Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

An Experimentally Efficient Method for (MSS,CoMSS) Partitioning

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

Abstract

The concepts of MSS (Maximal Satisfiable Subset) and CoMSS (also called Minimal Correction Subset) play a key role in many A.I. approaches and techniques. In this paper, a novel algorithm for partitioning a Boolean CNF formula into one MSS and the corresponding CoMSS is introduced. Extensive empirical evaluation shows that it is more robust and more efficient on most instances than currently available techniques.

Introduction

The concept of MSS (Maximal Satisfiable Subset) has long been playing a key role in many logic-based A.I. approaches and techniques. Especially, in the knowledge representation domain, reasoning on the basis of inconsistent premises is often modeled using MSSes as basic building blocks (in e.g. belief revision, knowledge fusion, argumentation theory or nonmonotonic reasoning (see surveys and edited volumes of early seminal contributions in each of these fields in e.g. (Fermé and Hansson 2011), (Grégoire and Konieczny 2006), (Besnard and Hunter 2008) and (Ginsberg 1987)).

The complement of one MSS in an inconsistent set of formulas, noted CoMSS, is often called Minimal Correction Subset: it forms a minimal subset of formulas to be dropped in order to restore consistency. Consequently, the concept of CoMSS is a crucial paradigm in both model-based diagnosis (see seminal works in (Hamscher, Console, and de Kleer 1992)) and consistency-restoring techniques in knowledge-bases.

Beyond logic-based frameworks, MSSes and CoMSSes find similar roles in constraint reasoning when a problem is over-constrained and thus yields no solution (see for example in constraint networks (Rossi, van Beek, and Walsh 2006; Lecoutre 2009) or in optimisation (Chinneck 2008)).

In this paper, we are interested in the ubiquitous problem within the aforementioned domains that consists in partitioning an unsatisfiable Boolean CNF formula into one MSS and the corresponding CoMSS, when maximality is considered with respect to set-theoretical inclusion (vs. cardinality). Computing one such partition is in $\Delta_2^P = P^{NP}$ (Papadimitriou 1993). Despite this heavy cost in the worst case, several computational approaches to extract one (MSS,CoMSS) partition have been proposed that prove empirically viable for many instances.

Clearly, extracting one (MSS,CoMSS) pair is close to the basic version of the MAX-SAT problem that involves soft clauses only and consists in extracting one maximal (with respect to cardinality) satisfiable subset of a Boolean CNF formula. Every solution to MAX-SAT is one MSS whereas every MSS is not necessarily a solution to MAX-SAT. Interestingly, as advocated by (Marques-Silva et al. 2013), specific algorithms to compute one MSS can prove faster than MAX-SAT-dedicated ones; in this respect, computing one MSS can be used to deliver one approximate solution to MAX-SAT when a solution to this latter problem is out of reach.

Noticeably, CoMSS and MUS (namely, Minimal Unsatisfiable Subset) are dual concepts. One MUS is an unsatisfiable subset such that dropping any one of its clauses leads to satisfiability. MUSes can be computed as hitting sets of CoMSSes since any CoMSS contains one clause of each MUS (Bailey and Stuckey 2005; Grégoire, Mazure, and Piette 2007a; 2007b; Liffiton and Sakallah 2008).

In the paper, a novel algorithm to compute one (MSS,CoMSS) partition of an unsatisfiable Boolean CNF formula is thus introduced. Extensive empirical evaluation shows that it is more robust and more efficient on most instances than currently available techniques to extract one MSS or one CoMSS.

Logical preliminaries and basic concepts

Let $\mathcal{L}$ be a standard Boolean logical language built on a finite set of Boolean variables and usual connectives (namely, $\wedge$, $\vee$, $\neg$ and $\rightarrow$ standing for conjunction, disjunction, negation and material implication, respectively). Any formula in $\mathcal{L}$ can be represented (while preserving satisfiability) in clausal normal form (in short, a CNF) using a set (interpreted conjunctively) of clauses, where a clause is a formula made of a finite disjunction of literals and where a literal is Boolean variable that can negated. Formulas will be noted using lower-case Greek letters such as α, β, etc. Sets of formulas will be represented using letters like Γ, Δ, Σ, etc. We note by $\overline{\alpha}$ the opposite of the clause α, i.e. the set of unit clauses formed of the opposite of the literals of α.

An interpretation $\mathcal{I}$ assigns values from $\{0, 1\}$ to every

Boolean variable, and, following usual compositional rules, to all formulas of $\mathcal{L}$. A formula α is consistent or satisfiable when there exists at least one interpretation $\mathcal{I}$ that satisfies it, i.e. such that $\mathcal{I}(\alpha) = 1$; $\mathcal{I}$ is then called a model of α and is represented by the set of variables that it satisfies. α can be deduced from Σ, noted $\Sigma \models \alpha$, when α is satisfied in all models of Σ.

SAT is the NP-complete problem that consists in checking whether or not a CNF is satisfiable, i.e., whether or not there exists a model of all clauses in the CNF. In the paper, we often refer to CDCL-SAT solvers, which are currently the most efficient logically complete SAT solvers and that exploit so-called Conflict-Direct Clause Learning features (see e.g. (Marques-Silva and Sakallah 1996; Moskewicz et al. 2001; Zhang et al. 2001; Zhang and Malik 2002; Eén and Sörensson 2004; Audemard and Simon 2009; 2012)).

Let Σ be a CNF.

