Computing MUS-Based Inconsistency Measures[⋆]

Isabelle Kuhlmann[1][0000−0001−9636−122X], Andreas Niskanen[2][0000−0003−3197−2075], and Matti Järvisalo[2][0000−0003−2572−063X]

[1] University of Hagen, Germany
[2] University of Helsinki, Finland

Abstract. We detail two instantiations of a generic algorithm for the problematic and MUS-variable-based inconsistency measures, based on answer set programming and Boolean satisfiability (SAT). Empirically, the SAT-based approach allows for more efficiently computing the measures when compared to enumerating all minimal correction subsets of a knowledge base.

Keywords: Inconsistency measurement · minimal unsatisfiability

1 Introduction

Inconsistency measurement [27, 29] aims to provide a quantitative assessment of the level of inconsistency in knowledge bases. However, inconsistency measurement in propositional knowledge bases is highly non-trivial under essentially any reasonable quantitative measure of inconsistency [61]. Despite this, algorithmic approaches to inconsistency measurement have been developed [41, 42, 39, 40, 57] based on declarative techniques. However, various inconsistency measures based on minimally unsatisfiable subsets (MUSes) of knowledge bases [32, 21, 28, 63] cannot be directly captured with a single call to an NP optimizer due to higher complexity of MUS-based inconsistency measures [61]. Less attention has been paid so-far on developing algorithms for such measures [34].

We develop algorithms for the *problematic* (P) [28] and *MUS-variable-based*[3] (MV) [63] inconsistency measures. Both can be determined by enumerating all MUSes in the knowledge base (KB) in terms of the KB formulas: the former measure is the number of KB formulas that occur in the union of the MUSes, the latter the number of *variables* that occur in the union of the MUSes (relative to the number of variables in the KB). By hitting set duality [60], instead of enumerating all MUSes, the measures can alternatively be computed by enumerating all MCSes of the KB [63] using SAT-based MCS enumerators [58, 59, 30, 10, 6] as extensions of MCS extractors [4, 31, 48, 51, 53]; MCS enumeration is known to be often easier than MUS enumeration [45, 7, 55, 11, 9, 8]. However, MCS enumeration algorithms are not specifically developed with inconsistency measurement in mind.

[⋆] Work financially supported by Deutsche Forschungsgemeinschaft (grant 506604007 / IK) and by Academy of Finland (grants 347588 / AN and 322869, 356046 / MJ).

[3] Not to be confused with the notion of variable minimal unsatisfiability [18, 5].

We develop a generic algorithmic approach specifically for computing the P and MV inconsistency measures, and detail two of its instantiations: one based on iteratively calling an answer set programming (ASP) solver [26, 56] on a sequence of queries under a disjunctive answer set program specific to P and MV, and another based on SAT-based counterexample-guided abstraction refinement (CEGAR) [19, 20]. The SAT-based CEGAR instantiation empirically outperforms both ASP and state-of-the-art MCS enumerators.

2 Preliminaries

A *knowledge base* (KB) is a finite set of propositional formulas. The signature $\mathsf{At}(\cdot)$ of a formula or knowledge base is the set of atoms (or variables) appearing in the formula/KB. A *(truth) assignment* $\tau : \mathsf{At} \to \{0,1\}$ assigns a truth value (1, *true* or 0, *false*) to each atom in At. An assignment τ satisfies a formula ϕ (and ϕ is satisfiable) iff $\tau(\phi) = 1$, i.e., ϕ evaluates to 1 under τ. A KB $\mathcal{K}$ is consistent if there is an assignment that satisfies all formulas in $\mathcal{K}$, and otherwise inconsistent. Let $\mathbb{K}$ be the set of all knowledge bases. Formally, an *inconsistency measure* is a function $\mathcal{I} : \mathbb{K} \to \mathbb{R}^{\infty}_{\geq 0}$ for which $\mathcal{I}(\mathcal{K}) = 0$ iff $\mathcal{K}$ is consistent for all $\mathcal{K} \in \mathbb{K}$. The *problematic* (P) inconsistency measure [28] counts the number of formulas in a given KB participating in some conflict. Similarly, the *MUS-variable-based* (MV) inconsistency measure [63] counts the number of atoms in the signature of a KB that are involved in some conflict. A conflict is defined by the notion of a *minimal unsatisfiable subset* (MUS). A set of logical formulas $S \subseteq \mathcal{K}$ is a *minimal unsatisfiable subset (MUS)* of $\mathcal{K}$ if S is inconsistent, and all $S' \subsetneq S$ are consistent. Now, let $\mathsf{MUS}(\mathcal{K})$ be the set of MUSes of a given KB $\mathcal{K}$.

