Reasoning About Strong Inconsistency in ASP ⋆

Carlos Mencía[1] and Joao Marques-Silva[2]

[1] University of Oviedo, Spain menciacarlos@uniovi.es
[2] ANITI, University of Toulouse, France joao.marques-silva@univ-toulouse.fr

Abstract. The last decade has witnessed remarkable improvements in the analysis of inconsistent formulas, namely in the case of Boolean Satisfiability (SAT) formulas. However, these successes have been restricted to monotonic logics. Recent work proposed the notion of *strong inconsistency* for a number of non-monotonic logics, including Answer Set Programming (ASP). This paper shows how algorithms for reasoning about inconsistency in monotonic logics can be extended to the case of ASP programs, in the concrete case of strong inconsistency. Initial experimental results illustrate the potential of the proposed approach.

1 Introduction

The last decade and a half witnessed a remarkable evolution in algorithms for reasoning about inconsistency. This is the case with algorithms for the extraction and enumeration of minimal unsatisfiable subsets (MUSes) [4–6,8,9,33,34,37,42] and minimal correction subsets (MCSes) [3, 6, 21, 26, 27, 36, 39, 40, 45], but also algorithms for maximum satisfiability (MaxSAT) [1, 2, 18, 35, 41]. This work was motivated by earlier important advances [7, 17, 28, 29, 32, 48]. Although most of this work was proposed in the context of propositional formulas it is also the case that most of the algorithms are amenable to generalization for different fragments of First-Order Logic (FOL). These algorithms specifically addressed monotonic logics, with propositional logic as a concrete example.

In the case of non-monotonic logics, minimal inconsistency is uninteresting [16], because of non-monotonicity. Recent work proposed the concept of *strong inconsistency* for non-monotonic logics [15, 16], which enabled demonstrating that well-known properties of inconsistent sets in monotonic logics also apply in the case of strong inconsistency, with a reference example being the minimal hitting set relationship between minimal inconsistent subsets and minimal correction subsets [46]. Nevertheless, a limitation of this earlier work is that the algorithms proposed aim at being illustrative, consisting of simple set enumeration approaches, known not to scale in practice [34].

This paper changes this state of affairs. Concretely, the paper proposes novel simple insights, which enable *any* algorithm for reasoning about inconsistency

⋆ This research is supported by the Spanish Government under project TIN2016-79190-R, by the Principality of Asturias under grant IDI/2018/000176, and by ANITI, funded by the French program "Investing for the Future – PIA3" under Grant agreement n^o ANR-19-PI3A-0004.

in the monotonic cases, to also be applicable to reasoning about strong inconsistency in the non-monotonic cases. The paper demonstrates the proposed ideas in the concrete setting of Answer Set Programming (ASP) [14, 22], but these can be applied in other settings provided mild conditions hold. The significance of being able to reason efficiently about (strong) inconsistency in ASP should be highlighted. Whereas SAT reasoners represent a remarkable (and unique) problem solving technology, ASP blends efficient problem solving (by exploiting the technologies that are the hallmark of SAT solvers) with a well-established and widely used knowledge representation paradigm. The proposed algorithms enable new applications of ASP based on reasoning about (strong) inconsistency.

2 Preliminaries

Boolean Satisfiability. We consider definitions and notation standard in Boolean Satisfiability (SAT) [10]. Concretely, we consider propositional formulas in *conjunctive normal form* (CNF), defined as a conjunction, or set, of clauses $\mathcal{F} = \{c_1, ..., c_m\}$ over a set of variables $V(\mathcal{F}) = \{x_1, ..., x_n\}$ where a clause is a disjunction of literals, and a literal is a variable x or its negation $\neg x$. An interpretation is a mapping μ: $V(\mathcal{F}) \rightarrow \{0, 1\}$. If μ satisfies $\mathcal{F}$, it is referred to as a *model* of $\mathcal{F}$. $\mathcal{F} \models \mathcal{G}$ means that all the models of $\mathcal{F}$ are models of $\mathcal{G}$. A minimal (resp. maximal) model is such that the set of variables assigned value 1 (resp. 0) is irreducible. A formula is satisfiable ($\mathcal{F} \not\models \bot$) if it has a model; and otherwise unsatisfiable ($\mathcal{F} \models \bot$). In the latter case, the following definitions apply:

Definition 1 (MUS/MCS). *$\mathcal{M} \subseteq \mathcal{F}$ is a* minimal unsatisfiable subset *(MUS) if and only if $\mathcal{M} \models \bot$ and for all $\mathcal{M}' \subsetneq \mathcal{M}$, $\mathcal{M}' \not\models \bot$. $\mathcal{C} \subseteq \mathcal{F}$ is a* minimal correction subset *(MCS) if and only if $(\mathcal{F} \setminus \mathcal{C}) \not\models \bot$ and for all $\mathcal{C}' \subsetneq \mathcal{C}$, $\mathcal{F} \setminus \mathcal{C}' \models \bot$.*

MUSes are minimal explanations of unsatisfiability, while MCSes are irreducible sets of clauses whose removal renders satisfiability. The complement of an MCS is a *maximal satisfiable subset* (MSS). MUSes and MCSes are hitting set duals: Every MCS is a minimal hitting set of all MUSes and vice versa [11, 46].

