Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

On the Extraction of One Maximal Information Subset That Does Not Conflict with Multiple Contexts

Éric Grégoire, Yacine Izza, Jean-Marie Lagniez
CRIL
Université d'Artois & CNRS
rue Jean Souvraz SP18
F-62307 Lens France
{gregoire,izza,lagniez}@cril.fr

Abstract

The efficient extraction of one maximal information subset that does not conflict with multiple contexts or additional information sources is a key basic issue in many A.I. domains, especially when these contexts or sources can be mutually conflicting. In this paper, this question is addressed from a computational point of view in clausal Boolean logic. A new approach is introduced that experimentally outperforms the currently most efficient technique.

Introduction

The efficient extraction of one maximal information subset that does not conflict with multiple contexts or additional information sources is a key basic issue in many A.I. domains, especially when these contexts or sources can be mutually conflicting. As illustrated in (Besnard, Grégoire, and Lagniez 2015), it is central in belief change, model-based diagnosis, planning, decision making, argumentation and non-monotonic reasoning, among other A.I. fields.

Consider for example a logic-based agent that needs to take her decisions under very tight time constraints and on the basis of the part of her information that does not conflict with mutually conflicting assumptions about what she does not know for sure. Specifically, her decisions must be compatible with any of these assumptions since it can eventually be true. Due to the time constraints, she quickly extracts one maximal information subset that does not contradict any assumption and considers this subset as a satisfactory basis for decision making. Indeed, every assumption remains compatible with this set and adopting any additional piece of the available information would violate at least one of the assumptions. Alternatively, this extraction can be the kernel of a method that aims at enumerating all these maximal subsets when enough computing time is available. It can also be adapted to obey some preference ordering between information pieces.

The extraction of one maximal information subset that does not contradict possibly mutually conflicting sources can also play a role in negotiation and multi-agent systems since it can amount to finding one maximal subset that does not conflict with any goal of any of the involved agents or negotiators.

In this paper, this issue is investigated from a computational point of view in clausal Boolean logic, when maximality is about set-theoretic inclusion. An original method is proposed that experimentally outperforms the most efficient current technique.

The paper is organized as follows. In the next section, the main logical and computational concepts used in the paper are recalled. Then, the problem is defined formally. The currently most efficient approaches are described before our original algorithm is introduced in a progressive way. Experimentations that compare the new approach with its competitors are then reported and discussed. Finally, the conclusion elaborates on promising paths for further research.

Technical Background

We consider standard clausal Boolean logic. $\neg$, $\vee$, $\wedge$ and $\Rightarrow$ represent the negation, disjunction, conjunction and material implication connective, respectively. A clause is a disjunction of literals and literals are possibly negated Boolean variables, denoted by $a, b \ldots$, which can thus be assigned either *true* or *false*. Clauses are denoted by $\alpha, \beta, \gamma, \ldots$ Sets of clauses are denoted by $\Delta, \Sigma, \ldots$ Sets of sets of clauses are denoted by $\mathcal{C}, \mathcal{D}, \ldots$ The cardinality of a set Δ is written *card*(Δ). A set of clauses Δ is *satisfiable* iff there exists at least one model of Δ, namely a truth assignment of all Boolean variables of Δ making all clauses of Δ to be *true* according to usual compositional rules. SAT is the NP-complete problem that consists in checking whether a finite set of clauses is satisfiable (Cook 1971).

From now on, we assume that Δ is a finite set of clauses, that Φ and Ψ are subsets of Δ and that $\mathcal{C}$ is a finite set of finite sets Γ_i of Boolean clauses, each of the Γ_i being (individually) satisfiable and representing one context. Key concepts in this paper are inclusion-Maximal and Minimal Correction Subsets.

Definition 1. Φ is an (inclusion-)Maximal Satisfiable Subset of Δ (in short, Φ is a $\mathrm{MSS}(\Delta)$) iff Φ is satisfiable and $\forall \alpha \in \Delta \setminus \Phi$, $\Phi \cup \{\alpha\}$ is unsatisfiable.

A Minimal Correction Subset (MCS) (also called Co-MSS) of Δ is the set-theoretical complement in Δ of the corresponding MSS.

Definition 2. Ψ is a Minimal Correction Subset of Δ (in short, Ψ is a MCS(Δ)) iff $\Psi = \Delta \setminus \Phi$ where Φ is a MSS of Δ.

Accordingly, Δ can always be partitioned into a pair made of one MSS and one MCS. Unless P=NP, extracting one such partition is intractable in the worst case since this problem belongs to the $FP^{NP}[wit, log]$ class, i.e., the set of function problems that can be computed in polynomial time by executing a logarithmic number of calls to an NP oracle that returns a witness for the positive outcome (Marques-Silva and Janota 2014). Techniques to compute one such partition that prove very often efficient are described for example in (Grégoire, Lagniez, and Mazure 2014; Marques-Silva et al. 2013; Mencía, Previti, and Marques-Silva 2015). Note that in the worst case the number of MSSes is exponential in the number of clauses in Δ.

The well-known concept of Minimally Unsatisfiable Subset (MUS) of an unsatisfiable set of clauses Δ will also be useful in the sequel: one MUS of Δ is an unsatisfiable subset of Δ that is minimal in the sense that dropping any clause from the subset makes this latter one become satisfiable. See for example (Bailey and Stuckey 2005; Birnbaum and Lozinskii 2003; Previti and Marques-Silva 2013; Liffiton and Malik 2013; Belov, Heule, and Marques-Silva 2014; Grégoire, Mazure, and Piette 2009; Liffiton and Sakallah 2008) among numerous studies about MUS.

