On Computing Minimum Unsatisfiable Cores

Inês Lynce and João Marques-Silva

IST/INESC-ID, Technical University of Lisbon, Portugal
{ines,jpms}@sat.inesc-id.pt

Abstract. Certifying a SAT solver for unsatisfiable instances is a computationally hard problem. Nevertheless, in the utilization of SAT in industrial settings, one often needs to be able to generate unsatisfiability proofs, either to guarantee the correctness of the SAT solver or as part of the utilization of SAT in some applications (e.g. in model checking). As part of the process of generating unsatisfiable proofs, one is also interested in unsatisfiable sub-formulas of the original formula, also known as unsatisfiable cores. Furthermore, it may by useful identifying the *minimum* unsatisfiable core of a given problem instance, i.e. the *smallest* number of clauses that make the instance unsatisfiable. This approach is be very useful in AI problems where identifying the minimum core is crucial for correcting the minimum amount of inconsistent information (e.g. in knowledge bases).

1 Introduction

The utilization of SAT in industrial settings has motivated work on certifying SAT solvers [14]. Given a problem instance, the certifier needs to be able to verify that the computed truth assignments indeed satisfy a satisfiable instance and that, for an unsatisfiable instance, a proof of unsatisfiability can be generated. Certifying a SAT solver for a satisfiable instance is easy; certifying a SAT solver for an unsatisfiable instance is hard. This paper concerns with the objective of certifying SAT solvers for unsatisfiable instances, and further for generating *minimum* proofs of unsatisfiability. Besides focusing on generating a proof of unsatisfiability for a target unsatisfiable formula, this paper addresses the problem of identifying a sub-formula that is also unsatisfiable (i.e. an *unsatisfiable core*), and also of computing the *smallest* sub-formula in the number of clauses that is also unsatisfiable (i.e. the *minimum unsatisfiable core*).

Besides the theoretical interest of computing unsatisfiable cores, or minimum unsatisfiable cores, the recent utilization of SAT technology in Unbounded Model Checking [11] also relies extensively on the ability of SAT solvers for generating proofs of unsatisfiability and for computing unsatisfiable cores. As a result, the utilization of SAT solvers in Model Checking requires their ability for efficiently generating proofs of unsatisfiability and also for computing *small* unsatisfiable cores.

Moreover, the identification of *inconsistent* kernels in propositional knowledge bases is a problem where the identification of a subset with the minimum number of clauses may be crucial. Observe that repairing inconsistent knowledge in real-world applications is a quite often problem to be addressed [10], although to the extent of our knowledge the identification of the *minimum* inconsistent kernel has never been addressed. Another application domain is interactive applications requiring explanations. For this domain it is crucial identifying precisely why a set of constraints is inconsistent and to correct it with the least number of modifications [8].

The paper is organized as follows. The next section surveys the work on computing unsatisfiable cores. This motivates the question of how to compute the minimum unsatisfiable core, and section 3 proposes a first model for solving this problem. Besides the model, the paper also presents preliminary experimental results.

2 Computing Unsatisfiable Cores

It is well-known that a CNF formula is unsatisfiable if it is possible to generate an empty clause by resolution from the original clauses. The set of original clauses involved in the derivation of the empty clause is referred to as the *unsatisfiable core*.

Definition 1 (Unsatisfiable Core). *Given a formula φ, UC is an unsatisfiable core for φ iff UC is a formula φ_c s.t. φ_c is unsatisfiable and $\varphi_c \subseteq \varphi$.*

Observe that an unsatisfiable core can be defined as any subset of the original formula that is unsatisfiable. Consequently, there may exist many different unsatisfiable cores, with different number of clauses, for the same problem instance, such that some of these cores are subsets of others. Also, and in the worst case, the unsatisfiable core corresponds exactly to the set of original clauses.

In the recent past, there have been different contributions to research on unsatisfiable cores. Research work can be distinguished between theoretical and experimental work. In the theoretical field, unsatisfiable cores complexity has been analyzed and formal algorithms have been proposed [3, 5, 6, 13]. Experimental work includes contributions of Bruni and Sassamo [2] and Zhang and Malik [14] [1]. Both approaches extract unsatisfiable cores. The first proposes an adaptative search guided by clauses hardness. The second approach is motivated by considering that a CNF formula is unsatisfiable if it is possible to generate an empty clause by resolution from the original clauses. This approaches basically extracts unsatisfiable cores based on the conflict analysis procedure [9]. The unsatisfiable core is given by the set of original clauses involved in the derivation of the empty clause. Recent work by Oh *et al.* [12] proposes extracting a minimal unsatisfiable core.

