

Tracking MUSes and Strict Inconsistent Covers

Éric Grégoire
CRIL-CNRS & IRCICA
Université d'Artois
rue Jean Souvraz SP18
F-62307 Lens Cedex France
gregoire@cril.univ-artois.fr

Bertrand Mazure
CRIL-CNRS & IRCICA
Université d'Artois
rue Jean Souvraz SP18
F-62307 Lens Cedex France
mazure@cril.univ-artois.fr

Cédric Piette
CRIL-CNRS & IRCICA
Université d'Artois
rue Jean Souvraz SP18
F-62307 Lens Cedex France
piette@cril.univ-artois.fr

Abstract—In this paper, a new heuristic-based approach is introduced to extract minimally unsatisfiable subformulas (in short, MUSes) of SAT instances. It is shown that it often outperforms current competing methods. Then, the focus is on inconsistent covers, which represent sets of MUSes that cover enough independent sources of infeasibility for the instance to regain satisfiability if they were repaired. As the number of MUSes can be exponential with respect to the size of the instance, it is shown that such a concept is often a viable trade-off since it does not require us to compute all MUSes but provides us with enough mutually independent infeasibility causes that need to be addressed in order to restore satisfiability.

I. INTRODUCTION

In this paper, the focus is on computational approaches to detect minimal unsatisfiable subformulas (MUSes) of unsatisfiable SAT instances. Detecting MUSes can prove valuable in many applications. For example, when we check the consistency of knowledge-bases, we prefer knowing which clauses are contradicting one another rather than only knowing that the whole base is inconsistent. MUSes provide such an information, as they represent the smallest explanations (in terms of the number of involved clauses) for unsatisfiability.

Unfortunately, computing MUSes exhibits a high worst-case complexity. Indeed, checking whether a set of clauses is a MUS is DP-complete [1], and checking whether a formula belongs to the set of MUSes of an unsatisfiable instance or not, is in Σ_2^P [2]. Moreover, the number of MUSes can be exponential in the size of the instance. Indeed, the number of MUSes of an n -clauses instance is $C_n^{n/2}$ in the worst case. However, let us stress that the number of MUSes remains often tractable in real-life situations. For example, in model-based diagnosis [3], based on experimental studies, it is often assumed that single faults occur, which is often translated by a limited number of MUSes.

Recently, several approaches have been proposed to approximate or compute MUSes. Unfortunately, they concern specific classes of clauses or they remain tractable for small instances, only. Among them, let us mention Bruni's work [4], who has shown how a MUS can be extracted in polynomial time through linear programming techniques for clauses exhibiting a so-called integral point property. However, only restrictive classes of clauses obey such a property (mainly Horn, renameable Horn, extended Horn, balanced and matched ones). Other studies about the complexity and algorithmic properties

of extracting MUSes for specific classes of clauses can be found in [5], [6] and [7]. In [8], Bruni has also proposed an approach that approximates MUSes by means of an adaptative search guided by clauses hardness. Zhang and Malik have described in [9] a way to extract MUSes by learning nogoods involved in the derivation of the empty clause by resolution. In [10], Lynce and Marques-Silva have proposed a complete and exhaustive technique to extract smallest MUSes. Oh and her co-authors have presented in [11] a Davis, Putnam, Logemann and Loveland DPLL-oriented approach that is based on a marked clauses concept to allow one to approximate MUSes. Liffiton and Sakallah have shown how MUSes can be computed through the dual concept of maximally satisfiable subsets [12].

In this paper, a new heuristic-based approach to approximate and compute MUSes is introduced. It is based on a concept of *critical clauses w.r.t. an interpretation* that allows us to refine the approach by [13] to locate approximations of MUSes. Although it exploits a heuristic information, let us stress that the approach is complete in the sense that it always delivers a MUS for any unsatisfiable instance. Then, a concept of *inconsistent cover* is introduced. Inconsistent covers represent sets of MUSes that cover enough independent sources of infeasibility that would allow the instance to regain satisfiability if they were fixed. As the number of MUSes can be exponential with respect to the size of the instance, such a concept can be a viable trade-off since it does not require us to compute all MUSes but provides us with enough infeasibility causes that would allow the instance to become satisfiable if they were all repaired.

The paper is organized as follows. In the next section, the concepts of MUS and inconsistent cover are presented formally. In section III, the crucial notion of critical clause w.r.t. an interpretation is introduced and analyzed. In section IV, the new approach to approximate or compute one MUS is presented. Extensive experimental results are given in section V. Before we conclude, section VI shows how the approach can be extended to compute strict inconsistent covers.

