Boosting a Complete Technique to Find MSS and MUS thanks to a Local Search Oracle

Éric Grégoire and **Bertrand Mazure** and **Cédric Piette**
CRIL-CNRS & IRCICA – Université d'Artois
rue Jean Souvraz SP18 F-62307 Lens Cedex France
{gregoire,mazure,piette}@cril.univ-artois.fr

Abstract

In this paper, a new complete technique to compute Maximal Satisfiable Subsets (MSS) and Minimally Unsatisfiable Subformulas (MUS) of sets of Boolean clauses is introduced. The approach improves the currently most efficient complete technique in several ways. It makes use of the powerful concept of critical clause and of a computationally inexpensive local search oracle to boost an exhaustive algorithm proposed by Liffiton and Sakallah. These features can allow exponential efficiency gains to be obtained. Accordingly, experimental studies show that this new approach outperforms the best current existing exhaustive ones.

1 Introduction

This last decade, the SAT problem, namely the issue of checking whether a set of Boolean clauses is satisfiable or not, has received much attention from the AI research community. Indeed, SAT appears to be a cornerstone in many domains, like e.g. nonmonotonic reasoning, automated reasoning, model-based diagnosis, planning and knowledge bases verification and validation. However, only knowing that a SAT instance is unsatisfiable is often not satisfactory since we might prefer knowing what goes *wrong* with the instance when this latter one is expected to be satisfiable.

In this respect, the MUS (*Minimally Unsatisfiable Subformula*) concept can be crucial since a MUS can be seen as an irreducible cause for infeasibility. Indeed, a MUS is an unsatisfiable set of clauses that is such that any of its subsets is satisfiable. It thus provides one explanation for unsatisfiability that cannot be made shorter in terms of the number of involved clauses. Restoring the satisfiability of an instance cannot be done without fixing all its MUS.

Unfortunately, a same instance can exhibit several MUS. Actually, the number of these MUS can be exponential since a n-clauses SAT instance can exhibit $C_n^{n/2}$ MUS in the worst case. Moreover, computing MUS is intractable in the general case. Indeed, checking whether a set of clauses is a MUS or not is DP-complete [Papadimitriou and Wolfe, 1988] and checking whether a formula belongs to the set (clutter) of MUS of an unsatisfiable SAT instance or not, is in Σ_2^p [Eiter and Gottlob, 1992]. Fortunately, the number of MUS remains often tractable in real-life applications. For example, in model-based diagnosis [Hamscher *et al.*, 1992], it is often assumed that single faults occur most often, which can entail small numbers of MUS.

A dual concept is the notion of *Maximal Satisfiable Subset* (MSS) of a SAT instance, and the complement of a MSS in a SAT instance is called a CoMSS. The complete sets of MUS and MSS are an implicit encoding of the other [Liffiton and Sakallah, 2005]. Specifically, a CoMSS is a hitting set of the set of MUS and represent minimal sets of clauses that should be dropped in order to restore consistency. In this paper, we are interested in exhaustive approaches to compute these correlated concepts in the Boolean clausal framework.

Recently, several approaches have been proposed to approximate or compute MUS and MSS, both in the Boolean framework and for other types of constraints. Some of them concern specific classes of clauses or remain tractable for small instances, only. Among them, let us mention the approach in [Bruni, 2005], where it is shown how a MUS can be extracted in polynomial time through linear programming techniques for clauses exhibiting a so-called integral property. However, only restrictive classes of clauses obey such a property (mainly Horn, renamable Horn, extended Horn, balanced and matched ones). Let us also mention [Büning, 2000; Davydov *et al.*, 1998; Fleischner *et al.*, 2002], which are other important studies in the complexity and the algorithmic aspects of extracting MUS for specific classes of clauses. In [Bruni, 2003], an approach is proposed that approximates MUS by means of an adaptative search guided by clauses hardness. In [Zhang and Malik, 2003] a technique is described, that extracts MUS by learning nogoods involved in the derivation of the empty clause by resolution. In [Lynce and Marques-Silva, 2004], a complete and exhaustive technique to extract smallest MUS is introduced. In [Oh *et al.*, 2004], a DPLL-oriented approach has been presented that is based on a marked clauses concept to allow one to approximate MUS. In [Grégoire *et al.*, 2006a], a heuristic-based incomplete approach to compute MUS has been introduced, which outperforms competing ones from a computational point of view.

