Conflict-Driven Answer Set Enumeration

Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub[⋆]

Institut für Informatik,
Universität Potsdam,
August-Bebel-Str. 89, D-14482 Potsdam, Germany

Abstract. We elaborate upon a recently proposed approach to finding an answer set of a logic program based on concepts from constraint processing and satisfiability checking. We extend this approach and propose a new algorithm for enumerating answer sets. The algorithm, which to our knowledge is novel even in the context of satisfiability checking, is implemented in the *clasp* answer set solver. We contrast our new approach to alternative systems and different options of *clasp*, and provide an empirical evaluation.

1 Introduction

Answer set programming (ASP; [1]) has become a primary tool for declarative problem solving. Although the corresponding solvers are highly optimized (cf. [2,3]), their performance does not match the one of state-of-the-art solvers for satisfiability checking (SAT; [4]). While SAT-based ASP solvers like *assat* [5] and *cmodels* [6] exploit SAT solvers, the underlying techniques are not yet established in genuine ASP solvers. We addressed this deficiency in [7] by introducing a new computational approach to ASP solving, centered around the constraint processing (CSP; [8]) concept of a *nogood*. Apart from the fact that this allows us to easily integrate solving technology from the areas of CSP and SAT, it also provided us with a uniform representation of inferences from logic program rules, unfounded sets, as well as nogoods learned from conflicts.

While we have detailed in [7] how a single answer set is obtained, we introduce in what follows an algorithm for enumerating answer sets. In contrast to systematic backtracking approaches, the passage from computing a single to multiple solutions is non-trivial in the context of backjumping and clause learning. A popular approach consists in recording a found solution as a nogood and exempting it from nogood deletion. However, such an approach is prone to blow up in space in view of the exponential number of solutions in the worst case. Unlike this, our algorithm runs in polynomial space and is (to the best of our knowledge) even a novelty in the context of SAT.

After establishing the formal background, we describe in Section 3 the constraint-based specification of ASP solving introduced in [7]. Based on this uniform representation, we develop in Section 4 algorithms for answer set enumeration, relying on conflict-driven learning and backjumping. In Section 5, we provide a systematic empirical evaluation of different approaches to answer set enumeration, examining different systems as well as different options within our conflict-driven answer set solver *clasp*.

⋆ Affiliated with Simon Fraser University, Canada, and Griffith University, Australia.

2 Background

Given an alphabet $\mathcal{P}$, a (normal) *logic program* is a finite set of rules of the form $p_0 \leftarrow p_1, \dots, p_m, \mathit{not}\ p_{m+1}, \dots, \mathit{not}\ p_n$ where $0 \leq m \leq n$ and $p_i \in \mathcal{P}$ is an *atom* for $0 \leq i \leq n$. A *body literal* is an atom p or its negation $\mathit{not}\ p$. For a rule r, let $\mathit{head}(r) = p_0$ be the *head* of r and $\mathit{body}(r) = \{p_1, \dots, p_m, \mathit{not}\ p_{m+1}, \dots, \mathit{not}\ p_n\}$ be the *body* of r. The set of atoms occurring in a logic program Π is denoted by $\mathit{atom}(\Pi)$. The set of bodies in Π is $\mathit{body}(\Pi) = \{\mathit{body}(r) \mid r \in \Pi\}$. For regrouping rule bodies sharing the same head p, define $\mathit{body}(p) = \{\mathit{body}(r) \mid r \in \Pi, \mathit{head}(r) = p\}$. In ASP, the semantics of a program Π is given by its *answer sets*. For a formal introduction to ASP, we refer the reader to [1].

We consider Boolean *assignments*, A, over the *domain* $\mathit{dom}(A) = \mathit{atom}(\Pi) \cup \mathit{body}(\Pi)$. Formally, an assignment A is a sequence $(\sigma_1, \dots, \sigma_n)$ of *signed literals* σ_i of form $\mathbf{T}p$ or $\mathbf{F}p$ for $p \in \mathit{dom}(A)$ and $1 \leq i \leq n$; $\mathbf{T}p$ expresses that p is *true* and $\mathbf{F}p$ that it is *false*. (We omit the attribute *signed* for literals whenever clear from the context.) We denote the complement of a literal σ by $\overline{\sigma}$, that is, $\overline{\mathbf{T}p} = \mathbf{F}p$ and $\overline{\mathbf{F}p} = \mathbf{T}p$. We let $A \circ B$ denote the sequence obtained by concatenating assignments A and B. We sometimes abuse notation and identify an assignment with the set of its contained literals. Given this, we access true and false members of A via $A^{\mathbf{T}} = \{p \in \mathit{dom}(A) \mid \mathbf{T}p \in A\}$ and $A^{\mathbf{F}} = \{p \in \mathit{dom}(A) \mid \mathbf{F}p \in A\}$.

A *nogood* is a set $\{\sigma_1, \dots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment that contains $\sigma_1, \dots, \sigma_n$. An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = \mathit{dom}(A)$ and $A^{\mathbf{T}} \cap A^{\mathbf{F}} = \emptyset$ is a *solution* for a set Δ of nogoods if $\delta \not\subseteq A$ for all $\delta \in \Delta$. For a nogood δ, a literal $\sigma \in \delta$, and an assignment A, we say that $\overline{\sigma}$ is *unit-resulting* for δ wrt A if (1) $\delta \setminus A = \{\sigma\}$ and (2) $\overline{\sigma} \notin A$. By (1), σ is the single literal from δ that is not contained in A. This implies that a violated constraint does not have a unit-resulting literal. Condition (2) makes sure that no duplicates are introduced: If A already contains $\overline{\sigma}$, then it is no longer unit-resulting. For instance, literal $\mathbf{F}q$ is unit-resulting for nogood $\{\mathbf{F}p, \mathbf{T}q\}$ wrt assignment $(\mathbf{F}p)$, but neither wrt $(\mathbf{F}p, \mathbf{F}q)$ nor wrt $(\mathbf{F}p, \mathbf{T}q)$. Note that our notion of a unit-resulting literal is closely related to the *unit clause rule* of DPLL (cf. [4]). For a set Δ of nogoods and an assignment A, we call *unit propagation* the iterated process of extending A with unit-resulting literals until no further literal is unit-resulting for any nogood in Δ.