II. MUSes AND INCONSISTENT COVERS

Let \mathcal{L} be a standard Boolean logical language built on a finite set of Boolean variables, denoted a , b , etc. Formulas will be denoted using upper-case letters such as C . Sets of

formulas will be represented using Greek letters like Γ or Σ . An interpretation is a truth assignment function that assigns values from $\{true, false\}$ to every Boolean variable.

A formula is consistent or satisfiable when there is at least one interpretation that satisfies it, i.e. that makes it become *true*. An interpretation will be denoted by upper-case letters like I and will be represented by the set of literals that it satisfies. Actually, any formula in \mathcal{L} can be represented (while preserving satisfiability) using a set (interpreted as a conjunction) of clauses, where a clause is a finite disjunction of literals and where a literal is Boolean variable that can be negated. SAT is the canonical NP-complete problem that consists in checking whether a set of Boolean clauses is satisfiable or not, i.e. whether there exists an interpretation that satisfies all clauses in the set or not. Let us also recall the SAT-related optimization problem, namely max-SAT.

Definition 1 *Given a SAT instance Γ , max-SAT consists in finding the maximum number of clauses of Σ that can be satisfied under a same interpretation.*

When a SAT instance is unsatisfiable, it exhibits at least one minimally unsatisfiable subformula, in short one *MUS*.

Definition 2 *A MUS Γ of a SAT instance Σ is a set of clauses s.t.*

- 1) $\Gamma \subseteq \Sigma$
- 2) Γ is unsatisfiable
- 3) Every proper subset of Γ is satisfiable

In the following, another crucial concept is the notion of (*strict*) *inconsistent cover*.

Definition 3 *Two sets of clauses are independent if and only if their intersection is empty. One (strict) inconsistent cover IC of an unsatisfiable SAT instance Σ is a set-theoretic union of (independent) MUSes of Σ s.t. $\Sigma \setminus IC$ is satisfiable.*

It should be noted that a same unsatisfiable instance can exhibit several different strict inconsistent covers. For example, let $\Sigma = \{\neg a, \neg b, a \vee b, c, \neg d \vee b, \neg c \vee a, d\}$. Σ contains 3 MUSes: $MUS_1 = \{a \vee b, \neg a, \neg b\}$, $MUS_2 = \{d, \neg b, \neg d \vee b\}$ and $MUS_3 = \{c, \neg c \vee a, \neg a\}$. Σ contains 2 strict inconsistent covers, namely $IC_1 = MUS_1$ and $IC_2 = MUS_2 \cup MUS_3$.

Although a strict inconsistent cover does not provide us with the set of all MUSes that may be present in a formula, it gives us a series of minimal explanations for infeasibility that are sufficient to explain and potentially repair enough sources of unsatisfiability in order for the whole formula to regain satisfiability.

A straightforward result is that a strict inconsistent cover enables us to get a lower-bound of the number of unsatisfied clauses in a max-SAT solution of an unsatisfiable SAT instance.

Property 1 *Let Σ be an unsatisfiable SAT instance. Let IC be a strict inconsistent cover of Σ . Let $|IC|$ be the number of*

independent MUSes contained in IC . For any interpretation I of Σ , at least $|IC|$ clauses of Σ are falsified under I .

III. A NEW HEURISTIC TO DETECT MUSES

In [13] it is shown how local search can be helpful for approximating MUSes. The basic idea is that clauses that are often falsified during a failed local search for satisfiability belong most probably to MUSes, when the instance is actually unsatisfiable. When the score of a clause is the number of times it has been falsified during a failed local search (in short, failed LS), discriminating the clauses with a high score can deliver a good approximation of the set of MUSes. Such a heuristic has been studied in an extensive manner in [14] and [13]. It has also been extended in several ways to address decision and optimization problems that belong to higher levels of the polynomial hierarchy (see e.g. [15], [16], [17] and [18]).

In the following, we assume that the SAT instance is unsatisfiable. The above heuristic can require us to increment the score of clauses even when they do not actually belong to any MUS. Unless we solve the problem of finding MUSes itself, we can only rely on some heuristic indications about the extent to which a currently falsified clause could or could not belong to a MUS. In this respect, we claim that some relevant parts of the neighborhood of the current interpretation can be checked and provide more information about whether a currently falsified clause C should be counted or not. The idea is to take the structure of C into account and to increment the score of C only when it cannot be satisfied without conducting other clauses to be falsified in their turn. We shall see that this technique implements definitions that approximate a proposition that is intrinsic to clauses belonging to MUSes.

To illustrate this concept, let us use the following example. Let $\Delta = \{a \vee b \vee c, \neg a \vee b, \neg b \vee c, \neg c \vee a, \neg a \vee \neg b \vee \neg c\}$. Δ is unsatisfiable and is its own MUS. Let $I = \{a, b, c\}$ an interpretation. Under this interpretation, only the $\neg a \vee \neg b \vee \neg c$ clause is falsified. In the following, the *once-satisfied* clause concept will prove useful.