Interestingly, in [Grégoire *et al.*, 2006b] the same authors have introduced a concept of inconsistent cover to circumvent the possible intractable number of MUS, and presented a technique to compute such covers. Roughly, an inconsistent cover of an unsatisfiable SAT instance represents a set of MUS that covers enough independent causes of inconsistency that would allow the instance to regain consistency if they were repaired. Although an inconsistent cover does not provide us with the set of all MUS that may be present in a formula, it does however provide us with a series of minimal explanations of inconsistency that are sufficient to explain and potentially "fix" enough causes of inconsistency in order for the whole instance to regain consistency.

These latter techniques are incomplete ones in the sense that they do not necessarily deliver all MUS. However, in some application domains, it can be necessary to find the set of *all* MUS, because diagnosing infeasibility is hard, if not impossible, without a complete view of its causes [Liffiton and Sakallah, 2005]. Obviously enough, such techniques can only remain tractable provided that the number of MUS remains itself tractable. Likewise, the number of MSS and CoMSS can be exponential in the worst case. It should be noted that many domains in Artificial Intelligence like belief revision (e.g. [Bessant *et al.*, 2001]) involve conceptual approaches to handle unsatisfiability that can require the complete sets of MUS, MSS, and CoMSS to be computed in the worst case, even when additional epistemological ingredients like stratification are introduced in the logical framework.

In this paper, the focus is on complete techniques. We introduce a new complete technique to compute all MUS, MSS and CoMSS of a SAT instance, provided obvious tractability limitations. It improves the currently most efficient complete technique, namely Liffiton and Sakallah'one [Liffiton and Sakallah, 2005] (in short L&S), which in turn was shown more competitive than previous approaches by Bailey and Stuckey [Bailey and Stuckey, 2005], and by de la Banda, Stuckey and Wazny [de la Banda *et al.*, 2003], which were introduced in somewhat different contexts.

Our approach exhibits two main features. First, it is a hybridization of the L&S complete approach with a local search pretreatment. A local search technique is indeed used as an oracle to find potential CoMSS of the SAT instance, which are themselves hitting sets of MUS. We show that such a hybridization can yield exponential efficiency gains. Second, the efficiency of the approach relies on the crucial concept of critical clause, which appears to be a powerful ingredient of our technique to locate MUS.

The rest of the paper is organized as follows. First, the reader is provided with the necessary background about SAT, MUS and the dual concepts of MSS and CoMSS. Then, Liffiton and Sakallah's exhaustive approach is briefly presented. In Section 4, we show how this technique can be valuably hybridized with a local search pretreatment, making use of the critical clause concept. It is shown how this pretreatment can be theoretically valuable from a computational point of view. In Section 5, we compare this new approach with L&S.

2 Background

In this section, we provide the reader with the basic background about SAT, MUS, MSS and CoMSS.

Let $\mathcal{L}$ be the Boolean logical language built on a finite set of Boolean variables, noted a, b, c, etc. The $\wedge$, $\vee$, $\neg$ and $\Rightarrow$ symbols represent the standard conjunctive, disjunctive, negation and material implication connectives, respectively.

Formulas and clauses will be noted 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 satisfiable when there is at least one interpretation (called model) that satisfies it, i.e. that makes it become *true*. An interpretation will be noted 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, where a literal is a Boolean variable that is possibly negated. SAT is the 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.

When a SAT instance is unsatisfiable, it exhibits at least one *Minimally Unsatisfiable Subformula*, in short one *MUS*.

Definition 1. *A* MUS Γ *of a SAT instance* Σ *is a set of clauses s.t.* $\Gamma \subseteq \Sigma$, Γ *is unsatisfiable and* $\forall \Delta \subset \Gamma$, Δ *is satisfiable.*

Example 1. *Let* $\Sigma = \{a, \neg c, \neg b \vee \neg a, b, \neg b \vee c\}$. Σ *exhibits two MUS, namely* $\{a, b, \neg b \vee \neg a\}$ *and* $\{\neg c, b, \neg b \vee c\}$.

A dual concept is the notion of *Maximal Satisfiable Subset* (MSS) of a SAT instance.

