

# Does This Set of Clauses Overlap with at Least One MUS?

Éric Grégoire, Bertrand Mazure, and Cédric Piette

Université Lille-Nord de France, Artois  
CRIL-CNRS UMR 8188  
F-62307 Lens Cedex, France  
{gregoire,mazure,piette}@cril.fr

**Abstract.** This paper is concerned with the problem of checking whether a given subset  $\Gamma$  of an unsatisfiable Boolean CNF formula  $\Sigma$  takes part in the basic causes of the inconsistency of  $\Sigma$ . More precisely, an original approach is introduced to check whether  $\Gamma$  overlaps with at least one minimally unsatisfiable subset (MUS) of  $\Sigma$ . In the positive case, it intends to compute and deliver one such MUS. The approach re-expresses the problem within an evolving coarser-grained framework where clusters of clauses of  $\Sigma$  are formed and examined according to their levels of mutual conflicts when they are interpreted as basic interacting entities. It then progressively refines the framework and the solution by splitting most promising clusters and pruning the useless ones until either some maximal preset computational resources are exhausted, or a final solution is discovered. The viability and the usefulness of the approach are illustrated through benchmarks experimentations.

## 1 Introduction

These last years, various research studies have concentrated on explaining *why* a SAT instance is inconsistent in terms of minimal subsets of clauses that are actually conflicting. Indeed, although some SAT solvers (e.g. [1,2]) can deliver the trace of a proof of inconsistency, this trace is often not guaranteed to deliver a set of conflicting clauses that would become consistent if any of its clauses was dropped. Accordingly, several recent contributions have investigated various computational issues about computing Minimal Unsatisfiable Subsets (in short, MUSes) of unsatisfiable SAT instances ([3,4], see [5] for a recent survey). From a worst-case complexity analysis, several major issues arise. First, an  $n$ -clauses SAT instance  $\Sigma$  can exhibit  $C_n^{n/2}$  MUSes in the worst case. Then, checking whether a given formula belongs to the set of MUSes of another CNF formula or not is a  $\sum_2^P$ -hard problem [6]. However, approaches to compute one MUS that appear viable in many circumstances have been proposed [7,8]. In order to circumvent the possibly exponential number of MUSes in  $\Sigma$ , variant problems have been defined and addressed in the literature, like the problem of computing a so-called cover of MUSes, which is a subset of clauses that contains enough minimal sources of conflicts to explain all unrelated reasons leading to the inconsistency of  $\Sigma$ , without computing all MUSes of  $\Sigma$  [8]. Finally, algorithms to compute the complete sets of

MUSes of  $\Sigma$  have also been proposed and shown viable [12,11], at least to some extent due to the possible combinatorial blow-up.

When a user is faced with an unsatisfiable SAT instance, she (he) can have some beliefs about which subset  $\Gamma$  of clauses is actually causing the conflicts within  $\Sigma$ . In this respect, she (he) might want to check whether her (his) beliefs are grounded or not. Apart from the situation where  $\Sigma \setminus \Gamma$  is satisfiable, she (he) might wish to check whether  $\Gamma$  shares a non-empty set-theoretical intersection with at least one MUS of  $\Sigma$ . In the positive case, she (he) might wish to be delivered such an MUS as well. Current techniques to find and compute MUSes do not accommodate those wishes without computing all MUSes of  $\Sigma$ .

An original approach is introduced in the paper to address those issues, without computing all MUSes of  $\Sigma$ . To circumvent the high worst-case complexity (at least, to some extent), it re-expresses the problem within an evolving coarser-grained framework where clusters of clauses of  $\Sigma$  are formed and examined according to their levels of mutual conflicts when they are interpreted as basic interacting entities. It then progressively refines the framework and the solution by splitting most promising clusters and pruning useless ones until either some maximal preset computational resources are exhausted, or a final solution is discovered. Interestingly, the levels of mutual conflicts between clusters are measured using a form of Shapley's values [9].

The paper is organized as follows. In the next Section, formal preliminaries are provided, together with basic definitions and properties about MUSes and Shapley's measure of conflicting information. The main principles guiding the approach are described in Section 3. The sketch of the any-time algorithm is provided in Section 4, while the empirical results are given and discussed in Section 5. Main other relevant works are then described before paths for future research are provided.

## 2 Logical Preliminaries, MUSes and Shapley's Measure

### 2.1 Logical Preliminaries and SAT

Let  $L$  be the propositional language of formulas defined in the usual inductive way from a set  $P$  of propositional symbols (represented using plain letters like  $a, b, c$ , etc.), the Boolean constants  $\top$  and  $\perp$ , and the standard connectives  $\neg, \wedge, \vee, \Rightarrow$  and  $\Leftrightarrow$ . A SAT instance is a propositional formula in conjunctive normal form (CNF for short), i.e. a conjunction (often represented through a set) of clauses, where a clause is a disjunction (also often represented as a set) of literals, a literal being a possibly negated propositional variable. In the following, plain letters like  $l, m, n$ , etc. will be used to represent formulas of  $L$ . Upper-case Greek letters like  $\Delta, \Gamma$ , etc. will be used to represent sets of clauses, and lower-case ones like  $\alpha, \beta, \gamma$  etc. to represent clauses.

SAT is the canonic NP-complete decision problem consisting in checking whether a SAT instance is satisfiable or not. In the following, the words satisfiable (resp. unsatisfiable) and consistent (resp. inconsistent) are used indifferently. Along the paper,  $\Sigma$  is an unsatisfiable SAT instance and  $\Gamma$  is a set of clauses s.t.  $\Gamma \subset \Sigma$ .