The first step for computing an unsatisfiable core consists in identifying the clauses (either original or recorded) that were involved in the steps that led to deriving the empty clause, and thus proving unsatisfiability. However, and since the unsatisfiable core must only include original clauses, it is necessary to develop a procedure for producing a trace from the recorded to the original clauses. This procedure is based on an iterative marking scheme for the clauses. Initially, only the clauses involved in deriving the empty clause are marked. At the end, the marked original clauses correspond to the unsatisfiable core.

The existing experimental work described above has very little concern regarding extraction of *minimum size* unsatisfiable cores, although in [14] the unsatisfiable core is reduced after being extracted (and it can be reduced to the minimal core in the best case). Also, recent work in [12] proposes a minimally unsatisfiable sub-formula extractor. However, in some practical applications it may be useful identifying the minimum unsatisfiable core of a given problem instance, i.e. the *smallest* number of clauses that make the instance unsatisfiable. We should note that in some cases the size of a minimal unsatisfiable core may be much larger than the size of a minimum unsatisfiable core.

3 Computing the Minimum Unsatisfiable Core

In this section we present the basic ideas of our model and algorithm to compute the minimum unsatisfiable core.

Definition 2 (Minimum Unsatisfiable Core). *Consider a formula φ and the set of all unsatisfiable cores for φ: $\{UC_1, ..., UC_j\}$. Then, $UC_k \in \{UC_1, ..., UC_j\}$ is a minimum unsatisfiable core for φ iff $\forall UC_i \in \{UC_1, ..., UC_j\}, 0 < i \leq j : |UC_i| \geq |UC_k|$.*

From the above definition, one may conclude that all unsatisfiable formulas have at least one minimum unsatisfiable core. Next, we illustrate our model and algorithm using the following example:

Example 1. Consider the CNF formula φ having the variables $X = \{x_1, x_2, x_3\}$ and the clauses $\Omega = \{\omega_1, ..., \omega_6\}$:

$$\begin{array}{lll} \omega_1 = x_1 \vee \neg x_3 & \omega_3 = \neg x_2 \vee x_3 & \omega_5 = x_2 \vee x_3 \\ \omega_2 = x_2 & \omega_4 = \neg x_2 \vee \neg x_3 & \omega_6 = \neg x_1 \vee x_2 \vee \neg x_3 \end{array}$$

From the formula φ above, we can identify nine different unsatisfiable cores UC_1, ..., UC_9:

[1] A similar approach has been proposed in [7]. Basically, in [14] the information required is recorded *during* the search, whereas in [7] it is computed *after* the search.

$UC_1 = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6\}$ $UC_4 = \{\omega_1, \omega_3, \omega_4, \omega_5, \omega_6\}$ $UC_7 = \{\omega_2, \omega_3, \omega_4, \omega_5\}$
$UC_2 = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5\}$ $UC_5 = \{\omega_2, \omega_3, \omega_4, \omega_5, \omega_6\}$ $UC_8 = \{\omega_2, \omega_3, \omega_4, \omega_6\}$
$UC_3 = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_6\}$ $UC_6 = \{\omega_1, \omega_2, \omega_3, \omega_4\}$ $UC_9 = \{\omega_2, \omega_3, \omega_4\}$

It is straightforward to conclude that $UC_9 = \{\omega_2, \omega_3, \omega_4\}$ is the minimum unsatisfiable core, i.e. the unsatisfiable core with the smallest cardinality. Observe that the algorithms referred in the previous section do not offer any guarantees of optimality regarding the size of the computed unsatisfiable core. Hence, any of the enumerated cores UC_1, ..., UC_9 could be correctly returned by any of the algorithms.

Definition 3 (Minimal Unsatisfiable Core). *An unsatisfiable core UC for φ is a minimal unsatisfiable core iff removing any clause $\omega \in UC$ from UC implies that $UC - \{\omega\}$ is not an unsatisfiable core.*

The work in [14] proposes an iterative solution for reducing the size of the computed unsatisfiable core, by iteratively invoking the SAT solver on each computed sub-formula. Also, the work in [12] proposes extracting a minimal unsatisfiable core. However, these solutions albeit capable of reducing the size of computed unsatisfiable cores, do not provide *any* guarantees regarding the size of the unsatisfiable core with the least number of clauses. Suppose that the first iteration of the algorithm returns $UC_1 = \{\omega_1, \omega_3, \omega_4, \omega_5, \omega_6\}$. Thus, in one of the following iterations this result can be improved to the *minimal* core $UC_4 = \{\omega_1, \omega_3, \omega_4, \omega_5, \omega_6\}$, but never to the *minimum* core $UC_9 = \{\omega_2, \omega_3, \omega_4\}$.