Definition 2. *A* MSS Γ *of a SAT instance* Σ *is a set of clauses s.t.* $\Gamma \subseteq \Sigma$, Γ *is satisfiable and* $\forall \Delta \subseteq (\Sigma \setminus \Gamma)$ *s.t.* $\Delta \neq \emptyset$, $\Gamma \cup \Delta$ *is unsatisfiable.*

The set-theoretical complement of a MSS w.r.t. a SAT instance is called a *CoMSS*.

Definition 3. *The CoMSS of a MSS* Γ *of a SAT instance* Σ *is given by* $\Sigma \setminus \Gamma$.

Example 2. *Let us consider the formula* Σ *from the previous example.* Σ *exhibits five CoMSS:* $\{b\}$, $\{\neg c, a\}$, $\{\neg c, \neg b \vee \neg a\}$, $\{\neg b \vee c, \neg b \vee \neg a\}$ *and* $\{\neg b \vee c, a\}$.

As shown by several authors [Liffiton and Sakallah, 2005], these concepts are correlated. Mainly, a CoMSS contains at least one clause from each MUS. Actually, a CoMSS is an irreducible hitting set of the set of MUS. In a dual way, every MUS of a SAT instance is an irreducible hitting set of the CoMSS. Accordingly, as emphasized by [Liffiton and Sakallah, 2005] although `MINIMAL-HITTING-SET` is a NP-hard problem, irreducibility is a less strict requirement than minimal cardinality. Actually, a MUS can be generated in polynomial time from the set of CoMSS.

3 Liffiton and Sakallah's Exhaustive Approach

Liffiton and Sakallah's approach [Liffiton and Sakallah, 2005] to compute all MUS (in short L&S) is based on the strong duality between MUS and MSS. To the best of our knowledge, it is currently the most efficient one. First it computes all MSS before it extracts the corresponding set of

MUS. Here, the focus is on L&S first step since we shall improve it and adopt the second step as such.

L&S is integrated with a modern SAT solver and takes advantage of it. Roughly, every ith clause $C_i = x_1 \vee ... \vee x_m$ of the SAT instance is augmented with a negated clause selector variable y_i to yield $C_i^{'} = x_1 \vee ... \vee x_m \vee \neg y_i$. While solving these new clauses, assigning y_i to *false* has the effect of disabling or removing C_i from the instance. Accordingly, a MSS can be obtained by finding a satisfying assignment with a minimal number of y_i variables assigned *false*. The algorithm makes use of a sliding objective approach allowing for an incremental search. A bound on the number of y_i that may be assigned to *false* is set. For each value of the bound, starting at 0 and incrementing by 1, an exhaustive search is performed for all satisfiable assignments to the augmented formula $C_i^{'}$, which will find all CoMSS having their size equal to the bound. Whenever one solution is found, it is recorded, and a corresponding clause forcing out that solution (and any supersets of it) is inserted. This blocking clause is a disjunction of the y_i variables for the clauses in that CoMSS.

Before beginning the search with the next bound, the algorithm checks that the new instance augmented with all the blocking clauses is still satisfiable without any bound on the y_i variables. If this formula is unsatisfiable, the entire set of CoMSS has been found and the algorithm terminates.

The second part of the algorithm computes the complete set of MUS from the set of CoMSS in a direct way. The approach that we shall introduce will include this second step as such.

4 Local Search and Critical Clauses

In this section, it is shown how the aforementioned exhaustive search algorithm can be improved in a dramatic manner by hybridizing it with an initial local search step, which provides valuable oracles for the subsequent exhaustive search process. We shall call the new approach HYCAM (HYbridization for Computing All Mus).

First, let us motivate our approach in an intuitive manner. Clearly, a (fast) initial local search run for satisfiability on the initial instance might encounter some actual MSS. Whenever this phenomenon happens, it can prove valuable to record the corresponding CoMSS in order to avoid computing them during the subsequent exhaustive search. Moreover, rather than checking whether we are faced with an actual MSS or not, it can prove useful to record the corresponding candidate CoMSS that will be checked later during the exhaustive search. Obviously enough, we must study which interpretations encountered during the local search process yield candidate MSS and criteria must be defined in order to record a limited number of potentially candidate CoMSS only. In this respect, a concept of critical clause will prove extremely valuable in the sense that it allows us to state necessary conditions for being a CoMSS that can be checked quickly. When all the remaining candidate CoMSS are recorded, the incremental approach by Liffiton and Sakallah allows us to exploit this information in a very valuable and efficient way. Let us describe this in a more detailed manner.