Definition 1 (MSS) $\Gamma \subseteq \Sigma$ *is a Maximal Satisfiable Subset (*MSS*) of* Σ *iff* Γ *is satisfiable and* $\forall \alpha \in \Sigma \setminus \Gamma$, $\Gamma \cup \{\alpha\}$ *is unsatisfiable.*

Definition 2 (CoMSS) $\Gamma \subseteq \Sigma$ *is a Minimal Correction Subset (*MCS *or* CoMSS*) of* Σ *iff* $\Sigma \setminus \Gamma$ *is satisfiable and* $\forall \alpha \in \Gamma$, $\Sigma \setminus (\Gamma \setminus \{\alpha\})$ *is unsatisfiable.*

Accordingly, Σ can always be partitioned into a pair made of one MSS and one CoMSS. Obviously, such a partition needs not be unique.

A *core* of Σ is a subset of Σ that is unsatisfiable. Minimal cores, with respect to set-theoretical inclusion, are called MUSes.

Definition 3 (MUS) $\Gamma \subseteq \Sigma$ *is a Minimal Unsatisfiable Subset (*MUS*) of* Σ *iff* Γ *is unsatisfiable and* $\forall \alpha \in \Gamma$, $\Gamma \setminus \{\alpha\}$ *is satisfiable.*

Under its basic form where all clauses are considered as being soft, i.e., not compulsory satisfiable, MAX-SAT consists in delivering one maximal (with respect to cardinality) subset of Σ. As emphasized earlier, every solution to MAX-SAT is one MSS and there exists a hitting set duality between CoMSSes and MUSes.

Our algorithm to partition a CNF is based on the *transition clause* concept.

Definition 4 (Transition clause) *Let* Σ *be an unsatisfiable* CNF. *A clause* $\alpha \in \Sigma$ *is a transition clause of* Σ *iff* $\Sigma \setminus \{\alpha\}$ *is satisfiable.*

The transition clause concept has proved crucial in approaches to MUS extraction. Especially, (Belov and Marques-Silva 2011) takes advantage of the following property in order to enhance the performance of a MUS extractor.

Property 1 *Let* Σ *be an unsatisfiable* CNF. *When* $\alpha \in \Sigma$ *is a transition clause,* α *belongs to every* MUS *of* Σ.

Likewise, a concept of transition constraint has proved powerful for the extraction of minimal unsatisfiable sets of constraints in the general setting of constraint networks (Hemery et al. 2006; Grégoire, Lagniez, and Mazure 2013a; 2013b). The transition clause concept will be a key concept in our partitioning algorithm, too.

Algorithm 1: BLS (Basic Linear Search)

input : one CNF Σ
output: one (MSS,CoMSS) partition of Σ

$\Pi \leftarrow \Omega \leftarrow \emptyset$;
foreach $\alpha \in \Sigma$ **do**
  if SAT $(\Pi \cup \alpha)$ **then** $\Pi \leftarrow \Pi \cup \{\alpha\}$;
  else $\Omega \leftarrow \Omega \cup \{\alpha\}$;
return (Π,Ω) ;

The simplest algorithm to partition a CNF into one (MSS,CoMSS) pair is given in Algorithm 1 and called Basic Linear Search (in short, BLS). It inserts within the MSS under construction every clause that is satisfiable together with this subset. Clearly, when the number of clauses in Σ is n, BLS requires n calls to a SAT-solver. This basic algorithm is only given here because we believe that it is the best starting point to understand the new approach that we are going to introduce. A survey of the currently more efficient approaches is provided for example in (Marques-Silva et al. 2013).

CMP: a novel partitioning algorithm

Algorithm 2: CMP
(Computational Method for Partitioning)

input : one CNF Σ
output: one (MSS,CoMSS) partition of Σ

$(\Pi, \Psi) \leftarrow$ `ApproximatePartition` (Σ) ;
$\Omega \leftarrow \emptyset$;
$\Gamma \leftarrow \Psi$; `// Working subset of` Ψ
while $\Psi \neq \emptyset$ **do**
  `extendSatPart` $(\Pi, \Gamma, \Omega, \Psi)$;
  if $\Psi \neq \emptyset$ **then**
    $\alpha \leftarrow$ `selectClause` (Γ) ;
    $(\mathcal{I}, \Delta) \leftarrow$ `solve` $(\Pi \cup (\Gamma \setminus \{\alpha\}) \cup \overline{\alpha})$;
    `//Returns a model` $\mathcal{I}$ `if SAT and`
    `//(`$\mathcal{I} = \emptyset$ `and core` Δ`) otherwise`
    if $\mathcal{I} \neq \emptyset$ **then** `// Transition clause`
      $\Omega \leftarrow \Omega \cup \{\alpha\}$;
      $\Pi \leftarrow \Pi \cup \{\beta \in (\Psi \setminus \{\alpha\})$ s.t. $\mathcal{I}(\beta) = 1\} \cup \overline{\alpha}$;
      $\Gamma \leftarrow \Psi \leftarrow \{\beta \in (\Psi \setminus \{\alpha\})$ s.t. $\mathcal{I}(\beta) = 0\}$;
    else
      $\Gamma \leftarrow \Gamma \setminus \{\alpha\}$;
      `exploitCore` $(\Pi, \Gamma, \Psi, \Delta, \alpha)$;
return $(\Pi \cap \Sigma,\Omega)$;

The novel algorithm for partitioning an unsatisfiable CNF into one MSS and its corresponding CoMSS is called CMP for Computational Method for Partitioning. Its skeleton is

Procedure `ApproximatePartition(`Σ`)`

1 **if** `SAT(`Σ`)` **then** `// Satisfiable instance`
2 | **return** $(\Sigma, \emptyset)$;
3 **else**
4 | Let $\mathcal{P}$ be the progress-saving interpretation delivered by the previous call to the CDCL-SAT solver ;
5 | **return** $(\{\beta \in \Sigma \text{ s.t. } \mathcal{P}(\beta)=1\}, \{\beta \in \Sigma \text{ s.t. } \mathcal{P}(\beta)=0\})$;

depicted in Algorithm 2, where framed boxes in the algorithm are optional and will be discussed in a subsequent section.