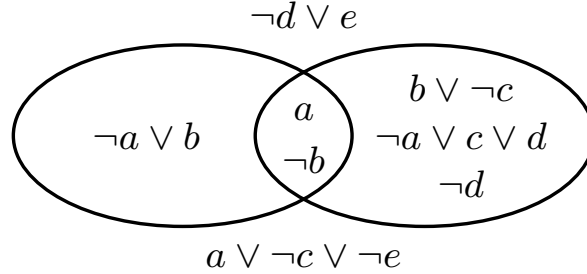


Fig. 1. MUSes of  $\Sigma$  from Example 1

## 2.2 MUSes of a SAT Instance

An MUS of an unsatisfiable SAT instance  $\Sigma$  is an unsatisfiable subset of clauses of  $\Sigma$  that cannot be made smaller without restoring its satisfiability. An MUS thus represents a minimal cause of inconsistency, expressed by means of conflicting clauses.

**Definition 1.** A set of clauses  $\Gamma$  is a *Minimally Unsatisfiable Subset (MUS)* of  $\Sigma$  iff

1.  $\Gamma \subseteq \Sigma$
2.  $\Gamma$  is unsatisfiable
3.  $\forall \Delta \subset \Gamma, \Delta$  is satisfiable.

The following example illustrates that MUSes can overlap one another.

**Example 1.** Let  $\Sigma = \{\neg d \vee e, b \vee \neg c, \neg d, \neg a \vee b, a, a \vee \neg c \vee \neg e, \neg a \vee c \vee d, \neg b\}$ .  $\Sigma$  is unsatisfiable and contains 2 MUSes which are illustrated in Figure 1, namely  $\{a, \neg a \vee b, \neg b\}$  and  $\{b \vee \neg c, \neg d, a, \neg a \vee c \vee d, \neg b\}$ .

This example also illustrates how clauses in  $\Sigma$  can play various roles w.r.t. the unsatisfiability of  $\Sigma$ . Following [10], clauses that belong to all MUSes of  $\Sigma$  are *necessary* (w.r.t. the unsatisfiability of  $\Sigma$ ). Removing any necessary clause from  $\Sigma$  restores consistency. *Potentially necessary* clauses of  $\Sigma$  belong to at least one MUS of  $\Sigma$  but not to all of them: removing one potentially necessary clause does not restore the consistency of  $\Sigma$ . Clauses that do not belong to any of those two categories (i.e. clauses belonging to no MUS at all) are called *never necessary*. Removing any combination of never necessary clauses cannot restore consistency. Obviously enough, an unsatisfiable SAT instance does not always contain necessary clauses since it can exhibit non-overlapping MUSes.

**Example 1 (cont'd).** In this example, both clauses “ $a$ ” and “ $\neg b$ ” are necessary w.r.t. the inconsistency of  $\Sigma$ , “ $\neg d \vee e$ ” and “ $a \vee \neg c \vee \neg e$ ” are never necessary whereas all the other clauses of  $\Sigma$  are potentially necessary.

Numerous approaches have been proposed to extract one MUS from  $\Sigma$  [5]. However, these approaches cannot take into account *a priori* conditions stating that this MUS

should overlap with  $\Gamma$ . Accordingly, in order to find such an overlapping MUS using the best current available techniques, one must resort to tools like [11,12] that compute all MUSes of  $\Sigma$ . Note that the most powerful of these techniques compute the Maximal Satisfiable Subsets (MSSes) of  $\Sigma$  [11] as a first necessary step, and then derive MUSes as hitting-sets of all the MSSes. Accordingly, these techniques are hardly tractable and cannot be used for many large and difficult SAT instances, even when the actual number of MUSes is not too large.

### 2.3 Shapley's Measure of Inconsistency

There exist various studies about the possible levels of inconsistency within a set of formulas. In the following, one form of Shapley's measure [9] of inconsistency proposed in [13] will be used. It can be formulated as follows.

Shapley's inconsistency measure of a clause  $\alpha$  in a SAT instance  $\Sigma$  delivers a score that takes the number of MUSes of  $\Sigma$  containing  $\alpha$  into account, as well as the size of each of these MUSes, enabling the involvement or "importance" of the clauses within each source of inconsistency to be considered, too.

**Definition 2.** [13] Let  $\Sigma$  be a SAT instance and  $\alpha \in \Sigma$ . Let  $\cup_{MUS_\Sigma}$  be the set of all MUSes of  $\Sigma$ . The measure of inconsistency  $MI$  of  $\alpha$  in  $\Sigma$ , noted  $MI(\Sigma, \alpha)$ , is defined as

$$MI(\Sigma, \alpha) = \sum_{\{\Delta s.t. \Delta \in \cup_{MUS_\Sigma} \text{ and } \alpha \in \Delta\}} \frac{1}{|\Delta|}$$

Properties of such a measure are investigated in [13]. Roughly, a clause that takes part in many conflicts is assigned a higher score, while at the same time a clause that occurs in a large MUS is given a lower score than a clause that occurs in an MUS containing a smaller number of clauses.

**Example 2.** Considering the CNF  $\Sigma$  of Example 1, we have:

- $MI(\Sigma, \neg a \vee b) = 1/3$
- $MI(\Sigma, b \vee \neg c) = 1/5$
- $MI(\Sigma, \neg d \vee e) = 0$
- $MI(\Sigma, a) = MI(\Sigma, \neg b) = 1/3 + 1/5 = 8/15$

In the following, this measure will be extended to characterize the conflict levels between conjunctive formulas.

## 3 Main Principles Underlying the Approach

### 3.1 Problem Definition

A set of Boolean clauses  $\Gamma$  *actually* participates in the inconsistency of a SAT instance  $\Sigma$  when  $\Gamma$  contains at least one clause that belongs to at least one MUS of  $\Sigma$ , namely a clause that is necessary or potentially necessary w.r.t. the inconsistency of  $\Sigma$ . In the positive case,  $\Gamma$  is said to *participate in the inconsistency* of  $\Sigma$  and the goal is to deliver one such MUS. More formally:

**Definition 3.** Let  $\cup_{MUS_{\Sigma}}$  be the set of MUSes of a set of Boolean clauses  $\Sigma$  and let  $\Gamma$  be a set of Boolean clauses, too.  $\Gamma$  participates in the inconsistency of  $\Sigma$  iff  $\exists \alpha \in \Gamma, \exists \Delta \in \cup_{MUS_{\Sigma}}$  s.t.  $\alpha \in \Delta$ .