A local search algorithm is thus run on the initial SAT instance. The goal is to record as many candidate CoMSS as possible, based on the intuitive heuristics that local search often converges towards local minima, which could translate possibly good approximations of MSS. A straightforward approach would consist in recording for each visited interpretation the set of unsatisfied clauses. Obviously enough, we do not need to record supersets of already recorded candidate CoMSS since they cannot be actual CoMSS as they are not minimal with respect to set-theoretic inclusion. More generally, we have adapted the technique proposed by Zhang in [Zhang, 2005] to sets of clauses in order to record the currently smaller candidate CoMSS among the already encountered series of sets of unsatisfied clauses. Now, crucial ingredients in our approach are the concepts of once-satisfied and critical clauses. The latter concept has already proved valuable for locating MUS and inconsistent covers using an incomplete technique based on local search [Grégoire *et al.*, 2006a; 2006b].

Algorithm 1: Local Search approximation

Input: a CNF formula Σ **Output**: Set of candidate CoMSS

begin
  $candidates \longleftarrow \emptyset$; $\#fail \longleftarrow 0$;
  $I \longleftarrow generate\_random\_interpretation()$;
  while *($\#fail <$ PRESET_#FAILURES_AUTHORIZED)* **do**
    $newcandidates \longleftarrow$ FALSE ;
    for $j = 1$ **to** *#FLIPS* **do**
      Let Δ be the set of falsified clauses by I ;
      if $\forall C \in \Delta$, *C is critical*
      and Δ *is not implied in candidates* **then**
        removeAllSetImplied(Δ,$candidates$) ;
        $candidates \longleftarrow \Delta \cup candidates$;
        $newcandidates \longleftarrow$ TRUE ;
      flip(I) ;
    if $not(newcandidates)$ **then** $\#fail \longleftarrow \#fail + 1$;
  return $candidates$;
end

Algorithm 2: The HYCAM algorithm

Input: a CNF formula Σ **Output**: All MSS of Σ

begin
  $cand \longleftarrow$ LS_approximation(Σ) ; $k \longleftarrow 0$;
  $\Sigma_y \longleftarrow$ addSelectorClauses(Σ) ; $MSS \longleftarrow \emptyset$;
  while *SAT(Σ_y)* **do**
    removeAllSetImplied($\{\Sigma \backslash C | C \in MSS\}$,$cand$) ;
    $\Sigma_y \longleftarrow$ addBlockingClausesOfSize(k,$cand$) ;
    $MSS \longleftarrow MSS \cup \{\Sigma \backslash C | C \in cand$ and $|C| = k\}$;
    $MSS \longleftarrow MSS \cup$ SAT_with_bound(k,Σ_y) ;
    $k \longleftarrow k + 1$;
  return MSS ;
end

Definition 4. *A clause C is once-satisfied by an interpretation I iff exactly one literal of C is satisfied by I. A clause C that is falsified by the interpretation I is critical w.r.t. I iff the opposite of each literal of C belongs to at least one once-satisfied clause by I.*

Intuitively, a critical clause is thus a falsified clause that requires at least another one to be falsified in order to become satisfied, performing a flip. Property 1 shows how this concept allows us to eliminate wrong candidate CoMSS.

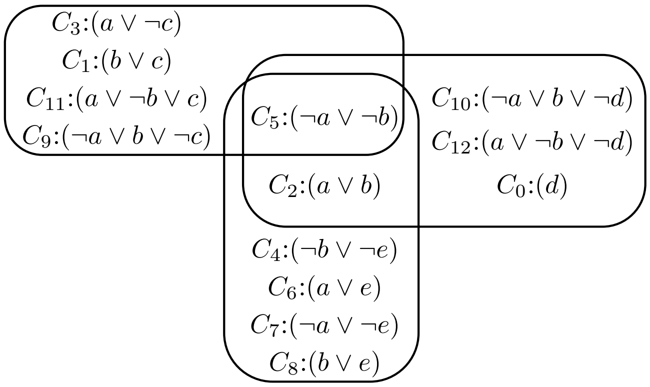

Figure 1: MUS of Example 3

Property 1. *Let Σ be a SAT instance and let I be an interpretation. Let Γ be a non-empty subset of Σ s.t. all clauses of Γ are all falsified by I. When at least one clause of Γ is not critical w.r.t. I, then Γ is not a CoMSS of Σ.*