Definition 1. *The* problematic *(P) inconsistency measure* $\mathcal{I}_{\mathrm{p}} : \mathbb{K} \to \mathbb{R}^{\infty}_{\geq 0}$ *is* $\mathcal{I}_{\mathrm{p}}(\mathcal{K}) = |\bigcup \mathsf{MUS}(\mathcal{K})|$. *The* MUS-variable-based *(MV) inconsistency measure* $\mathcal{I}_{\mathrm{mv}} : \mathbb{K} \to \mathbb{R}^{\infty}_{\geq 0}$ *is* $\mathcal{I}_{\mathrm{mv}}(\mathcal{K}) = |\bigcup_{M \in \mathsf{MUS}(\mathcal{K})} \mathsf{At}(M)| / |\mathsf{At}(\mathcal{K})|$.

Example 1. Let $\mathcal{K}_1 = \{x \wedge y, \neg x, \neg y, y \vee z\}$. Then $\mathsf{MUS}(\mathcal{K}_1) = \{\{x \wedge y, \neg x\}, \{x \wedge y, \neg y\}\}$. Hence $|\bigcup \mathsf{MUS}(\mathcal{K}_1)| = |\{x \wedge y, \neg x, \neg y\}| = 3$, so $\mathcal{I}_{\mathrm{p}}(\mathcal{K}_1) = 3$. Moreover, $|\bigcup_{M \in \mathsf{MUS}(\mathcal{K}_1)} \mathsf{At}(M)| = |\mathsf{At}(\{x \wedge y, \neg x\}) \cup \mathsf{At}(\{x \wedge y, \neg y\})| = |\{x, y\} \cup \{x, y\}| = |\{x, y\}| = 2$, and $|\mathsf{At}(\mathcal{K}_1)| = |\{x, y, z\}| = 3$. Therefore $\mathcal{I}_{\mathrm{mv}}(\mathcal{K}_1) = \frac{2}{3}$.

A set $S \subseteq \mathcal{K}$ is a *minimal correction set (MCS)* if $\mathcal{K} \setminus S$ is consistent, and for all $S' \subsetneq S$, $\mathcal{K} \setminus S'$ is inconsistent. In words, MCSes identify fragments of KBs whose removal resolves inconsistency. By hitting set duality between MUSes and MCSes [60], we have $\bigcup \mathsf{MUS}(\mathcal{K}) = \bigcup \mathsf{MCS}(\mathcal{K})$ for any KB $\mathcal{K}$, i.e., the union of MUSes is the same as the union of MCSes. In turn, $\mathcal{I}_{\mathrm{p}}(\mathcal{K}) = |\bigcup \mathsf{MCS}(\mathcal{K})|$. The MV measure is equivalently defined by considering atoms in MCSes [63].

3 Algorithms for the P and MV Inconsistency Measures

The P and MV measures can be computed via the union of MCSes of the input KB $\mathcal{K}$. This is (naively) achieved by enumerating all MCSes, as suggested

Algorithm 1 Generic algorithm for the P and MV inconsistency measures. Input: knowledge base $\mathcal{K}$, measure $\mathcal{I} \in \{\mathcal{I}_{\mathrm{p}}, \mathcal{I}_{\mathrm{mv}}\}$.

```
1: $Q \leftarrow \mathcal{K}$, $C \leftarrow \emptyset$
2: while $Q \neq \emptyset$ do
3:     $mcs \leftarrow$ MCSOVERLAP$(\mathcal{K}, Q)$
4:     if $mcs = \perp$ then break
5:     if $\mathcal{I} = \mathcal{I}_{\mathrm{p}}$ then
6:         $C \leftarrow C \cup mcs$; $Q \leftarrow Q \setminus mcs$
7:     else if $\mathcal{I} = \mathcal{I}_{\mathrm{mv}}$ then
8:         $C \leftarrow C \cup \mathsf{At}(mcs)$; $Q \leftarrow Q \setminus \{\phi \in \mathcal{K} \mid \mathsf{At}(\phi) \subseteq C\}$
9: if $\mathcal{I} = \mathcal{I}_{\mathrm{p}}$ then return $|C|$ else if $\mathcal{I} = \mathcal{I}_{\mathrm{mv}}$ then return $|C|/|\mathsf{At}(\mathcal{K})|$
```

for the MV measure [63]. However, this may result in the extraction of MCSes redundant w.r.t. the measure: for P, an MCS which contains only formulas encountered in previous MCSes, and for MV an MCS whose signature is in the signature of previous MCSes, does not affect the inconsistency value. The computation of irredundant MCSes can be formalized as the *MCS overlap problem*: find an MCS M of $\mathcal{K}$ which intersects a given query $Q \subseteq \mathcal{K}$ of interest. The corresponding decision problem is Σ_2^p-complete, as it is equivalent to the MUS overlap problem [43, 37] which in turn captures the Σ_2^p-complete problem of deciding whether a given clause occurs in an MUS [44]. As at most a linear number of NP oracle calls are needed for extracting an MCS [50], it is not plausible that MCS enumeration algorithms could avoid computing redundant MCSes.