3.1 The Proposed Model

We assume that each formula φ is defined over n variables, $X = \{x_1, \ldots, x_n\}$, and that the formula has m clauses, $\Omega = \{\omega_1, \ldots, \omega_m\}$. We start by defining a set S of m new variables, $S = \{s_1, \ldots, s_m\}$, and create a new formula φ' defined on $n+m$ variables, $X \cup S$, and with m clauses $\Omega' = \{\omega'_1, ..., \omega'_m\}$. Each clause $\omega'_i \in \varphi'$ is defined from a corresponding clause $\omega_i \in \varphi$ and from a variable s_i as follows:

$$\omega'_i = \{\neg s_i\} \cup \omega_i$$

Example 2. Considering the CNF formula φ given in Example 1, the new formula φ' is defined on variables $X \cup S = \{x_1, x_2, x_3, s_1, ..., s_6\}$ and clauses $\Omega' = \{\omega'_1, ..., \omega'_6\}$, such that:

$\omega'_1 = \neg s_1 \vee x_1 \vee \neg x_3$ $\omega'_3 = \neg s_3 \vee \neg x_2 \vee x_3$ $\omega'_5 = \neg s_5 \vee x_2 \vee x_3$
$\omega'_2 = \neg s_2 \vee x_2$ $\omega'_4 = \neg s_4 \vee \neg x_2 \vee \neg x_3$ $\omega'_6 = \neg s_6 \vee \neg x_1 \vee x_2 \vee \neg x_3$

Observe that variables S can be interpreted as *clause selectors* which allow considering or not each clause ω_i. Clearly, φ' is readily satisfiable by setting all s_i variables to 0. Now, for each assignment to the S variables, the resulting sub-formula may be satisfiable or unsatisfiable. For each unsatisfiable sub-formula, the number of S variables assigned value 1 indicates how many clauses are contained in the unsatisfiable core [2] (since the other clauses are satisfied by the S variables assigned value 0). The *minimum unsatisfiable core* is obtained from the unsatisfiable sub-formula with the *least* number of S variables assigned value 1.

One can adapt a state-of-the-art SAT solver to implement the proposed model. The problem instance variables are organized into two disjoint sets: the S variables and the X variables. Decisions are first made on the S variables (defining the S space) and afterwards on the X variables (defining the X space); hence, each assignment to the S variables defines a potential core. If for a given assignment all clauses become satisfied, then the search simply backtracks to the most recently untoggled S variable. Otherwise, each time the search backtracks from a decision level associated with an X variable to a decision level associated with a S variable, we have identified an unsatisfiable core, defined by the S variables assigned value 1. After all assignments to the S variables have been (implicitly) evaluated, the unsatisfiable core with the least number of utilized clauses corresponds to the minimum unsatisfiable core.

[2] Observe that this unsatisfiable core may be reduced if we restrict the core to clauses involved in the derivation of the empty clause.

3.2 Optimizations to the Model

The key challenge of the proposed model is the search space. For the original problem instance the search space is 2^n, where n is the number of variables, whereas for the transformed problem instance the search space becomes 2^{n+m}, where m is the number of clauses. Nevertheless, a few key optimizations can be applied, namely by introducing a *cardinality constraint* and an *additional clause recording* scheme.

First, the SAT-based algorithm can start with an upper bound on the size of the minimum unsatisfiable core. For this purpose, the algorithm described in [14] can be used. Hence, when searching for the minimum unsatisfiable core, we just need to consider assignments to the S variables which yield smaller unsatisfiable cores. This additional constraint can be modeled as a cardinality constraint. Moreover, each computed unsatisfiable core can be used for backtracking *non-chronologically* on the S variables, thus further potentially reducing the search space. Observe that an unsatisfiable core is computed whenever the search backtracks from the X space to the S space, meaning that there is no solution to the formula given the current S assignments.

