Extracting MUSes

Éric GRÉGOIRE and **Bertrand MAZURE** and **Cédric PIETTE** [1]

Abstract. Minimally unsatisfiable subformulas (in short, MUSes) represent the smallest explanations for the inconsistency of SAT instances in terms of the number of involved clauses. Extracting MUSes can thus prove valuable because it circumscribes the sources of contradiction in an instance. In this paper, a new heuristic-based approach to approximate or compute MUSes is presented. It is shown that it often outperforms current competing ones.

1 INTRODUCTION

SAT is the NP-complete decision problem that consists in checking whether a set of Boolean clauses admits at least one truth assignment that satisfies all clauses. These last years, many researchers have focused on the more difficult task of extracting minimally unsatisfiable subformulas (in short, MUSes) of unsatisfiable SAT instances. Although this problem exhibits a high worst case complexity since e.g. checking whether a formula belongs to the set of MUSes of an inconsistent instance or not is in Σ_2^p [9], extracting MUSes can prove valuable because it circumscribes what is *wrong* with an instance. Indeed, many application domains like model-based diagnosis, knowledge-base validation or VLSI correctness checking, require such explanations to be delivered. When e.g. a knowledge-base is checked for consistency, we often prefer to know which clauses are actually contradicting one another, rather than just being told that the whole base is inconsistent.

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, renamable Horn, extended Horn, balanced and matched ones). Let us also mention [5, 7, 10] as they are other important studies in the complexity and the algorithmic aspects of extracting MUSes for specific classes of clauses. In [3], 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 [23] a way to extract MUSes by learning nogoods involved in the derivation of the empty clause by resolution. In [17], Lynce and Marques-Silva have proposed a complete and exhaustive technique to extract smallest MUSes. Oh and his co-authors have presented in [20] a Davis, Putnam, Logemann and Loveland DPLL-oriented approach that is based on a marked clause 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 [16].

In [19], a heuristic was also proposed to approximate MUSes, based on the experimental finding that clauses that are most often falsified during a failed local search often belong to MUSes. It has also been used to improve the performance of DPLL-like complete algorithms [6]. In this paper, a new variant and original extensions of this heuristic are studied. During the local search run, relevant parts of the neighborhood of the current truth assignment are explored in order to decide whether an unsatisfied clause during this local search should be actually counted or not. It is then extended in order to compute sets of MUSes. This new approach is shown to often outperform the current competing ones from an experimental point of view.

The paper is organized as follows. In the next section, the concept of MUS is presented formally. In section 3, a crucial notion of critical clause is introduced and analyzed. In section 4, the new approach to approximate or compute one MUS is presented. Extensive experimental results are given in section 5. Before we conclude, section 6 shows how the approach can be extended to compute sets of MUSes.

2 MINIMALLY UNSATISFIABLE SUBFORMULA (MUS)

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, where a literal is a Boolean variable that can be negated. SAT is the well-known 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.*
1. $\Gamma \subseteq \Sigma$
2. Γ *is unsatisfiable*
3. *Every proper subset of* Γ *is satisfiable*

Computing MUSes is a heavy computational task in the worst case. Indeed, checking whether a set of clauses is a MUS is DP-complete [21], and checking whether a formula belongs to the set of

[1] CRIL-CNRS & IRCICA,
Université d'Artois, rue Jean Souvraz SP18, F-62307 Lens Cedex France
E-mail: {gregoire,mazure,piette}@cril.univ-artois.fr

MUSes of an inconsistent instance or not, is in Σ_2^p [9]. Let us note that although the set of MUSes of an n-clauses instance is $C_n^{n/2}$ in the worst case, this number is often tractable in real-life situations. For example, in model-based diagnosis [13], based on experimental studies, it is often assumed that single faults occur most often, which is translated by a limited number of MUSes.

3 A NEW HEURISTIC TO DETECT MUSes

In [19], 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 [18, 19]. It has also been extended in several ways to address decision and optimisation problems that belong to higher levels of the polynomial hierarchy [11, 2, 12, 1].

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 property 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 $I = \{a, b, c\}$ be an interpretation. Under this interpretation, only the clause $\neg a \vee \neg b \vee \neg c$ is falsified. In the following, the *once-satisfied* clause concept will prove useful.

Definition 2 *A clause C is once-satisfied by an interpretation I iff only one literal of C is satisfied by 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 3 *A clause C falsified by the interpretation I is critical w.r.t. I iff 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 by 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 1 *Let C be a critical clause w.r.t. the interpretation I, then any flip from I to I' that is such that C is satisfied under I' will conduct I' to falsify at least one clause that was satisfied by 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. Indeed, this strategy exploits a property that is obeyed by clauses belonging to MUSes.