Proof. By definition, when a clause C_f of Γ is not critical w.r.t. I, there exists at least one literal $c \in C_f$ whose truth-value can be inversed (i.e. flipped) without falsifying any other clause of Σ. Accordingly, Γ is not minimal and cannot be a CoMSS of Σ. □

In practice, testing whether all falsified clauses are critical or not can be performed quickly and prevents many sets of clauses to be recorded as candidate CoMSS. Using these features, the local search run on the initial SAT instance yields a series of candidate CoMSS. This information proves valuable and allows us to boost L&S complete search.

L&S is incremental in the sense that it computes CoMSS of increasing sizes, progressively. After n iterations have been performed, all CoMSS of cardinality lower or equal than n have been obtained. Accordingly, if we have recorded candidate CoMSS containing $n+1$ clauses, and if they are not supersets of already obtained CoMSS, we are sure that they are actual CoMSS. In this respect, we do not need to search them, and their corresponding blocking clauses can be inserted directly. Moreover, we do not need to perform the SAT test at the end of the n-th iteration, since we are then aware of the existence of larger CoMSS.

It is also easy to show that the insertion of these blocking clauses can allow both NP-complete and CoNP-complete tests to be avoided. Let us illustrate this on an example.

Example 3. *Let Σ be the following unsatisfiable SAT instance and let Σ' be the corresponding augmented SAT instance using L&S clauses selector variables y_i.*

$$\Sigma = \begin{cases} C_0 \ :(d) & C_1 \ :(b \vee c) & C_2:(a \vee b) \\ C_3 \ :(a \vee \neg c) & C_4 \ :(\neg b \vee \neg e) & C_5:(\neg a \vee \neg b) \\ C_6 \ :(a \vee e) & C_7 \ :(\neg a \vee \neg e) & C_8:(b \vee e) \\ C_9 \ :(\neg a \vee b \vee \neg c) & C_{10}:(\neg a \vee b \vee \neg d) & \\ C_{11}:(a \vee \neg b \vee c) & C_{12}:(a \vee \neg b \vee \neg d) & \end{cases}$$

Σ is an unsatisfiable SAT instance made of 13 clauses and making use of 5 variables. It exhibits 3 MUS, which are illustrated in Figure 1, and admits 19 MSS. Assume that both L&S and HYCAM are run on this instance. Its clauses are augmented by the $\neg y_i$ negated clause selector variables. Assume also that the local search performed by HYCAM provides 4 candidate CoMSS: $\{C_5\}$, $\{C_3, C_2\}$, $\{C_0, C_1, C_2\}$ and $\{C_3, C_8, C_{10}\}$.

If the branching variables are chosen based on the lexical order, then a and b are assigned to $true$ and C_5 is falsified. Thus, L&S tries to prove that this clause forms a CoMSS, which requires a NP-complete test (because it has to find a model of $\Sigma \backslash \{\neg a \vee \neg b\} \cup \{a, b\}$). On the contrary, when HYCAM is run, the blocking clause y_5 is added before the first iteration of the complete algorithm is performed, since the local search has already delivered this CoMSS. In consequence, when a and b are assigned $true$, the DPLL-algorithm backtracks immediately as the $\{y_5, \neg y_5\}$ unsatisfiable set has been obtained, without requiring any further NP-complete test.

Similarly, the introduction of additional clause selector variables by HYCAM can reduce the number of CoNP-complete tests. For example, let us assume that e is the first branching variable, that e is assigned $false$ and that the next variables are selected according to the lexical order. When a and b are assigned $true$, L&S tries to prove that $\{C_5\}$ is a CoMSS. Since $\neg e$ is tautological consequence of $\Sigma \backslash \{\neg a \vee \neg b\} \cup \{a, b\}$, no model exists for $\Sigma \backslash \{\neg a \vee \neg b\} \cup \{a, b, \neg e\}$. Clearly, such a test is in CoNP. Thanks to the previously delivered candidate CoMSS, HYCAM avoids this part of the search space to be explored. Indeed, since we know that $\{C_5\}$ is a CoMSS, when a and b are assigned $true$, no further CoNP tests are performed with respect to this partial assignment.