CMP delivers a partition (Π, Ω) of a CNF Σ where Π is one MSS of Σ (and, consequently, where Ω is the corresponding CoMSS). At this point, it is important to stress that:

1. The loop structure of CMP makes a search for transition clauses that differs from BLS in a fundamental original way: the iteration loops in both algorithms are very different. In the worst case, CMP makes $O(n^2)$ calls to a SAT-solver whereas BLS makes a linear number of such calls, only. However, our extensive experimental studies show that even without the optional features given in the framed boxes, the basic structure of CMP is more efficient than every current competitor on many instances.
2. The full version of CMP thus includes several additional optional features encapsulated within framed boxes in Algorithm 2. We will show that each of them incrementally increases the performance of the approach: some of the options were already included in other approaches (`extendSatPart`), other ones are exploited in an original way (especially a concept that we will call "opposite enforcement on backbones") and the last ones are new concepts (parts of `exploitCore`).

CMP: step-by-step presentation

CMP starts by computing one MSS approximation of Σ using the `ApproximatePartition` procedure (Alg.2:line1), which calls a CDCL-SAT solver to check the satisfiability of Σ. In the positive case, `ApproximatePartition` delivers the partition $(\Sigma, \emptyset)$ and since $\Psi = \emptyset$, CMP does not enter its main loop and ends. In the negative case, `ApproximatePartition` partitions Σ into a couple (Π, Ψ), where Π is a subset of the MSS under construction that is obtained (Proc.`ApproximatePartition`:line5) by making use of the so-called *progress-saving interpretation* (Pipatsrisawat and Darwiche 2007) $\mathcal{P}$ of Σ that has been delivered by the call to the CDCL-SAT solver (Proc.`ApproximatePartition`:line1): all clauses of Σ satisfied by $\mathcal{P}$ are recorded inside Π. In CDCL-SAT solvers, the progress-saving interpretation captures the final state of variables which can hopefully exhibit a "good" approximation of an MSS of Σ. Moreover, the facilities provided by CDCL-SAT solvers to record useful information like the final state of variables ordering, the polarity cache and clause deletion/restart strategies for subsequent calls to the solver, can be exploited as well (Audemard et al. 2011; Guo and Lagniez 2011).

Π will evolve to deliver the final MSS whereas the final CoMSS Ω under construction is initialized to the empty set (Alg.2:line 2). When the optional parts of the algorithm are taken into account, Π might not only be enriched with clauses that are moved from Ψ but also with additional (entailed) clauses that are expected to speed the whole process. Actually, in this case, $\Sigma = (\Pi \cap \Sigma) \cup \Psi \cup \Omega$ will be an invariant of the loop of the algorithm.

Let us leave apart the optional framed boxes for the moment and see also Figure 1. At the beginning (Alg.2:line 3) and whenever a clause α is added into Ω (Alg.2:line 10), a working subset Γ of Ψ is initialized to Ψ (Alg.2:line 12). While every clause of Ψ has not been dispatched either in the CoMSS or in the MSS under construction, the algorithm searches for a clause $\alpha \in \Psi$ that is a transition clause of $\Pi \cup \Gamma \cup \{\alpha\}$ where $\Gamma \cup \{\alpha\} \subseteq \Psi$, i.e., such that $\Pi \cup \Gamma \cup \{\alpha\}$ is unsatisfiable whereas $\Pi \cup \Gamma$ is satisfiable. When such an α is discovered, it is moved inside the CoMSS under construction Ω (Alg.2:line 10). All clauses of $\Pi \cup \Psi$ that are satisfied (resp. falsified) by the exhibited model $\mathcal{I}$ from the last satisfiability test of $\Pi \cup \Gamma$ are moved inside Π (Alg.2:line 11) (resp. remain in Ψ (Alg.2:line 12)). The working subset Γ is then initialized to Ψ according to the new value of this latter set (Alg.2:line 12). When α is not such a transition clause (Alg.2:line 14), α is expelled from the working subset $\Gamma \in \Psi$ (Alg.2:line 15). The `selectClause` function (line 7) selects one clause α from Γ. Our implementation of this function is based on the well-known VSIDS (Zhang and Malik 2002) heuristic.

CMP: proof of correctness

For space reasons, we only give here the main keys of the proof, together with useful observations that help to understand why CMP delivers the intended results.

First, it is easy to see that the loop always terminates. Remember that Γ is initialized to Ψ (alg.2:lines 3 &12). Since Σ is unsatisfiable whereas Π is always satisfiable, there always exists at least one $\alpha \in \Gamma$ such that the satisfiability test (Alg.2:line 9) is positive thanks to the fact the previously negatively tested α are expelled from Γ, successively (Alg.2:line 14). When this positive test occurs, Ψ decreases (Alg.2:line 12). An extreme case occurs when satisfiability is only found when $\Gamma = \{\alpha\}$.

Now, let us explain why the final Ω is a CoMSS of Σ. First, consider the first time when some α is going to be moved inside Ω (Alg.2:line 10) because α has been recognized as a transition clause of $\Pi \cup \Gamma$. This corresponds to Figure 1, where $\Omega = \emptyset$. The following property ensures that α can be inserted within Ω since when α is a transition clause of $\Pi \cup \Gamma$, α belongs to any Θ that is a CoMSS of Σ and that is such that $\Theta \subseteq \Psi \setminus (\Gamma \setminus \{\alpha\})$.