Property 2 *Let I be an interpretation giving an optimal result for max-SAT on an inconsistent instance Σ. Then, any clause C of Σ falsified by I belongs to at least one MUS of Σ and is critical w.r.t. I. Moreover, at least one clause linked to C that is once-falsified by I also belongs to a MUS of Σ.*

In this respect, a direct implementation of this technique would thus yield an approximation one in the sense that clauses and their linked ones are considered during the whole search, and not only at the best step of a max-SAT procedure. Such a technique can be easily grafted to a LS algorithm.

However, 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 consistent. $\neg e \vee \neg f$ is falsified by $I = \{a, b, d, e, f\}$ and is critical w.r.t. I. Moreover, a clause from a MUS that is falsified by 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 inconsistent set of clauses. $\neg a \vee \neg b$ is falsified by $I = \{a, b, d, e, f\}$. 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 3 *Let Γ be a MUS of Σ. For all clause $C \in \Gamma$, there exists an interpretation I s.t. C is critical w.r.t. I.*

This property ensures that any clause that takes part in 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 4 ensures that all falsified clauses are critical in local or global minima.

Definition 4 *A local minimum is an interpretation s.t. no flip can increase the number of satisfied clauses. A global minimum (or max-SAT solution) is an interpretation delivering the maximal number of satisfied clauses.*

Property 4 *In (local or global) minima, all falsified clauses are critical.*

A corollary even 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.*

4 APPROXIMATING AND COMPUTING ONE MUS

In the following, it is shown that a meta-heuristic based on scoring critical clauses can be the cornerstone of a novel complete method 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 UNSAT instance. While LS fails to find a model of Σ (each time within a preset amount of computing resources), Σ is recorded on a stack and clauses that exhibit the lowest scores are removed from Σ. Then, the inconsistency of the last version of Σ for which LS has failed to find a model is checked using a complete search algorithm. If it is indeed inconsistent, then it is an upper-approximation of a MUS of the initial instance. Else, this inconsistency test is repeated on the next version of Σ from the stack until unsatisfiability is proved. Roughly, this algorithm is described in the following AOMUS procedure.

```
Procedure AOMUS(Σ) // Approximate One MUS
begin
 stack := ∅ ;
 while (LS+Score(Σ) fails to find a model)
 do
   push(Σ) ;
   Σ:=Σ \ LowestScore(Σ) ;
 done
 repeat
   Σ:=pop() ;
 until (Σ is UNSAT)
end
```

Then an exact MUS can be obtained by a step-by-step minimization of the upper-approximation until the remaining clauses are proved to form a MUS (see [14] for an alternative method). This process is called fine-tune. The order of tested clauses can be guided by the score of each clause.

```
Procedure fine-tune(Σ)
begin
   foreach clauses c ∈ Σ  // sorted by scores
     If (Σ \ c is inconsistent) then Σ := Σ \ c ;
   done
end
```

The efficiency of this procedure 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 inconsistency tests are performed (when a clause belongs to the MUS, the test amounts to a consistency check).

Actually, we have refined this basic procedure in the following manner. Whenever a unique clause remains falsified during any of the LS runs, then we are sure that this clause belongs to all MUSes. We mark it as protected and it cannot be removed from Σ thereafter. When the remaining falsified clauses contain protected clauses only, they form one MUS: indeed, removing any one of these clauses will restore consistency. Moreover, when all clauses are protected and Σ is unsatisfiable, we are sure that Σ is a MUS and the fine-tune step can be omitted. It appears that this refinement proves valuable for many instances, and allows a dramatic gain in the efficiency of the procedure. The OMUS algorithm includes the AOMUS procedure together with the fine-tune one with this refinement.

The parameters for these methods that were selected are as follows. Wsat [15] 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 $(min\text{-}score + \frac{\#Flips}{\#Clauses})$ are dropped, where $min\text{-}score$ 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 UNSAT instances from several difficult benchmarks from DIMACS [8] and from the annual SAT competitions [22], and compared with other published approaches to compute MUSes, as described in the next section.

5 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 inconsistent sets of clauses are actually MUSes, we provide both the results of applying our algorithm with and without the fine-tune routine. Without the fine-tune routine, the approach delivers upper-approximations of MUSes, and is called AOMUS (Approximate One MUS). However, on many instances, these approximations are actual MUSes. Moreover, it appeared very often that the last subformula for which LS failed to find a model was in fact unsatisfiable. Thus, in practice the last loop of the AOMUS algorithm often reduces to a a unique inconsistency test. Let us stress that our approach is complete in the sense that it always delivers an approximate MUS for any UNSAT instance and a MUS when the fine-tune routine is run.