In addition, clause recording is used to reduce the search space. Interestingly, after a conflict that implies recording a clause that allows backtracking from the X space to the S space, an unsatisfiable core can be easily obtained from the new recorded clause. For example, given formula φ' from Example 2, recording clause $\omega'_7 = s_2 \vee s_3 \vee s_4$ means that core $UC_9 = \{\omega_2, \omega_3, \omega_4\}$ has been identified.

Besides the *traditional* clause recording scheme [9], where each new clause corresponds to a sequence of resolution steps, a new clause is recorded whenever a solution is found. The new clause contains all the S literals responsible for not selecting the corresponding clause [3], *except* for those clauses that would be satisfied by the X variables in the computed solution.

Example 3. Consider again formula φ' from Example 2, and suppose that the current set of assignments is $\{s_1{=}0, s_2{=}0, s_3{=}1, s_4{=}1, s_5{=}0, s_6{=}1, x_1{=}1, x_2{=}0, x_3{=}0\}$. At this stage of the search, all clauses are satisfied, and therefore a solution is found. Consequently, a new clause is recorded to avoid finding again the same solution and also to force finding an unsatisfiable core in the future. For this example, a new clause $\omega'_8 = s_2 \vee s_5$ is recorded. Observe that clause ω'_1 is satisfied by assigning $x_1{=}1$. The new clause means that for finding an unsatisfiable core either clause ω_2 or clause ω_5 has to be part of the formula.

As a final remark, observe that by recording a clause whenever a solution is found allows the search to prove unsatisfiability, thus terminating.

Example 4. Again for formula φ', finding all possible solutions will add to the formula (at least) the following clauses:

$$\omega'_9 = s_3 \qquad \omega'_{11} = s_1 \vee s_2 \qquad \omega'_{13} = s_2 \vee s_6$$
$$\omega'_{10} = s_4 \qquad \omega'_{12} = s_2 \vee s_5$$

It is straightforward to conclude that, by resolution between these clauses and the clauses from the original problem specification, the empty clause is derived.

4 Experimental Results

In what follows experimental results are given and evaluated. We start by giving results for instances where the main goal is finding the minimum unsatisfiable core. Afterwards, we give results for instances where the minimum unsatisfiable core is not proved to be found (in the given CPU time) and therefore we consider the smallest unsatisfiable core found so far.

For all the results, two main steps were followed:

1. First, an unsatisfiable core is obtained using zChaff most recent version [4]. This version (zChaff 2003.12.04) can produce an unsatisfiable core from an unsatisfiable formula [14]. Moreover, the unsatisfiable core is reduced iteratively: the solver runs until a fixed point on the size of the unsatisfiable core is reached.

[3] Such literals are assigned value 1 in a clause that is part of the original specification.
[4] Available from `http://ee.princeton.edu/~chaff/zchaff.php`.

Table 1. Results on aim instances.

Instance	#Vars	#Cls	zChaff	Minimum	Time Minimum (s)
aim-60-2_0-no-1	60	120	76	72	259
aim-60-2_0-no-2	60	120	79	78	510
aim-60-2_0-no-3	60	120	71	70	68
aim-60-2_0-no-4	60	120	64	(48)	>3600
aim-75-2_0-no-1	75	120	96	86	1735
aim-75-2_0-no-2	75	120	72	72	1153
aim-75-2_0-no-3	75	120	62	61	2937
aim-75-2_0-no-4	75	120	89	(89)	>3600

Table 2. Results on uuf50 instances.

Instance	#Vars	#Cls	zChaff	Smallest Value
uuf50-021	50	218	141	132
uuf50-032	50	218	117	107
uuf50-041	50	218	111	101
uuf50-0110	50	218	129	122
uuf50-0112	50	218	104	100
uuf50-0119	50	218	141	123
uuf50-0206	50	218	107	101
uuf50-0207	50	218	127	124
uuf50-0474	50	218	142	127

2. Afterwards, a tool for extracting the *minimum* unsatisfiable core is used. In this tool, the size of the unsatisfiable core computed in 1. is used as a cardinality constraint when finding the minimum unsatisfiable core [5].

The algorithm for identifying the minimum unsatisfiable core was implemented in the CQuest SAT solver. CQuest is implemented in C++ and includes most of the most competitive techniques for industrial benchmarks. For all experimental results a P-IV@1.7 GHz Linux machine with 1 GByte of physical memory was used. The limit CPU time was 3600 seconds.