3 Nogoods of Logic Programs

Our approach is guided by the idea of Lin and Zhao [5] and decomposes ASP solving into (local) inferences obtainable from the *Clark completion* of a program [9] and those obtainable from loop formulas.

We begin with nogoods capturing inferences from the Clark completion of a program Π. The latter can be defined as follows:

$$\begin{aligned} &\{p_\beta \equiv p_1 \wedge \dots \wedge p_m \wedge \neg p_{m+1} \wedge \dots \wedge \neg p_n \mid \\ &\qquad\qquad \beta \in \mathit{body}(\Pi), \beta = \{p_1, \dots, p_m, \mathit{not}\ p_{m+1}, \dots, \mathit{not}\ p_n\}\} && (1) \\ &\cup \{p \equiv p_{\beta_1} \vee \dots \vee p_{\beta_k} \mid p \in \mathit{atom}(\Pi), \mathit{body}(p) = \{\beta_1, \dots, \beta_k\}\}\ . && (2) \end{aligned}$$

This formulation relies on auxiliary atoms representing bodies; this avoids an exponential blow-up of the corresponding set of clauses. The first type of equivalences in (1) takes care of bodies, while the second one in (2) deals with atoms.

For obtaining the underlying set of constraints, we begin with the body-oriented equivalence in (1). Consider a body $\beta \in \mathit{body}(\Pi)$. The equivalence in (1) can be decomposed into two implications. First, we get $p_\beta \rightarrow p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n$, which is equivalent to the conjunction of $\neg p_\beta \vee p_1, \ldots, \neg p_\beta \vee p_m, \neg p_\beta \vee \neg p_{m+1}, \ldots, \neg p_\beta \vee \neg p_n$. These clauses express the following set of nogoods:

$$\Delta(\beta) = \{\, \{\mathbf{T}\beta, \mathbf{F}p_1\}, \ldots, \{\mathbf{T}\beta, \mathbf{F}p_m\}, \{\mathbf{T}\beta, \mathbf{T}p_{m+1}\}, \ldots, \{\mathbf{T}\beta, \mathbf{T}p_n\} \,\} \;.$$

As an example, consider the body $\{x, \mathit{not}\ y\}$. We obtain the nogoods $\Delta(\{x, \mathit{not}\ y\}) = \{\, \{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{F}x\}, \{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{T}y\} \,\}$. Similarly, the converse of the previous implication, viz. $p_\beta \leftarrow p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n$, gives rise to the nogood

$$\delta(\beta) = \{\mathbf{F}\beta, \mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\} \;.$$

Intuitively, $\delta(\beta)$ forces the truth of β or the falsity of a body literal in β. For instance, for body $\{x, \mathit{not}\ y\}$, we get the nogood $\delta(\{x, \mathit{not}\ y\}) = \{\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x, \mathbf{F}y\}$.

Proceeding analogously with the atom-based equivalences in (2), we obtain for an atom $p \in \mathit{atom}(\Pi)$ along with its bodies $\mathit{body}(p) = \{\beta_1, \ldots, \beta_k\}$ the nogoods

$$\Delta(p) = \{\, \{\mathbf{F}p, \mathbf{T}\beta_1\}, \ldots, \{\mathbf{F}p, \mathbf{T}\beta_k\} \,\} \quad \text{and} \quad \delta(p) = \{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\} \;.$$

For example, for an atom x with $\mathit{body}(x) = \{\{y\}, \{\mathit{not}\ z\}\}$, we get the nogoods $\Delta(x) = \{\, \{\mathbf{F}x, \mathbf{T}\{y\}\}, \{\mathbf{F}x, \mathbf{T}\{\mathit{not}\ z\}\} \,\}$ and $\delta(x) = \{\mathbf{T}x, \mathbf{F}\{y\}, \mathbf{F}\{\mathit{not}\ z\}\}$.

Combining the four types of nogoods leads us to the following set of nogoods:

$$\begin{aligned} \Delta_\Pi = \; & \{\delta(\beta) \mid \beta \in \mathit{body}(\Pi)\} \cup \{\delta \in \Delta(\beta) \mid \beta \in \mathit{body}(\Pi)\} \\ & \cup \{\delta(p) \mid p \in \mathit{atom}(\Pi)\} \cup \{\delta \in \Delta(p) \mid p \in \mathit{atom}(\Pi)\} \;. \end{aligned} \tag{3}$$

The nogoods in Δ_Π capture the *supported models* of a program [10]. Any answer set is a supported model, but the converse only holds for *tight* programs [11]. The mismatch on non-tight programs is caused by *loops* [5], responsible for cyclic support among true atoms. Such cyclic support can be prohibited by loop formulas. As shown in [12], the answer sets of a program Π are precisely the models of Π that satisfy the loop formulas of all non-empty subsets of $\mathit{atom}(\Pi)$.

For a program Π and some $U \subseteq \mathit{atom}(\Pi)$, we define the *external bodies* of U for Π as $\mathit{EB}_\Pi(U) = \{\mathit{body}(r) \mid r \in \Pi, \mathit{head}(r) \in U, \mathit{body}(r) \cap U = \emptyset\}$. The (disjunctive) *loop formula* of U for Π is

$$\neg\big(\textstyle\bigvee_{\beta \in \mathit{EB}_\Pi(U)} (\bigwedge_{p \in \beta^+} p \wedge \bigwedge_{p \in \beta^-} \neg p)\big) \rightarrow \neg\big(\bigvee_{p \in U} p\big)$$

where $\beta^+ = \beta \cap \mathit{atom}(\Pi)$ and $\beta^- = \{p \mid \mathit{not}\ p \in \beta\}$. The loop formula of a set U of atoms forces all elements of U to be false if U is not *externally supported* [12]. To capture the effect of a loop formula induced by a set $U \subseteq \mathit{atom}(\Pi)$ such that $\mathit{EB}_\Pi(U) = \{\beta_1, \ldots, \beta_k\}$, we define the *loop nogood* of an atom $p \in U$ as

$$\lambda(p, U) = \{\mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k, \mathbf{T}p\} \;.$$

Overall, we get the following set of loop nogoods for a program Π:

$$\Lambda_\Pi = \bigcup_{U \subseteq atom(\Pi), U \neq \emptyset} \{\lambda(p, U) \mid p \in U\} \ . \quad (4)$$

As shown in [7], completion and loop nogoods allow for characterizing answer sets.