Problem Statement

We follow (Besnard, Grégoire, and Lagniez 2015) where Maximal Satisfiable and Minimal Correction Subsets *under a set of assumptive contexts* are defined as follows.

Definition 3. Φ is a Maximal Satisfiable Subset of Δ under a set of assumptive contexts $\mathcal{C}$, (Φ is an AC-MSS($\Delta, \mathcal{C}$) for short), iff
1. Φ is a satisfiable subset of Δ, and
2. $\forall\, \Gamma_i \in \mathcal{C}, \Phi \cup \Gamma_i$ is satisfiable, and
3. $\forall \alpha \in \Delta \setminus \Phi$, $\Phi \cup \{\alpha\} \cup \Gamma_i$ is unsatisfiable for some $\Gamma_i \in \mathcal{C}$.

Example 1. *Let* $\Delta = \{a \vee b, a \vee c, d \vee b, e \vee c, \neg b, \neg c\}$ *and* $\mathcal{C} = \{\{\neg a\}, \{\neg d\}, \{\neg e\}\}$. *Note that this example is simple in the sense that (1)* Δ *is satisfiable, (2) all contexts are merely unary clauses, and (3) the contexts are not mutually logically conflicting. In the general case these three properties need not be satisfied. Fig. 1 depicts all clauses from both sets. In this example, the figure shows how the various contexts of* $\mathcal{C}$ *conflict with clauses in* Δ. *The encircled sets of clauses represent the MUSes from* $\Delta \cup \mathcal{C}$ *containing one context from* $\mathcal{C}$. *In this example,* $\{a \vee b, a \vee c, d \vee b, e \vee c\}$ *is one* `AC-MSS`$(\Delta, \mathcal{C})$, *among several ones.*

Definition 4. Ψ is a Minimal Correction Subset of Δ under a set of assumptive contexts $\mathcal{C}$, denoted AC-MCS($\Delta, \mathcal{C}$), iff $\Delta = \Psi \cup \Phi$ where Φ is an AC-MSS($\Delta, \mathcal{C}$).

Example 2. *In Example 1,* $\{a \vee b, a \vee c, d \vee b, e \vee c\}$, $\{a \vee b, d \vee b, \neg c\}$, $\{a \vee c, e \vee c, \neg b\}$ *and* $\{\neg c, \neg b\}$ *are all* `AC-MCS`$(\Delta, \mathcal{C})$.

We will focus on the extraction of either one AC-MSS($\Delta, \mathcal{C}$) or one AC-MCS($\Delta, \mathcal{C}$) interchangeably in the paper, according to the corresponding algorithm that is the easiest one to present. When one such set is obtained, its dual can be derived in a straightforward manner since they partition Δ.

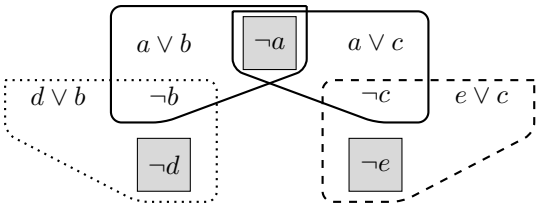

Figure 1: Example 1.

Useful approximations of AC-MSS($\Delta, \mathcal{C}$) and AC-MCS($\Delta, \mathcal{C}$) are defined as follows.

Definition 5. Φ is a *Satisfiable Subset* of Δ under a set of assumptive contexts $\mathcal{C}$, in short AC-SS($\Delta, \mathcal{C}$), iff there exists at least one set of clauses Φ' s.t. $\Phi \subseteq \Phi' \subseteq \Delta$ and Φ' is an AC-MSS($\Delta, \mathcal{C}$).

Definition 6. Ψ is a *Correction Subset* of Δ under a set of assumptive contexts $\mathcal{C}$, in short AC-CS($\Delta, \mathcal{C}$), iff there exists at least one set of clauses Ψ' s.t. $\Psi' \subseteq \Psi \subseteq \Delta$ and Ψ' is an AC-MCS($\Delta, \mathcal{C}$).

Notice that neither Δ nor the conjunction of all the assumptive contexts Γ_i of $\mathcal{C}$ are required to be satisfiable in the definitions. The Γ_i are used in a point-wise manner: any AC-MSS($\Delta, \mathcal{C}$) must be satisfiable together with any context Γ_i, taken individually. Examples showing why the Γ_i cannot be replaced by one single context made of their conjunction or disjunction, even when the Γ_i do not conflict one another, are given in (Besnard, Grégoire, and Lagniez 2015).

Transformational Approach

The currently most efficient technique to extract one AC-MSS($\Delta, \mathcal{C}$) for large Δ has been proposed in (Besnard, Grégoire, and Lagniez 2015). It was motivated by the limits of the `Greedy-AC-SS` procedure, which is useful to recall here. `Greedy-AC-SS` starts with one MSS of Δ as temporary result, denoted Δ'. Then, it considers each Γ_i, iteratively. At each iteration step, it expels a minimal number of clauses from Δ' so that Δ' becomes satisfiable with Γ_i. However, the final set Δ' is not necessary an AC-MSS($\Delta, \mathcal{C}$). Indeed, some clauses that are dropped at some iteration step can sometimes render unnecessary the previous elimination of other clauses. Actually, `Greedy-AC-SS`$(\Delta, \mathcal{C})$ merely partitions Δ into one AC-CS($\Delta, \mathcal{C}$) and one AC-SS($\Delta, \mathcal{C}$).