3.1 Generic Algorithm

Our generic algorithm (Algorithm 1) avoids computing redundant MCSes by iteratively solving the MCS overlap problem instead of enumerating MCSes. Assume that a procedure MCSOVERLAP is available, returning for a given KB $\mathcal{K}$ and query Q an MCS mcs with $mcs \cap Q \neq \emptyset$, or $\perp$ if no such MCS exists. We start by initializing Q to $\mathcal{K}$ and C (covered elements) to $\emptyset$ (line 1). Then, while Q remains nonempty, we extract an MCS intersecting Q (line 3). If no such MCS exists, we exit the loop (line 4). How the query Q and the set C is updated depends on the measure. For the P measure, we add the MCS to C and remove it from Q (lines 5–6). For MV, we add the signature of the MCS to C and remove from Q all formulas whose signature is included in C (lines 7–8). Finally, we either return the size of C for the P measure, or divide it by the size of the signature of the KB for the MV measure (line 9).

3.2 Instantiation via Disjunctive ASP

First, we detail a disjunctive ASP [17, 25, 47] approach, directly capturing Σ_2^p, to the MCS overlap problem; see Listing 1.1. Its idea is to guess a candidate set S_{cs} of formulas and check whether it is a maximal satisfiable subset (MSS)—the

```
 1 1{inCs(X): kbElement(X)}.
 2 inComplement(F):- kbElement(F), not inCs(F).
 3 atomInComplement(A):- atomInFormula(A,F), inComplement(F).
 4 validCS:- 1{atomInComplement(A): queryAtom(A)}.
 5 :- not validCS.
 6 atomInCs(A):- atomInFormula(A,F), inCs(F).
 7 1{truthValueCS(A,T): tv(T)}1 :- atomInCs(A).
 8 numElementsInCs(X):- X = #count{F: inCs(F)}.
 9 csIsSat:- numElementsInCs(X), X{truthValueCS(F,t): inCs(F), kbElement(F)}X.
10 :- not csIsSat.
11 numSupersets(X):- numElementsInCs(Y), numKbElements(Z), X=Z-Y.
12 superset(1..X):- numSupersets(X), X>0.
13 1{addElement(F,S): inComplement(F)}1 :- superset(S).
14 supersetEq(S1,S2):- superset(S1), superset(S2), S1!=S2, addElement(F1,S1),
        addElement(F2,S2), F1==F2.
15 :- supersetEq(S1,S2).
16 inSuperset(F,S):- inCs(F), superset(S).
17 inSuperset(F,S):- addElement(F,S), superset(S).
18 atomInSuperset(A,S):- atomInCs(A), superset(S).
19 atomInSuperset(A,S):- addElement(F,S), atomInFormula(A,F).
20 truthValueSet(A,S,t) | truthValueSet(A,S,f):- atomInSuperset(A,S),
        superset(S).
21 truthValueSet(A,S,t):- isUnsat(S), atomInSuperset(A,S), superset(S).
22 truthValueSet(A,S,f):- isUnsat(S), atomInSuperset(A,S), superset(S).
23 isUnsat(S):- truthValueSet(F,S,f), inSuperset(F,S).
24 :- not isUnsat(S), superset(S).
```

Listing 1.1: Disjunctive ASP encoding for MCS overlap.