Theorem 1 ([7]). *Let Π be a logic program, let Δ_Π and Λ_Π as given in (3) and (4). Then, a set X of atoms is an answer set of Π iff $X = A^{\mathbf{T}} \cap atom(\Pi)$ for a (unique) solution A for $\Delta_\Pi \cup \Lambda_\Pi$.*

The nogoods in $\Delta_\Pi \cup \Lambda_\Pi$ describe a set of constraints that must principally be checked for computing answer sets. While the size of Δ_Π is linear in $atom(\Pi) \times body(\Pi)$, the one of Λ_Π is exponential. Thus, answer set solvers use dedicated algorithms that explicate loop nogoods in Λ_Π only on demand, either for propagation or model verification.

4 Answer Set Enumeration

We presented in [7] an algorithm for computing one answer set that is based upon *Conflict-Driven Clause Learning* (CDCL; [4]). In what follows, we combine ideas from the *First-UIP scheme* of CDCL and *Conflict-directed BackJumping* (CBJ; [13]) with particular propagation mechanisms for ASP in order to obtain an algorithm for enumerating a desired number of answer sets (if they exist). Our major objective is to use First-UIP learning and backjumping in the enumeration of solutions, while avoiding repeated solutions and the addition of (non-removable) nogoods to the original problem.

In fact, First-UIP backjumping constitutes a "radical" strategy to recover from conflicts: It jumps directly to the point where a conflict-driven assertion takes effect, undoing all portions of the search space in between. The undone part of the search space is not necessarily exhausted, and some portions of it can be reconstructed in the future. On the one hand, the possibility to revisit parts of the search space makes the termination of CDCL less obvious than it is for other search procedures. (For a proof of termination, see for instance [14].) On the other hand, avoiding repetitions in the enumeration of solutions becomes non-trivial: When a solution has been found and a conflict occurs after flipping the value of some variable(s) in it, then a conflict-driven assertion might reestablish a literal from the already enumerated solution, and after backjumping, the same solution might be feasible again. This is avoided in CDCL solvers by recording "pseudo" nogoods for prohibiting already enumerated solutions. Such a nogood must not be removed, which is different from conflict nogoods that can be deleted once they are obsolete. Of course, an enumeration strategy that records nogoods for prohibiting solutions runs into trouble if there are numerous solutions, in which case the solver blows up in space.

Unlike First-UIP backjumping, CBJ, which has been designed for CSP and is also used in the SAT solver *relsat* [15], makes sure that backjumping only undoes exhausted search spaces. In particular, if there is a solution, then an unflipped decision literal of

Algorithm 1. NOGOODPROPAGATION

```
Input  : A program $\Pi$, a set $\nabla$ of nogoods, and an assignment $A$.
Output : An extended assignment and set of nogoods.

1  $U \leftarrow \emptyset$                                   // set of unfounded atoms
2  loop
3      while $\varepsilon \not\subseteq A$ for all $\varepsilon \in \Delta_\Pi \cup \nabla$ and
4      there is some $\delta \in \Delta_\Pi \cup \nabla$ st $\delta \setminus A = \{\sigma\}$ and $\overline{\sigma} \notin A$ do
5          $A \leftarrow A \circ (\overline{\sigma})$
6          $dlevel(\overline{\sigma}) \leftarrow max(\{dlevel(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\})$
7      if $\varepsilon \subseteq A$ for some $\varepsilon \in \Delta_\Pi \cup \nabla$ or TIGHT($\Pi$) then
8          return $(A, \nabla)$
9      else
10         $U \leftarrow U \setminus A^{\mathbf{F}}$
11         if $U = \emptyset$ then $U \leftarrow$ UNFOUNDEDSET($\Pi, A$)
12         if $U = \emptyset$ then return $(A, \nabla)$
13         else let $p \in U$ in
14             $\nabla \leftarrow \nabla \cup \{\lambda(p, U)\}$
15             if $\mathbf{T}p \in A$ then return $(A, \nabla)$
16             else
17                 $A \leftarrow A \circ (\mathbf{F}p)$
18                 $dlevel(\mathbf{F}p) \leftarrow max(\{dlevel(\rho) \mid \rho \in \lambda(p, U) \setminus \{\mathbf{T}p\}\} \cup \{0\})$
```

that solution cannot be jumped over, as no nogood excludes the search space below it. Only the fact that all solutions containing a certain set of decision literals have been enumerated justifies retracting one of them. This is reflected by CBJ, where a decision literal can only be retracted if the search space below it is exhausted.

Our strategy to enumerate solutions combines First-UIP learning and backjumping with CBJ. As long as no solution has been found, we apply the First-UIP scheme as usual (cf. [7]). Once we have found a solution, its decision literals must be backtracked chronologically. That is, we cannot jump over any unflipped decision literal contributing to a solution. (Other decision literals are treated as usual.) Only if a search space is exhausted, we flip the value of the last decision literal contained in a solution. Note that the First-UIP scheme can be applied even if some decision literals belong to a solution as long as only other decision literals are jumped over.