Example 3. *Let* $\Delta = \{\neg a \vee b, \neg b, b \vee d\}$. *Let* $\mathcal{C} = \{\Gamma_1 = \{a\}, \Gamma_2 = \{\neg d\}\}$. `Greedy-AC-SS`$(\Delta, \mathcal{C})$ *works as follows. Assume that* Γ_1 *is considered first and* $\neg a \vee b$ *is selected and dropped from* Δ *at this step. Then, assume that* $\neg b$ *is dropped to make* Δ' *become satisfiable with* Γ_2. *The final set* $\Delta' = \{b \vee d\}$ *is not an* AC-MSS($\Delta, \mathcal{C}$). *Indeed, only dropping* $\neg b$ *from* Δ *would have delivered the bigger set* $\{\neg a \vee b, b \vee d\}$, *which appears to be one* AC-MSS($\Delta, \mathcal{C}$).

Accordingly, (Besnard, Grégoire, and Lagniez 2015) considered one elementary brute-force approach to extract one AC-MSS$(\Delta, \mathcal{C})$. This approach amounts to considering all possible orderings for the different Γ_i and for each ordering, to computing every possible solution of `Greedy-AC-SS`, before a final solution is extracted. As there can be an exponential number of MSSes in a set of clauses, this approach is intractable both in the worst case and in many expectedly-easier situations.

To address this computational blow up, at least to some extent, (Besnard, Grégoire, and Lagniez 2015) have proposed an encoding schema that allows the problem to be rewritten into *one* single computation of one (mere) MSS together with some additional calls to a SAT solver. More precisely, when m is the number of Γ_i in $\mathcal{C}$, n' the largest number of clauses in any Γ_i, n the number of clauses in Δ, that method requires in the worst case $O(m)$ calls to a SAT-solver on an instance of size $O(n + n')$, plus a logarithmic number of calls to a SAT-solver on an instance of initial size $O(m(n + n'))$ that is divided by two at each call. Not surprisingly, this `Transformational Approach` appears more efficient than the brute-force one for many instances. But, as the authors recognize it, large numbers m of different Γ_i remain problematic since the size of the transformed instance is proportional to m.

A Basic Incremental Algorithm

Algorithm 1: `Incremental`$_1$`-AC-MSS`

Input : Δ: a set of clauses;
$\mathcal{C} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$: a set of satisfiable sets of clauses;
Output : Φ s.t. Φ is an `AC-MSS`$(\Delta, \mathcal{C})$;

```
1 Φ ← Greedy-AC-SS(Δ, C);  Ψ ← (Δ \ Φ);
2 foreach α ∈ Ψ do
3     all-Γi-satisfied ← true;    i ← 1;
4     while i ≤ n and all-Γi-satisfied do
5         all-Γi-satisfied ← Φ ∪ {α} ∪ Γi is satisfiable;
6         i ← i + 1;
7     if all-Γi-satisfied then  Φ ← Φ ∪ {α};
8 return Φ;
```

Actually, it is possible to avoid both the Transformational approach and the brute-force one in order to partition Δ into one AC-MSS$(\Delta, \mathcal{C})$ and one AC-MCS$(\Delta, \mathcal{C})$.

`Incremental`$_1$`-AC-MSS` depicted in Algorithm 1 delivers such an alternative method. It starts with one AC-SS$(\Delta, \mathcal{C})$, denoted Φ and delivered by the aforementioned `Greedy-AC-SS` procedure. Then, it augments Φ with clauses α of $\Delta \setminus \Phi$ such that, at each step, $\Phi \cup \{\alpha\}$ is satisfiable with every Γ_i. The correctness of this algorithm is easily established by considering the three items of Definition 3: the satisfiability of Φ is ensured at each step; the final set Φ does not contradict any Γ_i by construction and every clause α from $\Delta \setminus \Phi$ is such that $\Phi \cup \{\alpha\} \cup \Gamma_i$ is unsatisfiable for at least one Γ_i.

Example 4. *Let us run* `Incremental`$_1$`-AC-MSS` *on Example 1. Assume that* $\{a \vee b, a \vee c, e \vee c\}$ *is delivered by* `Greedy-AC-SS`*.* Φ *is initialized with this set, and thus* $\Psi = \{d \vee b, \neg c, \neg b\}$ *(line 1). Each clause from* Ψ *is then considered, successively. When* $\alpha = d \vee b$, $\Phi \cup \{\alpha\}$ *is satisfiable with each context and* Φ *is thus augmented with* α*. When* $\alpha = \neg c$ *(and when* $\alpha = \neg b$*), as* $\Phi \cup \{\alpha\}$ *is unsatisfiable,* α *is not inserted in the solution under construction.*

Independently of the computational cost of the `Greedy-AC-SS` procedure, `Incremental`$_1$`-AC-MSS` requires mn calls to a SAT solver in the worst case, where m and n are $card(\mathcal{C})$ and $card(\Delta)$, respectively.