set-complement of which is an MCS. An MSS must be satisfiable while all of its supersets must be unsatisfiable. Moreover, we enforce that at least one atom from $\mathsf{At}(Q)$ (w.r.t. MV) or, respectively, formula from Q (w.r.t. P) must be included in the set-complement of the candidate set, i.e., an MCS. Following [39, 40], we encode the formulas in a KB $\mathcal{K}$ by representing each atom x in a formula ϕ as `atomInFormula(`x`,`ϕ`)`, and the number of formulas as `numKbElements(`$|\mathcal{K}|$`)`. Atoms and formulas are modeled as `atom/1` and `kbElement/1`, respectively. For MV, we represent each atom $x_q \in \mathsf{At}(Q)$ as `queryAtom(`x_q`)`, and, for P, each formula $\phi_q \in Q$ as `queryFormula(`ϕ_q`)`. Then, e.g., a conjunction $\phi = \phi_1 \wedge \phi_2$ is encoded as `conjunction(`ϕ`,`ϕ_1`,`ϕ_2`)`. Truth values 1 (`t`) and 0 (`f`) are represented by `tv(t,f)`. The evaluation of (sub)formulas is encoded following the semantics of the connectives: e.g., a conjunction $\phi = \phi_1 \wedge \phi_2$ evaluates to 1 by `truthValueCS(F,t):- conjunction(F,G,H), truthValueCS(G,t), truthValueCS(H,t)`. Note that we need to avoid the use of `not`, due to the use of saturation [23]. To check if supersets of a candidate set are unsatisfiable, we refer to a specific superset, i.e., instead of `truthValueCS(F,t)`, we use `truthValueSet(F,S,t)`, etc. A candidate set S_{cs} containing at least one formula $\phi \in \mathcal{K}$ is guessed (line 1). We check that at least one atom (w.r.t. MV) is in the set-complement (lines 2–5). (For P, line 3 is omitted and `atomInComplement` (line 4) replaced by `inComplement`, and `queryAtom` by `queryFormula`.) To check S_{cs} for satisfiability, each atom in S_{cs} (line 6) gets a truth value (line 7); S_{cs} is satisfiable iff all $|S_{\text{cs}}|$ of its formulas evaluate to 1 (lines 8–9). Only satisfiable candidate sets can be derived (line 10). To ensure that each superset of S_{cs} is unsatisfiable, we define $|\mathcal{K}| - |S_{\text{cs}}|$ supersets (lines 11–12), and add exactly one element from the set-complement to each (line 13). No two supersets are equal by lines 14–15. Lines 16–24 check if all supersets of S_{cs} are unsatisfiable. Lines 16–17 and 18–19 determine formulas (respectively

Algorithm 2 SAT-based CEGAR for solving the MCS overlap problem. Input: knowledge base $\mathcal{K}$, query $Q \subseteq \mathcal{K}$.

```
1: $B \leftarrow \top$
2: while true do
3:     $(result, \tau) \leftarrow \mathrm{SAT}(\bigvee_{\phi \in Q} \neg\phi \wedge B)$
4:     if $result = unsat$ then return $\bot$
5:     $S \leftarrow \{\phi \in \mathcal{K} \mid \tau(\phi) = 1\}$
6:     while true do
7:         $B \leftarrow B \wedge \bigvee_{\phi \in \mathcal{K} \setminus S} \phi$
8:         $(result, \tau) \leftarrow \mathrm{SAT}(\bigvee_{\phi \in Q} \neg\phi \wedge \bigwedge_{\phi \in S} \phi \wedge B)$
9:         if $result = unsat$ then break else $S \leftarrow \{\phi \in \mathcal{K} \mid \tau(\phi) = 1\}$
10:    $(result, \tau) \leftarrow \mathrm{SAT}(\bigwedge_{\phi \in Q} \phi \wedge \bigwedge_{\phi \in S} \phi \wedge B)$
11:    if $result = unsat$ then return $\mathcal{K} \setminus S$
12:    else $S \leftarrow \{\phi \in \mathcal{K} \mid \tau(\phi) = 1\}$, $B \leftarrow B \wedge \bigvee_{\phi \in \mathcal{K} \setminus S} \phi$
```

atoms) in a given superset. The unsatisfiability check is done by *saturation*: the rule in line 20 allows the atoms in a superset to be both 1 and 0. If both 1 and 0 are derived for each atom (lines 21–22) and the formula evaluates to 0 (line 23), the formula is unsatisfiable. The constraint on line 24 enforces each superset to be unsatisfiable. If the disjunctive ASP program does not have an answer set, no MCS containing at least one formula from Q (w.r.t. P) or at least one atom from $\mathsf{At}(Q)$ (w.r.t. MV) exists, and Algorithm 1 terminates. Otherwise, we extract the corresponding formulas or atoms (represented by `inComplement/1` / `atomInComplement/1`), and remove them from Q.

3.3 Instantiation via SAT-based CEGAR

We detail SAT-based CEGAR as a second approach to MCS overlap. The key idea in SAT-based CEGAR is to overapproximate the solutions to the problem via a propositional abstraction. By iteratively solving the abstraction we obtain candidate solutions, which are subsequently verified. This is done by searching for a counterexample for the candidate solution being a valid solution to the problem. If a counterexample is found, the abstraction is refined by adding constraints which rule out the candidate solution. Our SAT-based CEGAR algorithm is closely related to an earlier-proposed approach that reduces MCS overlap to propositional circumscription [37, 36] and employs CEGAR for circumscription [35, 2, 3] (which in itself is not directly applicable as it only supports computations over sets of individual clauses).