Example 1. Let $F_{ex} = \{(\neg x_1), (x_1), (x_1 \vee x_2), (\neg x_2)\}$. F_{ex} is unsatisfiable. It has two MUSes: $\mathcal{M}_1 = \{(x_1), (\neg x_1)\}$, $\mathcal{M}_2 = \{(\neg x_1), (x_1 \vee x_2), (\neg x_2)\}$; and three MCSes: $\mathcal{C}_1 = \{(\neg x_1)\}$, $\mathcal{C}_2 = \{(x_1), (x_1 \vee x_2)\}$, $\mathcal{C}_3 = \{(x_1), (\neg x_2)\}$.

Minimal Sets over a Monotone Predicate. Several problems in propositional logic can be reduced to computing a minimal set over a monotone predicate (MSMP) [37, 38][3]. In this setting, a predicate p: $2^{\mathcal{R}} \rightarrow \{0, 1\}$, defined over a reference set $\mathcal{R}$, is *monotone* if whenever $p(\mathcal{R}_0)$ holds, then $p(\mathcal{R}_1)$ also holds, with $\mathcal{R}_0 \subseteq \mathcal{R}_1 \subseteq \mathcal{R}$. $\mathcal{M} \subseteq \mathcal{R}$ is a minimal set over a predicate p if $p(\mathcal{M})$ holds and, for all $\mathcal{M}' \subsetneq \mathcal{M}$, $p(\mathcal{M}')$ does not hold. As an example, given $\mathcal{F} \models \bot$, by setting $\mathcal{R} \triangleq \mathcal{F}$, the MUSes of $\mathcal{F}$ are the minimal sets over the monotone predicate $p(\mathcal{W}) \triangleq \neg\mathsf{SAT}(\mathcal{W})$, with $\mathcal{W} \subseteq \mathcal{R}$. The MCSes of $\mathcal{F}$ are the minimal sets over $p(\mathcal{W}) \triangleq \mathsf{SAT}(\mathcal{R} \setminus \mathcal{W})$, with $\mathcal{W} \subseteq \mathcal{R}$.

[3] MSMP was proposed in [37, 38], but it was inspired by earlier work [12, 13].

Answer Set Programming & Strong Inconsistency. We review basic concepts in ASP. A more detailed account can be found in [14, 22].

A *(normal) logic program* $P = \{r_1, ..., r_n\}$ is a finite set of rules of the following form: $a \leftarrow b_1, ..., b_m$, not $c_{m+1}, ...,$ not c_n, where a, b_i and c_i are atoms. A literal is an atom or its default negation not a. Extended logic programs may include classical negation ($\neg$). For a rule r, $\mathsf{body}(r)$ denotes the literals $b_1, ..., b_m$, not $c_{m+1}, ...,$ not c_n and $\mathsf{head}(r)$ denotes the literal a. We write $B^+(r)$ for $b_1, ..., b_m$ and $B^-(r)$ for $c_{m+1}, ..., c_n$. A rule is a fact if it has an empty body. Further, we allow *choice rules* of the form $n \leq \{a_1, ..., a_k\}$, with $n \geq 0$. A program is *ground* if it does not contain any variables. A *ground instance* of a program P, denoted $\mathsf{grd}(P)$, is a ground program obtained by substituting the variables of P by all constants from its Herbrand universe.

The semantics of ASP programs can be defined via a *reduct* [25]. A set I of ground atoms is a *model* of a program P if $\mathsf{head}(r) \in I$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$ for every $r \in \mathsf{grd}(P)$. The reduct of P w.r.t. the set I, denoted P^I, is defined as $P^I = \{\mathsf{head}(r) \leftarrow B^+(r) \mid r \in \mathsf{grd}(P), I \cap B^-(r) = \emptyset\}$. The set I is an *answer set* of P if I is a minimal model of P^I. The inclusion of choice rule $n \leq \{a_1, ..., a_k\}$ guarantees that any answer set contains at least n atoms from $\{a_1, ..., a_k\}$. A program P is *consistent* if it has at least one consistent answer set; otherwise, P is *inconsistent*.