Table 1 gives results for selected *aim* instances [6]. For each instance, given values refer to the number of variables and clauses, the size of the unsatisfiable core given by zChaff and the size of the *minimum* unsatisfiable core given by our tool. In addition, the table shows the time spent on finding the minimum unsatisfiable core. (The time spent on finding the minimal unsatisfiable core is negligible.) For the instances where the search does not terminate in the allowed CPU time, the size of the minimum core is between parenthesis. For these instances, this value represents the size of the smallest unsatisfiable core found so far, which means that it is not guaranteed to correspond to the size of the minimum unsatisfiable core. Results in Table 1 clearly indicate that the size of the unsatisfiable core provided by zChaff may often be reduced.

Given the complexity of the algorithm for finding the minimum unsatisfiable core, interesting results may be obtained even when the search does not terminate in the allowed CPU time. In such cases, the smallest value found by the time the search finishes may already represent a significant reduction to the size of the unsatisfiable core given by zChaff. Table 2 gives results for selected *uuf50* instances [4]. These results are representative of the results obtained for the whole benchmark family. It is clear that the initial core may indeed be reduced, even though the best value found in 3600s is not proved to correspond to the size of the minimum unsatisfiable core.

Due to the complexity of the problem, the results obtained so far are limited to instances with a small number of variables and clauses. Nevertheless, we believe that further optimizations can be implemented in the future, both due to new advances in SAT technology and to new improvements specific to this optimization problem. However, it has become clear from these preliminary results that the size of an unsatisfiable core computed by zChaff can be somewhat far from the size of

[5] This value is used as an upper-bound for the number of S variables to be assigned value 1. During the search, the bound decreases whenever a new smaller unsatisfiable core is found.

[6] We used the *aim* generator for obtaining these instances [1].

the *minimum* unsatisfiable core. Consequently, it is indeed useful to use this tool with the goal of reducing the minimal unsatisfiable core given by zChaff.

5 Conclusions

This paper overviews algorithms for computing unsatisfiable cores, and proposes a model for computing the *minimum* unsatisfiable core. The proposed model represents a complex optimization problem, and a SAT-based algorithm has been proposed. Experimental results analyze the actual practical performance of the algorithm, and clearly indicate that the new approach can successfully complement the existing tools for identifying unsatisfiable cores. Given the complexity of the new algorithm, future research work entails improving and extending experimental evaluation of the proposed model and algorithm.

Acknowledgments

This work is partially supported by the European research project IST-2001-34607 and by Fundação para a Ciência e Tecnologia under research projects POSI/CHS/34504/2000 and POSI/SRI/41926/2001.

References

1. Y. Asahiro, K. Iwama, and E. Miyano. Random generation of test instances with controlled attributes. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge*, volume 26, pages 377–394. American Mathematical Society, 1993.
2. R. Bruni and A. Sassano. Restoring satisfiability or maintaining unsatisfiability by finding small unsatisfiable subformulae. In *LICS Workshop on Theory and Applications of Satisfiability Testing*, June 2001.
3. H. K. Büning. On subclasses of minimal unsatisfiable formulas. *Discrete Applied Mathematics*, 107(1-3):83–98, 2000.
4. P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 331–337, 1991.
5. 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.
6. H. Fleischner, O. Kullmann, and S. Szeider. Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. *Theoretical Computer Science*, 289(1):503–516, 2002.
7. E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for cnf formulas. In *Proceedings of the Design and Test in Europe Conference*, pages 10886–10891, March 2003.
8. U. Junker. QUICKXPLAIN: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI Workshop on Modeling and Solving Problems with Constraints*, August 2001.
9. J. P. Marques-Silva and K. A. Sakallah. GRASP: A new search algorithm for satisfiability. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design*, pages 220–227, November 1996.
10. B. Mazure, L. Sais, and E. Grégoire. Boosting complete techniques thanks to local search methods. *Annals of Mathmatics and Artificial Intelligence*, 22(3-4):319–331, 1998.
11. K. L. McMillan. Interpolation and SAT-based model checking. In *Proceedings of Computer Aided Verification*, 2003.
12. Y. Oh, M. Mneimneh, Z. S. Andraus, K. A. Sakallah, and I. L. Markov. Amuse: A minimally-unsatisfiable subformula extractor. In *Proceedings of the Design Automation Conference*, June 2004. Accepted for publication.
13. C. H. Papadimitriou and D. Wolfe. The complexity of facets resolved. *Journal of Computer and System Sciences*, 37(1):2–13, 1988.
14. L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of the Design and Test in Europe Conference*, pages 10880–10885, March 2003.