Property 2 *Let Σ be an unsatisfiable* CNF. *When $\Sigma = \Pi \cup \Psi$ with Π satisfiable, $\Gamma \subseteq \Psi$ and $\alpha \in \Gamma$ is a transition clause of $\Pi \cup \Gamma$, we have that $\forall \Theta \subseteq \Psi \setminus (\Gamma \setminus \{\alpha\})$ s.t. $\Theta \in$ CoMSS(Σ): $\alpha \in \Theta$.*

Σ an unsatisfiable CNF
$\Sigma = \Pi \cup \Psi \cup \Omega$
Π is satisfiable
$\Gamma \subseteq \Psi$
α transition clause of $\Pi \cup \Gamma$
Ω set of transition clauses

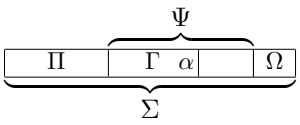

Figure 1: The various sets of clauses

Proof. By contradiction. Assume that $\exists\ \Theta \in \text{CoMSS}(\Sigma)$ s.t. $\Theta \subseteq \Psi \setminus (\Gamma \setminus \{\alpha\})$ and $\alpha \notin \Theta$, i.e. $\Theta \subseteq \Psi \setminus \Gamma$. If $\Theta \in \text{CoMSS}(\Sigma)$ then $\Sigma \setminus \Theta$ is satisfiable. Moreover, since $\Gamma \subseteq \Psi$ and $\Theta \subseteq \Psi \setminus \Gamma$, we have: $\Psi = (\Psi \setminus \Gamma) \cup \Gamma = ((\Psi \setminus \Gamma) \setminus \Theta) \cup \Theta \cup \Gamma$. Thus, $\Sigma = \Pi \cup \Psi = \Pi \cup \Gamma \cup ((\Psi \setminus \Gamma) \setminus \Theta) \cup \Theta$. By consequence, $(\Sigma \setminus \Theta) \supseteq (\Pi \cup \Gamma)$. As $\Sigma \setminus \Theta$ is satisfiable, we have also $\Pi \cup \Gamma$ satisfiable which is in contradiction with $\Pi \cup \Gamma$ being unsatisfiable (because α is a transition clause of $\Pi \cup \Gamma$). □

Now, thanks to the following property it is easy to prove by induction that every time a clause α is inserted within Ω, α belongs to a same CoMSS than the clauses that were previously inserted within Ω.

Property 3 *Let Σ be an unsatisfiable* CNF. *Assume $\Sigma = \Pi \cup \Psi$, Π satisfiable, $\Gamma \subseteq \Psi$ and that $\alpha \in \Gamma$ is a transition clause of $\Pi \cup \Gamma$. If $\Theta \subseteq (\Psi \setminus \Gamma)$ is a* CoMSS *of $\Sigma \setminus \{\alpha\}$ then $\Theta \cup \{\alpha\}$ is a* CoMSS *of Σ.*

Proof. By contraposition. Assume that $(\Theta \cup \{\alpha\})$ is not a CoMSS of Σ. Let us prove that Θ is not a CoMSS of $\Sigma \setminus \{\alpha\}$. $(\Theta \cup \{\alpha\})$ is not a CoMSS of Σ means that either $\Sigma \setminus (\Theta \cup \{\alpha\})$ is unsatisfiable (1) or $(\Theta \cup \{\alpha\})$ is not minimal, i.e. $(\Theta \cup \{\alpha\})$ is an upper-approximation of a CoMSS of Σ (2).

(1) $\Sigma \setminus (\Theta \cup \{\alpha\})$ unsatisfiable entails that $(\Sigma \setminus \{\alpha\}) \setminus \Theta$ is unsatisfiable and consequently Θ is not a CoMSS of $\Sigma \setminus \{\alpha\}$.

(2) $(\Theta \cup \{\alpha\})$ is an upper-approximation of a CoMSS of Σ implies that there exists a CoMSS Φ of Σ s.t. $\Phi \subset (\Theta \cup \{\alpha\})$. In other words, there exists $\beta \in (\Theta \cup \{\alpha\})$, s.t. $\beta \notin \Phi$. Property 2 ensures that $\alpha \in \Phi$, we obtain that $\alpha \neq \beta$.

$\Sigma \setminus (\Theta \cup \{\alpha\})$ is satisfiable (because $(\Theta \cup \{\alpha\})$ is an upper-approximation of a CoMSS of Σ). Thus, $\Sigma \setminus (\Theta \cup \{\alpha\} \cup \{\beta\})$ is also satisfiable. Therefore, $(\Sigma \setminus \{\alpha\}) \setminus (\Theta \setminus \{\beta\})$ is satisfiable and thus Θ is not a CoMSS of $\Sigma \setminus \{\alpha\}$. □

CMP: optional improvements

We have investigated four candidate improvements of CMP. They are encapsulated in the framed boxes of Algorithm 2.

First, following (Birnbaum and Lozinskii 2003) and (Marques-Silva et al. 2013), we have exploited the idea that checking the satisfiability of a logically weaker (and thus hopefully easier) CNF where some clauses (here, the clauses belonging to Γ) are replaced by their disjunction (here, noted $\bigvee \Gamma$) can prove informative. In case of unsatisfiability of $\Pi \cup \{\bigvee \Gamma\}$, $\Pi \cup \Gamma$ is also unsatisfiable. In case of satisfiability, the set of clauses from Γ that are containing literals satisfied in the exhibited model, is satisfiable together with Π and can thus be moved inside Π. This is the role of

Procedure `extendSatPart` $(\Pi, \Gamma, \Omega, \Psi)$

1 $(\mathcal{I}, \Delta) \leftarrow$ `solve` $(\Pi \cup \{\bigvee \Gamma\})$;
2 **if** $\mathcal{I} \neq \emptyset$ **then**
3 $\quad \Pi \leftarrow \Pi \cup \{\beta \in \Psi$ s.t. $\mathcal{I}(\beta) = 1\}$;
4 $\quad \Gamma \leftarrow \{\beta \in \Psi$ s.t. $\mathcal{I}(\beta) = 0\}$;
5 **else**
6 $\quad \Omega \leftarrow \Omega \cup \Gamma$;
7 $\quad \Gamma \leftarrow \Psi \leftarrow \Psi \setminus \Gamma$;

the `extendSatPart` function in line 5 of Algorithm 2. This feature proved important for the efficiency of the CLD method from (Marques-Silva et al. 2013).