Definition 4 *A clause C is once-satisfied by an interpretation I if and only if exactly only one literal of C is satisfied under I .*

In the above example, the clauses $\neg a \vee b$, $\neg b \vee c$ and $\neg c \vee a$ are once-satisfied by $I = \{a, b, c\}$.

Definition 5 *A clause C falsified under an interpretation I is critical w.r.t. I if and only if the opposite of every literal of C belongs to a clause that is once-satisfied by I . These once-satisfied clauses that are not tautological ones are called linked to C .*

In the example, $\neg a \vee \neg b \vee \neg c$ is falsified under I and is critical w.r.t. I ; its related linked clauses are the once-satisfied ones $\neg a \vee b$, $\neg b \vee c$ and $\neg c \vee a$.

The role of these definitions is easily understood thanks to the following property.

Property 2 Let C be a critical clause w.r.t. an interpretation I , then any flip from I to I' such that C is satisfied under I' will conduct I' to falsify at least one clause that was satisfied under I .

In order to discriminate clauses belonging to MUSes, the idea is to increment the scores of critical clauses during the search, together with their linked (satisfied) clauses, rather than increment the scores of all falsified clauses. Such a technique can be easily grafted to a LS algorithm and the updates can be easily computed. Actually, it implements definitions that approximate a property that is obeyed by clauses belonging to MUSes.

Property 3 Let I be an interpretation giving an optimal result for max-SAT on an unsatisfiable instance Σ . Then, any falsified clause C w.r.t. I belongs to at least one MUS of Σ and is critical w.r.t. I . Moreover, at least one once-satisfied clause linked to C also belongs to a MUS of Σ .

Our technique is thus an approximation one in the sense that clauses and their linked ones are considered during the whole search, and not w.r.t. interpretations that are solutions of max-SAT. Indeed, being a critical clause is neither a necessary nor a sufficient condition to belong to a MUS. As the following example illustrates, a critical clause w.r.t. an interpretation that is not an optimal one w.r.t. max-SAT for an unsatisfiable formula might not belong to a MUS. Let $\Delta = \{a \vee d, \neg a \vee \neg b, \neg d \vee e, f, \neg e \vee \neg f\}$. Clearly, Δ is satisfiable. $\neg e \vee \neg f$ is falsified under $I = \{a, b, d, e, f\}$ and is critical w.r.t. I . Moreover, a clause from a MUS that is falsified under a given interpretation I is not necessary critical w.r.t. I , as the following example shows. Let $\Delta = \{a \vee d, b, \neg a \vee \neg b, \neg d \vee e, f, \neg e \vee \neg f\}$. Clearly, Δ is a minimal unsatisfiable set of clauses. $\neg a \vee \neg b$ is falsified under $I = \{a, b, d, e, f\}$. However, it is not critical w.r.t. I . Fortunately, the following property ensures that all clauses from a MUS can be scored by the heuristic.

Property 4 Let Γ be a MUS. For all clauses $C \in \Gamma$, there exists an interpretation I s.t. C is critical w.r.t. I .

This property ensures that any clause that takes part to a MUS can be critical w.r.t. at least one interpretation. As such, this property does not guarantee that our scoring heuristic will allow us to exhibit all clauses belonging to MUSes. Indeed, it does not indicate that a LS run will necessary increment the score of all such clauses at least once since LS does not necessary visit all interpretations. However, the following property and its corollary provide us with a good indication that LS will probably visit interpretations where clauses belonging to MUSes are critical. Indeed, it is well-known that LS is in general attracted by local minima. Property 5 ensures that all falsified clauses are critical in local or global minima.

Definition 6 A local minimum is an interpretation s.t. no flip can increase the number satisfied clauses. A global minimum

Algorithm 1: AOMUS algorithm

Input: an unsatisfiable SAT formula Σ
Output: an Approximation of One MUS of Σ

```

1 begin
2   stack  $\leftarrow \emptyset$ ;
3   while (LS + Score( $\Sigma$ ) fails to find a model) do
4     push( $\Sigma$ );
5      $\Sigma \leftarrow \Sigma \setminus \text{LowestScore}(\Sigma)$ ;
6   repeat
7      $\Sigma \leftarrow \text{pop}()$ ;
8   until  $\Sigma$  is UNSAT;
9   return  $\Sigma$ ;
10 end
```

(or max-SAT solution) is an interpretation delivering the maximal number of satisfied clauses.

Property 5 In (local or global) minima, all falsified clauses are critical.

A corollary ensures that at least one clause per MUS is critical in such minima.

Corollary 1 In (local or global) minima, at least one clause per MUS is critical.