This paper focuses on the analysis of inconsistent ASP programs. Throughout, we will consider that programs are partitioned into two subsets: $P = B \cup S$, where B denotes *background knowledge*, assumed to be consistent and which cannot be relaxed, and S denotes the set of rules that can be dropped to achieve consistency. In contrast to propositional logic, logical entailment is not monotonic in ASP. Hence, supersets of an inconsistent program are not necessarily inconsistent, and a subset of a consistent program may be inconsistent. This way, MUSes and MCSes as defined for propositional logic do not capture their intended meaning and properties. To overcome this drawback, the notion of *strong inconsistency* [15, 16][4] was recently proposed: Given an inconsistent program $P = B \cup S$, $P' = B \cup S'$, with $S' \subseteq S$, is *strongly P-inconsistent* if for all S'', with $S' \subseteq S'' \subseteq S$, $B \cup S''$ is inconsistent. In other words, strong inconsistency denotes that all supersets (up to P) of a given subprogram are inconsistent. Minimal explanations and corrections of inconsistent ASP programs can be defined in terms of strong inconsistency, as follows:

Definition 2 (MSIS/MSICS). *Given an inconsistent program $P = B \cup S$, the subset $M \subseteq S$ is a* minimal strongly P-inconsistent subset *(MSIS) iff $B \cup M$ is strongly P-inconsistent and, for all $M' \subsetneq M$, $B \cup M'$ is not strongly P-inconsistent. $C \subseteq S$ is a* minimal strong P-inconsistency correction subset *(MSICS) iff $B \cup (S \setminus C)$ is not strongly P-inconsistent and, for all $C' \subsetneq C$, $B \cup (S \setminus C')$ is strongly P-inconsistent.*

The complement of an MSICS is a *maximal consistent subset*. Besides, every MSIS is a minimal hitting set of the set of all MSICSes and vice versa [15, 16].

[4] This notion was defined for arbitrary non-monotonic logics. We show it for ASP.

Example 2. Consider the inconsistent program $P_{ex} = B_{ex} \cup S_{ex}$, with $B_{ex} = \emptyset$ and $S_{ex} = \{r_1 : a \leftarrow \texttt{not}\ a, \texttt{not}\ b.,\ r_2 : b \leftarrow \texttt{not}\ a.,\ r_3 : \neg b.\}$. There are two MSISes: $M_1 = \{r_1, r_3\}$, $M_2 = \{r_2, r_3\}$; and two MSICSes: $C_1 = \{r_1, r_2\}$, $C_2 = \{r_3\}$. Notice that although $\{r_1\}$ is inconsistent, it is not strongly P_{ex}-inconsistent, since $\{r_1, r_2\}$ is consistent (with the only answer set $\{b\}$).

Related Work. Debugging ASP programs has attracted a large body of research (see [20] for a survey). Systems as `spock` [24] or `Ouroboros` [43, 44], based on meta-programming, enable pinpointing errors causing inconsistency, as unsupported atoms or unsatisfied rules. On the other hand, `DWASP` [19] allows for interactively debugging ASP programs by exploiting unsatisfiable cores. In contrast, our goal is computing MSISes and MSICSes, in the case of strong inconsistency. Our work is closely related to [30,31], which extended a number of algorithms for MSSes in SAT to maximal consistent subsets in ASP (and so MSICSes). Herein, we focus on computing MSISes as well, and on enumerating both kinds of sets. To our best knowledge, the only proposed approach for computing MSISes [15, 16] relies on exhaustive set enumeration and was not evaluated empirically.

3 Reasoning About Strongly Inconsistent ASP Programs

3.1 Strong Inconsistency & MSMP

Strong inconsistency exhibits a monotonicity property, that all the supersets (up to P) of a strongly P-inconsistent program are strongly P-inconsistent too:

Proposition 1. *Let $P = B \cup S$, and $P_U = B \cup U$, with $U \subseteq S$, be strongly P-inconsistent. Then, for all $U \subseteq U' \subseteq S$, $P_{U'} = B \cup U'$ is strongly P-inconsistent.*

Proof. Since P_U is strongly P-inconsistent, for all U' with $U \subseteq U' \subseteq S$, $B \cup U'$ is inconsistent. Hence, for any superset U' with $U \subseteq U' \subseteq S$, it holds that for all $U' \subseteq U'' \subseteq S$, $B \cup U''$ is inconsistent. So, $P_{U'}$ is strongly P-inconsistent. □

Throughout, for a given program $P = B \cup S$, and $R \subseteq S$, $\mathsf{SAT}^+(B, S, R)$ indicates whether there is a superset of R (up to S) that together with B is consistent, i.e. it is true iff there exists R', with $R \subseteq R' \subseteq S$, such that $P' = B \cup R'$ is consistent. Noticeably, $\mathsf{SAT}^+(B, S, R)$ is false iff $B \cup R$ is strongly P-inconsistent. We show that computing an MSIS is an instance of MSMP.