The second tentative improvement (Algo2:line 11) is related to backbone literals ((Monasson et al. 1999) and more recently e.g. (Kilby et al. 2005)); it has been investigated in CoMSS extraction procedures by (Marques-Silva et al. 2013), too. $\overline{\alpha}$ is the set of unit clauses made of the opposite literals of α. As α will here belong to the intended CoMSS, this entails that $\overline{\alpha}$ is a deductive consequence of the current value of Π (and any superset of it by monotonicity). These learnt unit clauses are put inside Π in the hope to speed up the future satisfiability tests of supersets of Π.

An original tentative improvement occurs when it is tested whether or not α is a transition clause (Algo2:line 8). In the positive case, $\Pi \cup \Gamma \setminus \{\alpha\}$ is satisfiable whereas $\Pi \cup \Gamma$ is unsatisfiable. Accordingly, a same result will be obtained by checking the satisfiability of $\Pi \cup (\Gamma \setminus \{\alpha\}) \cup \{\overline{\alpha}\}$, where the set of unit clauses $\overline{\alpha}$ is expected to speed up the test. To the best of our knowledge, this use of backbones literals in a logically stronger set of premises that is not only consistent with $\overline{\alpha}$ but entails $\overline{\alpha}$ is novel in the search for CoMSS. We call this feature *opposite enforced*, in short *oe*. Interestingly, CMP allows this additional feature because a transition clause is found in case of satisfiability whereas other approaches find transition constraints when unsatisfiability is encountered.

Procedure `exploitCore` $(\Pi, \Gamma, \Psi, \Delta, \alpha)$

1 **if** $\Delta \cap \overline{\alpha} = \emptyset$ **then** $\Gamma \leftarrow \Gamma \cap \Delta$;
2 **elsif** $\Delta \cap \Gamma = \emptyset$ **then**
3 $\quad \Pi \leftarrow \Pi \cup \{\alpha\}$;
4 $\quad \Psi \leftarrow \Psi \setminus \{\alpha\}$;
5 **else** $\Pi \leftarrow \Pi \cup \{\Delta \setminus \overline{\alpha} \vDash \alpha\}$;

A fourth improvement is the `exploitCore` function (Algo2:line 15) that refines in the following original way Γ thanks to the computed core Δ. Three cases need be distinguished.

1. When Δ does not contain any clause of $\overline{\alpha}$, Γ can be assigned its set-theoretic intersection with Δ as a transition constraint can always be found within this intersection.
2. Otherwise, if no clause of Γ belongs to Δ then this entails that Δ is only made of clauses of $\overline{\alpha}$ and Π. Consequently, $\Pi \wedge \overline{\alpha}$ is unsatisfiable, i.e., α is a deductive consequence of Π. Thus, α can be moved into the satisfiable part Π.

3. Else, Δ is built from clauses of $\overline{\alpha}$ and Γ (and possibly from clauses of Π, too). In this case, since at least one clause of $\overline{\alpha}$ is in Δ and since Δ is unsatisfiable, we have that $(\Delta \setminus \overline{\alpha}) \wedge \overline{\alpha}$ is unsatisfiable, i.e., α is a deductive consequence of $\Delta \setminus \overline{\alpha}$. Although it cannot be represented by a set of clauses in a direct manner, this information can be exploited and implemented easily (and without significant additional space cost) using clause selectors markers *à la* (Oh et al. 2004), as follows. Each clause β of Σ is associated to a new corresponding dedicated literal noted λ_β, called selector, and β is replaced by $\beta \vee \neg\lambda_\beta$ in Σ. A clause is active in (i.e., belongs to) a set of clauses if its selector is assigned to 1. With these selectors, $\Delta \setminus \overline{\alpha} \models \alpha$ can be represented by the clause $(\neg\lambda_{\delta_1} \vee \cdots \vee \neg\lambda_{\delta_m} \vee \lambda_\alpha)$ where δ_i ($i \in [1..m]$) are the m clauses of $\Delta \setminus \overline{\alpha}$. When all clauses of $\Delta \setminus \overline{\alpha}$ are active then α must be also active.

It is easy to show that all those improvements do not alter the correctness of the algorithm.

Related work

Handling over-constrained systems though maximal satisfiable or minimal correction subsets has long been an active subject of research in A.I. (see pioneering work for example in (Meseguer et al. 2003)). In the general constraint networks setting, a seminal approach called QUICKXPLAIN has been described in (Junker 2004). In the same framework, improved techniques can be found in e.g. (Hemery et al. 2006).

In order to solve a constraint network instance or extract its MSSes and CoMSSes, it is often more efficient to encode the network through a set of Boolean clauses and benefit from SAT technology (provided that the size of the Boolean instance does not blow-up). In the SAT domain, many approaches have been proposed to compute MSSes and CoMSSes. The earliest approaches were based on DPLL-based SAT solvers (see e.g. (Birnbaum and Lozinskii 2003)) but are quite inefficient compared to more recent approaches (Liffiton and Sakallah 2008) based on MAX-SAT. As stressed in (Marques-Silva et al. 2013), the latter approaches also proved more efficient than various algorithms based on iterative calls to a SAT solver like (Bailey and Stuckey 2005). Much research has also been conducted in model-based diagnosis. Noticeably, a recent approach, called FastDiag (hereafter noted BFD for Basic FastDiag), (Felfernig, Schubert, and Zehentner 2012) has adapted QUICKXPLAIN from (Junker 2004) to the Boolean case and proves often very competitive.