IV. APPROXIMATING AND COMPUTING ONE MUS

In the following, it is shown that a meta-heuristic based on scoring critical clauses is viable in order to approximate or compute MUSes. Actually, due to implementation efficiency constraints, we update the scores of critical clauses only. Updating the scores of their linked clauses does not lead to dramatic performance improvements, at least w.r.t. our selected LS algorithm and tested benchmarks.

The main idea is as follows. Let Σ be an unsatisfiable SAT instance. While local search fails to find a model of Σ , we remove clauses of Σ with the lowest scores. We record the obtained sub-formulas on a stack. Next, the unsatisfiability of the last subformula where LS fails to find a model is checked. If this subformula is unsatisfiable, then it is an approximation of a MUS of Σ . Otherwise, this unsatisfiability test is repeated on the lastly recorded supersets of clauses, until one of them is proved unsatisfiable. This algorithm, called AOMUS (Approximate One MUS), is described in Algorithm 1.

Then an exact MUS can be obtained by a step-by-step minimization of the upper-approximation until the remaining clauses are proven to form a MUS (see [19] for an alternative method). This process is called *fine-tune* (see Algorithm 2). The order of tested clauses can be guided by the score of each clause.

Let us stress that this algorithm is complete in the sense that it always delivers one MUS for any unsatisfiable instance. The combination of AOMUS algorithm and *fine-tune* procedure

Algorithm 2: *fine-tune* procedure

Input: an approximation of a MUS of Σ **Output:** a MUS extracted from Σ

```
1 begin
2   foreach clause  $c \in \Sigma$  sorted w.r.t. their scores do
3     if ( $\Sigma \setminus \{c\}$  is unsatisfiable) then
4        $\Sigma \leftarrow \Sigma \setminus \{c\}$ ;
5   return  $\Sigma$  ;
6 end
```

is called *OMUS* (find One MUS) and is described in Algorithm 3.

Its efficiency directly depends on the quality of the upper-approximation. In the next section, experimental results show that the approximation delivered by *AOMUS* is often of a good quality, because a very small set of clauses is removed by the *fine-tune* step and in consequence a very small number of unsatisfiability tests are performed (when a clause belongs to the MUS, the test amounts to a consistency check).

Actually, we refined this basic procedure in the following manner. Assume that the current computed subformula is actually unsatisfiable; whenever a unique clause remains falsified during the LS run, we are sure that this clause belongs to all MUSes of this current subformula.

We mark these clauses as *protected* and they cannot be removed from Σ thereafter. Moreover, this information is kept all along the process because it can prove very useful during the minimization procedure. After the approximation is computed, the idea is to remove each clause to verify whether it participates to the cause of unsatisfiability of the formula or not. However, protected clauses do not need to be tested, because we know that removing one of them restores satisfiability. Furthermore, when the remaining falsified clauses contain protected clauses only, they form one exact MUS. In this case, the *fine-tune* step can be omitted since an exact MUS has been already extracted from the formula.

It appears that this refinement proves useful for many instances, and allows dramatic efficiency gains for both *AOMUS* and *OMUS* algorithms.

The parameters that were selected for these methods are as follows. As a case study, *Wsat* [20] with the *Rnovelty+* option was chosen as the LS procedure. The following parameters were fine-tuned based on extensive tests on various benchmarks. After each flip of the LS, the score of critical clauses is increased by the number of their linked clauses. This technique allows us to take the length of critical clauses into account, since the number of linked clauses depends on the length of the critical clause in terms of the number of involved literals. Now, clauses whose score is lower than $(\text{min-score} + \frac{\#Flips}{\#Clauses})$ are dropped, where *min-score* is the lowest score for a clause of Σ ; *#Flips* and *#Clauses* are the number of performed flips and the number of clauses in Σ , respectively. This procedure was tested extensively on various

Algorithm 3: *OMUS* algorithm

Input: an unsatisfiable SAT formula Σ **Output:** One MUS of Σ

```
1 begin
2    $\Sigma \leftarrow \text{AOMUS}(\Sigma)$  ;
3    $\Sigma \leftarrow \text{fine-tune}(\Sigma)$  ;
4   return  $\Sigma$  ;
5 end
```

UNSAT instances from several difficult benchmarks from DIMACS [21] and from the annual SAT competitions [22], and compared with other published approaches to compute MUSes, as described in the next section.