Clearly enough, if  $\Gamma$  contains at least one necessary clause w.r.t. the inconsistency of  $\Sigma$  then  $\Gamma$  participates in the inconsistency of  $\Sigma$ . In this specific case, checking participation in inconsistency requires at most  $k$  calls to a SAT solver, where  $k$  is the number of clauses of  $\Gamma$ , since there exists  $\alpha$  in  $\Gamma$  s.t.  $\Sigma \setminus \{\alpha\}$  is satisfiable. Moreover, any MUS of  $\Sigma$  overlaps with  $\Gamma$  and provides a solution to the problem. However, in the general case, whether at least one necessary clause w.r.t. the satisfiability of  $\Sigma$  exists or not within  $\Gamma$  is unknown.

### 3.2 Using Clusters to Move to a Simplified Level

Accordingly, and in order to circumvent to some extent the high-level computational complexity of the problem, the approach in this paper resorts to a strategy that consists in partitioning  $\Sigma$  into a prefixed  $m$  number of subsets of clauses, called *clusters*, as a first step.

**Definition 4.** Let  $\Sigma$  be a SAT instance. An  $m$ -clustering of clauses  $\Pi$  of  $\Sigma$  is a partition of  $\Sigma$  into  $m$  subsets  $\Pi_j$  where  $j \in [1..m]$ .  $\Pi_j$  is called the  $j^{th}$  cluster of  $\Sigma$  (w.r.t. the clustering  $\Pi$  of  $\Sigma$ ).

For convenience reasons, the same notation  $\Pi_j$  will be used to represent both the set of clauses forming the  $j^{th}$  cluster of  $\Sigma$  and the conjunctive formula made of those clauses. Also, a set-theoretical union of clusters will be identified as the conjunction of the clauses that they contain.

All the clauses inside a cluster are then considered conjunctively to deliver a formula that is interpreted as being an indivisible interacting entity within  $\Sigma$ . The clauses of  $\Gamma$  are treated in the same way to form an additional conjunctive formula. At this point the initial problem is moved to a coarser-grained one that consists in checking how the conjunctive formula of  $\Gamma$  treated as a whole takes part in the inconsistency of the new representation of  $\Sigma$  that is now made of  $m$  individual subformulas that are considered as basic interacting entities.

### 3.3 From MUSes to MUSCes

As the problem is moved to a framework where inconsistency is analysed using clusters as basic interacting entities, the MUS concept should be adapted accordingly, giving rise to a concept of Minimally Unsatisfiable Set of Clusters for a clustering  $\Pi$  ( $MUSC_{\Pi}$  for short) of  $\Sigma$ .

**Definition 5.** Let  $\Sigma$  be a SAT instance and  $\Pi$  an  $m$ -clustering of clauses of  $\Sigma$ . A Minimally Unsatisfiable Set of Clusters for  $\Pi$  ( $MUSC_{\Pi}$  for short)  $M_{\Pi}$  of  $\Sigma$  is a set of clusters s.t.:

1.  $M_{\Pi} \subseteq \Pi$
2.  $M_{\Pi}$  is unsatisfiable
3.  $\forall \Phi \in M_{\Pi}, M_{\Pi} \setminus \{\Phi\}$  is satisfiable

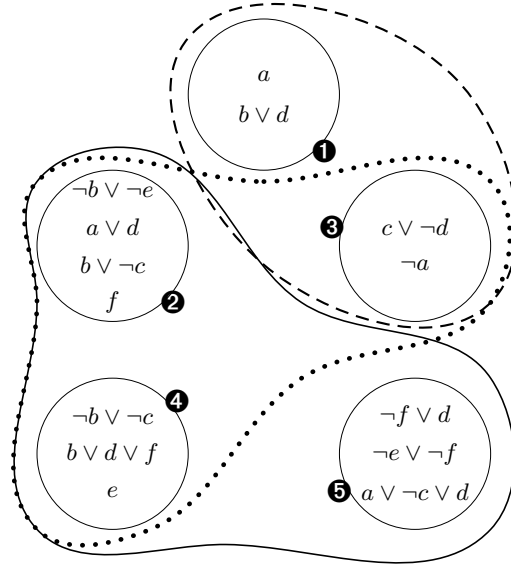


Fig. 2. MUSes of  $\Sigma$  from Example 3

The set of  $MUSC_{\Pi}$  of  $\Sigma$  is noted  $\cup_{MUSC_{\Pi}}$ .

The standard MUS concept is a particular instantiation of the MUSC one, where each cluster is actually one clause. Accordingly, MUSes represent the finest-grained level of abstraction in order to explain inconsistency of  $\Sigma$  by means of contradicting clauses. The MUSC concept allows more coarser-grained clusterings to be used, as ensured by the following property.

Let us consider the clauses that we want to prove they are involved in the inconsistency of  $\Sigma$ . When those clauses appear in an  $MUSC_{\Pi}$  of  $\Sigma$  (for some clustering  $\Pi$ ), they also participate in the inconsistency of  $\Sigma$  when this issue is represented at the lowest level of abstraction, namely at the level of clauses. More formally:

**Proposition 1.** *Let  $\Sigma$  be a SAT instance,  $\Pi$  an  $m$ -clustering of  $\Sigma$  and  $M_{\Pi}$  an  $MUSC_{\Pi}$  of  $\Sigma$ . If  $\Pi_i \in M_{\Pi}$  then  $\exists \alpha \in \Pi_i, \exists \Psi \in \cup_{MUS_{\Sigma}}$  s.t.  $\alpha \in \Psi$ .*

Such a problem transformation exhibits interesting features from a computational point of view that will be exploited by the approach. But there is no free lunch. As the new problem is a simplification of the initial one, some MUSes might disappear in the process. However, this simplification remains compatible with the initial goal and an algorithm will be proposed that is sound and complete in the sense that it always delivers an MUS of  $\Sigma$  overlapping with  $\Gamma$ , whenever this one exists and enough computational resources are provided.