Proposition 2. *Computing an MSIS is an instance of the MSMP problem.*

Proof. Let $p(\mathcal{W}) \triangleq \neg\mathsf{SAT}^+(B, S, \mathcal{W})$ with $\mathcal{W} \subseteq \mathcal{R}$, and $\mathcal{R} \triangleq S$. We prove that p is monotone and that any minimal set over p is an MSIS of $P = B \cup S$.
Monotonicity: If $p(\mathcal{W})$ holds, $B \cup \mathcal{W}$ is strongly P-inconsistent. By Proposition 1, for all $\mathcal{W}'$, with $\mathcal{W} \subseteq \mathcal{W}' \subseteq S$, $B \cup \mathcal{W}'$ is strongly P-inconsistent, so $p(\mathcal{W}')$ holds.
Correctness: Let $\mathcal{M}$ be a minimal set for which $p(\mathcal{M})$ holds, i.e. $B \cup \mathcal{M}$ is strongly P-inconsistent. Since $\mathcal{M}$ is minimal, for any $\mathcal{M}' \subsetneq \mathcal{M}$, $p(\mathcal{M}')$ does not hold, i.e. $B \cup \mathcal{M}'$ is not strongly P-inconsistent. Thus, by Definition 2, $\mathcal{M}$ is an MSIS. □

Computing an MSICS can also be reduced to MSMP. The proof is analogous, by defining $p(\mathcal{W}) \triangleq \mathsf{SAT}^+(B, S, S \setminus \mathcal{W})$ with $\mathcal{W} \subseteq \mathcal{R}$, and $\mathcal{R} \triangleq S$.

Algorithm 1: Deletion-based minimal set computation

Input: p: Monotone predicate, $\mathcal{R}$: Reference set
Output: $\mathcal{M}$: Minimal set

1 $\mathcal{M} \leftarrow \mathcal{R}$; // $\mathcal{M}$ is over-approximation
2 **foreach** $u \in \mathcal{M}$ **do** // *Inv:* $p(\mathcal{M})$
3 **if** $p(\mathcal{M} \setminus \{u\})$ **then** // Do we need u?
4 $\mathcal{M} \leftarrow \mathcal{M} \setminus \{u\}$; // If not, drop it
5 **return** $\mathcal{M}$; // Final $\mathcal{M}$ is a minimal set

3.2 Computing Minimal Explanations and Corrections

The reductions above enable computing MSISes and MSICSes by using *any* algorithm for MSMP and an oracle implementing $\mathsf{SAT}^+(B, S, R)$.

Extracting a Single Minimal Set. Algorithms for computing a single minimal set in MSMP include Deletion [17], Progression [37] or QuickXplain [32], among others [8]. Herein we focus on the deletion-based approach, shown in Algorithm 1.

Given an inconsistent program $P = B \cup S$, by setting the predicate to $p(\mathcal{W}) \triangleq \neg\mathsf{SAT}^+(B, S, \mathcal{W})$ with $\mathcal{W} \subseteq \mathcal{R}$, and $\mathcal{R} \triangleq S$, Algorithm 1 proceeds as follows: Starting with $\mathcal{M} = \mathcal{R}$, the algorithm iteratively picks a rule $u \in \mathcal{M}$ and tests whether $B \cup (\mathcal{M} \setminus \{u\})$ is strongly P-inconsistent. If it is, u is removed from $\mathcal{M}$; otherwise u is kept in $\mathcal{M}$. After considering all the rules in $\mathcal{R}$, $\mathcal{M}$ is an MSIS.

An MSICS of P can be computed using *basic linear search* (BLS) [6, 36]: Starting with $\mathcal{S} = \emptyset$, iteratively pick a rule in $u \in S \setminus \mathcal{S}$ and test whether $\mathsf{SAT}^+(B, S, \mathcal{S} \cup \{u\})$ holds. If it does, $B \cup (\mathcal{S} \cup \{u\})$ is not strongly P-inconsistent, and u is added to $\mathcal{S}$. On termination, the set of rules not added to $\mathcal{S}$ is an MSICS (and $\mathcal{S}$ is a maximal consistent subset). Besides, if the oracle for $\mathsf{SAT}^+(B, S, \mathcal{S} \cup \{u\})$ returns a witness after positive answers, all the elements in S satisfied can be added to $\mathcal{S}$, saving some predicate tests. BLS is equivalent to Algorithm 1 using the predicate $p(\mathcal{W}) \triangleq \mathsf{SAT}^+(B, S, S \setminus \mathcal{W})$, with $\mathcal{W} \subseteq \mathcal{R}$, and $\mathcal{R} \triangleq S$.