Algorithm 2: `Greedy-AC-SS`

Input : Δ: a set of clauses; $\mathcal{C}$: a set of satisfiable sets of clauses;
Output: $\Phi \subseteq \Delta$ s.t. Φ is one `AC-SS`$(\Delta, \mathcal{C})$;

```
1 Ψ ← Δ;  Φ ← ∅;  cpt ← 0;
2 repeat
3     ϒ ← Ψ;
4     foreach Γi ∈ C do
5         Let I be a model of Γi ∪ Φ;
6         ϒ ← ϒ ∩ {α ∈ Ψ s.t. I(α) = true};
7     Φ ← Φ ∪ ϒ;  Ψ ← Ψ \ ϒ;  cpt ← cpt + 1;
8 until ϒ = ∅ or cpt > #iteration-max ;
9 return Φ;
```

Actually, we have opted for an experimentally more efficient version of the `Greedy-AC-SS` procedure (see Algorithm 2). It refines through a main iteration loop a partition (Φ,Ψ) of Δ by incrementally extending Φ, the current `AC-SS`$(\Delta, \mathcal{C})$, and downsizing Ψ, the current `AC-CS`$(\Delta, \mathcal{C})$. The exit conditions are either the absence of any refinement during the last iteration step or a preset maximal number of iterations that has been reached. For each Γ_i, a model I of $\Gamma_i \cup \Phi$ is searched. Υ records all the clauses of Ψ that are satified by all those I; Φ (resp. Ψ) is then assigned $\Psi \cup \Upsilon$ (resp. $\Psi \setminus \Upsilon$). Following (Grégoire, Lagniez, and Mazure 2014) the selection for initial interpretations is performed by calling a SAT solver on Δ; the so-called *progress saving* interpretation, which gives the largest encountered number of satisfied clauses, gives an initial assignment of variables for the search of a model of $\Gamma_i \cup \Phi$.

We will present and discuss the actual performance of the `Incremental`$_1$`-AC-MSS` algorithm with this pretreatment on benchmarks later in the paper. First, let us present the more elaborate technique that we have developed and that is experimentally more efficient.

Useful Properties

To build more efficient incremental-like algorithms, we focus on computationally-exploitable circumstances under which some additional clauses can be permanently moved inside the AC-MSS$(\Delta, \mathcal{C})$ or the AC-MCS$(\Delta, \mathcal{C})$, when these sets are under progressive construction. From now on, we will focus on the extraction of one AC-MCS$(\Delta, \mathcal{C})$ since this will ease the presentation of our results.

A first useful property is intuitively as follows. *If we discover a clause* α *such that the current AC-CS augmented with* α *makes the corresponding AC-SS become now satisfiable not only with an additional context but with all of them then we can augment the current AC-CS with* α*. Indeed,in this case there always exists at least one AC-MCS that is*

included in the current AC-CS augmented with α. Furthermore, all these AC-MCS necessarily contain α. Formally:

Property 1. *Assume that $\Theta \subset \Delta$ and that $\alpha \in (\Delta \setminus \Theta)$.*

If $\exists\, \mathcal{C}'$ s.t. $\mathcal{C}' \subseteq \mathcal{C}$ and $\exists\, \Gamma_j \in \mathcal{C}'$ s.t. $\forall\, \Gamma_i \in \mathcal{C} \setminus \mathcal{C}'$

1. *$(\Delta \setminus \Theta) \cup \Gamma_i$ is satisfiable;*
2. *$(\Delta \setminus \Theta) \cup \Gamma_j$ is unsatisfiable;*
3. *$\forall\, \Gamma_k \in \mathcal{C}'$, $(\Delta \setminus (\Theta \cup \{\alpha\})) \cup \Gamma_k$ is satisfiable*

then there exists at least one $\texttt{AC-MCS}(\Delta, \mathcal{C})$, say Ψ, s.t.
(1) $\Psi \subseteq \Theta \cup \{\alpha\}$, and
(2) $\forall \Psi$ that are $\texttt{AC-MCS}(\Delta, \mathcal{C})$ s.t. $\Psi \subseteq \Theta \cup \{\alpha\}$: $\alpha \in \Psi$.

To grasp the intuition of why this property holds, simply notice that (1) follows in part from the fact that, as expressed by condition 3, removing α from Δ will allow the so-far unsatisfied or unchecked contexts to become satisfied. (2) is easily understood by reasoning by contradiction. Assume at the same time $\exists\ \Psi$ such that Ψ is one $\texttt{AC-MCS}(\Delta, \mathcal{C})$, $\Psi \subseteq \Theta \cup \{\alpha\}$ and $\alpha \notin \Psi$. Accordingly, $\Psi \subseteq \Theta$ and thanks to condition 2 we have that $(\Delta \setminus \Psi) \cup \Gamma_j$ is unsatisfiable, which contradicts the hypothesis that Ψ is one $\texttt{AC-MCS}(\Delta, \mathcal{C})$.

Example 5. *Consider Δ and $\mathcal{C}$ from Example 1. Assume that $\Theta = \{a \vee b, d \vee b, \neg b\}$, $\mathcal{C}' = \{\{\neg a\}, \{\neg e\}\}$, $\alpha = \neg c$ and $\Gamma_j = \{\neg a\}$. All conditions for Property 1 to apply are satisfied. Thus, α belongs to all the $\texttt{AC-MCS}$ that can be obtained from Δ and $\mathcal{C}$ and that are included in $\{a \vee b, d \vee b, \neg b, \neg c\}$ (i.e., $\{\{a \vee b, b \vee d, \neg c\}, \{\neg b, \neg c\}\}$).*

Actually, we have exploited a generalization of this property in order to propose an algorithm that extracts one AC-MCS$(\Delta, \mathcal{C})$. The following notation *à la* Partial-MaxSAT is useful to express the extended property.

Definition 7. Let Ω_1 and Ω_2 be two sets of clauses, where Ω_1 is satisfiable. The $\texttt{Partial-MCS}(\Omega_1, \Omega_2)$ procedure delivers one inclusion-minimal set of clauses Σ of Ω_2 s.t. $\Omega_1 \cup (\Omega_2 \setminus \Sigma)$ is satisfiable. The clauses from Ω_1 are called strict and clauses of Ω_2 are called soft. For convenience, we will also use the name of this procedure to represent its output.