In fact, from a computational point of vue, the preliminary non-expensive local search eliminates nodes in the search tree, avoiding both NP and CoNP tests.

5 Experimental Evaluation

HYCAM has been implemented and compared to L&S from a practical point of view. For both algorithms, the complete search step is based on the use of MiniSat [Eén and Sörensson, 2004], which is currently one of the best modern SAT solvers. As a case study, we used Walksat [Kautz and Selman, 2004] for the local search pretreatment. The number of flips and tries of Walksat is related to the number of candidate CoMSS already found. For each try, a small number of flips is performed. If no new candidate is found during a try then a counter is incremented. When this counter exceeds a threshold (experimentally set to 30), we consider that no new candidate could be found by the local search. This way to end the local search pretreatment offers a good trade-off between the number of candidates found and the time spent. Besides, for all experiments, the time consumed by the local search step was less than 5% of the global time. All our experimental studies have been conducted on Intel Xeon 3GHz under Linux CentOS 4.1. (kernel 2.6.9) with a RAM memory size of 2Go. In the following, a time-out limit has been set to 3 CPU hours.

First, in Table 1a, we report experimental results about the computation of MSS on pigeon-hole and xor-chains benchmarks, which are globally unsatisfiable in the sense that removing any one of their clauses makes the instance become satisfiable. Obviously enough, such instances exhibit a num-

(a)

Inst.	(#v,#c)	#MSS	#CoMSS cand.	#CoMSS act.	L&S (sec.)	HYCAM (sec.)
hole6	(42,133)	133	133	133	0.040	0.051
hole7	(56,204)	204	204	204	0.75	0.33
hole8	(72,297)	297	293	293	33	1.60
hole9	(90,415)	415	415	415	866	30
hole10	(110,561)	561	559	559	7159	255
x1_16	(46,122)	122	122	122	0.042	0.041
x1_24	(70,186)	186	186	186	7.7	0.82
x1_32	(94,250)	250	241	241	195	28
x1_40	(118,314)	314	314	314	2722	486

- **Inst.**, **Instance**: benchmark name
- **(#v,#c)**: numbers of variables and clauses
- **#MSS**: number of MSS of the SAT instance
- **#CoMSS cand., act.**: numbers of candidate and actual CoMSS exhibited by HYCAM local search pretreatment respectively
- **L&S**: time in seconds for Liffiton and Sakallah's algorithm
- **HYCAM**: time in seconds for HYCAM

(b)

Instance	(#v,#c)	#MSS	#CoMSS cand.	#CoMSS act.	L&S (sec.)	HYCAM (sec.)
rand_net40-25-10	(2000,5921)	5123	4318	2729	893	197
rand_net40-25-5	(2000,5921)	4841	6950	598	650	174
rand_net40-30-10	(2400,7121)	5831	3458	2405	1748	386
rand_net40-30-1	(2400,7121)	7291	4380	662	1590	1325
rand_net40-30-5	(2400,7121)	5673	2611	2507	2145	402
ca032	(558,1606)	1173	1159	1159	4	1
ca064	(1132,3264)	2412	2324	2263	59	3
ca128	(2282,6586)	4899	2878	2422	691	18
ca256	(4584,13236)	9882	9553	9064	*t.o.*	277
2pipe	(892,6695)	3571	2094	1849	130	36
2pipe_1_ooo	(834,7026)	3679	1822	1587	52	30
2pipe_2_ooo	(925,8213)	5073	2286	1825	148	61
3pipe_1_ooo	(2223,26561)	17359	7327	3481	5153	2487
am_5_5	(1076,3677)	1959	3250	65	68	57
c432	(389,1115)	1023	1019	1019	4	1
c880	(957,2590)	2408	2141	1866	28	3
bf0432-007	(1040,3668)	10958	3332	2136	233	98
velev-sss-1.0-cl	(1453,12531)	4398	2987	2154	1205	513

Table 1: L&S vs. HYCAM on globally unsatisfiable instances (a) and on various difficult SAT instances (b)

ber of CoMSS equals to their number of clauses, and the size of any of their CoMSS is one. A significant time gap can be observed in favor of HYCAM. The efficiency gain ratio is even more significant when the size of the instance increased. For these instances, the local search run often succeeds in finding all CoMSS, and the complete step often reduces to an unsatisfiability test. On the contrary, L&S explores many more nodes in the search space to deliver the CoMSS.