V. EXPERIMENTAL RESULTS

All experiments have been conducted on Pentium IV, 3Ghz under linux Fedora Core 4. As our results show, this approximation delivers an exact result most of the time. Moreover, the *fine-tune* procedure ensures that a MUS is actually obtained. As most current approaches do not guarantee that the delivered unsatisfiable sets of clauses are actually MUSes, we provide both the results of applying *AOMUS* and *OMUS*. However, on many instances, approximations delivered thanks to *AOMUS* appeared to be actual MUSes. Moreover, very often, the last subformula where LS fails to find a model is in fact unsatisfiable. Thus, in practice, the last loop of the *AOMUS* algorithm reduces to a unique inconsistency test, most of the time.

We compared our approach with an adaptation of *AOMUS* where *Score* is the basic heuristic of [13], which simply counts the number of times a clause is falsified. We also compared our approach with *zCore*, the core extractor of *zChaff* [9]. *zChaff* is currently one of the most efficient SAT solvers. We also ran Lynce and Marques-Silva's procedure [10], and took Bruni's [8] experimental results into account. For Bruni's technique, we only mention the experimental results obtained by the author, since this system is not available. Although a comparison with Bruni's technique is thus difficult to achieve from an experimental side, it appears that Bruni's technique has been experimented on small instances, only. *zCore* proved competitive for single-MUS instances but failed to deliver good results when several MUSes are present. Indeed, *zCore* does not concentrate on finding one MUS, but on finding proofs of unsatisfiability. Not surprisingly, our approach proved more efficient than the similar one where *Score* is based on [13] heuristic. Most often, it proved to be more competitive than all the other considered techniques when very large and difficult multi-MUSes instances were considered. Noticeably, it was also the only technique to perform in a competitive way on all benchmarks. Let us stress that the Lynce-Silva's procedure computes the smallest MUS, that *zCore* delivers an approximation of a MUS, whereas our *OMUS* and *AOMUS* procedures deliver one exact and one approximate MUS, respectively. Moreover, it should

TABLE I
EXPERIMENTAL RESULTS: APPROXIMATE ONE MUS (AOMUS) AND FIND ONE MUS (OMUS)

Instance	#var	#cla	Lynce&Silva [10]		Bruni [4]		zCore [9]		Scoring like [13]		AOMUS		OMUS	
			#cla	Time	#cla	#cla	Time	#cla	Time	#cla	Time	#cla	Time	
fpga10_11	220	1122		Time out	-	561	28.51	561	18.26	561	13.06	561	13.75	
fpga10_12	240	1344		Time out	-	672	71.27	561	30.11	561	16.9	561	17.03	
fpga10_13	260	1586		Time out	-	793	166.99	561	51.67	561	25.95	561	31.89	
fpga10_15	300	2130		Time out	-	1065	570.3	561	128.05	561	44.18	561	68.17	
fpga11_12	264	1476		Time out	-	738	112.53	738	66.8	738	65.49	738	66.3	
fpga11_13	286	1742		Time out	-	871	504.97	738	180.66	738	56.71	738	84.74	
fpga11_14	308	2030		Time out	-	1015	1565.6	738	415.32	738	69.55	738	304.4	
fpga11_15	330	2340		Time out	-		Time out	738	568.79	738	52.14	738	85.2	
aim100-1_6-no-2	100	160	53	224	54	54	0.05	53	0.268	53	0.38	53	0.38	
aim100-2_0-no-1	100	200		Time out	19	19	0.09	19	0.216	19	0.19	19	0.23	
aim200-1_6-no-3	200	320		Time out	86	83	0.07	83	0.37	83	0.44	83	0.83	
aim200-2_0-no-3	200	400		Time out	37	37	0.23	37	0.39	37	0.49	37	0.54	
aim50-1_6-no-4	50	80	20	1.18	20	20	0.04	20	0.163	20	0.16	20	0.17	
aim50-2_0-no-4	50	100	21	3.49	21	21	0.14	21	0.208	21	0.22	21	0.27	
2bitadd_10	590	1422		Time out	-	815	343.48	1212	42.752	806	189.47	716	268.5	
barrel2	50	159		Time out	-	77	0.04	100	0.35	77	0.36	77	0.44	
jnh10	100	850		Time out	161	68	0.88	128	9.35	79	42.25	79	42.9	
jnh20	100	850		Time out	120	102	0.23	104	21.68	87	48.93	87	75.76	
jnh5	100	850		Time out	125	86	0.39	140	12.653	88	46.2	86	46.87	
jnh8	100	850		Time out	91	90	0.22	162	28.964	69	90.53	67	99.07	
homer06	180	830		Time out	-	415	15.96	415	10.97	415	9.04	415	9.3	
homer07	198	1012		Time out	-	506	21.6	415	12.59	415	10.67	415	19.19	
homer08	216	1212		Time out	-	606	44.46	554	23.43	415	19.79	415	24.65	
homer09	270	1920		Time out	-	960	141.48	415	93.19	504	60.9	415	81.23	
homer10	360	3460		Time out	-	940	624.11	1614	148.27	503	466.94	415	513.11	
homer11	220	1122		Time out	-	561	23.44	561	41.68	561	15.6	561	16.32	
homer12	240	1344		Time out	-	672	76.19	708	25.92	564	41.03	561	62.34	
homer13	260	1586		Time out	-	793	152.13	579	67.38	561	76.66	561	78.51	
homer14	300	2130		Time out	-	1065	714.03	561	347.19	561	28.03	561	30.64	
homer15	400	3840		Time out	-		Time out	677	247.84	561	1048.28	561	1104.13	
homer16	264	1476		Time out	-	738	115.49	738	78.44	738	61.31	738	62.91	
homer17	286	1742		Time out	-	871	369.11	870	127.43	738	68.28	738	87.4	