Algorithm 1 refines the propagation algorithm introduced in [7]. The major change is given in ll. 3–6: For every unit-resulting literal $\overline{\sigma}$ that is added to A, the value of $dlevel(\overline{\sigma})$ is explicated. Instead of the current decision level, we assign the greatest value $dlevel(\rho)$ of any literal $\rho \in \delta \setminus \{\sigma\}$. So $dlevel(\overline{\sigma})$ is the smallest decision level such that $\overline{\sigma}$ is unit-resulting for δ wrt A. In Line 18, $dlevel(\mathbf{F}p)$ is determined in the same way for $\lambda(p, U)$. See [7] for details on the unchanged parts of Algorithm 1.

Algorithm 2 implements our approach to enumerating a given number of answer sets. Its key element is the chronological backtracking level bl. At any state of the computation, its value holds the greatest decision level such that (1) the corresponding

Algorithm 2. CDNL-ENUM-ASP

```
Input : A program $\Pi$ and a number $s$ of solutions to enumerate.
1  $A \leftarrow \emptyset$                                  // assignment over $atom(\Pi) \cup body(\Pi)$
2  $\nabla \leftarrow \emptyset$                            // set of (dynamic) nogoods
3  $dl \leftarrow 0$                                          // decision level
4  $bl \leftarrow 0$                                          // (systematic) backtracking level
5  loop
6      $(A, \nabla) \leftarrow$ NogoodPropagation$(\Pi, \nabla, A)$
7      if $\varepsilon \subseteq A$ for some $\varepsilon \in \Delta_\Pi \cup \nabla$ then
8          if $dl = 0$ then exit
9          else if $bl < dl$ then
10             $(\delta, \sigma_{UIP}, k) \leftarrow$ ConflictAnalysis$(\varepsilon, \Pi, \nabla, A)$
11             $\nabla \leftarrow \nabla \cup \{\delta\}$
12             $dl \leftarrow max(\{k, bl\})$
13             $A \leftarrow A \setminus \{\sigma \in A \mid dl < dlevel(\sigma)\}$
14             $A \leftarrow A \circ (\overline{\sigma_{UIP}})$
15             $dlevel(\overline{\sigma_{UIP}}) \leftarrow k$
16         else
17             $\sigma_d \leftarrow dliteral(dl)$
18             $dl \leftarrow dl - 1$
19             $bl \leftarrow dl$
20             $A \leftarrow A \setminus \{\sigma \in A \mid dl < dlevel(\sigma)\}$
21             $A \leftarrow A \circ (\overline{\sigma_d})$
22             $dlevel(\overline{\sigma_d}) \leftarrow dl$
23     else if $A^{\mathbf{T}} \cup A^{\mathbf{F}} = atom(\Pi) \cup body(\Pi)$ then
24         print $A^{\mathbf{T}} \cap atom(\Pi)$
25         $s \leftarrow s - 1$
26         if $s = 0$ or $dl = 0$ then exit
27         else
28             $\sigma_d \leftarrow dliteral(dl)$
29             $dl \leftarrow dl - 1$
30             $bl \leftarrow dl$
31             $A \leftarrow A \setminus \{\sigma \in A \mid dl < dlevel(\sigma)\}$
32             $A \leftarrow A \circ (\overline{\sigma_d})$
33             $dlevel(\overline{\sigma_d}) \leftarrow dl$
34     else
35         $\sigma_d \leftarrow$ Select$(\Pi, \nabla, A)$
36         $dl \leftarrow dl + 1$
37         $A \leftarrow A \circ (\sigma_d)$
38         $dlevel(\sigma_d) \leftarrow dl$
39         $dliteral(dl) \leftarrow \sigma_d$
```

decision literal has not (yet) been flipped and (2) some enumerated solution contains all decision literals up to decision level bl. To guarantee that no solution is repeated, we have to make sure that backjumping does not retract decision level bl without flipping a

Algorithm 3. CONFLICTANALYSIS

```
Input  : A violated nogood $\delta$, a program $\Pi$, a set $\nabla$ of nogoods, and an assignment $A$.
Output : A derived nogood, a UIP, and a decision level.
1 let $\sigma \in \delta$ st $A = B \circ (\sigma) \circ B'$ and $\delta \setminus B = \{\sigma\}$
2 while $\{\rho \in \delta \mid dlevel(\rho) = dlevel(\sigma)\} \neq \{\sigma\}$ do
3     let $\varepsilon \in \Delta_\Pi \cup \nabla$ st $\overline{\sigma} \in \varepsilon$ and $\varepsilon \setminus B = \{\overline{\sigma}\}$
4     $\delta \leftarrow (\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$
5     let $\sigma \in \delta$ st $B = C \circ (\sigma) \circ C'$ and $\delta \setminus C = \{\sigma\}$
6     $B \leftarrow C$
7 $k \leftarrow max(\{dlevel(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\})$
8 return $(\delta, \sigma, k)$
```

decision literal whose decision level is smaller than or equal to bl.[1] We exclude such a situation in Algorithm 2 by denying backjumps "beyond" decision level bl, if a conflict is encountered at a decision level $dl > bl$, or by enforcing the flipping of the decision literal of decision level bl, if a conflict (or a solution) is encountered at decision level bl. The latter means that the search space below decision level bl is exhausted, that is, all its solutions have been enumerated, so that the decision literal of decision level bl needs to be flipped for enumerating any further solutions.

As in Algorithm 1, we explicitly assign $dlevel(\sigma)$ whenever some literal σ is added to assignment A in Algorithm 2. Also, we set $dliteral(dl)$ to σ_d in Line 39 when decision literal σ_d is added to A at decision level dl. In this way, no confusion about the decision level of a literal or the decision literal of a decision level is possible.[2]

Conflict analysis in Algorithm 3 follows the approach in [7]; it assumes that there is a *Unique Implication Point* (UIP) at the decision level where the conflict has been encountered. This is always the case: A look at ll. 8–22 in Algorithm 2 reveals that the conflict to be analyzed is a consequence of the last decision, and not caused by flipping a decision literal in order to enumerate more solutions. (Note that flipping a decision literal does not produce a new decision level, hence, we have $bl = dl$ if a deliberate flipping causes a conflict. In such a case, we do not analyze the respective conflict.)