Interestingly, $\texttt{Partial-MCS}(\Omega_1, \Omega_2)$ is thus a variant of a procedure that extracts one MCS, and can be implemented as such: it can naturally benefit from the aforementioned recent practical progress about computing MCSes (e.g., (Grégoire, Lagniez, and Mazure 2014; Mencía, Previti, and Marques-Silva 2015)).

The generalization is as follows. We can replace α in Property 1 by any $\texttt{Partial-MCS}((\Delta \setminus \Upsilon) \cup \Gamma_j, \Upsilon \setminus \Theta)$. *Intuitively, we can augment the AC-CS under construction with a minimal set of clauses that allows the corresponding AC-SS to become satisfiable with an additional context, provided that the augmented AC-CS actually makes the corresponding AC-SS become satisfiable with all contexts.*

Property 2. *Assume that Υ is an $\texttt{AC-CS}(\Delta, \mathcal{C})$, $(\Theta \cup \Sigma) \subseteq \Upsilon$ and $\Theta \cap \Sigma = \emptyset$.*

If $\exists\, \mathcal{C}'$ s.t. $\mathcal{C}' \subseteq \mathcal{C}$ and $\exists\, \Gamma_j \in \mathcal{C}'$ s.t. $\forall\, \Gamma_i \in \mathcal{C} \setminus \mathcal{C}'$

1. *$(\Delta \setminus \Theta) \cup \Gamma_i$ is satisfiable;*
2. *$(\Delta \setminus \Theta) \cup \Gamma_j$ is unsatisfiable;*
3. *Σ is one $\texttt{Partial-MCS}((\Delta \setminus \Upsilon) \cup \Gamma_j, \Upsilon \setminus \Theta)$;*
4. *$\forall\, \Gamma_k \in \mathcal{C}'$, $(\Delta \setminus (\Theta \cup \Sigma)) \cup \Gamma_k$ is satisfiable*

then there exists at least one Ψ that is an $\texttt{AC-MCS}(\Delta, \mathcal{C})$s.t.
(1) $\Psi \subseteq \Theta \cup \Sigma$, and
(2) $\forall \Psi$ s.t. Ψ is an $\texttt{AC-MCS}(\Delta, \mathcal{C})$ and $\Psi \subseteq \Theta \cup \Sigma$:$\Sigma \subseteq \Psi$.

Example 6. *Let us consider again Δ and $\mathcal{C}$ from Example 1. Let $\Theta = \{d \vee b\}$, $\mathcal{C}' = \{\{\neg a\}, \{\neg e\}\}$, $\Sigma = \{\neg c, \neg b\}$ and $\Gamma_j = \{\neg a\}$. All conditions of Property 2 are satisfied. Thus, Σ belongs to every $\texttt{AC-MCS}(\Delta, \mathcal{C})$ included in $\{d \vee b, \neg b, \neg c\}$.*

Now, the question is how to exploit this property to enhance the extraction of one $\texttt{AC-MCS}(\Delta, \mathcal{C})$. That is what we show in the next section.

An Enhanced Incremental Algorithm

An original algorithm for the extraction of one $\texttt{AC-MCS}(\Delta, \mathcal{C})$ is depicted in Algorithm 3. Let us describe it progressively and in an intuitive manner.

By abuse of notation, we will write that Σ solves Γ_j when Property 2 applies (referring to Σ and Γ_j in Property 2): in this case Σ is a subset of clauses from Δ that will belong to the final AC-MCS that is currently under construction.

Ψ is the AC-MCS under construction: it is initialized to the empty set (line 1). Importantly, we make use of a mapping $\sigma(\Gamma_i)$ that associates to each Γ_i one subset of Δ. Initially, i is assigned 1 and all these subsets are empty, except $\sigma(\Gamma_1)$, which is assigned $\Delta \setminus \texttt{Greedy-AC-SS}(\Delta, \Gamma)$. For easy understanding, assume for the moment that there is no preprocessing and that line 4 actually contains $\sigma(\Gamma_i) \leftarrow \Delta$.

Then we enter a main iteration loop. In this iteration loop, we make sure that $\Psi \cup \bigcup_{\Gamma_i \in \mathcal{C}} \sigma(\Gamma_i)$ always remains one AC-CS$(\Delta, \mathcal{C})$. This constraint is obviously satisfied before entering the loop since this set is Δ. Consider the first iteration loop. $i = 1$ and in line 7, we compute $\texttt{Partial-MCS}(\Gamma_1, \Delta)$ and store the result in Σ. Several cases can occur. Either $\Sigma = \sigma(\Gamma_i)$: in this first iteration, we actually then have $\Delta = \Sigma = \sigma(\Gamma_i)$. In such a situation, we have all the conditions for Property 2 to apply that are satisfied and we can thus augment the solution under construction Ψ with Σ (line 8). We insert the empty set in $\sigma(\Gamma_i)$ to express that Γ_i has been solved. i is then decremented and the loop terminates since i contains now 0: the output is Ψ. In another very basic case, $i = m = 1$: there is one context in $\mathcal{C}$, only. Again, we have found the solution, which is Σ, but contrary to the previous case, Σ might be a strict subset of Δ. In the last case (*else* case), we know that Σ is a set of clauses whose removal would allow to satisfy Γ_i. However, not all Γ_j have been successfully solved so far and there is no guarantee that Σ can be fully included in the solution since Property 2 does not apply. Hence, we put the remaining clauses $\sigma(\Gamma_i) \setminus \Sigma$ within $\sigma(\Gamma_{i+1})$ whereas Σ is recorded within $\sigma(\Gamma_i)$. The idea is that we keep trace in $\sigma(\Gamma_i)$ of the subset of clauses that would have been sufficient to solve Γ_i, whereas all the other *currently* remaining clauses will now be considered with respect to Γ_{i+1}. In some sense, there is thus a forward move in σ that pushes ahead and makes the remaining clauses to consider available for the next iteration,