**Example 3.** *Let  $\Sigma$  consists of 14 clauses, as depicted in Figure 2, forming 9 MUSes. For example, when the participation of the clauses “ $\neg a$ ” and “ $c \vee \neg d$ ” in the inconsistency of  $\Sigma$  needs to be checked and, in the positive case, when a corresponding MUS*

needs to be extracted, computing the exhaustive set of MUSes can appear necessary in the worst case. Let us cluster the clauses in the way depicted in Figure 2, giving rise to 5 clusters; the cluster ③ is the set of clauses  $\Gamma$  accordingly to our previous notation. At this level of abstraction, only 3  $MUSC_{\Pi}$  remain, i.e.  $M_1 = \{\mathbf{1}, \mathbf{3}\}$ ,  $M_2 = \{\mathbf{2}, \mathbf{3}, \mathbf{4}\}$  and  $M_3 = \{\mathbf{2}, \mathbf{4}, \mathbf{5}\}$ , using the labels of clusters represented in the Figure. This clustering limits the number of entities that are sources of inconsistency and that need to be considered in order to prove the membership of one of the two clauses “ $\neg a$ ” and “ $c \vee \neg d$ ” to an MUS of  $\Sigma$ .

This example illustrates the interest in clustering clauses in that it can avoid the need of extracting all MUSes in order to answer the question of the existence of an overlapping MUS. However, if the clauses are clustered and do not appear in an  $MUSC$  (for some clustering  $\Pi$ ), this does not entail that they do not participate in the inconsistency of the formula at the clauses level.

**Proposition 2.** *Let  $\Sigma$  be a SAT instance,  $\Pi$  an  $m$ -clustering of  $\Sigma$  and  $\Pi_i \in \Pi$ . Proposition 1 ensures that:*

$$\text{if } (\exists M_{\Pi} \in \cup_{MUSC_{\Pi}} \text{ s.t. } \Pi_i \in M_{\Pi}) \text{ then } (\exists \alpha \in \Pi_i, \exists \Psi \in \cup_{MUS_{\Sigma}} \text{ s.t. } \alpha \in \Psi)$$

*The converse does not hold.*

Accordingly, the fact that a cluster does not appear in any MUSC does not imply that no one of its clauses is not involved in an MUS of the considered instance. This is illustrated by the following example:

**Example 4.** *Let  $\Sigma = \{\neg c, \neg a, b \vee c, a, \neg b\}$ .  $\Sigma$  exhibits 2 MUSes:  $MUS_{\Sigma}^1 = \{\neg a, a\}$  and  $MUS_{\Sigma}^2 = \{\neg b, b \vee c, \neg c\}$ . Let us now consider a clustering  $\Pi$  of  $\Sigma$  s.t.  $\Pi_1 = \{\neg b, a\}$ ,  $\Pi_2 = \{\neg a, b \vee c\}$  and  $\Pi_3 = \{\neg c\}$ . Considering a table where each column represents a cluster and each line represents an MUS of  $\Sigma$ , we get:*

	$\Pi_1$	$\Pi_2$	$\Pi_3$
$MUS_{\Sigma}^1$	$a$	$\neg a$	
$MUS_{\Sigma}^2$	$\neg b$	$b \vee c$	$\neg c$

*At this level of abstraction, the only  $MUSC_{\Pi}$  is  $\{\Pi_1, \Pi_2\}$ , capturing  $MUS_{\Sigma}^1$ . However,  $MUS_{\Sigma}^2$  is not represented in this unique  $MUSC_{\Pi}$ , since its clauses are splitted in each cluster of  $\Pi$ . Particularly, this clustering does not permit the involvement of “ $\neg c$ ” in the inconsistency of  $\Sigma$  to be shown.*

Accordingly, a form of “subsumption” phenomenon between causes of inconsistency can occur when clauses are clustered in such a way. On the one hand, the number of detectable MUSes in  $\Sigma$  can decrease accordingly, and a subsequent process could be allowed to focus on selected remaining ones only. On the other hand, it should be ensured that too much information relevant to the inconsistency of  $\Sigma$  is not hidden in the process, making the goal of finding an overlapping MUS impossible to reach, although such an MUS would exist. To some extent, the last example also illustrates that the

way according to which the clusters are formed and thus how MUSes are disseminated amongst clusters can drastically influence the efficiency of the approach.

### 3.4 Measuring Conflicts amongst MUSCses

The measure of inconsistency is based on a form of Shapley's values described in Section 2.3, which needs to be adapted as follows to consider clusters as basic interacting entities.

**Definition 6.** Let  $\Sigma$  be a SAT instance and  $\Pi$  an  $m$ -clustering of  $\Sigma$ . The measure of inconsistency  $MI$  of a cluster  $\Pi_i$  of  $\Pi$  is defined as follows:

$$MI(\Pi_i) = \sum_{\{M \mid M \in \cup_{MUSC \Pi} \text{ and } \Pi_i \in M\}} \frac{1}{|M|}$$

Once the set of MUSCes has been extracted, this measure can be computed in polynomial time.

**Example 5.** Let  $\Sigma$  be the SAT instance and  $\Pi$  be a clustering corresponding to Example 3. We have:

- $MI(\Pi_1) = 1/2$
- $MI(\Pi_2) = MI(\Pi_4) = 2 \times 1/3 = 2/3$
- $MI(\Pi_3) = 1/2 + 1/3 = 5/6$
- $MI(\Pi_5) = 1/3$

The cluster that exhibits the highest score is  $\Pi_3$ . This is partly due to the fact that  $\Pi_3$  belongs to 2 MUSCes.  $\Pi_2$  and  $\Pi_4$  also appear inside two MUSCes, but each of them only represents a third of both sources of conflict, whereas  $\Pi_3$  is involved in an MUSC made of 2 clusters. The role of this latter cluster is thus very important in this source of conflict.

### 3.5 Initiating the Process

In order to form the initial clustering of clauses, a cheap scoring heuristic is used to estimate for each clause  $\alpha$  of  $\Sigma$  its probability of belonging to at least one MUS of  $\Sigma$ . This heuristic exploits a failed local-search for satisfiability of  $\Sigma$  and a so-called concept of *critical clause* that takes a relevant partial neighborhood of each explored interpretation into account [8].