The CEGAR algorithm (Algorithm 2) for the MCS overlap problem takes as input a KB $\mathcal{K}$ and a subset of query formulas $Q \subseteq \mathcal{K}$, with the goal of finding an MCS of $\mathcal{K}$ that intersects Q. This is equivalent to finding an MSS of $\mathcal{K}$ which excludes at least one $\phi \in Q$. As the abstraction, we drop the requirement on maximality, and consider satisfiable subsets of $\mathcal{K}$. To avoid finding assignments which do not correspond to such MSSes, we initialize a set B of blocking clauses (line 1). Since SAT solvers operate on formulas in conjunctive normal form (CNF), each $\phi \in \mathcal{K}$ is encoded in a standard way [62] to a set of clauses $\mathrm{Cls}(\phi)$ and a variable $\mathrm{Var}(\phi)$ so that ϕ is satisfiable iff $\mathrm{Cls}(\phi) \wedge \mathrm{Var}(\phi)$ is. Thus by initializing a SAT solver with $\bigwedge_{\phi \in \mathcal{K}} \mathrm{Cls}(\phi)$, we can query for the satisfiability of any subset $S \subseteq \mathcal{K}$ with additional unit clauses $\bigwedge_{\phi \in S} \mathrm{Var}(\phi)$.

In the main CEGAR loop (lines 2–12), we iteratively ask the SAT solver for an assignment which falsifies some $\phi \in \mathcal{K}$ (line 3). If there is no such assignment, there is no MCS which overlaps Q, and we return $\perp$ (line 4). Otherwise, a satisfying assignment gives a satisfiable subset S of $\mathcal{K}$ (line 5) and $\mathcal{K} \setminus S$ is a correction set. We subset-maximize S iteratively under the constraint that some $\phi \in \mathcal{K}$ is falsified (lines 6–9). Finally, we check if S is an MSS of $\mathcal{K}$ by asking for a counterexample, i.e., an assignment satisfying every ϕ in $Q \cap S$ (line 10). If there is no such assignment, $\mathcal{K} \setminus S$ is an MCS which intersects Q (line 11). Otherwise we block all subsets of the obtained satisfiable subset, including the candidate MSS S (line 12). The number of iterations is bounded by the number of candidate MSSes, and Algorithm 2 terminates. Note that the CEGAR approach allows for several optimizations, in addition to using an incremental SAT solver. Since so-called autark variables cannot be included in any MUS [38], the lean kernel, i.e., the set of clauses not touched by any autarky [52, 12], is an overapproximation of the union of MUSes. A *maximum autarky* $\mathcal{A}$ of $\mathcal{K}$ is obtained with an MCS extraction call [49]; $\mathcal{A}$ can be safely removed from every query Q in Algorithm 1. Further, *disjoint cores* can be extracted by iteratively querying the SAT solver for pairwise disjoint MUSes; their union $\mathcal{D}$ is an underapproximation of the union of MUSes and hence the elements (formulas for P, atoms for MV) in $\mathcal{D}$ are known to be covered in the set $\mathcal{C}$.

4 Empirical Evaluation

For implementations of the SAT-based CEGAR and ASP instantiations of Algorithm 1, see `https://bitbucket.org/coreo-group/sat4im`. We use the ASP solver Clingo [24] 5.5.1, and we implemented the SAT-based CEGAR approach via PySAT [33] 0.1.8.dev3 using the CaDiCaL 1.5.3 [16] SAT solver. We compare the performance of the ASP and SAT-based CEGAR instantiations to mcscache [58] as a state-of-the-art MCS enumerator and umuser [52] computing the union of MUSes. Each KB formula $\phi \in \mathcal{K}$ is encoded into CNF via $\mathrm{Cls}(\phi)$ and $\mathrm{Var}(\phi)$, so that MCSes (resp. MUSes) of $\mathcal{K}$ can be computed as group-MCSes (resp. group-MUSes [46, 54]) over $\{\mathrm{Var}(\phi) \mid \phi \in \mathcal{K}\}$ with $\bigwedge_{\phi \in \mathcal{K}} \mathrm{Cls}(\phi)$ as hard constraints. mcscache extracts one MCS of the KB at a time. We keep track of the set of formulas (P) or variables (MV) currently covered by some MCS. We terminate mcscache once all of the elements are covered.

We consider three variants of CEGAR: (i) CEGAR: with subset-maximization, disjoint cores, and autarky trimming. (ii) CEGAR/no CM: Subset-maximization of candidate MSSes (lines 6–9) is disabled, instead the SAT solver is directly asked for a counterexample (line 10) and instead (line 11), the satisfiable subset for an MSS is maximized. (iii) CEGAR/no AT: No *autarky trimming*.