More extensive results can be downloaded from <http://www.cril.univ-artois.fr/~piette>

be emphasized that MUSes that are discovered by the various approaches are not necessarily the same ones.

In Table I, some typical experimental results are given. Except for Bruni’s results which are just size results that we have extracted from [8], we provide both the experimental size of the discovered smallest unsatisfiable subsets, together with the CPU time in seconds to get them. Time-out indicates that no result has been obtained within 1 hour CPU time. For example, for the `homer14` instance, AOMUS delivered an approximate MUS made of 561 clauses within 28.03 s. Actually, this was an exact MUS, as it was found by OMUS in 30.64 s. Note that an AOMUS version based on [13] delivered the same result in 347.19 s. zCore delivered an approximate MUS made of 1065 clauses within 714 s. Actually, this approximate MUS was a superset of the MUS discovered by both AOMUS and OMUS. Also, it can be seen e.g. on the `fpga` benchmarks that AOMUS (i.e. our approach without the `fine-tune` procedure) delivered smaller unsatisfiable subsets than any other considered method, most often. Let us also emphasize that even on small instances like the `aim` ones, OMUS proved very competitive, as well.

VI. EXTRACTING STRICT INCONSISTENT COVERS

These last years, several approaches have been proposed to compute *all* MUSes of a SAT instance ([23], [12]).

Unfortunately, these computational approaches remain often intractable since, among other things, the number of MUSes in a formula can be exponential in the size of the formula. In this section, a novel method is introduced to compute *independent* MUSes, i.e. MUSes that do not share any clause. The idea motivating this approach is that independent MUSes express independent –uncorrelated– causes of infeasibility inside a same formula. This lead us to the concept of *strict inconsistent cover* of an unsatisfiable instance.

Although an inconsistent cover does not provide us with the set of all MUSes that may be present in a formula, it does however provide us with a series of minimal explanations for unsatisfiability that are sufficient to explain and potentially repair enough sources of infeasibility in order for the whole formula to regain satisfiability. Moreover, formulas can have MUSes that are very small with respect to the size of the formula; dropping or repairing such MUSes can sometimes be sufficient to regain satisfiability. Since strict inconsistent covers are composed of independent MUSes, one way to obtain a strict inconsistent cover is to compute a single MUS, then remove it from the formula, and repeat these operations until the remaining subformula becomes satisfiable. This method is described in the following *ICMUS* (find a strict Inconsistent Cover of MUSes) algorithm (see Algorithm 4).

In Table II, some typical experimental results are provided.

Algorithm 4: ICMUS algorithm

Input: an unsatisfiable SAT formula Σ **Output:** a strict Inconsistent Cover of Σ

```
1 begin
2    $IC \leftarrow \emptyset$  ;
3   while ( $\Sigma$  is unsatisfiable) do
4      $MUS \leftarrow \text{OMUS}(\Sigma)$  ;
5      $IC \leftarrow IC \cup MUS$  ;
6      $\Sigma \leftarrow \Sigma \setminus MUS$  ;
7   return  $IC$  ;
8 end
```

Unsurprisingly, a lot of instances exhibit very small inconsistent covers in terms of the number of involved clauses. For example, let us consider `ezfact16_2`. This formula contains 1113 clauses, and possesses one MUS that is composed of 41 clauses. This MUS is in fact a strict inconsistent cover because its removal restores the formula satisfiability. The industrial-related formula `term1_gr_rcs_w3` exhibits a strict inconsistent cover made of 11 small MUSes. Thanks to this result, we can deduce that all interpretations falsify at least 11 clauses. Indeed, as shown in Property 1 a strict inconsistent cover enables us to get an lower-bound of unsatisfied clauses in a max-SAT solution of an unsatisfiable instance. Indeed, the extracted MUSes do not share constraints and we know that all interpretations falsify at least one clause per MUS. Let us also note that inconsistent covers on `fpga` formulas suggest the presence of a form of symmetry in this type of instances.