$\Sigma$  is divided into a preset number  $m$  of clusters in the following way. The  $\frac{|\Sigma \setminus \Gamma|}{m}$  clauses of  $\Sigma \setminus \Gamma$  that exhibit the highest scores are (probably) taking part in a number of conflicts and their presence in  $\Sigma$  could be the cause of many MUSes. They are assembled to form the  $\Pi_1$  cluster. The remaining clauses of  $\Sigma \setminus \Gamma$  are then sorted in the same manner according to their scores to deliver the remaining  $m - 1$  clusters of  $\Pi$  and to provide a  $m$ -clustering of  $\Sigma \setminus \Gamma$ . The clauses of  $\Gamma$  are then conjuncted together to yield a  $m + 1^{th}$  cluster, noted  $\Pi_\Gamma$ . Accordingly, a  $m + 1$ -clustering of  $\Sigma$  is obtained.

### 3.6 Analysing the Conflicts at the Clusters Level

Each cluster is thus interpreted as a conjunctive formula and the interaction between these entities is analysed in terms of mutually contradicting (sets of) clusters. This conflict analysis is intended to prune the problem by rejecting clusters that are not conflicting with  $\Gamma$ , and in concentrating on the clusters that are most conflicting with  $\Gamma$ . The exhaustive set of MUSCs of this clustering (i.e.  $\cup_{MUSC_{\Pi}}$ ) is thus computed and three different cases might occur:

1. *Global inconsistency*: a form of global inconsistency occurs at the level of clusters when for every cluster  $\Pi_i$  ( $i \in [1..m]$ ) of the  $m$ -clustering  $\Pi$ ,  $\Pi \setminus \Pi_i$  is consistent. Such a situation occurs when at least one clause of each cluster is needed to form a conflict. The clustering does not provide any other useful information. The parameter  $m$  is increased to obtain a finer-grained clustering. Note that this situation does not require a lot of computational resources to be spent, since the current clustering contains only one MUSC and a linear number of maximal satisfiable sets of cluster need to be checked.
2.  *$\Gamma$  appears in at least one conflict*: some minimal conflicts can be detected between clusters, and at least one such conflict involves the  $\Pi_{\Gamma}$  cluster formed with  $\Gamma$ . More precisely,  $\exists M \in \cup_{MUSC_{\Pi}}$  s.t.  $\Pi_{\Gamma} \in M$ . The idea is to focus on such an MUSC since it involves the proof that some clauses in  $\Gamma$  minimally conflict with  $\Sigma$ . More precisely, there exists an MUS of  $\Sigma$  that necessarily contains clauses of  $\Gamma$  and clauses occurring in the other clusters of the MUSC. Accordingly, the current set of clauses can be pruned by dropping all clauses that are occurring in clusters not present in this particular MUSC.
3.  *$\Gamma$  does not appear in any conflict*: some minimal conflicts can be detected between clusters but no such conflict involves the cluster formed with  $\Gamma$ . More precisely,  $\nexists M \in \cup_{MUSC_{\Pi}}$  s.t.  $\Pi_{\Gamma} \in M$ . In such a situation, a refinement of the clustering is undertaken by splitting its *most conflicting* current cluster, using a measure that is described in paragraph 3.4. A prefixed number of clusters with the highest scores are splitted, motivated by the heuristic that they should be the most probable ones that could include and hide a number of other sources of inconsistency, and hopefully MUSes in which  $\Gamma$  is involved.

### 3.7 Iterating and Ending the Process

The clustering is thus modified following one of the three possible situations described above, allowing the approach to increase the number of clusters, split and focus on most promising subparts of  $\Sigma$ , mainly. This process is iterated. Accordingly, a  $|\Sigma|$ -clustering (i.e. a clustering where each cluster is actually a single clause) can be obtained in a finite time, corresponding to the “classical” computation of exhaustive approaches, on a reduced number of clauses of  $\Sigma$ . Actually, when the current size of the cluster that is most conflicting with  $\Gamma$  becomes manageable to envision the computation of all the MUSes of the set-theoretical union of  $\Gamma$  with  $\Sigma$ , such a computation is performed.

**Algorithm 1.**The look4MUS algorithm.

---

**Input:**  
 $\Sigma$ : a CNF  
 $\Gamma$ : a CNF s.t.  $\Gamma \subset \Sigma$   
 $m$ : size of the initial clustering  
 $inc$ : size of increment of the clustering (global inconsistency case)  
 $s$ : number of clusters to split (local inconsistency case)  
**Output:** An MUS of  $\Sigma$  overlapping  $\Gamma$  when such a MUS exists,  $\emptyset$  otherwise

```

1 begin
2    $S_\Sigma \leftarrow \text{scoreClauses}(\Sigma)$ ;
3    $\Pi \leftarrow \text{clusterClauses}(\Sigma \setminus \Gamma, S_\Sigma, m) \cup \{\Gamma\}$ ;
4    $\cup_{MUSC_\Pi} \leftarrow \text{HYCAM}^*(\Pi)$ ;
5   if  $\Pi$  is a  $|\Sigma|$ -clustering then
6     if  $\exists M \in \cup_{MUSC_\Pi}$  s.t.  $\Gamma \subseteq M$  then return  $M$ ;
7     else return  $\emptyset$ ;
8   else
9     if  $\forall \Pi_i \in \Pi, \Sigma \setminus \Pi_i$  is satisfiable then
10      // global inconsistency
11      return look4MUS( $\Sigma, \Gamma, m + inc, inc, s$ );
12    else
13      while  $\nexists M \in \cup_{MUSC_\Pi}$  s.t.  $\Gamma \in M$  do
14        // local inconsistency
15         $\Pi' \leftarrow \emptyset$ ;
16        for  $j \in [1..s]$  do
17           $\Pi_{best} \leftarrow \emptyset$ ;
18          foreach  $\Pi_i \in \Pi$  s.t.  $\Pi \neq \Gamma$  do
19            if  $\text{MI}(\Pi_i, \cup_{MUSC_\Pi}) > \text{MI}(\Pi_{best}, \cup_{MUSC_\Pi})$  then
20               $\Pi_{best} \leftarrow \Pi_i$ ;
21             $\Pi \leftarrow \Pi \setminus \Pi_{best}$ ;
22             $\Pi' \leftarrow \Pi' \cup \text{clusterClauses}(\Pi_{best}, S_\Sigma, 2)$ ;
23           $\Pi \leftarrow \Pi \cup \Pi'$ ;
24           $\cup_{MUSC_\Pi} \leftarrow \text{HYCAM}^*(\Pi)$ ;
25         $M_\Gamma \leftarrow \text{selectOneMUSC}(\cup_{MUSC_\Pi}, \Gamma)$ ;
26        return look4MUS( $M_\Gamma, \Gamma, m, inc, s$ );
27    end
28  end