We illustrate answer set enumeration by CDNL-ENUM-ASP on the schematic example in Figure 1. Thereby, we denote by σ_d^i the ith decision literal picked by SELECT in Line 35 of Algorithm 2. We denote by σ_a^i the complement of a UIP, asserted in Line 14 of Algorithm 2, after decision literal σ_d^i led to a conflict. For a literal σ, we write $\sigma[n]$ to indicate the decision level of σ, that is, $dlevel(\sigma) = n$. Note that, in Figure 1, we represent assignments only by their decision and asserted literals, respectively, and omit any literals derived by NOGOODPROPAGATION. We underline the decision literal of the chronological backtracking level bl. If an assignment contains such a literal,

[1] A backjump without flipping could happen if we would exclusively use the First-UIP scheme. An assertion at a decision level $dl < bl$ would be such that the complement of the corresponding UIP has been present in a solution enumerated before. Hence, reassigning all decision literals between dl (exclusive) and bl (inclusive) would lead to an already enumerated solution.

[2] We assume that $\sigma_d \notin A$ and $\overline{\sigma_d} \notin A$ for any decision literal σ_d returned by SELECT(Π, ∇, A) in Line 35 of Algorithm 2.

$A_1 = (\sigma_d^1[1], \sigma_d^2[2], \sigma_d^3[3], \sigma_d^4[4], \sigma_d^5[5])$	conflict at $dl = 5$
$A_2 = (\sigma_d^1[1], \sigma_d^2[2], \sigma_d^3[3], \sigma_a^5[3])$	assertion at $dl = 3$
$A_3 = (\sigma_d^1[1], \sigma_d^2[2], \sigma_d^3[3], \sigma_a^5[3], \sigma_d^6[4])$	solution at $dl = 4$
$A_4 = (\sigma_d^1[1], \sigma_d^2[2], \underline{\sigma_d^3[3]}, \sigma_a^5[3], \overline{\sigma_d^6}[3])$	backtracking to $bl = 3$
$A_5 = (\sigma_d^1[1], \sigma_d^2[2], \underline{\sigma_d^3[3]}, \sigma_a^5[3], \overline{\sigma_d^6}[3], \sigma_d^7[4], \sigma_d^8[5])$	conflict at $dl = 5$
$A_6 = (\sigma_d^1[1], \sigma_a^8[1], \sigma_d^2[2], \underline{\sigma_d^3[3]}, \sigma_a^5[3], \overline{\sigma_d^6}[3])$	assertion at $dl = 1$
	backtracking to $bl = 3$
$A_7 = (\sigma_d^1[1], \sigma_a^8[1], \sigma_d^2[2], \underline{\sigma_d^3[3]}, \sigma_a^5[3], \overline{\sigma_d^6}[3], \sigma_d^9[4])$	solution at $dl = 4$
$A_8 = (\sigma_d^1[1], \sigma_a^8[1], \sigma_d^2[2], \underline{\sigma_d^3[3]}, \sigma_a^5[3], \overline{\sigma_d^6}[3], \overline{\sigma_d^9}[3])$	backtracking to $bl = 3$
	solution/conflict at $dl = 3 = bl$
$A_9 = (\sigma_d^1[1], \sigma_a^8[1], \underline{\sigma_d^2[2]}, \overline{\sigma_d^3}[2])$	backtracking to $bl = 2$ ...

Fig. 1. Answer set enumeration example

it must not be retracted unless the search space below it is exhausted, that is, unless a conflict or a(nother) solution is encountered at decision level bl.

Consider assignment A_1 in Figure 1, and assume that NOGOODPROPAGATION yields a violated nogood after decision literal σ_d^5 has been selected at decision level $dl = 5$. Let CONFLICTANALYSIS return a nogood δ such that $\overline{\sigma_a^5} \in \delta$ and $max(\{dlevel(\sigma) \mid \sigma \in \delta \setminus \{\overline{\sigma_a^5}\}\} \cup \{0\}) = 3$, that is, $\overline{\sigma_a^5}$ is a UIP. Given that no solution has been found yet, we have $bl = 0$. Thus, CDNL-ENUM-ASP jumps back to decision level 3 and asserts σ_a^5, yielding assignment A_2. Up to this point, the enumeration of solutions is similar to the search for a single solution. We next select decision literal σ_d^6 at decision level $dl = 4$. Assume that NOGOODPROPAGATION on assignment A_3 yields a solution. Since we are enumerating solutions, we cannot stop here. Rather, we continue with assignment A_4 obtained by flipping σ_d^6, and $bl = 3$ is the greatest decision level of any unflipped decision literal. Note that $\overline{\sigma_d^6}$ at decision level $dlevel(\overline{\sigma_d^6}) = 3 = bl$ is not asserted by any nogood. We continue by selecting decision literals σ_d^7 and σ_d^8, yielding assignment A_5. Suppose that NOGOODPROPAGATION yields again a violated nogood at decision level $dl = 5$ and that CONFLICTANALYSIS returns a nogood δ with $\overline{\sigma_a^8} \in \delta$ and $max(\{dlevel(\sigma) \mid \sigma \in \delta \setminus \{\overline{\sigma_a^8}\}\} \cup \{0\}) = 1$. That is, σ_a^8 is asserted by δ at decision level 1. Given that the previous solution included $\sigma_d^1 = dliteral(1)$, it must also have contained σ_a^8; otherwise, some nogood had been violated after NOGOODPROPAGATION. If we would now jump back to decision level 1 and assert σ_a^8, then the already enumerated solution would be feasible again, and CDNL-ENUM-ASP would repeat it. After asserting σ_a^8, we thus have to return to decision level $dl = 3 = bl$, rather than to 1, yielding assignment A_6. Note that A_6 still contains $\overline{\sigma_d^6}$, so that the solution encountered after selecting σ_d^6 (cf. A_3) cannot be repeated. Assume that selecting decision literal σ_d^9 at decision level $dl = 4$ yields a second solution for assignment A_7. Then, we backtrack to decision level $dl = 3 = bl$ and flip σ_d^9, yielding assignment A_8. Note that $\overline{\sigma_d^9}$ is not asserted by any nogood. If now NOGOODPROPAGATION yields a third solution, then decision level 3 is exhausted, that is, all solutions containing σ_d^1, σ_d^2, and σ_d^3 have been enumerated. Hence, we let $bl = 2$ and flip σ_d^3, yielding assignment A_9. Otherwise, if NOGOODPROPAGATION on A_8 leads to a violated nogood, then we do not analyze the conflict because $dl = 3 = bl$. In fact, flipped decision literals $\overline{\sigma_d^6}$ and $\overline{\sigma_d^9}$ lack asserting nogoods, so that the result of CONFLICTANALYSIS would be undefined. If NOGOODPROPAGATION yields a conflict for A_8, we thus proceed with A_9, as in the case that a solution is found for A_8.