Recently (Marques-Silva et al. 2013) experimentally compared the best MSS and CoMSS extraction approaches in the Boolean setting, namely the above BFD algorithm, BLS (depicted in Algorithm 1), EFD and ELS for Enhanced FastDiag and Enhanced Linear Search through additional features discussed in (Marques-Silva et al. 2013). The same authors also proposed a novel algorithm for computing CoMSSes that they experimentally showed to be the most efficient and robust one for CoMSSes computation, compared with the aforementioned list of approaches. Roughly, this algorithm, called CLD, extracts one CoMSS by using the `extendSatPart` principle iteratively (and also using backbone considerations, among other things). The authors showed that CLD experimentally outperforms all the competing approaches.

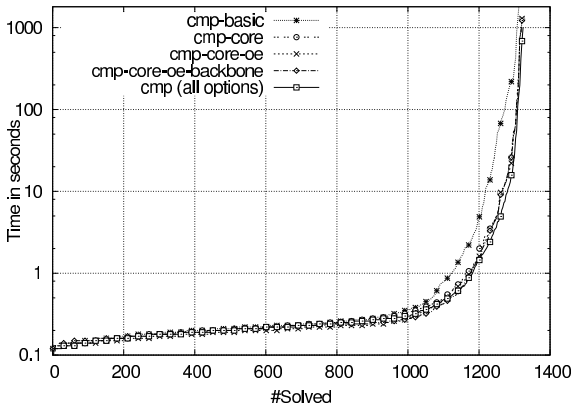

Figure 2: CMP variants on SAT/MAX-SAT instances

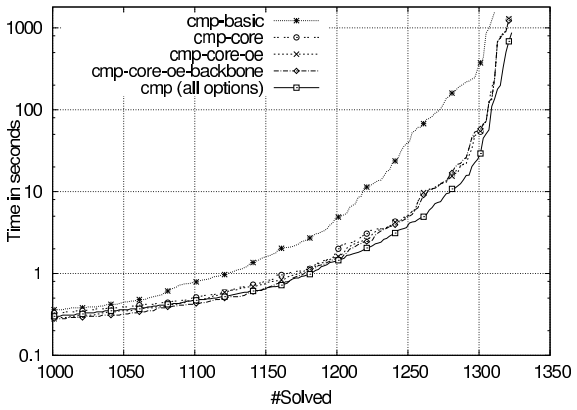

Figure 3: Zoom in on Figure 2

Although it implements the backbones and `extendSatPart` features, CMP is very different in nature as its key principle is the search for transition constraints rather than iterating on `extendSatPart`. Moreover, it includes the novel *opposite enforced* and `exploitCore` features.

Experimental results

All experimentations have been conducted on Intel Xeon E5-2643 (3.30GHz) processors with 7.6Gb RAM on Linux CentOS. Time limit was set to 30 minutes.

Two series of benchmarks have been considered. The first one was made of the 1343 benchmarks used and referred to in (Marques-Silva et al. 2013): they are small-sized industrial-based instances from SAT competitions www.satcompetition.org and structured instances from the MAX-SAT evaluations maxsat.ia.udl.cat:81. We enriched this experimentation setting by also considering a second series of benchmarks, made of *all* the 295 instances used for the 2011 MUS competition organized in parallel with the SAT one. Note that in all Figures that we are going to present,

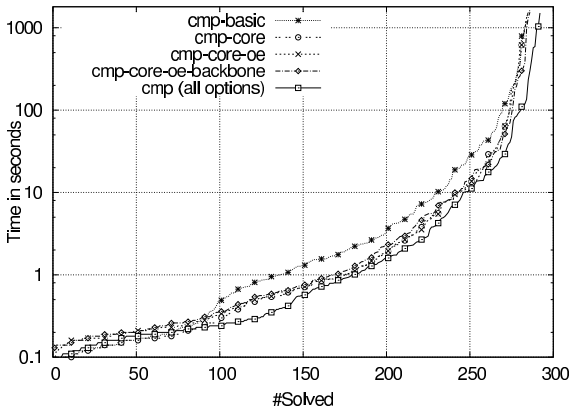

Figure 4: CMP variants on MUS instances

	SAT/MAX-SAT inst.	MUS inst.
Number of instances	1343	295
CMP-*basic*	1312	285
CMP-*core*	1321	285
CMP-*core-oe*	1321	286
CMP-*core-oe-backbone*	1322	286
CMP (all options)	1327	292

Table 1: Number of successfully partitioned instances

when the y-values represent the CPU times to partition a given number x of instances through different approaches, this does not mean that the x partitioned instances are necessarily identical and that each instance is partitioned in an identical way by all the considered approaches.

CMP is implemented in C++. MINISAT (Eén and Sörensson 2004) was selected as the CDCL SAT-solver. CMP and all experimentation data are available from www.cril.univ-artois.fr/documents/cmp/.

First, the actual benefits of the four optional parts of CMP have been assessed experimentally.