while at the same time, we keep in $\sigma(\Gamma_i)$ a discovered set of clauses that could make Γ_i satisfiable with the AC-MSS under construction together with $\bigcup_{k=1}^{i-1} \sigma(\Gamma_k)$. Conversely, when at line 7 Property 2 applies, i is decremented. This gives rise to a kind of one-step backward move: Γ_i has been solved, we can now go back and examine Γ_{i-1} at the next iteration step. All this is done in such a way that $\sigma(\Gamma_j) = \emptyset$ for all $j > i$ at the beginning and the end of each iteration step.

Now, we are in a better position to understand the contents of the while loop and its first instruction at line 6. $\Delta \setminus (\Psi \cup \bigcup_{j=1}^{i} \sigma(\Gamma_j))$ gives the AC-SS under construction. By construction, it is satisfiable with Γ_i. Now, `Partial-MCS`$((\Delta \setminus (\Psi \cup \bigcup_{j=1}^{i} \sigma(\Gamma_j))) \cup \Gamma_i, \sigma(\Gamma_i))$ gives one smallest subset of all the remaining clauses that have been cumulated in $\sigma(\Gamma_i)$ that need to be dropped from $\sigma(\Gamma_i)$ in order for the corresponding remaining subset of $\sigma(\Gamma_i)$ to become satisfiable with the first argument of `Partial-MCS`. The idea is that when all clauses of $\sigma(\Gamma_i)$ must be dropped in this way, namely when $\Sigma = \sigma(\Gamma_i)$ (line 7), we are exactly under the conditions for Property 2 to apply. Indeed, we have found $\Sigma =$ `Partial-MCS`$((\Delta \setminus \Upsilon) \cup \Gamma_i, \Upsilon \setminus \Theta)$ such that increasing the current AC-CS with Σ will lead to a corresponding AC-CS that is satisfiable with each Γ_i since there will be no remaining clauses in $\sigma(\Gamma_i)$. When $i = m$, a same operation can be done: indeed, by making the current AC-SS satisfiable with Γ_m, we have ensured that all Γ_i are solved. In both cases, we can now decrement i: in the next iteration loop, we will try to solve Γ_{i-1} for which clauses were possibly recorded in $\sigma(\Gamma_{i-1})$. Indeed, when Property 2 cannot apply, we keep in $\sigma(\Gamma_i)$ the subset of clauses found to solve Γ_i (but that were not able to solve all other Γ_j at the same time) and put in a forward movement the remaining clauses in σ_{i+1} that will be addressed at the next iteration step, which will attempt to solve Γ_{i+1}. Note that when $\Sigma(\Gamma_i) = \emptyset$, the call to `Partial-MCS` yields the empty set, leading to decrement i at line 10.

Example 7. *Let us run* `Incremental`$_2$`-AC-MCS` *on Example 1, where* $\Gamma_1 = \{\neg a\}$, $\Gamma_2 = \{\neg e\}$ *and* $\Gamma_3 = \{\neg d\}$. *Let* $\{a \vee b, a \vee c, e \vee c\}$ *be the AC-SS delivered by* `Greedy-AC-SS`. *Before the iteration loop begins, we have* $i = 1$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c, d \vee b\}$ *and* $\sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.

- *Iteration loop #1:* $\Sigma = \{\neg b, \neg c\}$. *Because* $\Sigma \neq \sigma(\Gamma_1)$ *and* $i \neq 3$, *the* `else` *part is run. Thus,* $i = 2$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$, $\sigma(\Gamma_2) = \{d \vee b\}$ *and* $\sigma(\Gamma_3) = \emptyset$.
- *Iteration loop #2:* $\Sigma = \emptyset$. *Similarly to the second iteration, the* `else` *part is reached. Thus,* $i = 3$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$, $\sigma(\Gamma_2) = \emptyset$ *and* $\sigma(\Gamma_3) = \{d \vee b\}$.
- *Iteration loop #3:* $\Sigma = \emptyset$. *Since* $i = n$, *the* `if` *part is run. Thus,* $i = 2$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$ *and* $\sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.
- *Iteration loop #4:* $\Sigma = \emptyset$. *Since* $\sigma(\Gamma_2) = \emptyset$, *the* `if` *part is selected. Thus,* $i = 1$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$ *and* $\sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.
- *Iteration loop #5:* $\Sigma = \{\neg b, \neg c\}$, $\Sigma = \sigma(\Gamma_1)$. *The* `if` *part is selected. Thus,* $i = 0$, $\Psi = \{\neg b, \neg c\}$, $\sigma(\Gamma_1) = \sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.

The loop ends and the algorithm returns Ψ, *which is an* `AC-MCS`$(\Delta, \mathcal{C})$.