We now introduce the notions of *correctness*, *completeness*, and *redundancy-freeness* for an answer set enumeration algorithm.

Definition 1. *For a logic program Π, we define an enumeration algorithm as*

– *correct, if every enumerated solution is an answer set of Π;*
– *complete, if all answer sets of Π are enumerated;*
– *redundancy-free, if no answer set of Π is enumerated twice.*

Furthermore, we need the following property **(UF)**:

For any assignment A, let UNFOUNDEDSET(Π, A) *in Algorithm 1 return some non-empty unfounded set $U \subseteq atom(\Pi) \setminus A^{\mathbf{F}}$ for Π wrt A, if there is such a set U, and return the empty set $\emptyset$, otherwise.*[3]

By letting Π be a logic program and $s \in \mathbb{Z}$, we have the following correctness result.

Theorem 2. CDNL-ENUM-ASP(Π, s) *is correct, provided that **(UF)** holds.*

For a program Π and $X \subseteq atom(\Pi)$, we say that X *agrees* with a nogood δ if one of the following conditions holds, where $\overline{X} = atom(\Pi) \setminus X$:

– $\mathbf{F}p \in \delta$ for some $p \in X$,
– $\mathbf{T}p \in \delta$ for some $p \in \overline{X}$,
– $\mathbf{F}\beta \in \delta$ for some $\beta \in body(\Pi)$ such that $\beta \subseteq X \cup \{not\ p \mid p \in \overline{X}\}$, or
– $\mathbf{T}\beta \in \delta$ for some $\beta \in body(\Pi)$ such that $\beta \cap (\overline{X} \cup \{not\ p \mid p \in X\}) \neq \emptyset$.

Intuitively, the notion of agreement expresses that $\delta \not\subseteq A$ for the total assignment A corresponding to X. That is, $\mathbf{T}p \in A$ for all atoms $p \in X$, $\mathbf{F}p \in A$ for all atoms $p \in \overline{X}$, and for a body $\beta \in body(\Pi)$, $\mathbf{T}\beta \in A$ if all body literals of β are true wrt X and $\mathbf{F}\beta \in A$ otherwise. We can show that any answer set X of Π agrees with all nogoods dealt with by CDNL-ENUM-ASP, both static and dynamic ones.

We first consider static nogoods in $\Delta_\Pi \cup \Lambda_\Pi$, as given in (3) and (4).

Proposition 1. *Any answer set of Π agrees with all nogoods in $\Delta_\Pi \cup \Lambda_\Pi$.*

Given this, we can show that the answer sets of Π agree with all nogoods added to ∇.

Proposition 2. *At every state of* CDNL-ENUM-ASP(Π, s)*, any answer set of Π agrees with all nogoods in ∇, provided that **(UF)** holds.*[4]

This leads us to the completeness of CDNL-ENUM-ASP, when invoked with $s = 0$.

Theorem 3. CDNL-ENUM-ASP$(\Pi, 0)$ *is complete, provided that **(UF)** holds.*

Finally, we can show that CDNL-ENUM-ASP is redundancy-free.

Theorem 4. CDNL-ENUM-ASP(Π, s) *is redundancy-free.*

[3] A set U of atoms is unfounded for Π wrt A if we have $EB_\Pi(U) \subseteq A^{\mathbf{F}}$.
[4] We here stipulate **(UF)** for making sure that the result of CONFLICTANALYSIS is well-defined at every invocation.

Table 1. Experiments enumerating answer sets

No	Instance	#Sol	*clasp*$_a$	*clasp*$_{ar}$	*clasp*$_b$	*clasp*$_{br}$	*smodels*	*smodels*$_r$	*smodels*$_{cc}$	*cmodels*
1	hc_19	10^4	7.2	7.2	7.7	7.2	•	•	•	•
2	hc_19	10^5	71.4	77.1	83.5	91.2	•	•	•	•
3	hc_20	10^4	9.3	9.5	10.9	9.5	•	•	•	•
4	hc_20	10^5	103.4	117.2	115.8	109.9	•	•	•	•
5	mutex3IDFD	10^5	1.4	1.4	35.4	35.9	5.5	5.8	240.6	•
6	mutex3IDFD	10^6	14	13.9	•	•	55.9	52.8	•	•
7	mutex4IDFD	10^4	20.8	27.4	43.8	37	44.7	574.7	47.5	•
8	mutex4IDFD	10^5	52.2	63.2	596.7	585.7	273.4	•	•	•
9	pigeon_15	10^5	2.7	2.7	4	3.9	7.1	8.6	126.7	•
10	pigeon_15	10^6	26.1	26.5	53	54.7	71.8	73.6	•	•
11	pigeon_15	10^7	260.7	262.8	•	•	•	•	•	•
12	pigeon_16	10^5	3.2	3.1	4.4	4.6	7.8	9.9	175.2	•
13	pigeon_16	10^6	30.1	30.5	57.7	59.6	78.5	80.9	•	•
14	pigeon_16	10^7	303	304.5	•	•	•	•	•	•
15	queens_19	10^4	14.4	17.1	13.1	15.1	47.1	115	49	427.49
16	queens_19	10^5	141.5	143.8	135.9	162.7	265.1	358.1	•	•
17	queens_20	10^4	14.1	15.8	13.1	15.3	127	172.1	48.3	569.15
18	queens_20	10^5	147.2	170.5	149.6	178.6	380.3	•	•	•
19	schur-n29-m44	10^4	22.4	26.4	19.8	22.7	17.4	49.4	15.6	•
20	schur-n29-m44	10^5	203.1	212.5	177.2	246.4	132.4	175.9	353.2	•
21	schur-n29-m45	10^4	24.7	21.8	21.5	24.6	17.2	50.2	16.1	•
22	schur-n29-m45	10^5	231.6	265.6	190.7	199.9	133.3	176	397.3	•

