear based shrinking*. For unsatisfiable cores of size 4 or smaller, as those considered in Example 5, the two shrinking procedures coincide, while in general linear based shrinking performs more stable model searches.

4 Implementation and Experiment

Algorithm ONE [Alviano *et al.*, 2015c] has been implemented in WASP, an ASP solver based on completion [Alviano and Dodaro, 2016c] also supporting, among other algorithms, *linear search sat-unsat* (LINSU). Within LINSU, a first stable model is searched to obtain an upper bound of the optimum cost, and subsequent searches are constrained to improve the current upper bound, until an incoherence arises. The implementation of ONE optionally includes the two shrinking procedures described in Section 3.1, so that both underestimates and overestimates can be produced by WASP in any case, weighted or unweighted. Currently, the time budget of function *solve_with_budget* is fixed to 10 seconds, but the architecture of WASP can easily accommodate alternative options, such as a budget proportional to the time required to find the unsatisfiable core to be shrank.

WASP also implements *disjoint cores analysis*, which is essentially a preliminary step where only soft literals in the input are passed as assumptions to function *solve*, while new

Table 1: Number of solved instances within a given error estimation (140 testcases).

	WASP			CLASP+ best *lb* by WASP	
$\varepsilon(ub, lb)$	ONE	LSHR	PSHR	LINSU	OLL
0.00%	84	88	90	77	84
$\leq$ 6.25%	86	95	94	90	87
$\leq$ 12.50%	86	101	97	96	88
$\leq$ 25.00%	92	105	103	99	99
$\leq$ 50.00%	102	105	105	99	101
$\leq$ 100.00%	104	105	105	107	105

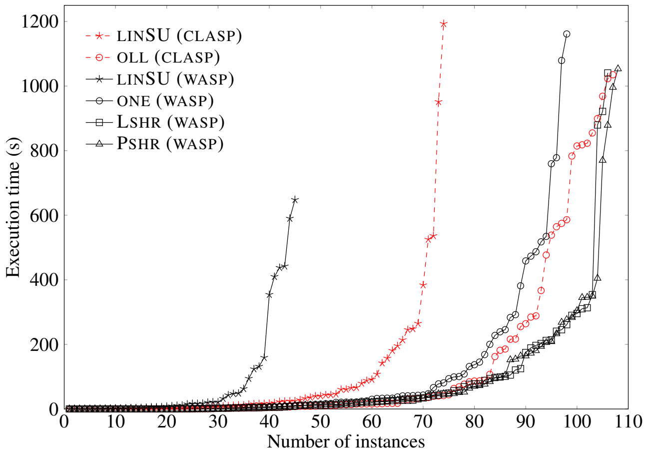

Figure 1: Solved instances within a bound on running time.

soft literals introduced by the analysis of detected cores are temporarily ignored. Disjoint cores analysis terminates with the detection of a stable model, and after that algorithm ONE is run not distinguishing between initial and new soft literals.

In order to assess empirically the impact of these shrinking procedures to the performance of WASP, benchmarks (with soft literals) from the ASP Competition 2015 [Gebser *et al.*, 2015b] were considered, namely Abstract Dialectal Framework, Still Life, Crossing Minimization, Max Clique, MaxSAT, Steiner Tree, System Synthesis, Valves Location, and Video Streaming. Moreover, WASP was also compared with CLASP [Gebser *et al.*, 2015a], which implements linear search sat-unsat (strategy `bb,1`) and OLL (strategy `usc,1` with disjoint cores analysis; [Andres *et al.*, 2012; Morgado *et al.*, 2014]), a core based algorithm that inspired the definition of ONE. Both solvers were tested with the disjoint cores analysis. The experiments were run on an Intel Xeon 2.4 GHz with 16 GB of memory, and time and memory were limited to 20 minutes and 15 GB, respectively.

An overview of the obtained results is given in Figure 1. As a first comment, the fact that CLASP is in general faster than WASP to complete stable model searches is confirmed by comparing the performance of the two solvers running linear search sat-unsat (CLASP solves 29 instances more than WASP) or the core based algorithms (difference of 9 instances). This gap is completely filled by adding the shrinking procedures. Concerning the average execution time of the tested algorithms, the graph highlights that core based algorithms are faster than linear search sat-unsat in more testcases. Moreover, and more important, the addition of core shrinking does not add overhead to WASP. The main reason for this performance improvement is that shrinking a core often implies that subsequently found unsatisfiable cores are smaller: The cumulative number of literals in the analyzed cores is reduced by at least 68% when shrinking is performed (excluding Steiner Tree, System Synthesis and Video Streaming, for which WASP found few unsatisfiable cores). The budget is reached at least once in each problem, and often no more than 2 times, with a peak of 20–25 times on average for instances of Max Clique and Still Life.

Another advantage of unsatisfiable core shrinking is that better and better stable models are possibly discovered while computing an optimum stable model. In order to measure the impact of our strategies within this respect, let us define the *estimate error* ε of the last stable model produced by Algorithm 1 as follows:

$$\varepsilon(ub,lb) := \begin{cases} \frac{ub-lb}{lb} & \text{if } ub \neq \infty \text{ and } lb \neq 0; \\ \infty & \text{if } ub = \infty, \text{ or both } ub \neq 0 \text{ and } lb = 0; \\ 0 & \text{if } ub = lb = 0. \end{cases}$$

Hence, the cost associated with the stable model returned by Algorithm 1 is at most $\varepsilon(ub,lb)$ times greater than the cost of an optimum stable model. Such a measure is not applicable to instances of Abstract Dialectical Framework and System Synthesis because of technicality not discussed in this paper.