Algorithm 3: `Incremental`$_2$`-AC-MCS`

Input : Δ: a set of clauses;
$\mathcal{C} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$: a set of satisfiable sets of clauses;
Output: Ψ s.t. Ψ is an `AC-MCS`$(\Delta, \mathcal{C})$;

```
1  Ψ ← ∅;
2  σ a map that associates to each Γ_i ∈ C a set of clauses, initially all empty;
3  i ← 1;
4  σ(Γ_i) ← Δ \ Greedy-AC-SS(Δ, C);
5  while i > 0 do
6      Σ ← Partial-MCS((Δ \ (Ψ ∪ ⋃_{j=1}^{i} σ(Γ_j))) ∪ Γ_i, σ(Γ_i));
7      if Σ = σ(Γ_i) or i = m then
8          Ψ ← Ψ ∪ Σ;
9          σ(Γ_i) ← ∅;
10         i ← i − 1;
11     else
12         σ(Γ_{i+1}) ← σ(Γ_i) \ Σ;
13         σ(Γ_i) ← Σ;
14         i ← i + 1;
15 return Ψ;
```

Property 3. *`Incremental`$_2$`-AC-MCS`$(\Delta,\mathcal{C})$ always returns one AC-MCS*$(\Delta,\mathcal{C})$. *In the worst case, this procedure requires* $O(nm)$ *calls to* `Partial-MCS`(Ω_1, Ω_2), *where* $n = card(\Delta)$ *and* $m = card(\mathcal{C})$.

Before we present our experimental study, let us stress that the structure of the `Incremental`$_2$`-AC-MCS` procedure allows us to benefit from incremental SAT-solvers. More precisely, we have implemented `Partial-MCS`, using a technique similar to the one presented in (Audemard, Lagniez, and Simon 2013), on top of `Glucose` (Audemard and Simon 2014) (http://www.labri.fr/perso/lsimon/glucose/) in order to get a system that reuses as much as possible the search already performed during the calls to the SAT solver (inside a same call to `partial-MCS` and from one call to the next one). Let us stress that our actual implementation benefits also from other optimizations. Especially, in the main loop, useless calls to `partial-MCS`$(\ldots, \emptyset)$ are avoided. It also uses all the very recent advanced technical features for computing MSSes and MCSes, as proposed recently in (Grégoire, Lagniez, and Mazure 2014), and used in (Besnard, Grégoire, and Lagniez 2015) in the `Transformational Approach`.

Experimental Study

We have compared the `Transformational Approach`, `Incremental`$_1$`-AC-MSS` and `Incremental`$_2$`-AC-MCS` through extensive experimentations. To this end, we have focused on the 295 different unsatisfiable benchmarks from the last MUS extraction competition http://www.cril.univ-artois.fr/SAT11/results/results.php?idev=48 as instances of Δ. We have considered the following values for $card(\mathcal{C})$: $2, 5, 10, 15, 20, 25, 30, 35, 40, 45$ and 50, thus representing various numbers of different contexts. Then, using the variables from Δ, we have randomly generated each Γ_i as a satisfiable set of 50 binary clauses. All experimentations

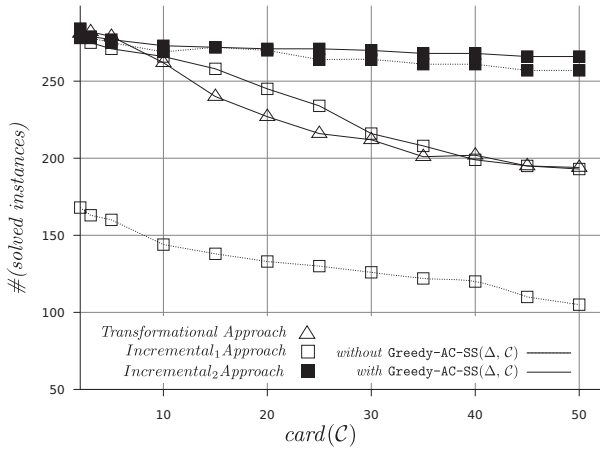

Figure 2: Number of solved instances.

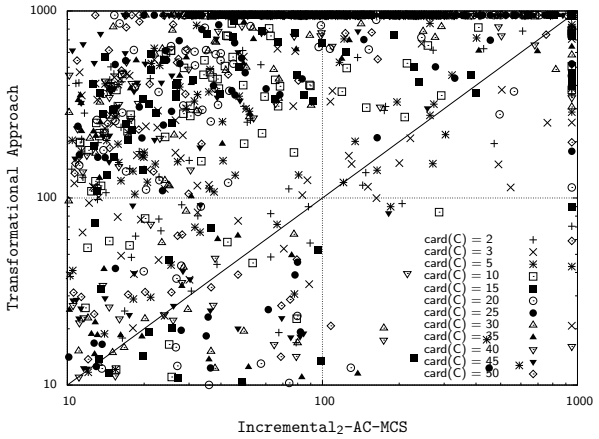

Figure 3: `Incremental`$_2$`-AC-MCS` vs. `Transformational Approach`.

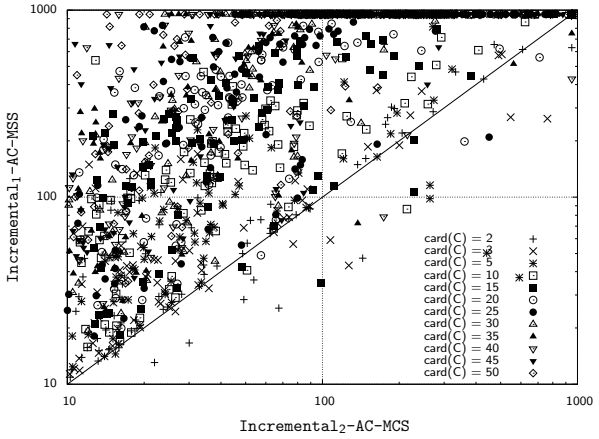

Figure 4: `Incremental`$_2$`-AC-MCS` vs. `Incremental`$_1$`-AC-MSS`.

have been conducted on Intel Xeon E5-2643 (3.30GHz) processors with 8Gb RAM on Linux CentOS. Time limit was set to 900 seconds per instance and per test. We have run the `Transformational Approach` software available from its authors at http://www.cril.fr/AAAI15-BGL. We have implemented all the other algorithms in C++ on top of `Glucose` (http://www.labri.fr/perso/lsimon/glucose/). Our software, as well as all data and results from these experimentations are available at http://www.cril.fr/AAAI16-GIL. The *iteration-max* constant in `Greedy-AC-SS` was set to 10.