Table 1: Number of solved instances (#solved) and cumulative runtimes (CRT).

		SRS (90 KBs)		ML (100 KBs)		ARG (100 KBs)	
	Approach	**#solved**	**CRT (s)**	**#solved**	**CRT (s)**	**#solved**	**CRT (s)**
P	CEGAR	**87**	1300.16	**55**	2576.94	51	298.78
	CEGAR/no AT	**87**	1302.10	46	19.98	**52**	516.61
	CEGAR/no CM	85	937.42	52	796.21	43	198.07
	mcscache	**87**	1318.20	46	27.86	42	365.05
	umuser	71	699.87	38	11.32	40	455.72
	ASP	21	1783.06	15	651.65	10	203.71
MV	CEGAR	**90**	4.99	**93**	4775.11	**53**	453.82
	CEGAR/no AT	**90**	5.07	46	20.17	52	629.98
	CEGAR/no CM	**90**	5.62	85	2353.93	51	356.5
	mcscache	**90**	6.72	46	29.06	46	387.78
	umuser	71	699.87	38	11.32	40	455.72
	ASP	36	1119.48	17	2645.44	10	339.04

We use KBs from three sources. (i) SRS [42, 39, 40]: 90 KBs, generated using *SyntacticRandomSampler* from `https://tweetyproject.org/`, under 9 parameter combinations, randomly selecting 10 KBs per combination; (ii) ML [39, 40]: 100 randomly selected KBs from the *Animals with Attributes* dataset (`http://attributes.kyb.tuebingen.mpg.de`), interpreting association rules mined with Apriori [1] as implications; (iii) ARG [40]: 100 randomly selected KBs consisting of CNF clauses of a standard SAT encoding [15] for stable extensions of abstract argumentation frameworks [22] from the ICCMA 2019 competition with the constraint that a random subset of 20 % of arguments are in the stable extension. The experiments were run on Intel Xeon E5-2643 v3 3.40-GHz CPUs with 192-GB RAM under Ubuntu 20.04.5 using a per-instance 900-s time limit.

The CEGAR approach performs the best, followed by mcscache and umuser; see Table 1. Default CEGAR performs consistently well, solving significantly more instances in particular on the ML and ARG datasets. Disjunctive ASP solves significantly fewer instances than the other approaches. For CEGAR, disabling autarky trimming (CEGAR/no AT) leads to more timeouts especially on ML benchmarks and MV. Disabling subset-maximization (CEGAR/no CM) also yields more timeouts, especially on the ARG dataset and P. (Disjoint cores did not have a noticeable impact.) Every benchmark instance solved by mcscache is also solved by the CEGAR approach, with the exception of a single ARG instance for the P measure. CEGAR altogether outperforms mcscache on a great majority of the benchmarks.

Overall, the CEGAR approach empirically outperformed ASP as well as state-of-the-art MCS enumerators. Our results motivate the development of specialized algorithms for other computationally notably complex inconsistency measures, such as ones based on counting MCSes [13] and MUSes [12, 14].