5 Experiments

Our empirical evaluation addresses the following two questions: First, how does our algorithm improve on solution recording (via nogoods) and, second, in how far are backjumps hampered by the backtracking level. Our comparison considers *clasp* (RC4) in two different modes: (a) the one with bounded backjumping (and learning), using the algorithms from Section 4 (referred to by *clasp*$_a$), and (b) the one using unlimited backjumping (and learning) in conjunction with solution recording (*clasp*$_b$). Note that a *solution nogood* consists of decision literals only. The same strategy is pursued by *smodels*$_{cc}$ [16], but with decisions limited to atoms. In contrast, *cmodels* provides a whole answer set as solution nogood to the underlying (learning) SAT solver. Given that restarts are disabled in *clasp*$_a$ and *clasp*$_b$, our experiments also include both variants augmented with bounded and unbounded restarts, respectively (indicated by an additional subscript *r*). The bounded restart variant, *clasp*$_{ar}$, is allowed to resume search from the backtracking level (cf. Algorithm 2),[5] while *clasp*$_{br}$ can perform unlimited restarts. We also incorporate standard *smodels* (2.32) and the variant *smodels*$_r$ with activated restart option, *smodels*$_{cc}$ (1.08) with option "nolookahead" as recommended by the developers, and *cmodels* (3.67) using *zchaff* (2004.11.15). All experiments were run on a 2.2GHz PC on Linux. We report results in seconds, taking the average of 10 runs, each restricted to 600s time and 512MB memory. A timeout (in all 10 runs) is indicated by "•". The benchmark instances and extended results are available at [17].

In Table 1, we report results for enumerating a vast number of answer sets. The instances are from the areas of Hamiltonian cycles in complete graphs (1-4), bounded

[5] In order to guarantee redundancy-freeness, restarts must not discard the backtracking level with its flipped decision literals.

Table 2. Experiments illustrating backjumping and backtracking behavior

No	Instance	#Sol	Backtracks	Backjumps	Bounded Jumps	Skippable Levels	Skipped Levels (%)	Jump Length (max)	Bounded Length (max)	Jump Length (avg)	Bounded Length (avg)	Time
1	gryzzles.3	1	0	311	0	771	100	17	0	2.5	0	0.1
2	gryzzles.3	857913	618092	208373	4135	321000	97.6	21	15	1.5	1	178.4
3	gryzzles.7	1	0	675	0	1931	100	22	0	2.9	0	0.1
4	gryzzles.7	10^6	895612	215995	2783	324951	98	23	22	1.5	2	246.9
5	gryzzles.18	1	0	599	0	2026	100	27	0	3.4	0	0.1
6	gryzzles.18	10^6	811593	51219	1605	92953	96.1	27	18	1.7	2	235
7	mutex4IDFD	1	0	280	0	26698	100	590	0	95.4	0	17
8	mutex4IDFD	10^6	0	280	0	26698	100	590	0	95.4	0	579.7
9	sequence2-ss2	1	0	156	0	674	100	35	0	4.3	0	13.3
10	sequence2-ss2	38	64	2875	225	13915	53	73	43	2.6	29	17.9
11	sequence3-ss3	1	0	10921	0	40213	100	65	0	3.7	0	66.9
12	sequence3-ss3	332	315	55111	731	121435	97.8	65	20	2.2	3	361

model checking (5-8), pigeonhole (9-14), n-queens (15-18), and Schur numbers (19-22). We have chosen these combinatorial problems because of their large number of answer sets. This allows us to observe the effect of an increasing number of answer sets on the performance of the respective approaches. The number of requested (and successfully enumerated) solutions is given in the third column. Comparing the two variants of *clasp*, we observe that *clasp*$_a$ and *clasp*$_{ar}$ scale better than *clasp*$_b$ and *clasp*$_{br}$. This is most intelligible on examples from bounded model checking (5-8) and pigeonhole problems (9-14). Solutions for the former contain many decision literals, and the large solution nogoods significantly slow down *clasp*$_b$ and *clasp*$_{br}$. The pigeonhole problems are structurally simple, so that all decisions yield solutions. Since the number of easy-to-compute solutions is massive, the sheer number of recorded solution nogoods slows down *clasp*$_b$ and *clasp*$_{br}$. Also note that the time that *smodels* spends in lookahead is wasted here. With Hamiltonian cycles (1-4), n-queens (15-18), and Schur numbers (19-22), the picture is rather indifferent. That is, solving time tends to dominate enumeration time, and the recorded solution nogoods are not as critical as with the aforementioned problems. Notably, *smodels* is very effective on Schur numbers. We verified that all *clasp* variants make the same number of decisions (or choices) as *smodels*, so we conjecture that different run-times come from implementation differences: counter-based propagation in *smodels* versus watched literals in *clasp*. Regarding the other systems, we see that *smodels*$_{cc}$ is slower than *smodels* as regards enumeration (9-14) but sometimes faster if search is needed (17), and *cmodels* is clearly outperformed. Comparing *clasp*$_a$ to *clasp*$_{ar}$ and *clasp*$_b$ to *clasp*$_{br}$, we see that restarts make (almost) no difference on the problems in Table 1. Indeed, *clasp* hardly ever restarts on these problems, so that the effect is negligible. However, this indifference does not account for *smodels*$_r$, where restarts turn out to be quite counterproductive on our combinatorial problems.