VII. CONCLUSION

In this paper, we have introduced a concept of clauses that are critical with respect to a given interpretation during a local search run. The main properties of this concept have been analyzed formally. We have shown that it plays a crucial role in a new heuristic-based approach to approximate or compute MUSes. Accordingly, our experimental results show the very good performance of this new approach which tracks MUSes according to the trace of a failed local search run for consistency checking. We have also introduced a strict inconsistent cover concept. This concept allows us to avoid computing all MUSes when the goal is to explain enough causes of unsatisfiability that would allow the instance to regain satisfiability if they were all repaired. Accordingly, our heuristic-based approach has been extended to address the search for inconsistent covers. Once again, the approach proves efficient on many difficult benchmarks. In the future, we plan to explore how the critical clause concept could be refined in order to further improve the performance of our algorithms.

ACKNOWLEDGEMENT

This study has been supported by the EC under a FEDER grant and by the *Région Nord/Pas-de-Calais*. The authors thank the reviewers for their valuable comments.

REFERENCES

- [1] C. Papadimitriou and D. Wolfe, “The complexity of facets resolved,” *Journal of Computer and System Sciences*, vol. 37, no. 1, pp. 2–13, 1988.
- [2] T. Eiter and G. Gottlob, “On the complexity of propositional knowledge base revision, updates and counterfactual,” *Artificial Intelligence*, vol. 57, pp. 227–270, 1992.
- [3] C. L. Hamscher W. and de Kleer J., Eds., *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.
- [4] R. Bruni, “On exact selection of minimally unsatisfiable subformulae,” *Annals of Mathematics and Artificial Intelligence*, vol. 43, no. 1, pp. 35–50, 2005.
- [5] H. Büning, “On subclasses of minimal unsatisfiable formulas,” *Discrete Applied Mathematics*, vol. 107, no. 1–3, pp. 83–98, 2000.
- [6] G. Davydov, I. Davydova, and H. Büning, “An efficient algorithm for the minimal unsatisfiability problem for a subclass of cnf,” *Annals of Mathematics and Artificial Intelligence*, vol. 23, no. 3–4, pp. 229–245, 1998.
- [7] H. Fleischner, O. Kullman, and S. Szeider, “Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference,” *Theoretical Computer Science*, vol. 289, no. 1, pp. 503–516, 2002.
- [8] R. Bruni, “Approximating minimal unsatisfiable subformulae by means of adaptive core search,” *Discrete Applied Mathematics*, vol. 130, no. 2, pp. 85–100, 2003.
- [9] L. Zhang and S. Malik, “Extracting small unsatisfiable cores from unsatisfiable Boolean formula,” in *International Conference on Theory and Applications of Satisfiability Testing (SAT’03)*, Portofino (Italy), 2003.
- [10] I. Lynce and J. Marques-Silva, “On computing minimum unsatisfiable cores,” in *International Conference on Theory and Applications of Satisfiability Testing (SAT’04)*, Vancouver, 2004.
- [11] Y. Oh, M. Mneimneh, Z. Andraus, K. Sakallah, and I. Markov, “Amuse: a minimally unsatisfiable subformula extractor,” in *Design Automation Conference (DAC’04)*, 2004, pp. 518–523.
- [12] M. Liffiton and K. Sakallah, “On finding all minimally unsatisfiable subformulas,” in *International Conference on Theory and Applications of Satisfiability Testing (SAT’05)*, 2005, pp. 173–186.
- [13] B. Mazure, L. Saïs, and É. Grégoire, “Boosting complete techniques thanks to local search,” *Annals of Mathematics and Artificial Intelligence*, vol. 22, pp. 319–322, 1998.
- [14] B. Mazure, L. Saïs, and É. Grégoire, “A powerful heuristic to locate inconsistent kernels in knowledge-based systems,” in *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU’96)*, Granada (Spain), 1996, pp. 1265–1269.
- [15] É. Grégoire and D. Ansart, “Overcoming the christmas tree syndrome,” *International Journal on Artificial Intelligence Tools (IJAIT)*, vol. 9, no. 2, pp. 97–111, 2000.
- [16] L. Brisoux, É. Grégoire, and L. Saïs, “Checking depth-limited consistency and inconsistency in knowledge-based systems,” *International Journal of Intelligent Systems*, vol. 16, no. 3, pp. 333–360, 2001.
- [17] É. Grégoire, B. Mazure, and L. Saïs, “Using failed local search for SAT as an oracle for tackling harder A.I. problems more efficiently,” in *International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA’02)*, LNCS 2443, Springer, Varna (Bulgaria), 2002, pp. 51–60.
- [18] F. Boussemart, F. Hémerly, C. Lecoutre, and L. Saïs, “Boosting systematic search by weighting constraints,” in *European Conference on Artificial Intelligence (ECAI’04)*, Valencia (Spain), 2004, pp. 146–150.
- [19] J. Huang, “MUP: A minimal unsatisfiability prover,” in *Asia and South Pacific Design Automation Conference (ASP-DAC’05)*, Shanghai, China, 2005, pp. 432–437.
- [20] H. Kautz, B. Selman, and D. McAllester, “Walksat in the SAT 2004 competition,” in *International Conference on Theory and Applications of Satisfiability Testing (SAT’04)*, Vancouver, 2004.
- [21] DIMACS, “Benchmarks on SAT,” <ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/>.
- [22] SATLIB, “Benchmarks on SAT,” <http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html>.
- [23] J. Bailey and P. J. Stuckey, “Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization,” in *PADL*, 2005, pp. 174–186.