In Table 1b, experimental results on more difficult benchmarks from the annual SAT competition [SATLIB, 2000] are described. Their number of MSS is often exponential, and computing them often remains intractable. Accordingly, we have limited the search to CoMSS of restricted sizes, namely we have set a size limit to 5 clauses. As our experimental results illustrate, HYCAM outperforms L&S. For example, let us consider `rand_net40-30-10`. This instance contains 5831 MSS (with the size of their corresponding CoMSS less than 5). L&S and HYCAM deliver this result in 1748 and 386 seconds, respectively. For the `ca256` instance, HYCAM has extracted 9882 MSS in less than 5 minutes whereas L&S did not manage to produce this result within 3 hours. Let us note that HYCAM also delivers CoMSS made of 5 clauses after its computation is ended since we know that all sets of 5 falsified clauses recorded by the local search run and that are not supersets of the obtained smaller CoMSS are actually also CoMSS.

In Table 2, experimental results on hard instances to compute the complete set of MSS and MUS are reported. As explained above, both L&S and HYCAM approaches require all MSS to be obtained before MUS are computed. By allowing complete sets of MSS to be delivered in a shorter time, HYCAM allows the complete set of MUS to be computed for more instances and in a faster manner than L&S does. Obviously enough, when the number of MSS or MUS are exponential, computing and enumerating all of them remain intractable.

For instance, L&S was unable to compute all MSS of the `php-012-011` instance within 3 hours CPU time, and could thus not discover its single MUS. HYCAM extracted it in 2597 seconds. On all instances exhibiting unique or a non-exponential number of MUS, HYCAM was clearly more efficient than L&S. For example, on the `dlx2_aa` instance, L&S and HYCAM discovered the 32 MUS within 3.12 and 0.94 seconds, respectively. Let us note that the additional time spent to compute all MUS from the set of MSS is often very small unless of course the number of MUS is exponential.

6 Conclusions and Future Research

Computing all MSS, CoMSS and MUS are highly intractable issues in the worst case. However, it can make sense to attempt to compute them for some real-life applications. In this paper, we have improved the currently most efficient exhaustive technique, namely Liffiton and Sakallah's method, in several ways. Our experimental results show dramatic efficiency gains for MSS, CoMSS and MUS extracting. One interesting feature of the approach lies in its anytime character for computing MSS. MSS of increasing sizes are computed gradually. Accordingly, we can put a bound on the maximum size of the CoMSS to be extracted, limiting the computing resources needed to extract them. To some extent, both L&S and HYCAM prove more adapted to extract complete sets of MSS and CoMSS than complete sets of MUS. Indeed, the procedure involves computing MSS (and thus CoMSS) first. In this respect, we agree with Liffiton and Sakallah that an interesting path for future research concerns the study of how MUS could be computed progressively from the growing set of extracted MSS.

Many artificial intelligence research areas have studied various problems involving the manipulation of MUS, MSS and CoMSS, like model-based diagnosis, belief revision, inconsistency handling in knowledge and belief bases, etc. These studies are often conducted from a conceptual point of view,

Instance	(#v,#c)	#MSS	#CoMSS cand.	#CoMSS act.	L&S (sec.)	HYCAM (sec.)	#MUS	MSS→MUS (sec.)
mod2-3cage-unsat-9-8	(87, 232)	232	232	232	3745	969	1	0.006
mod2-rand3bip-unsat-105-3	(105, 280)	280	280	280	2113	454	1	0.008
2pipe	(892, 6695)	10221	3142	1925	298	226	$> 211\,000$	*time out*
php-012-011	(132, 738)	738	734	734	*time out*	2597	1	0.024
hcb3	(45, 288)	288	288	288	10645	6059	1	0.006
1dlx_c_mc_ex_bp_f	(776, 3725)	1763	946	665	10.4	6.8	$> 350\,000$	*time out*
hwb-n20-02	(134, 630)	622	588	583	951	462	1	0.01
hwb-n22-02	(144, 688)	680	627	626	2183	811	1	0.025
ssa2670-141	(986, 2315)	1413	1374	1341	2.83	1.08	16	0.15
clqcolor-08-05-06	(116, 1114)	1114	1114	1114	107	62	1	0.007
dlx2_aa	(490, 2804)	1124	1020	970	3.12	0.94	32	0.023
addsub.boehm	(492, 1065)	1324	20256	347	35	29	$> 657\,000$	*time out*

Table 2: L&S vs. HYCAM on computing all MUS

or from a worst-case complexity point of view, only. We believe that the practical computational progresses as such as the ones obtained in this paper can prove valuable in handling these problems practically. In this respect, future research could concentrate on deriving specific algorithms for these AI issues, exploiting results like the ones described in this paper.