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. VLDB'94. pp. 487–499 (1994)
2. Alviano, M.: Model enumeration in propositional circumscription via unsatisfiable core analysis. Theory Pract. Log. Program. **17**(5-6), 708–725 (2017)
3. Alviano, M.: Query answering in propositional circumscription. In: Proc. IJCAI 2018. pp. 1669–1675. ijcai.org (2018)
4. Bacchus, F., Davies, J., Tsimpoukelli, M., Katsirelos, G.: Relaxation search: A simple way of managing optional clauses. In: Proc. AAAI 2014. pp. 835–841. AAAI Press (2014)
5. Belov, A., Ivrii, A., Matsliah, A., Marques-Silva, J.: On efficient computation of variable MUSes. In: Proc. SAT 2012. Lecture Notes in Computer Science, vol. 7317, pp. 298–311. Springer (2012)
6. Bendík, J.: On decomposition of maximal satisfiable subsets. In: Proc. FMCAD 2021. pp. 212–221. IEEE (2021)
7. Bendík, J., Benes, N., Cerná, I., Barnat, J.: Tunable online MUS/MSS enumeration. In: Proc. FSTTCS 2016. LIPIcs, vol. 65, pp. 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)
8. Bendík, J., Cerná, I.: MUST: minimal unsatisfiable subsets enumeration tool. In: Proc. TACAS 2020. Lecture Notes in Computer Science, vol. 12078, pp. 135–152. Springer (2020)
9. Bendík, J., Cerná, I.: Replication-guided enumeration of minimal unsatisfiable subsets. In: Proc. CP 2020. Lecture Notes in Computer Science, vol. 12333, pp. 37–54. Springer (2020)
10. Bendík, J., Cerna, I.: Rotation based MSS/MCS enumeration. In: Proc. LPAR 2020. EPiC Series in Computing, vol. 73, pp. 120–137. EasyChair (2020)
11. Bendík, J., Cerná, I., Benes, N.: Recursive online enumeration of all minimal unsatisfiable subsets. In: Proc. ATVA 2018. Lecture Notes in Computer Science, vol. 11138, pp. 143–159. Springer (2018)
12. Bendík, J., Meel, K.S.: Approximate counting of minimal unsatisfiable subsets. In: Proc. CAV 2020. Lecture Notes in Computer Science, vol. 12224, pp. 439–462. Springer (2020)
13. Bendík, J., Meel, K.S.: Counting maximal satisfiable subsets. In: Proc. AAAI 2021. pp. 3651–3660. AAAI Press (2021)
14. Bendík, J., Meel, K.S.: Counting minimal unsatisfiable subsets. In: Proc. CAV 2021. Lecture Notes in Computer Science, vol. 12760, pp. 313–336. Springer (2021)
15. Besnard, P., Doutre, S., Herzig, A.: Encoding argument graphs in logic. In: Laurent, A., Strauss, O., Bouchon-Meunier, B., Yager, R.R. (eds.) Proc. IPMU 2014. Communications in Computer and Information Science, vol. 443, pp. 345–354. Springer (2014)
16. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
17. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
18. Chen, Z., Ding, D.: Variable minimal unsatisfiability. In: Proc. TAMC 2006. Lecture Notes in Computer Science, vol. 3959, pp. 262–273. Springer (2006)

19. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003)
20. Clarke, E.M., Gupta, A., Strichman, O.: SAT-based counterexample-guided abstraction refinement. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **23**(7), 1113–1123 (2004)
21. Doder, D., Raskovic, M., Markovic, Z., Ognjanovic, Z.: Measures of inconsistency and defaults. Int. J. Approx. Reason. **51**(7), 832–845 (2010)
22. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**(2), 321–358 (1995)
23. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Ann. Math. Artif. Intell. **15**, 289–323 (1995)
24. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory solving made easy with Clingo 5. In: Technical Communications of ICLP. pp. 2:1–2:15. OASICS, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
25. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer set solving in practice. Synthesis Lectures on Artificial Intelligence and Machine Learning **6**(3), 1–238 (2012)
26. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. ICLP/SLP. pp. 1070–1080. MIT Press (1988)
27. Grant, J.: Classifications for inconsistent theories. Notre Dame J. Formal Log. **19**(3), 435–444 (1978)
28. Grant, J., Hunter, A.: Measuring consistency gain and information loss in stepwise inconsistency resolution. In: Proc. ECSQARU 2011. pp. 362–373. Springer (2011)
29. Grant, J., Martinez, M.V. (eds.): Measuring Inconsistency in Information, Studies in Logic, vol. 73. College Publications (2018)
30. Grégoire, É., Izza, Y., Lagniez, J.: Boosting MCSes enumeration. In: Proc. IJCAI 2018. pp. 1309–1315. ijcai.org (2018)
31. Grégoire, É., Lagniez, J., Mazure, B.: An experimentally efficient method for (MSS, CoMSS) partitioning. In: Proc. AAAI 2014. pp. 2666–2673. AAAI Press (2014)
32. Hunter, A., Konieczny, S.: Measuring inconsistency through minimal inconsistent sets. In: Proc. KR 2008. pp. 358–366. AAAI Press (2008)
33. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: A Python toolkit for prototyping with SAT oracles. In: Proc. SAT 2018. Lecture Notes in Computer Science, vol. 10929, pp. 428–437. Springer (2018)
34. Jabbour, S., Ma, Y., Raddaoui, B., Sais, L., Salhi, Y.: A MIS partition based framework for measuring inconsistency. In: Baral, C., Delgrande, J.P., Wolter, F. (eds.) Proc. KR 2016. pp. 84–93. AAAI Press (2016)
35. Janota, M., Grigore, R., Marques-Silva, J.: Counterexample guided abstraction refinement algorithm for propositional circumscription. In: Proc. JELIA 2010. Lecture Notes in Computer Science, vol. 6341, pp. 195–207. Springer (2010)
36. Janota, M., Marques-Silva, J.: cmMUS: A tool for circumscription-based MUS membership testing. In: Proc. LPNMR 2011. Lecture Notes in Computer Science, vol. 6645, pp. 266–271. Springer (2011)
37. Janota, M., Marques-Silva, J.: On deciding MUS membership with QBF. In: Proc. CP 2011. Lecture Notes in Computer Science, vol. 6876, pp. 414–428. Springer (2011)
38. Kleine Büning, H., Kullmann, O.: Minimal unsatisfiability and autarkies. In: Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 571–633. IOS Press (2021)