Enumerating Minimal Sets. MARCO [33] is a successful approach for enumerating MUSes and MCSes of CNF formulas. This algorithm exploits the hitting set duality between MUSes and MCSes. Since this relationship also holds between MSISes and MSICSes, MARCO can be adapted to ASP, as shown in Algorithm 2.

For a given inconsistent program $P = B \cup S$, the algorithm associates a propositional variable p_i with each rule $r_i \in S$, and maintains a CNF formula $\mathcal{H}$ defined on these variables. The formula $\mathcal{H}$, initially empty, serves to subsequently avoid considering any superset (resp. subset) of previously found MSISes (resp. MSICSes). Iteratively, a maximal model MxM of $\mathcal{H}$ is computed, which induces the set of rules R whose associated variables are set to 1 in MxM. Then, if the program $B \cup R$ is strongly P-inconsistent (i.e. if $\mathsf{SAT}^+(B, S, R)$ does not hold), an MSIS $M \subseteq R$ of P is extracted (e.g. by using Algorithm 1, with $\mathcal{R} \triangleq R$), whose supersets are blocked by adding a negative clause on its associated variables to $\mathcal{H}$. Otherwise, R is a maximal consistent subset, and so $S \setminus R$ is an MSICS of P, whose subsets are blocked by adding a positive clause on its associated variables

Algorithm 2: Minimal set enumeration

Input: $P = B \cup S$: Inconsistent ASP program
Output: MSISes and MSICSes of P

```
1  I ← {p_i | r_i ∈ S};
2  H ← ∅;                                          // Block MSISes and MSICSes
3  while true do
4      (st, MxM) ← ComputeMaximalModel(H);
5      if not st then return;
6      R ← {r_i | p_i ∈ MxM};                      // Pick selected rules
7      if not SAT+(B, S, R) then
8          M ← ComputeMSIS(B, S, R);               // Extract MSIS from R
9          ReportMSIS(M);
10         b ← {¬p_i | r_i ∈ M};                   // Block the MSIS
11     else
12         ReportMSICS(S \ R);
13         b ← {p_i | p_i ∈ I \ MxM};              // Block the MSICS
14     H ← H ∪ {b};
```

to $\mathcal{H}$. The process is repeated until $\mathcal{H}$ becomes unsatisfiable, with the guarantee that all MSISes and MSICSes of P have been computed.

Algorithm 2 is organized to give (heuristic) preference to finding MSISes quickly. We refer to it as `eMax`. A variant giving preference to finding MSICSes can be easily obtained, by computing minimal models of $\mathcal{H}$ (instead of maximal ones) and extracting an MSICS whenever $\mathsf{SAT}^+(B, S, R)$ holds. This variant is referred to as `eMin`.

Implementing $\mathsf{SAT}^+(B, S, R)$. It remains to discuss the way $\mathsf{SAT}^+(B, S, R)$ can be implemented in ASP. We invoke an ASP solver on an modified program which includes *selector* atoms and choice rules. This approach was used in [30, 31] to compute maximal consistent subsets. For a set of atoms $\mathcal{A}$, $\mathsf{choice}(\mathcal{A})$ denotes the rule $0 \leq \{a_1, .., a_k\}$, with $a_i \in \mathcal{A}$. Modern ASP solvers allow choice rules, and their inclusion does not increase the complexity beyond NP [47].

For a given program $P = B \cup S$, we first build the program $P_s = B \cup S_s$, where S_s is obtained from S as follows: for each rule $r_i \in S$ we introduce a fresh atom s_i, and add the rule $\mathsf{head}(r_i) \leftarrow \mathsf{body}(r_i), s_i$ to S_s. Note that if the fact s_i is added to P_s, the rule r_i is *activated*, and relaxed otherwise. For a given subset $R \subseteq S$, we use $s(R)$ to denote the set of selector atoms for rules in R in S_s, i.e. $s(R) = \{s_i \mid r_i \in R\}$. Then, the test $\mathsf{SAT}^+(B, S, R)$ is solved by invoking an ASP solver on the program $P' = P_s \cup \cup_{s \in s(R)}\{s\} \cup \mathsf{choice}(s(S \setminus R))$. Notice that each rule $r \in R$ is active in P'. Besides, the inclusion of the rule $\mathsf{choice}(s(S \setminus R))$ allows for activating any (or none) of the rules in $S \setminus R$ when looking for answer sets of P'. Hence, P' is consistent iff the program $B \cup R$ is not strongly P-inconsistent.