Table 1 reports the number of instances for which WASP produced a stable model within a given error estimate. In particular, the first row shows the number of instances for which an optimum stable model was computed (error estimate is 0). The last row, instead, shows the number of instances solved with error estimate bounded by 1, and smaller values for the error estimate are considered in the intermediate rows. It is interesting to observe that the stable model produced after the analysis of all disjoint cores is already sufficient to obtain an error estimate bounded by 100% for many tested instances. However, many of these stable models have an error estimate greater than 25%. In this case, adding core shrinking leads to better results.

For the sake of completeness, also CLASP is included in Table 1. However, since CLASP does not print any lower bound, the best value for *lb* produced by WASP is combined with the upper bounds given by CLASP running LINSU and OLL. If an error estimate of 100% is acceptable, then the number of stable models produced by CLASP is aligned with WASP, or even better. However, when the error estimate must be less or equal than 50%, the combination of disjoint cores analysis and core shrinking implemented by WASP leads to better results in this benchmark.

5 Conclusion

The combination of ASP programs and soft literals is important to ease the modeling of optimization problems. However, the computation of optimum stable models is often very hard, and suboptimal stable models may be the only affordable solutions in some cases. Despite that fact, efficient algorithms based on unsatisfiable core analysis are not anytime. A concrete strategy to turn them into anytime algorithms is given by a shrinking procedure applied to unsatisfiable cores before their analysis: better and better stable models are produced, and eventually a performance gain is obtained thanks to the reduced size of the analyzed unsatisfiable cores. (An alternative technique was introduced in MaxSAT, where one literal is iteratively removed from the unsatisfiable core, either obtaining a smaller unsatisfiable core, or a necessary literal in the processed unsatisfiable core [Nadel, 2010; Nadel *et al.*, 2014].) On the instances of the Sixth ASP Competition, our implementation is often able to provide (suboptimal) stable models with a guarantee of distance to the optimum cost of around 10%.

References

[Alviano and Dodaro, 2016a] Mario Alviano and Carmine Dodaro. Answer set enumeration via assumption literals. In Giovanni Adorni, Stefano Cagnoni, Marco Gori, and Marco Maratea, editors, *AI*IA 2016: Advances in Artificial Intelligence - XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 - December 1, 2016, Proceedings*, volume 10037 of *LNCS*, pages 149–163. Springer, 2016.

[Alviano and Dodaro, 2016b] Mario Alviano and Carmine Dodaro. Anytime answer set optimization via unsatisfiable core shrinking. *TPLP*, 16(5-6):533–551, 2016.

[Alviano and Dodaro, 2016c] Mario Alviano and Carmine Dodaro. Completion of disjunctive logic programs. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 886–892. IJCAI/AAAI Press, 2016.

[Alviano *et al.*, 2014] Mario Alviano, Carmine Dodaro, and Francesco Ricca. Anytime Computation of Cautious Consequences in Answer Set Programming. *TPLP*, 14(4-5):755–770, 2014.

[Alviano *et al.*, 2015a] Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski, editors, *Proceedings of Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015*, volume 9345 of *LNCS*, pages 40–54. Springer, 2015.

[Alviano *et al.*, 2015b] Mario Alviano, Carmine Dodaro, Joao Marques-Silva, and Francesco Ricca. Optimum stable model search: algorithms and implementation. *Journal of Logic and Computation*, 2015.

[Alviano *et al.*, 2015c] Mario Alviano, Carmine Dodaro, and Francesco Ricca. A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 2677–2683. AAAI Press, 2015.

[Andres *et al.*, 2012] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012*, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[Bliem *et al.*, 2016] Bernhard Bliem, Benjamin Kaufmann, Torsten Schaub, and Stefan Woltran. ASP for Anytime Dynamic Programming on Tree Decompositions. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*. AAAI Press, 2016.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Buccafurri *et al.*, 2000] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Trans. Knowl. Data Eng.*, 12(5):845–860, 2000.

[Eiter *et al.*, 2009] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer Set Programming: A Primer. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Tutorial Lectures*, volume 5689 of *LNCS*, pages 40–110. Springer, 2009.

[Gebser *et al.*, 2015a] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Javier Romero, and Torsten Schaub. Progress in clasp Series 3. In Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski, editors, *LPNMR 2015*, volume 9345 of *LNCS*, pages 368–383. Springer, 2015.

[Gebser *et al.*, 2015b] Martin Gebser, Marco Maratea, and Francesco Ricca. The Design of the Sixth Answer Set Programming Competition. In Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski, editors, *Proceedings of Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015*, volume 9345 of *LNCS*, pages 531–544. Springer, 2015.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[Lifschitz, 2008] Vladimir Lifschitz. What Is Answer Set Programming? In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 1594–1597. AAAI Press, 2008.

[Marek *et al.*, 2008] Victor W. Marek, Ilkka Niemelä, and Miroslaw Truszczynski. Logic programs with monotone abstract constraint atoms. *TPLP*, 8(2):167–199, 2008.

[Morgado *et al.*, 2014] António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-Guided MaxSAT with Soft Cardinality Constraints. In *Proceedings of Principles and Practice of Constraint Programming - 20th International Conference, CP 2014*, pages 564–573, Lyon, France, September 2014. Springer.

[Nadel *et al.*, 2014] Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Accelerated Deletion-based Extraction of Minimal Unsatisfiable Cores. *JSAT*, 9:27–51, 2014.

[Nadel, 2010] Alexander Nadel. Boosting minimal unsatisfiable core extraction. In Roderick Bloem and Natasha Sharygina, editors, *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010*, pages 221–229. IEEE, 2010.

[Niemelä, 1999] Ilkka Niemelä. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.