Table 2 provides statistics regarding the backjumping and backtracking of *clasp*$_a$ upon the enumeration of answer sets. The first three instances are Hamiltonian path problems (1-6), the fourth is from bounded model checking (7,8), and the last two from compiler superoptimization (9-12). For every instance, we provide two rows: the backjump statistics for one answer set versus that for a certain number of answer sets. The "Backtracks" column shows the number of chronological backtracks, that is, conflicts on the backtracking level, while "Backjumps" indicates conflicts on greater decision levels. The number of backjumps that were forced to stop at the backtracking level is given by “Bounded Jumps”. The “Skippable Levels” are the sum of backjump lengths (not counting backtracks), and “Skipped Levels” shows the percentage of levels that have effectively been skipped. We also provide the maximum “Jump Length” and the maximum “Bounded Length”. The latter is the maximum number of skippable levels that have not been retracted in a jump because of hitting the backtracking level. Finally, we show the average “Jump Length”, the average “Bounded Length”, and the time. On the Hamiltonian path problems (1-6), we observe that the number of backtracks dominates that of backjumps. Indeed, we also observed on other problems, not shown here, that the “hard” part of the search was before finding the first answer set; afterwards, the number of conflicts above the backtracking level decreased significantly. We see this very drastically on the bounded model checking instance (7,8) where 280 long backjumps are performed (jump length 95 on average). After this “warm-up” phase, no further conflicts are encountered, even not on the backtracking level (0 backtracks). Finally, the superoptimization examples (9-12) are rather sparse regarding answer sets, and backjumps are still noticeable after the first solution has been found. In Line 10, we observe an exceptionally low percentage of skipped levels, approximately half of the skippable levels are kept. The few bounded jumps that are done have a significant length (29 unskipped levels on average). However, on all instances in Table 2, the jump of maximum length was unbounded and thus effectively executed. The average “Jump Length” and the average “Bounded Length” are usually small, except for 7, 8, and 10.

6 Discussion

We introduced a new approach to enumerating answer sets, centered around First-UIP learning and backjumping. To the best of our knowledge, our solution enumeration approach is novel even in the context of SAT. Unlike *relsat* [15], applying the *Last*-UIP scheme, our approach uses First-UIP backjumping as long as systematic backtracking is unnecessary. Different from the #SAT solver *cachet* [18], using so-called “component caching”, our approach combines CDCL with CBJ for avoiding the repetition of solutions. Recent approaches to adopt SAT and CSP techniques in ASP solving [16,19,20] are rather implementation-specific and lack generality. Unlike this, we provided a uniform CSP-based approach by viewing ASP inferences as unit propagation on nogoods, which allowed us to directly incorporate techniques from CSP and SAT.

The *clasp* system implements state-of-the-art techniques from Boolean constraint solving, avoiding a SAT translation as done by *assat* [5], *cmodels* [6], and *sag* [19]. Also, *clasp* records loop nogoods only when ultimately needed for unit propagation; this is different from *assat* and *sag*, which determine loop formulas for all “terminating” loops. Unlike genuine ASP solvers *smodels* [2] and *dlv* [3], *clasp* does not determine greatest unfounded sets. Rather, it applies local propagation directly after an unfounded set has been found. Different from *smodels*$_{cc}$ [16] and *dlv* with backjumping [20], the usage of rule bodies in nogoods allows for a straightforward extension of unit propagation to ASP, abolishing the need for multiple inference rules. Notably, our novel approach allows *clasp* to enumerate answer sets of a program without explicitly prohibiting already computed solutions by nogoods, as done by *cmodels* and *smodels*$_{cc}$.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
2. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
3. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM TOCL **7**(3) (2006) 499–562
4. Mitchell, D.: A SAT solver primer. Bulletin of EATCS **85** (2005) 112–133
5. Lin, F., Zhao, Y.: ASSAT: computing answer sets of a logic program by SAT solvers. Artificial Intelligence **157**(1-2) (2004) 115–137
6. Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. Journal of Automated Reasoning (2007) To appear.
7. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. Proceedings IJCAI'07, AAAI Press/The MIT Press (2007) 386–392
8. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
9. Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press (1978) 293–322
10. Apt, K., Blair, H., Walker, A.: Towards a theory of declarative knowledge. In: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann (1987) 89–148
11. Fages, F.: Consistency of Clark's completion and the existence of stable models. Journal of Methods of Logic in Computer Science **1** (1994) 51–60
12. Lee, J.: A model-theoretic counterpart of loop formulas. Proceedings IJCAI'05, Professional Book Center (2005) 503–508
13. Dechter, R., Frost, D.: Backjump-based backtracking for constraint satisfaction problems. Artificial Intelligence **136**(2) (2002) 147–188
14. Ryan, L.: Efficient algorithms for clause-learning SAT solvers. MSc thesis, Simon Fraser University (2004)
15. Bayardo, R., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. Proceedings AAAI'97, AAAI Press/The MIT Press (1997) 203–208
16. Ward, J., Schlipf, J.: Answer set programming with clause learning. Proceedings LPNMR'04. Springer (2004) 302–313
17. (`http://www.cs.uni-potsdam.de/clasp`)
18. Sang, T., Bacchus, F., Beame, P., Kautz, H., Pitassi, T.: Combining component caching and clause learning for effective model counting. Proceedings SAT'04. (2004)
19. Lin, Z., Zhang, Y., Hernandez, H.: Fast SAT-based answer set solver. Proceedings AAAI'06, AAAI Press/The MIT Press (2006)
20. Ricca, F., Faber, W., Leone, N.: A backjumping technique for disjunctive logic programming. AI Communications **19**(2) (2006) 155–172