```

---

One interesting feature of this approach thus lies in its any-time property: whenever a preset amount of computing resources is exhausted, it can provide the user with the current solution, namely the currently focused subpart of  $\Sigma$  that is conflicting with  $\Gamma$ . The proposed approach based on those ideas is described in the next Section and sketched in Algorithm 1.

---

**Function** scoreClauses

---

**Input:**  $\Sigma$ : a CNF**Output:** A score vector of clauses of  $\Sigma$ 

```

1 begin
2   for each clause  $c$  of  $\Sigma$  do
3      $S_{\Sigma}(c) \leftarrow 0$ ;
4    $I \leftarrow$  a random assignment of each variable of  $\Sigma$ ;
5   while a preset maximum of steps is reached do
6     for each clause  $c$  of  $\Sigma$  do
7       if  $c$  is critical w.r.t  $I$  in  $\Sigma$  then
8          $S_{\Sigma}(c) ++$ ;
9        $I \leftarrow I'$  s.t.  $I$  and  $I'$  differs exactly by one assignment of a Boolean variable;
10  return  $S_{\Sigma}$ ;
11 end

```

---



---

**Function** clusterClauses

---

**Input:**  $\Sigma$ : a CNF,  $S_{\Sigma}$ : a score vector,  $m$ : the size of the clustering**Output:** A  $m$ -clustering of  $\Sigma$ 

```

1 begin
2    $\Pi \leftarrow \emptyset$ ;
3   for  $i \in [1..m-1]$  do
4      $\Pi_i \leftarrow$  the  $(\frac{|\Sigma|}{m})^{th}$  clauses with the highest scores in  $\Sigma$ ;
5      $\Pi \leftarrow \Pi \cup \{\Pi_i\}$ ;
6      $\Sigma \leftarrow \Sigma \setminus \Pi_i$ ;
7    $\Pi \leftarrow \Pi \cup \{\Sigma\}$ ;
8   return  $\Pi$ ;
9 end

```

---

## 4 Main Algorithm

In this section, the reader is provided with a sketch of the Algorithm. For convenience reasons, specific cases that are handled in the actual implementation (e.g. situations where  $\Gamma$  is inconsistent or where  $\Sigma$  is consistent) are not described. Also, it does not include the presentation of a syntactic preprocessing that removes all occurrences of identical clauses of  $\Sigma$  but one (left in  $\Gamma$  whenever possible), allowing the number of MUSes to be reduced exponentially. Such situations are important to handle in practice in order to avoid pathological cases: they are simple to implement and do not offer original algorithmic interest. The algorithm also refers to the HYCAM technique, that allows the exhaustive set of MUSes of a SAT instance to be computed. Actually, a new version of HYCAM that is able to deal with clusters of clauses has been implemented and used in the experimental study. We do detail the internal algorithmic techniques of HYCAM\* which is just a cluster-oriented version of the algorithm presented in [12]. Let

**Function** selectOneMUSC**Input:**  $\cup_{MUSC_{\Pi}}$ : the set of MUSC w.r.t.  $\Pi$ ,  $\Gamma$ : a CNF**Output:** An MUSC of  $\cup_{MUSC_{\Pi}}$  containing  $\Gamma$ 

```

1 begin
2    $M_{\Gamma} \leftarrow \Pi$ ;
3   foreach  $M$  s.t.  $M \in \cup_{MUSC_{\Pi}}$  do
4     if  $\Gamma \subseteq M$  and  $|M| < |M_{\Gamma}|$  then
5        $M_{\Gamma} \leftarrow M$ ;
6   return  $M_{\Gamma}$ ;
7 end

```

**Function** MI**Input:**  $\Pi_i$ : an element of the clustering  $\Pi$ ,  $\cup_{MUSC_{\Pi}}$ : the set of  $MUSC_{\Pi}$ **Output:** The measure of inconsistency of  $\Pi_i$  w.r.t.  $\Pi$ 

```

1 begin
2    $mi \leftarrow 0$ ;
3   foreach  $MUSC_{\Pi} \in \cup_{MUSC_{\Pi}}$  do
4     if  $\Pi_i \in MUSC_{\Pi}$  then
5        $mi \leftarrow mi + \frac{1}{|MUSC_{\Pi}|}$ ;
6   return  $mi$ ;
7 end

```

us focus on the different steps of the approach depicted in Algorithm 1. The approach takes as input a CNF  $\Sigma$  together with a subpart  $\Gamma$  of  $\Sigma$ . Moreover, it also needs 3 positive integer parameters, namely  $m$ ,  $inc$  and  $s$ .

First, a local search is run for heuristically scoring the clauses w.r.t. their “constraintness” within  $\Sigma$ . More precisely, scores of all clauses are initially set to 0. During the local search process, scores of clauses which are *critical* [8] w.r.t. the current explored interpretation are increased. Roughly, a clause is critical if it is falsified by the current interpretation and if any neighbor interpretation that satisfies this clause falsifies another clause previously satisfied. Hence, a prefixed number of interpretations is explored by the local search, giving rise to a score for each clause of the CNF (see function `scoreClause` for more details).