Example 3. Let $P = B \cup S$ be the program in Example 2, and consider the test $\mathsf{SAT}^+(B, S, \{r_1\})$. We first build $P_s = \{a \leftarrow \mathsf{not}\ a, \mathsf{not}\ b, s_1.,\ b \leftarrow \mathsf{not}\ a, s_2.,\ \neg b \leftarrow s_3.\}$. Then, we define $P' = P_s \cup \{s_1.\} \cup \mathsf{choice}(\{s_2, s_3\})$. P' is consistent (with the unique answer set $\{b, s_1, s_2\}$), indicating that $\{r_1\}$ is not strongly P-inconsistent.

4 Preliminary Results

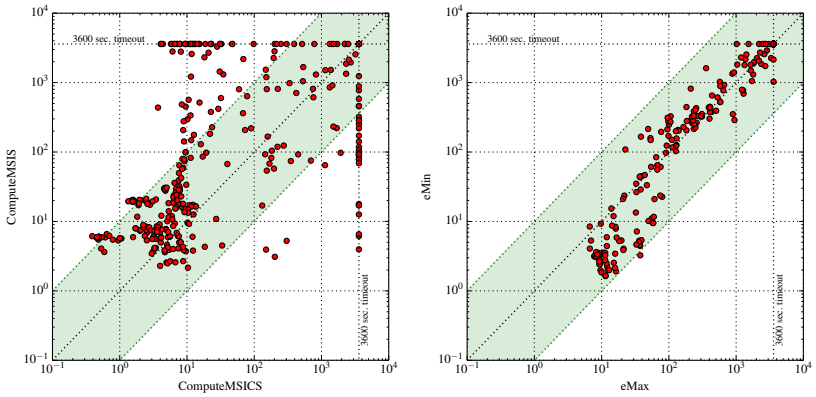
This section reports an initial experimental assessment of the proposed approaches. We implemented a prototype in Python 2.7, interfacing the ASP solver `clingo` [23] (v. 5.4.0), and ran a series of experiments on a Linux machine (2.26GHz, 128GB). Each process was limited to 3600s and 4GB. Below, `ComputeMSIS` (resp. `ComputeMSICS`) is Algorithm 1 using the predicate shown in Section 3 for computing an MSIS (resp. MSICS). Besides, witnesses are used in the extraction of MSICSes as an optimization, as described earlier. On the other hand, `eMax` corresponds to Algorithm 2, giving preference to finding MSISes quickly, and `eMin` is the variant that gives preference to MSICSes. In these cases, maximal and minimal models are computed using the tool mcsls [36][5].

Similarly to earlier work [31], we built a number of instances. We considered three problem domains (common in ASP competitions): *Graceful graphs*, *Knight tour with holes* and *Solitaire*. Each instance is an inconsistent ASP program $P = B \cup S$, where B contains the rules encoding the problem domain (assumed correct) and S contains the facts specific for each instance. Given the complexity of the tasks to solve, the instances are reasonably small. The benchmarks are as follows: *1) Graceful graphs*: Given a graph (V, E) the goal is to label its vertices with distinct integers in the range $0..|E|$ so that each edge is labeled with the absolute difference between the labels of its vertices and all edge labels are distinct. S contains the facts indicating the edges, so $|S| = |E|$. We considered values of $|V| \in \{10, 20\}$ and $|E| \in \{10, 20, 50\}$. *2) Knight tour with holes*: Given an $N \times N$ board with H holes, the problem asks if a knight chess piece can visit all non-hole positions of the boards exactly once returning to the initial position. S consists of facts with the positions of holes, so $|S| = H$. We considered values of $N \in \{7, 8\}$ and $H \in \{10, 20, 30\}$. *3) Solitaire*: Given a 7×7 board, with 2×2 corners removed (i.e. with 33 squares), an initial configuration is specified by facts $empty(L)$ and $full(L)$, indicating if each square L is empty or contains a stone. A stone can be moved by two squares if it jumps over another stone, which is removed. The goal is to perform T steps. S contains the facts $empty(L)$ and $full(L)$, so $|S| = 33$. We considered values of $T \in \{8, 10, 12, 14, 16, 18\}$. For each configuration, we built 20 random instances, making 360 in all.

The results are summarized in Fig. 1. Fig. 1a shows, for each instance, the running times needed for computing a single MSIS and an MSICS. `ComputeMSIS` and `ComputeMSICS` solved, respectively, 295 and 317 instances. The results vary across the set of instances, although in more cases computing an MSICS was performed faster than computing an MSIS. Fig. 1b compares `eMax` and `eMin`. In this case, complete enumeration was achieved for 172 and 167 instances respectively. However, as the plot indicates, there is no clear winner.