The results clearly show that `Incremental`$_2$`-AC-MCS` is better than its competitors. As illustrated in Figure 2, for every considered value for the cardinality of $\mathcal{C}$ it solves more instances than any of the other tested methods; the difference increases with *card*($\mathcal{C}$). Interestingly, these results do not change when `Greedy-AC-SS` is skipped in `Incremental`$_2$`-AC-MCS`, showing that the performance of the algorithm is really due to the specific properties of AC-MCSes that have been established in this study. `Incremental`$_1$`-AC-MSS` proves less competitive but often solves more instances than the `Transformational Approach`, provided that it includes `Greedy-AC-SS`. The number of instances that were solved by `Incremental`$_2$`-AC-MCS` ranged from 266 to 284 (on a total of 295 tested instances) depending on *card*($\mathcal{C}$). Likewise, most instances are solved in the fastest manner by `Incremental`$_2$`-AC-MCS`. Figure 3 compares this latter approach with the `Transformational` one in terms of the CPU time in seconds that was spent to solve each instance, for each value of *card*($\mathcal{C}$). This figure makes use of a logarithmic scale; it clearly shows that `Incremental`$_2$`-AC-MCS` is the most efficient approach, most often. Figure 4 shows similar results when `Incremental`$_2$`-AC-MCS` and `Incremental`$_1$`-AC-MSS` are compared. The full detailed results of our experimental study are available at http://www.cril.fr/AAAI16-GIL.

Conclusion

The ability to capture one maximal subset of information that must be satisfiable with a series of mutually-conflicting context is a key building-block in many A.I. subfields. In this paper, we have proposed an original method that allows one such subset to be extracted in clausal Boolean logic. Interestingly, it performs better than its competitors. Clearly, the results in this paper could be extended in several promising directions. Noticeably, the Partial-MCS procedure that exploits the incremental feature of advanced SAT-solvers could also benefit from the specific optimizations of other kinds of MCS finders like (Mencía, Previti, and Marques-Silva 2015). Also, the technique in this paper could be the kernel of an approach that enumerates all such maximal information subsets (at least when computational blow-up does not occur). A key issue in this respect would be how to re-use the information that could be derived during the search for one subset in the extraction of the next ones. Finally, ex-

porting the main results from this paper to build a *directly incremental* method to extract cardinality-maximal subsets that are satisfiable with multiple contexts remains an open but exciting challenge.

References

Audemard, G., and Simon, L. 2014. Lazy clause exchange policy for parallel SAT solvers. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing - SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, 197–205. Springer.

Audemard, G.; Lagniez, J.; and Simon, L. 2013. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing - SAT 2013*, volume 7962 of *Lecture Notes in Computer Science*, 309–317. Springer.

Bailey, J., and Stuckey, P. J. 2005. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages, 7th International Symposium, PADL 2005*, volume 3350 of *Lecture Notes in Computer Science*, 174–186. Springer.

Belov, A.; Heule, M.; and Marques-Silva, J. 2014. MUS extraction using clausal proofs. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing - SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, 48–57. Springer.

Besnard, P.; Grégoire, É.; and Lagniez, J. 2015. On computing maximal subsets of clauses that must be satisfiable with possibly mutually-contradictory assumptive contexts. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, Texas*, 3710–3716. AAAI Press.

Birnbaum, E., and Lozinskii, E. L. 2003. Consistent subsets of inconsistent systems: structure and behaviour. *J. Exp. Theor. Artif. Intell.* 15(1):25–46.

Cook, S. A. 1971. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM* 18(1):4–18.

Grégoire, É.; Lagniez, J.-M.; and Mazure, B. 2014. An experimentally efficient method for (MSS, CoMSS) partitioning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI'14)*, 2666–2673. AAAI Press.

Grégoire, É.; Mazure, B.; and Piette, C. 2009. Using local search to find msses and muses. *European Journal of Operational Research* 199(3):640–646.

Liffiton, M. H., and Malik, A. 2013. Enumerating infeasibility: Finding multiple MUSes quickly. In *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2013*, volume 7874 of *Lecture Notes in Computer Science*, 160–175. Springer.

Liffiton, M. H., and Sakallah, K. A. 2008. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning* 40(1):1–33.

Marques-Silva, J., and Janota, M. 2014. On the query complexity of selecting few minimal sets. *Electronic Colloquium on Computational Complexity (ECCC)* 21:31.

Marques-Silva, J.; Heras, F.; Janota, M.; Previti, A.; and Belov, A. 2013. On computing minimal correction subsets. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*.

Mencía, C.; Previti, A.; and Marques-Silva, J. 2015. Literal-based MCS extraction. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, 2015*, 1973–1979. AAAI Press.

Previti, A., and Marques-Silva, J. 2013. Partial MUS enumeration. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press.