We compared our approach with an adaptation of AOMUS where *Scoring* is the basic heuristic of [19], which simply counts the number of times a clause is falsified. We also compared our approach with zCore, the core extractor of zChaff [23]. zChaff is currently one of the most efficient SAT solvers. We also ran Lynce and Marques-Silva's procedure [17], and took Bruni's [3] 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 inconsistency. Not surprisingly, our approach proved more efficient than the similar one where *Scoring* is based on the heuristic from [19]. 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

Table 1. Experimental results: Approximate One MUS (AOMUS) and find One MUS (OMUS)

Instance	#var	#cla	Lynce&Silva [17]		Bruni [4]	Zcore [23]		Scoring like [19]		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

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

an approximation of a MUS, whereas our OMUS and AOMUS procedures deliver one exact and one approximate MUS, respectively. Moreover, it should be emphasized that MUSes that are discovered by the various approaches are not necessary the same ones.

In Table 1, some typical experimental results are given. Except for Bruni's results which are just size results that we have extracted from [3], we provide both the experimental size of the discovered smallest inconsistent 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 [19] 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 inconsistent 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.

6 APPROXIMATING THE SET OF MUSes

Based on the OMUS procedure, we now address the problem of computing the set of MUSes of unsatisfiable instances, also called *clutter* by Bruni [4]. Since a MUS can be "broken" by removing one of its clauses, a naive approach consists in removing one clause of a MUS after this latter one has been discovered by the OMUS procedure, and then in iterating the process. Such an approach would deliver the right result when any pair of MUSes exhibits an empty intersection. However, MUSes can have non-empty intersections. Accordingly, when we remove a clause from a MUS, we actually break all MUSes containing it. To prevent this drawback from occurring as much as possible, we should prefer dropping clauses that belong to a minimal number of MUSes. Accordingly, we have investigated the following heuristic.

As max-SAT is intended to deliver a minimal number of unsatisfied clauses, the remaining unsatisfied clauses in a max-SAT solution must belong to intersections of MUSes as much as possible. Accordingly, for each clause, we record the minimum number of clauses that have been falsified at the same time during a failed LS. After a MUS is detected, the clause in the MUS with the lowest score is removed from the instance.

Clearly, such an approach (that we note ASMUS (Approximate Set of MUS)) is incomplete. However, it delivers very good results with respect to current existing approaches, as illustrated by our experimental investigations summarized in Table 2. For these experimentations, the time-out was set to 20000 s. AleatX_Y_Z instances are standard generated (unsatisfiable) random ones, with X variables and Y clauses. XAIMY_Z instances are the mere concatenations of X AIMα_β instances, where $\alpha = \frac{Y}{X}$ and $\beta = \frac{Z}{Y}$.

We have compared the ASMUS method with the complete algorithm proposed in [16] from an experimental point of view. Table 2 shows that both approaches appear to deliver the *exact* sets of MUSes on the simple "aim" benchmarks, using similar run-times. On more difficult instances like Aleat30_75_*, ASMUS almost extracts all MUSes and its computation time is in general better than the complete method one.

Moreover, Liffiton and Sakallah's algorithm can get into trouble for larger instances, since a CNF formula can exhibit an exponential number of MUSes and since their approach aims at computing all MUSes individually, only after having computed all maximally sat-

Table 2. Finding as many MUSes as possible.

Instance	#var	#cla	L.&S. [16]		ASMUS	
			#MUS	Time	#MUS	Time
aim100-1_6-no-1	100	160	1	0.18	1	0.31
aim200-1_6-no-1	200	320	1	0.14	1	0.68
aim200-1_6-no-2	200	320	2	0.22	2	0.76
aim200-2_0-no-3	200	400	1	0.12	1	0.56
aim200-2_0-no-4	200	400	2	0.26	2	0.88
Aleat20_70_1	20	70	127510	6.9	6	4.9
Aleat20_70_2	20	70	114948	10.8	13	8.7
Aleat30_75_1	30	75	11	59.82	7	2.2
Aleat30_75_2	30	75	9	26.84	8	2.9
Aleat30_75_3	30	75	10	12.84	10	3.7
Aleat50_218_1000	50	218	Time out		67	173
Aleat50_218_100	50	218	Time out		39	126
2AIM100_160	100	160	2	0.21	2	0.69
2AIM400_640	400	640	2	14.9	2	3.1
3AIM150_240	150	240	3	73.84	3	1.46
4AIM200_320	200	320	Time out		4	2.82
dp02u01	213	376	Time out		14	26.12
Homer06	180	830	Time out		2	17.47

isfiable subformulas, which can be intractable. On the opposite, our approximation technique does not suffer from such a drawback and exhibits an anytime property since MUSes are directly computed one after the other. For example, let us consider the `Aleat20_70_2` random instance. It exhibits 70 clauses and these constraints form more than 114 000 MUSes. Due to the very small size of this formula, [16] has been able to compute all MUSes. For larger instances involving many MUSes, like `dp02u01` (213 atoms, 376 clauses), the set of MUSes could not be computed within 20000 s., while our approach extracted 14 MUSes in 26 s.