The basic version of CMP, i.e. Algorithm 2 without any of the options, was re-named CMP-*basic*. CMP-*basic* was then tentatively enhanced by taking the options into account in an incremental way. Only adding `exploitCore` (Algo2:line 15) gave rise to CMP-*core*; adding also the opposite-enforced feature (Algo2:line 8) yielded CMP-*core-oe*. CMP-*core-oe-backbone* was obtained by also activating the backbone literals tentative enhancement (Algo2:line 11). From now on, CMP denotes Algorithm 2 with all options, which thus included `extendSatPart` (Algo2:line 5) as well.

Figures 2 and 4 show gradual improvements when each of the options was taken into account in a cumulative way: each additional option allowed for some additional efficiency gain. Figure 3 is a focus on the rightest part of curves of Figure 2. Table 1 shows that the number of successful partitionings also increased according to the considered range of options. Noticeably, the `exploitCore` (*core*) option delivered the most significant improvement in terms of both computing time and number of successfully partitioned instances. Other ways to combine the options together allowed

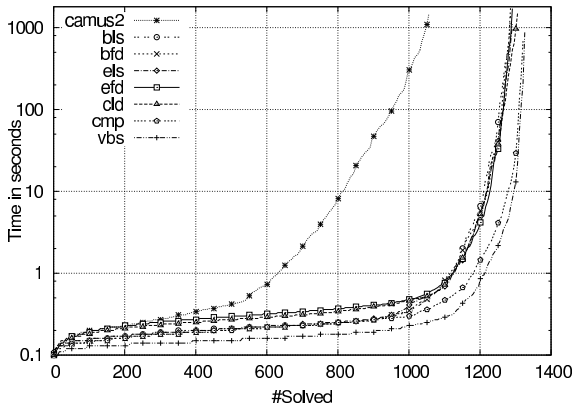

Figure 5: Comparison on SAT and MAX-SAT instances

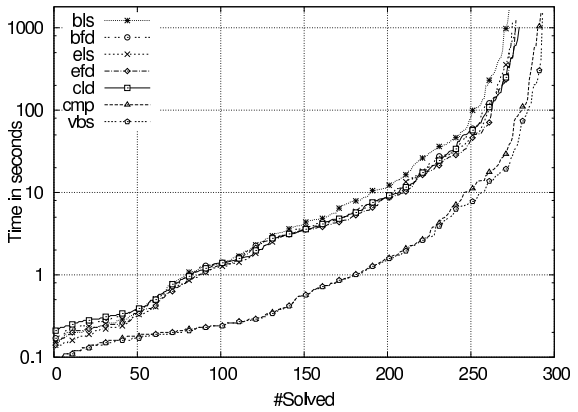

Figure 6: Comparison on MUS instances

similar observations to be made.

Then, we compared CMP with existing approaches that we mentioned earlier: namely, the BFD, BLS, CLD, EFD and ELS algorithms, which are described in (Marques-Silva et al. 2013) and implemented in the MCS_{LS} tool logos.ucd.ie/wiki/doku.php?id=mcsls. The version 2 of CAMUS (sun.iwu.edu/~mliffito/camus/ (Liffiton and Sakallah 2008; 2009)), named CAMUS2, was also tested on SAT/MAX-SAT benchmarks but this earlier system allowed a significantly smaller number of instances to be partitioned, only. Table 2 shows that CMP-*basic* itself allows more instances to be partitioned than any of the competitors. Figures 5 and 6 also compare the investigated approaches. We have also drawn the VBS (*Virtual Best Solver*) curve, which represents for each instance the best computing time amongst the tested methods. Clearly, CMP appeared best performing and was very close to VBS. For each method, Table 2 gives the number of instances that were successfully partitioned, with the highest score for CMP, too.

Discussion and perspectives

The extensive experimentations that we have conducted show that CMP is more robust than previous approaches and allows more Boolean instances to be successfully par-

	SAT/MAX-SAT inst.	MUS inst.
Number of instances	1343	295
BLS	1287	273
BFD	1287	276
ELS	1293	277
EFD	1291	277
CLD	1307	279
CMP-*basic*	1312	285
CMP (all options)	1327	292
VBS	1327	293

Table 2: Number of successfully partitioned instances

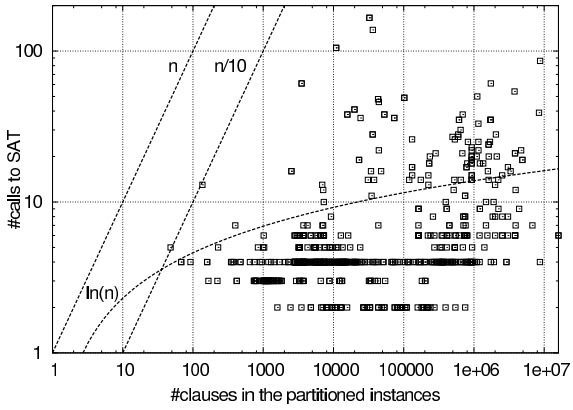

Figure 7: Number of calls by CMP to the SAT solver

titioned. Obviously, this does not entail that CMP would be more efficient for *every* instance. In this respect, it is worth mentioning that CMP exhibits a higher worst-case complexity than for example BLS, namely the basic linear search approach. Indeed, when n is the number of clauses in the instance, CMP requires $O(n^2)$ calls to a SAT solver in the worst case whereas BLS requires a linear number of such calls, only. Note however that the number of calls by CMP to the SAT solver always remains very significantly lower than n for all successfully partitioned instances (Figure 7).

We envision several paths for further research. First, the method could be enhanced by making use of incremental SAT solvers (Lagniez and Biere 2013; Audemard, Lagniez, and Simon 2013). Second, CMP can offer a way to approximate MAX-SAT when solving this latter problem is out of reach for hard instances. In this respect, although the features in CMP are not directly exportable to MAX-SAT algorithms, we believe that the way MSSes are computed in CMP can lead to novel ways to compute MAX-SAT. Also, CMP delivers approximate solutions for a basic version of MAX-SAT: in the future, we plan to study how to push the envelope in order to address *weighted* MAX-SAT. Finally, it would also be interesting to extend CMP to address the problem of enumerating CoMSS (or MSS) and study whether or not this could improve recent practical computational results about this issue (Marques-Silva et al. 2013).