39. Kuhlmann, I., Gessler, A., Laszlo, V., Thimm, M.: A comparison of ASP-based and SAT-based algorithms for the contension inconsistency measure. In: Proc. SUM 2022. pp. 139–153. Springer (2022)
40. Kuhlmann, I., Gessler, A., Laszlo, V., Thimm, M.: Comparison of SAT-based and ASP-based algorithms for inconsistency measurement. arXiv p. 2304.14832 (2023), preprint
41. Kuhlmann, I., Thimm, M.: An algorithm for the contension inconsistency measure using reductions to answer set programming. In: Proc. SUM 2020. pp. 289–296. Springer (2020)
42. Kuhlmann, I., Thimm, M.: Algorithms for inconsistency measurement using answer set programming. In: Proc. NMR 2021. pp. 159–168 (2021)
43. Kullmann, O.: Constraint satisfaction problems in clausal form II: minimal unsatisfiability and conflict structure. Fundam. Informaticae **109**(1), 83–119 (2011)
44. Liberatore, P.: Redundancy in logic I: CNF propositional formulae. Artif. Intell. **163**(2), 203–232 (2005)
45. Liffiton, M.H., Previti, A., Malik, A., Marques-Silva, J.: Fast, flexible MUS enumeration. Constraints An Int. J. **21**(2), 223–250 (2016)
46. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. J. Autom. Reason. **40**(1), 1–33 (2008)
47. Lifschitz, V.: Answer set programming. Springer Berlin (2019)
48. Marques-Silva, J., Heras, F., Janota, M., Previti, A., Belov, A.: On computing minimal correction subsets. In: Proc. IJCAI 2013. pp. 615–622. IJCAI/AAAI (2013)
49. Marques-Silva, J., Ignatiev, A., Morgado, A., Manquinho, V.M., Lynce, I.: Efficient autarkies. In: Proc. ECAI 2014. Frontiers in Artificial Intelligence and Applications, vol. 263, pp. 603–608. IOS Press (2014)
50. Marques-Silva, J., Mencía, C.: Reasoning about inconsistent formulas. In: Proc. IJCAI 2020. pp. 4899–4906. ijcai.org (2020)
51. Mencía, C., Ignatiev, A., Previti, A., Marques-Silva, J.: MCS extraction with sublinear oracle queries. In: Proc. SAT 2016. Lecture Notes in Computer Science, vol. 9710, pp. 342–360. Springer (2016)
52. Mencía, C., Kullmann, O., Ignatiev, A., Marques-Silva, J.: On computing the union of muses. In: Proc. SAT 2019. Lecture Notes in Computer Science, vol. 11628, pp. 211–221. Springer (2019)
53. Mencía, C., Previti, A., Marques-Silva, J.: Literal-based MCS extraction. In: Proc. IJCAI 2015. pp. 1973–1979. AAAI Press (2015)
54. Nadel, A.: Boosting minimal unsatisfiable core extraction. In: Proc. FMCAD 2010. pp. 221–229. IEEE (2010)
55. Narodytska, N., Bjørner, N.S., Marinescu, M.V., Sagiv, M.: Core-guided minimal correction set and core enumeration. In: Proc. IJCAI 2018. pp. 1353–1361. ijcai.org (2018)
56. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25**(3-4), 241–273 (1999)
57. Niskanen, A., Kuhlmann, I., Thimm, M., Järvisalo, M.: MaxSAT-Based inconsistency measurement. In: Proc. ECAI 2023. IOS Press (2023)
58. Previti, A., Mencía, C., Järvisalo, M., Marques-Silva, J.: Improving MCS enumeration via caching. In: Proc. SAT 2017. Lecture Notes in Computer Science, vol. 10491, pp. 184–194. Springer (2017)
59. Previti, A., Mencía, C., Järvisalo, M., Marques-Silva, J.: Premise set caching for enumerating minimal correction subsets. In: Proc. AAAI 2018. pp. 6633–6640. AAAI Press (2018)

60. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
61. Thimm, M., Wallner, J.P.: On the complexity of inconsistency measurement. Artif. Intell. **275**, 411–456 (2019)
62. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of Reasoning, pp. 466–483. Springer Berlin Heidelberg (1983)
63. Xiao, G., Ma, Y.: Inconsistency measurement based on variables in minimal unsatisfiable subsets. In: Proc. ECAI 2012, pp. 864–869. IOS Press (2012)