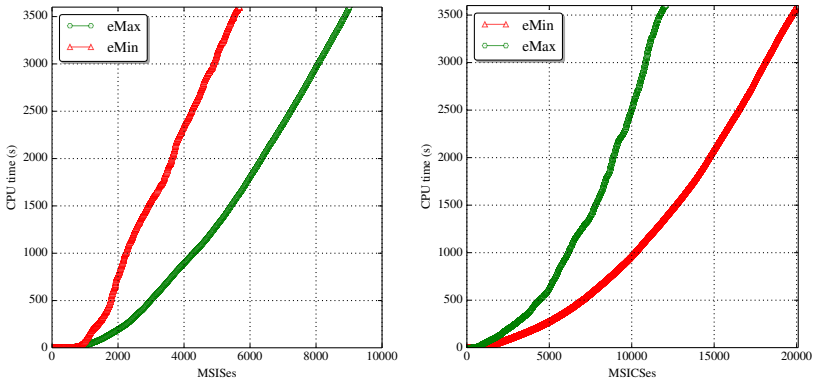
Fig. 2 shows the number of reported minimal sets over the whole benchmark set. By the time limit `eMax` reports 9008 MSISes and 12081 MSICSes, whereas `eMin` computes 5684 MSISes and 20057 MSICSes. As shown in Fig. 2a, `eMax` is

[5] Computing a minimal/maximal model can be reduced to computing an MCS. For this purpose, several alternatives can be used (e.g. [3, 36, 39, 40]).



(a) `ComputeMSIS` vs. `ComputeMSICS` (b) `eMin` vs `eMax`

Fig. 1: Running times



(a) Reported MSISes (b) Reported MSICSes

Fig. 2: Number of reported sets (`eMin` vs `eMax`)

much more efficient at computing MSISes, whereas `eMin` finds MSICSes faster (see Fig. 2b). Thus, each variant is effective at its intended purpose. These results suggest that a combination may be a good option for obtaning both sets quickly.

5 Conclusions

Recent work proposed the concept of *strong inconsistency* [15, 16], which provides a way of reasoning about inconsistency in non-monotonic logics. This paper shows how the large body of work for reasoning about (minimal) inconsistency in monotonic logics, originally developed in the context of SAT, can be readily applied to the case of reasoning about strong inconsistency in non-monotonic logics. Furthermore, the paper applies these insights to the case of ASP. Experimental results illustrate the scope and applicability of the proposed approach.

References

1. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial MaxSAT through satisfiability testing. In: SAT. pp. 427–440 (2009)
2. Ansótegui, C., Bonet, M.L., Levy, J.: SAT-based MaxSAT algorithms. Artif. Intell. **196**, 77–105 (2013)
3. Bacchus, F., Davies, J., Tsimpoukelli, M., Katsirelos, G.: Relaxation search: A simple way of managing optional clauses. In: AAAI. pp. 835–841 (2014)
4. Bacchus, F., Katsirelos, G.: Using minimal correction sets to more efficiently compute minimal unsatisfiable sets. In: CAV. pp. 70–86 (2015)
5. Bacchus, F., Katsirelos, G.: Finding a collection of MUSes incrementally. In: CPAIOR. pp. 35–44 (2016)
6. Bailey, J., Stuckey, P.J.: Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: PADL. pp. 174–186 (2005)
7. Bakker, R.R., Dikker, F., Tempelman, F., Wognum, P.M.: Diagnosing and solving over-determined constraint satisfaction problems. In: IJCAI. pp. 276–281 (1993)
8. Belov, A., Lynce, I., Marques-Silva, J.: Towards efficient MUS extraction. AI Commun. **25**(2), 97–116 (2012)
9. Bendík, J., Cerná, I., Benes, N.: Recursive online enumeration of all minimal unsatisfiable subsets. In: ATVA. pp. 143–159 (2018)
10. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
11. Birnbaum, E., Lozinskii, E.L.: Consistent subsets of inconsistent systems: structure and behaviour. J. Exp. Theor. Artif. Intell. **15**(1), 25–46 (2003)
12. Bradley, A.R., Manna, Z.: Checking safety by inductive generalization of counterexamples to induction. In: FMCAD. pp. 173–180 (2007)
13. Bradley, A.R., Manna, Z.: Property-directed incremental invariant generation. Formal Asp. Comput. **20**(4-5), 379–405 (2008)
14. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
15. Brewka, G., Thimm, M., Ulbricht, M.: Strong inconsistency in nonmonotonic reasoning. In: IJCAI. pp. 901–907 (2017)
16. Brewka, G., Thimm, M., Ulbricht, M.: Strong inconsistency. Artif. Intell. **267**, 78–117 (2019)
17. Chinneck, J.W., Dravnieks, E.W.: Locating minimal infeasible constraint sets in linear programs. ORSA Journal on Computing **3**(2), 157–168 (1991)
18. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: CP. pp. 225–239 (2011)
19. Dodaro, C., Gasteiger, P., Musitsch, B., Ricca, F., Shchekotykhin, K.M.: Interactive debugging of non-ground ASP programs. In: LPNMR. pp. 279–293 (2015)
20. Fandinno, J., Schulz, C.: Answering the "why" in answer set programming - A survey of explanation approaches. Theory Pract. Log. Program. **19**(2), 114–203 (2019)
21. Felfernig, A., Schubert, M., Zehentner, C.: An efficient diagnosis algorithm for inconsistent constraint sets. AI EDAM **26**(1), 53–62 (2012)
22. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012)
23. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo = ASP + control: Extended report. Tech. rep., University of Potsdam (2014)