References

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, IJCAI'09, 399–404.

Audemard, G., and Simon, L. 2012. Refining restarts strategies for sat and unsat. In *Principles and Practice of Constraint Programming - 18th International Conference (CP'12)*, volume 7514 of *Lecture Notes in Computer Science*, 118–126. Springer.

Audemard, G.; Lagniez, J.-M.; Mazure, B.; and Saïs, L. 2011. On freezing and reactivating learnt clauses. In *Proceedings of the 14th International Conference on Theory and Application of Satisfiability Testing (SAT'11)*, 188–200. Springer.

Audemard, G.; Lagniez, J.-M.; 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'13)*, 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 (PADL'2005)*, volume 3350 of *Lecture Notes in Computer Science*, 174–186. Springer.

Belov, A., and Marques-Silva, J. 2011. Accelerating MUS extraction with recursive model rotation. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design (FMCAD'11)*, 37–40. FMCAD Inc.

Besnard, P., and Hunter, A. 2008. *Elements of Argumentation*. The MIT 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.

Chinneck, J. W. 2008. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, volume 118 of *International Series in Operations Research & Management Science*. Springer.

Eén, N., and Sörensson, N. 2004. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03). Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, 502–518. Springer.

Felfernig, A.; Schubert, M.; and Zehentner, C. 2012. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM* 26(1):53–62.

Fermé, E. L., and Hansson, S. O. 2011. AGM 25 years - twenty-five years of research in belief change. *J. Philosophical Logic* 40(2):295–331.

Ginsberg, M. 1987. *Readings in nonmonotonic reasoning*. M. Kaufmann Publishers.

Grégoire, É., and Konieczny, S. 2006. Logic-based approaches to information fusion. *Information Fusion* 7(1):4–18.

Grégoire, É.; Lagniez, J.-M.; and Mazure, B. 2013a. Improving MUC extraction thanks to local search. *CoRR* abs/1307.3585.

Grégoire, É.; Lagniez, J.-M.; and Mazure, B. 2013b. Questioning the importance of WCORE-like minimization steps in MUC-finding algorithms. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'13)*, 923–930.

Grégoire, É.; Mazure, B.; and Piette, C. 2007a. Boosting a complete technique to find MSS and MUS thanks to a local search oracle. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, 2300–2305.

Grégoire, É.; Mazure, B.; and Piette, C. 2007b. Local-search extraction of MUSes. *Constraints* 12(3):325–344.

Guo, L., and Lagniez, J.-M. 2011. Dynamic polarity adjustment in a parallel SAT solver. In *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI'11)*, 67–73.

Hamscher, W.; Console, L.; and de Kleer, J., eds. 1992. *Readings in Model-Based Diagnosis*. Morgan Kaufmann.

Hemery, F.; Lecoutre, C.; Saïs, L.; and Boussemart, F. 2006. Extracting MUCs from constraint networks. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI'06)*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, 113–117. IOS Press.

Junker, U. 2004. QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI'04)*, 167–172. AAAI Press.

Kilby, P.; Slaney, J. K.; Thiébaux, S.; and Walsh, T. 2005. Backbones and backdoors in satisfiability. In *Proceedings, The 20th National Conference on Artificial Intelligence (AAAI'05)*, 1368–1373. AAAI Press / The MIT Press.

Lagniez, J.-M., and Biere, A. 2013. Factoring out assumptions to speed up mus extraction. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT'13)*, volume 7962 of *Lecture Notes in Computer Science*, 276–292. Springer.

Lecoutre, C. 2009. *Constraint Networks: Techniques and Algorithms*. Wiley.

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

Liffiton, M. H., and Sakallah, K. A. 2009. Generalizing core-guided Max-SAT. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *Lecture Notes in Computer Science*, 481–494. Springer.

Marques-Silva, J., and Sakallah, K. A. 1996. GRASP: a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design (ICCAD'96)*, 220–227. IEEE Computer Society.

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)*.

Meseguer, P.; Bouhmala, N.; Bouzoubaa, T.; Irgens, M.; and Sánchez, M. 2003. Current approaches for solving over-constrained problems. *Constraints* 8(1):9–39.

Monasson, R.; Zecchina, R.; Kirkpatrick, S.; Selman, B.; and Troyansky, L. 1999. Determining computational complexity from characteristic 'phase transitions'. *Nature* 400:133.

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 530–535. ACM.

Oh, Y.; Mneimneh, M. N.; Andraus, Z. S.; Sakallah, K. A.; and Markov, I. L. 2004. AMUSE: A minimally-unsatisfiable subformula extractor. In *Proceedings of the 41th Design Automation Conference (DAC'04)*, 518–523. ACM.

Papadimitriou, C. H. 1993. *Computational Complexity*. Addison-Wesley.

Pipatsrisawat, K., and Darwiche, A. 2007. A lightweight component caching scheme for satisfiability solvers. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, 294–299. Springer.

Rossi, F.; van Beek, P.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*. Elsevier.

Zhang, L., and Malik, S. 2002. The quest for efficient boolean satisfiability solvers. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, 17–36. Springer.

Zhang, L.; Madigan, C.; Moskewicz, M.; and Malik, S. 2001. Efficient conflict driven learning in boolean satisfiability solver. In *Proc. of the Proceedings of IEEE/ACM International Conference on Computer Design (ICCAD'01)*, 279–285.