7 CONCLUSION

In this paper, thanks to an original concept of critical clauses, a novel meta-heuristic-based approach to compute MUSes in SAT instances has been introduced. As our experimental results on difficult benchmarks illustrate it, the approach proves to be viable and often more competitive than previously published ones. The meta-heuristic is based on the intuitive idea that the most often falsified constraints during a failed local search are often the actual unsatisfiable ones. This idea has been refined to take the falsification propagation effect of these constraints. We believe that such a meta-heuristic could be applied to various difficult decision and optimisation problems. We plan to explore this in the near future.

REFERENCES

[1] F. Boussemart, F. Hémery, C. Lecoutre, and L. Saïs, 'Boosting systematic search by weighting constraints', in *European Conference on Artificial Intelligence (ECAI'04)*, pp. 146–150, (2004).

[2] L. Brisoux, É. Grégoire, and L. Saïs, 'Checking depth-limited consistency and inconsistency in knowledge-based systems', *International Journal of Intelligent Systems*, **16**(3), 333–360, (2001).

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

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

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

[6] M. Davis, G. Logemann, and D. Loveland, 'A machine program for theorem proving', *Journal of the Association for Computing Machinery*, **5**(7), 394–397, (1962).

[7] 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).

[8] DIMACS. Benchmarks on SAT. ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/.

[9] T. Eiter and G. Gottlob, 'On the complexity of propositional knowledge base revision, updates and counterfactual', *Artificial Intelligence*, **57**, 227–270, (1992).

[10] 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).

[11] É. Grégoire and D. Ansart, 'Overcoming the christmas tree syndrome', *International Journal on Artificial Intelligence Tools (IJAIT)*, **9**(2), 97–111, (2000).

[12] 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, pp. 51–60, (2002).

[13] *Readings in Model-Based Diagnosis*, eds., Console L. Hamscher W. and de Kleer J., Morgan Kaufmann Publishers Inc, 1992.

[14] J. Huang, 'Mup: A minimal unsatisfiability prover', in *Asia and South Pacific Design Automation Conference (ASP-DAC'05)*, pp. 432–437, (2005).

[15] 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)*, (2004).

[16] M.H. Liffiton and K.A. Sakallah, 'On finding all minimally unsatisfiable subformulas', in *International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pp. 173–186, (2005).

[17] I. Lynce and J. Marques-Silva, 'On computing minimum unsatisfiable cores', in *International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, (2004).

[18] 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)*, pp. 1265–1269, (1996).

[19] B. Mazure, L. Saïs, and É. Grégoire, 'Boosting complete techniques thanks to local search', *Annals of Mathematics and Artificial Intelligence*, **22**, 319–331, (1998).

[20] Y. Oh, M.N. Mneimneh, Z.S. Andraus, K.A. Sakallah, and I.L. Markov, 'Amuse: a minimally-unsatisfiable subformula extractor', in *Design Automation Confrence (DAC'04)*, pp. 518–523, (2004).

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

[22] SATLIB. Benchmarks on SAT. http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html.

[23] 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)*, (2003).

ANNEX: PROOFS

Proof of Property 1 *If C is critical w.r.t. I then for each literal l of C, $\exists\, C'$ s.t. C' is once-satisfied by I and $\bar{l}$ belongs to C'. C is falsified by I, thus l is false w.r.t. I and $\bar{l}$ is true w.r.t. I. $\bar{l}$ is the only literal of C' satisfied by I. Accordingly if the value of l is reversed then C' becomes falsified.* □

Proof of Property 2 *Any clause falsified by I belongs to a MUS of Σ because I is optimal w.r.t. the number of satisfied clauses and at least one clause of each MUS cannot be satisfied by I. The fact that any clause falsified by I is critical is proved thanks to property 4 since I is a global mimimum. 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 clause falsified by I belongs to a MUS of Σ.* □

Proof of Property 3 *Let Γ be a MUS of Σ and C be a clause of Γ. 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 were not falsified then Γ would exhibit a model M, which 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 unsatisfied since Γ is unsatisfiable. That means that this new unsatisfied clause was once-satisfied and linked to C. Accordingly, C is critical w.r.t. M.* □

Proof of Property 4 *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.* □