24. Gebser, M., Pührer, J., Schaub, T., Tompits, H.: A meta-programming technique for debugging answer-set programs. In: AAAI. pp. 448–453 (2008)
25. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP. pp. 1070–1080 (1988)
26. Grégoire, É., Izza, Y., Lagniez, J.: Boosting MCSes enumeration. In: IJCAI. pp. 1309–1315 (2018)
27. Grégoire, É., Lagniez, J., Mazure, B.: An experimentally efficient method for (MSS, CoMSS) partitioning. In: AAAI. pp. 2666–2673 (2014)
28. Grégoire, É., Mazure, B., Piette, C.: Extracting MUSes. In: ECAI. pp. 387–391 (2006)
29. Hemery, F., Lecoutre, C., Sais, L., Boussemart, F.: Extracting MUCs from constraint networks. In: ECAI. pp. 113–117 (2006)
30. Janota, M., Marques-Silva, J.: On minimal corrections in ASP. CoRR **abs/1406.7838** (2014), http://arxiv.org/abs/1406.7838
31. Janota, M., Marques-Silva, J.: On minimal corrections in ASP. In: RCRA. pp. 45–54 (2017), http://ceur-ws.org/Vol-2011/paper5.pdf
32. Junker, U.: QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In: AAAI. pp. 167–172 (2004)
33. Liffiton, M.H., Previti, A., Malik, A., Marques-Silva, J.: Fast, flexible MUS enumeration. Constraints **21**(2), 223–250 (2016)
34. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. J. Autom. Reasoning **40**(1), 1–33 (2008)
35. Manquinho, V.M., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: SAT. pp. 495–508 (2009)
36. Marques-Silva, J., Heras, F., Janota, M., Previti, A., Belov, A.: On computing minimal correction subsets. In: IJCAI. pp. 615–622 (2013)
37. Marques-Silva, J., Janota, M., Belov, A.: Minimal sets over monotone predicates in boolean formulae. In: CAV. pp. 592–607 (2013)
38. Marques-Silva, J., Janota, M., Mencía, C.: Minimal sets on propositional formulae. Problems and reductions. Artif. Intell. **252**, 22–50 (2017)
39. Mencía, C., Ignatiev, A., Previti, A., Marques-Silva, J.: MCS extraction with sublinear oracle queries. In: SAT. pp. 342–360 (2016)
40. Mencía, C., Previti, A., Marques-Silva, J.: Literal-based MCS extraction. In: IJCAI. pp. 1973–1979 (2015)
41. Morgado, A., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: A survey and assessment. Constraints **18**(4), 478–534 (2013)
42. Narodytska, N., Bjørner, N., Marinescu, M.V., Sagiv, M.: Core-guided minimal correction set and core enumeration. In: IJCAI. pp. 1353–1361 (2018)
43. Oetsch, J., Pührer, J., Tompits, H.: Catching the ouroboros: On debugging non-ground answer-set programs. Theory Pract. Log. Program. **10**(4-6), 513–529 (2010)
44. Polleres, A., Frühstück, M., Schenner, G., Friedrich, G.: Debugging non-ground ASP programs with choice rules, cardinality and weight constraints. In: LPNMR. pp. 452–464 (2013)
45. Previti, A., Mencía, C., Järvisalo, M., Marques-Silva, J.: Premise set caching for enumerating minimal correction subsets. In: AAAI. pp. 6633–6640 (2018)
46. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
47. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. **138**(1-2), 181–234 (2002)

48. de Siqueira N., J.L., Puget, J.: Explanation-based generalisation of failures. In: ECAI. pp. 339–344 (1988)