Then, clauses are clustered (line 1) into  $\Pi$ , invoking the `clusterClauses` function. This function consists in conjuncting clauses that exhibit the highest scores to form a first cluster, then the remaining highest-scored clauses to form a second cluster, etc. until  $m$  clusters are delivered. Next, the modified version of HYCAM, noted HYCAM\* (line 1), is called to compute all MUSCs of  $\Pi$ .

Now,  $\cup_{MUSC_{\Pi}}$  has been computed and several cases are possible. Either  $\Pi$  is globally inconsistent (lines 1-1): at this stage, no information can be exploited to find a subset of conflicting clauses with  $\Gamma$  (see Section 3.6.1) and the clustering is refined by adding  $inc$  more clusters (in the sketch of the algorithm, this is done through a recursive call to `look4MUS`). Another case occurs when some local inconsistency within the

clustering is detected. Here, either  $\Gamma$  is involved in at least one local inconsistency exhibited by the procedure (see Section 3.6.2), or no one of them contains  $\Gamma$  (see Section 3.6.3). Actually, while  $\Gamma$  is not involved in any local MUSC of  $\Pi$  (lines 1-1), the  $s$  most conflicting clusters of  $\Pi$  (w.r.t. the Shapley value in  $\text{MI}$  function) are split to attempt to reveal sources of conflict involving  $\Gamma$ . When such local inconsistencies are exhibited, one MUSC  $M_\Gamma$  containing  $\Gamma$  is selected (function `selectOneMUSC`, which chooses the MUSC that contains the least possible number of clauses). The computation continues only considering the  $M_\Gamma$  CNF, which is a subformula of  $\Sigma$  that contains one MUS involving  $\Gamma$ .

Unless the preset amount of computing time is exhausted, the algorithm can only terminate in lines 1 or 1, when each cluster is actually a single clause ( $|\Sigma|$ -clustering). At this point, a “classical” call to `HYCAM` is performed, delivering either MUSes (the “MUSC” line 1 is actually an MUS since each cluster is a clause) containing information from  $\Gamma$ , or proving that no such MUS exists. In the actual implementation, a  $|\Sigma|$ -clustering does not need to be reached in order to call the `HYCAM` procedure: when the current number of clusters is close enough to the number of clauses (i.e. when  $m < 2 \times |\Sigma|$ ), then the procedure is ended up with an `HYCAM` call.

Moreover, let us note that once line 22 is reached, an approximation of the original problem has been computed. Consequently, the algorithm could be terminated at any time with an approximated solution. The actual implementation is provided with an any-time feature in the sense that the approach delivers the current considered subformula conflicting with  $\Gamma$  when the preset amount of computing resources is exhausted.

## 5 Experimental Results

A C implementation of the algorithm has been completed. As a case study, the following values have been selected as parameters:  $m = 30$ ,  $inc = 10$  and  $s = \frac{m}{10}$ . The algorithm has been run on various SAT benchmarks from <http://www.satcompetition.org>. All the experimentations have been conducted on Intel Xeon 3GHz under Linux CentOS 4.1. (kernel 2.6.9) with a RAM limit of 2GB.

As a case study again, for each SAT instance it has been attempted to show that the subformula made of the clauses that are occurring at the 1<sup>st</sup>, 3<sup>rd</sup>, 5<sup>th</sup> and 7<sup>th</sup> positions within the standard DIMACS CNF-format file was participating in the inconsistency of the instance, and find an MUS overlapping with this set. A sample of the results are reported in Table 1. For each result, we report the name of the tested benchmark together with its number of variables (`#var`) and clauses (`#cla`). In addition, the size of the specific wanted MUS (`#claMUS`) in term of number of clauses and the time (in second) needed to extract it are also proposed.

This table shows that specific large (and thus difficult to extract) MUSes can be efficiently extracted from SAT instances. Note that the exhaustive set of MSSes of most of these instances cannot be delivered in reasonable time with neither `CAMUS` [11] nor `HYCAM` [12]. Thus, preexisting approaches cannot answer the question about the involvement of  $\Gamma$  in the inconsistency of  $\Sigma$ .

On the contrary, clustering clauses in order to hide sources of inconsistency and focus on the ones involving the wanted clauses proved valuable in practice. While no

**Table 1.** Extracting one MUS of  $\Sigma$  overlapping with  $\Gamma$ : sample of experimental results

name	#var	#cla	#cla <sub>MUS</sub>	time
ldlx_c_mc_ex_bp_f	776	3725	1440	705
bf0432-007	1040	3668	1223	119
bf1355-075	2180	6778	151	22
bf1355-638	2177	6768	154	21
bf2670-001	1393	3434	134	8
dp05u04	1571	3902	820	1254
dp06u05	2359	6053	1105	2508
ezfact16_1	193	1113	319	28
ezfact16_2	193	1113	365	43
ezfact16_3	193	1113	297	31
ezfact16_10	193	1113	415	54

approach was able to answer this query for various realistic problems, the approach in this paper drastically reduced the computational effort by limiting the combinatorial explosion of the number of MUSes. For instance, the `dp05u04` and `dp06u05` problems (encoding the dining philosophers problem *à la* bounded model checking) contain a number of MUSes large enough to prevent them from being computed (or even enumerated) using the current available technology. `look4MUS` succeeds to compute one of the wished minimal sources of inconsistency -without extracting all of them- in 20 and 42 minutes, respectively. The situation is similar for other considered CNFs. For the `ezfact16_*` factorization circuits family, an explanation involving particular clauses can be delivered in less than 1 minute while exhaustive approaches show their limits with such problems. Obviously enough, the considered benchmarks are smaller than the very large CNFs containing tens of thousand clauses that can now be solved by the best SAT solvers. Nevertheless, for some of those instances, thanks to its any-time feature, the procedure was able to iterate several times and compute MUSes, delivering a subformula conflicting with  $\Gamma$ .