TABLE II
INCONSISTENT COVERS FOR VARIOUS CLASSES OF FORMULAS

Instance	#var	#cla	Time	#MUSes in the IC	(#var,#cla) for each MUS
dp02u01	213	376	1.19	1	(47,51)
dp03u02	478	1007	362	1	(327,760)
23.cnf	198	474	2.68	1	(165,221)
42.cnf	378	904	9	1	(315,421)
fpga10_11_uns_rcr	220	1122	56	2	(110,561) (110,561)
fpga11_12_uns_rcr	264	1476	128	2	(132,738) (132,738)
ca002	26	70	0.61	1	(20,39)
ca004	60	168	1.11	1	(49,108)
ca008	130	370	5.26	1	(110,255)
term1_gr_rcs_w3	606	2518	6180	11	(12,22) (21,33) (30,58) (12,22) (12,22) (12,22) (12,22) (12,22) (12,22) (12,22) (24,39) (21,33)
C220_FV_RZ_14	1728	4508	28	1	(10,14)
C220_FV_RZ_13	1728	4508	46	1	(9,13)
C170_FR_SZ_96	1659	4955	18	1	(81,233)
C208_FA_SZ_121	1608	5278	21	1	(18,32)
C168_FW_UT_851	1909	7491	83	1	(7,9)
C202_FW_UT_2814	2038	11352	304	1	(15,18)
jnh208	100	800	14	1	(76,119)
jnh302	100	900	63	2	(27,28) (98,208)
jnh310	100	900	184	2	(12,13) (90,188)
ezfact16_1	193	1113	203	1	(37,54)
ezfact16_2	193	1113	104	1	(32,41)
ezfact16_3	193	1113	207	1	(63,128)
3col40_5_3	80	346	4.64	1	(64,136)
fphp-012-010	120	1212	57	1	(120,670)

ANNEX: PROOFS

Proof of property 1

By definition, strict inconsistent covers contain MUSes with empty intersections. Since at least one clause per MUS is falsified under any interpretation I , at least $|IC|$ clauses are thus falsified under I . ■

Proof of property 2

If C is critical then for each literal l of C , $\exists C'$ s.t. C' is once-satisfied w.r.t. I and \bar{l} belongs to C' . C is falsified under I , thus l is *false* under I and \bar{l} is *true* under I . \bar{l} is the only satisfied literal of C' , accordingly if the value of l is reversed then C' becomes falsified. ■

Proof of property 3

Any falsified clause under I belongs to a MUS because I is optimal w.r.t. the number of satisfied clauses and at least one clause of each MUS cannot be satisfied. The fact that any falsified clause under I is critical is proved thanks to Property 5 since I is a global minimum. I is optimal w.r.t. the number of satisfied clauses, thus at most one clause per MUS is falsified. Also, if one flip allows us to satisfy one of

theses clauses, another clause of the MUS becomes falsified. Accordingly, at least one once-satisfied clause linked to a falsified clause under I belongs to a MUS of Σ . ■

Proof of property 4

Let Γ be a MUS and C be a clause s.t. $C \in \Gamma$. By definition of a MUS, $\Gamma \setminus C$ is satisfiable. Let M be a model of $\Gamma \setminus C$. Let us prove that C is critical w.r.t. M . First, C is falsified. Indeed, if C is not falsified then Γ exhibits a model M . This is impossible because Γ is a MUS. Second, C is critical. Indeed, if any variable occurring in C is flipped w.r.t. M , then at least one clause of Γ becomes falsified since Γ is unsatisfiable. That means that this newly falsified clause was once-satisfied and linked to C . Accordingly, C is critical w.r.t. M . ■

Proof of property 5

If a variable occurring in a falsified clause w.r.t. a minimum is flipped, then this clause is satisfied and at least one previously satisfied clause becomes unsatisfied. That means that this new unsatisfied clause was once-satisfied. Accordingly, the initial falsified clause was critical. ■