Acknowledgments

We thank Mark Liffiton for making his system available to us. This work has been supported in part by the *Région Nord/Pas-de-Calais*.

References

[Bailey and Stuckey, 2005] J. Bailey and P. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Proceedings of PADL'05*, pages 174–186, 2005.

[Bessant *et al.*, 2001] B. Bessant, É. Grégoire, P. Marquis, and L. Saïs. *Iterated Syntax-Based Revision in a Nonmonotonic Setting*, volume 22 of *Applied Logic*, pages 369–391. Frontiers in Belief Revision, 2001.

[Bruni, 2003] R. Bruni. Approximating minimal unsatisfiable subformulae by means of adaptive core search. *Discrete Applied Mathematics*, 130(2):85–100, 2003.

[Bruni, 2005] R. Bruni. On exact selection of minimally unsatisfiable subformulae. *Annals of Mathematics and Artificial Intelligence*, 43(1):35–50, 2005.

[Büning, 2000] H. Büning. On subclasses of minimal unsatisfiable formulas. *Discrete Applied Mathematics*, 107(1–3):83–98, 2000.

[Davydov *et al.*, 1998] G. Davydov, I. Davydova, and H.K. Büning. An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. *Annals of Mathematics and Artificial Intelligence*, 23(3–4):229–245, 1998.

[de la Banda *et al.*, 2003] M. de la Banda, P. Stuckey, and J. Wazny. Finding all minimal unsatisfiable subsets. In *Proceedings of PPDL'03*, pages 32–43, 2003.

[Eiter and Gottlob, 1992] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactual. *Artificial Intelligence*, 57:227–270, 1992.

[Eén and Sörensson, 2004] N. Eén and N. Sörensson. MiniSat home page: http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat.

[Fleischner *et al.*, 2002] H. Fleischner, O. Kullman, and S. Szeider. Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. *Theoretical Computer Science*, 289(1):503–516, 2002.

[Grégoire *et al.*, 2006a] É. Grégoire, B. Mazure, and C. Piette. Extracting MUSes. In *Proceedings of ECAI'06*, pages 387–391, 2006.

[Grégoire *et al.*, 2006b] É. Grégoire, B. Mazure, and C. Piette. Tracking MUSes and strict inconsistent covers. In *Proceedings of FMCAD'06*, 2006.

[Hamscher *et al.*, 1992] W. Hamscher, L. Console, and J. de Kleer, editors. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.

[Kautz and Selman, 2004] H. Kautz and B. Selman. Walksat home page: http://www.cs.washington.edu/homes/kautz/walksat.

[Liffiton and Sakallah, 2005] M. Liffiton and K. Sakallah. On finding all minimally unsatisfiable subformulas. In *Proceedings of SAT'05*, pages 173–186, 2005.

[Lynce and Marques-Silva, 2004] I. Lynce and J. Marques-Silva. On computing minimum unsatisfiable cores. In *Proceedings of SAT'04*, 2004.

[Oh *et al.*, 2004] Y. Oh, M. Mneimneh, Z. Andraus, K. Sakallah, and I. Markov. AMUSE: a minimally-unsatisfiable subformula extractor. In *Proceedings of DAC'04*, pages 518–523, 2004.

[Papadimitriou and Wolfe, 1988] C. Papadimitriou and D. Wolfe. The complexity of facets resolved. *Journal of Computer and System Sciences*, 37(1):2–13, 1988.

[SATLIB, 2000] Benchmarks on SAT. http://www.satlib.org.

[Zhang and Malik, 2003] L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formula. In *Proceedings of SAT'03*, 2003.

[Zhang, 2005] L. Zhang. On subsumption removal and on-the-fly cnf simplification. In *Proceedings of SAT'05*, pages 482–489, 2005.