## 6 Other Relevant Works

Existing approaches to compute MUSes and address the specific problem investigated in this paper have been described in previous sections. Let us simply emphasize here that this piece of work involves a form of *abstraction* to reduce the computational cost of the initial problem. Moreover, due to its any-time property, it can be used as an *approximation* technique. In this respect, a large number of abstraction and approximation techniques have been proposed to address various problems related to propositional logic but that are different from the issue addressed in this paper. Let us mention some of them.

A seminal paper about the approximation of inference in propositional logic using limited resources is [14]. Also, approximate clausal entailment relationships where any step of the computation is decided in polytime have been proposed in [15,16], and extended to full classical logic in [17]. For an overview about propositional

approximations, the reader is referred to [18]. Finally, let us note that approximation of coherent-based reasoning has also been studied extensively in e.g. [19] and [20].

Finally, the idea of clustering clauses has already been proposed in the context of high-level constraints encoded within clauses [11]. Moreover, cases where no clause of  $\Gamma$  is involved in the inconsistency of the instance appear to be the worst case of the algorithm. However, if  $\Gamma$  is exclusively composed of never necessary clauses, then  $\Gamma$  can be satisfied by an *autarky* [21], defined as a partial interpretation satisfying each clause having one literal assigned. An algorithm [22] has been recently proposed for finding autarkies by modifying the CNF formula and considering an optimization problem.

## 7 Conclusions and Future Works

In this paper, an original “abstract-and-refine” technique for checking whether a set of clauses overlaps with at least one minimal unsatisfiable subset of a SAT instance has been described. Such an issue can prove useful when a user needs to restore the consistency of a CNF formula and has (or can get) some hints about where the conflicting clauses are located in the formula.

The approach proves orders of magnitude more efficiently than existing approaches, thanks to its step-by-step abstraction and reformulation paradigm. Especially, it does not require the whole set of MUSes of the instance to be computed. Another feature of the approach lies in its any-time character, allowing “rough” solutions to be provided when the preset computational resources are exhausted.

Interesting paths for future research include the investigation of several possible variants for the approach. For instance, a crucial point lies in the way according to which the clauses are clustered. Although the failed local-search heuristic and the concept of critical clauses proved to be efficient, it could be fruitful to investigate other forms of interactions between clauses in order to form clusters. Particularly, various techniques have been proposed to divide SAT instances into subinstances, and it would be interesting to study the viability of such approaches to form clusters. In addition, this study was focused on the “flat” Boolean framework where all clauses share a same importance. Users may have some qualitative or quantitative preferences about the information contained in  $\Sigma$ . In this case, they might want to extract the MUSC that best fits those preferences. In the future, we plan to investigate those issues.

## Acknowledgement

The authors gratefully thank the anonymous reviewers for their insightful comments. This work is partly supported by the “IUT de Lens”.

## References

1. Eén, N., Sörensson, N.: Minisat, <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat>
2. Biere, A.: Boolforce, <http://fmv.jku.at/boolforce>

3. Huang, J.: MUP: A minimal unsatisfiability prover. In: ASP-DAC 2005, Shanghai, China, pp. 432–437 (2005)
4. van Maaren, H., Wieringa, S.: Finding guaranteed MUSes fast. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 291–304. Springer, Heidelberg (2008)
5. Grégoire, É., Mazure, B., Piette, C.: On approaches to explaining infeasibility of sets of Boolean clauses. In: ICTAI 2008, vol. 1, pp. 74–83 (2008)
6. Eiter, T., Gottlob, G.: On the complexity of propositional knowledge base revision, updates and counterfactual. *Artificial Intelligence* 57, 227–270 (1992)
7. Zhang, L., Malik, S.: Extracting small unsatisfiable cores from unsatisfiable Boolean formula. In: SAT 2003, Portofino, Italy (2003)
8. Grégoire, É., Mazure, B., Piette, C.: Local-search extraction of MUSes. *Constraints Journal* 12(3), 325–344 (2007)
9. Shapley, L.: A value for n-person games. *Contributions to the Theory of Games II (Annals of Mathematics Studies 28)*, 307–317 (1953)
10. Kullmann, O., Lynce, I., Marques Silva, J.: Categorisation of clauses in conjunctive normal forms: Minimally unsatisfiable sub-clause-sets and the lean kernel. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 22–35. Springer, Heidelberg (2006)
11. Liffiton, M., Sakallah, K.: Algorithms for computing minimal unsatisfiable clause sets. *Journal of Automated Reasoning* 40(1), 1–33 (2008)
12. Grégoire, É., Mazure, B., Piette, C.: Boosting a complete technique to find MSS and MUS thanks to a local search oracle. In: IJCAI 2007, vol. 2, pp. 2300–2305. AAAI Press, Menlo Park (2007)
13. Hunter, A., Konieczny, S.: Measuring inconsistency through minimal inconsistent sets. In: KR 2008, Sydney, Australia, pp. 358–366 (2008)
14. Schaerf, M., Cadoli, M.: Tractable reasoning via approximation. *Artificial Intelligence* 74, 249–310 (1995)
15. Dalal, M.: Anytime families of tractable propositional reasoners. In: AI/MATH 1996, pp. 42–45 (1996)
16. Dalal, M.: Semantics of an anytime family of reasoners. In: ECAI 1996, pp. 360–364 (1996)
17. Finger, M.: Towards polynomial approximations of full propositional logic. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 11–20. Springer, Heidelberg (2004)
18. Finger, M., Wassermann, R.: The universe of propositional approximations. *Theoretical Computer Science* 355(2), 153–166 (2006)
19. Koriche, F., Sallantin, J.: A logical toolbox for knowledge approximation. In: TARK 2001, pp. 193–205. Morgan Kaufmann Publishers Inc., San Francisco (2001)
20. Koriche, F.: Approximate coherence-based reasoning. *Journal of Applied Non-Classical Logics* 12(2), 239–258 (2002)
21. Kullmann, O.: Investigations on autark assignments. *Discrete Applied Mathematics* 107, 99–137 (2000)
22. Liffiton, M., Sakallah, K.: Searching for autarkies to trim unsatisfiable clause sets. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 182–195. Springer, Heidelberg (2008)