

## A new local search heuristic to compute inconsistent kernels

Éric Grégoire    Bertrand Mazure    Cédric Piette

CRIL-CNRS & IRCICA

Université d'Artois, rue Jean Souvraz SP18 F-62307 Lens Cedex France

{piette,mazure,gregoire}@cril.univ-artois.fr

### 1 Abstract

Whereas computing a SAT instance consists in checking whether the instance is consistent or not, computing kernels of an inconsistent SAT instance consists in finding smallest inconsistent subsets of clauses in this instance. In the paper, a new variant of a local-search heuristic that approximates or computes kernels is introduced and investigated. The initial heuristic is based on the finding that clauses that are most often falsified during a failed local search often belong to kernels. A variant of this heuristic takes relevant parts of the neighbourhood of the interpretation into account in order to decide whether an unsatisfied clauses should be actually counted or not.

### 2 Introduction

SAT is a well-studied decision problem that consists in checking whether a set of Boolean clauses admits at least one truth assignment that satisfies all clauses. There is a large and very active research community involved with both the theoretical and experimental algorithmic studies of SAT (see e.g. [www.SATlive.org](http://www.SATlive.org)).

In this paper, we address an extension of the SAT problem that consists in finding or approximating smallest subsets of inconsistent clauses, also called kernels, for inconsistent SAT instances. Clearly, these subsets of clauses express the smallest explanations for inconsistency in terms of the number of involved clauses. In fact, many application domains like model-based diagnosis, knowledge-bases validation or VLSI correctness checking, require such explanations to be delivered. Indeed, when e.g. a knowledge-base is checked for consistency, we often prefer to know *what* is actually wrong with the knowledge-base, namely which smallest subsets of clauses are actually contradicting one another, rather than just being told that the knowledge-base is inconsistent. In this paper, a new local search heuristic addressing this problem is introduced and investigated.

In previous works, some of us have proposed to detect kernels using the trace of local search, relying on a heuristic asserting that clauses that are most often falsified during a failed local search often belong to kernels [14]. This heuristic has also been used to improve the

**Vienna, Austria, August 22–26, 2005**

performance of the Davis, Logemann and Loveland DPLL-like complete algorithms [6,14]. In this paper, a new variant of this heuristic is studied. It explores relevant parts of the neighbourhood of the current truth assignment in order to decide whether an unsatisfied clauses should be actually counted or not.

The paper is organised as follows. In the next section, the concept of kernel is presented formally. In section 3, the heuristic by [14] is briefly recalled. In section 4, a crucial notion of critical clause is introduced and analysed. In section 5, it is shown how this can lead to a new approach to approximate or compute one kernel. Extensive experimental results are given in section 6. Finally, some perspectives are drawn in the conclusion.

### 3 Kernels

Let  $\mathbf{L}$  be a standard Boolean logical language built on a finite set of Boolean variables, noted  $a$ ,  $b$ , etc. Formulas will be noted using upper-case letters such as  $C$ . Sets of formulas will be represented using Greek letters like  $\Gamma$  or  $\Sigma$ . 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 noted by upper-case letters like  $I$  and will be represented by the set of literals that it satisfies. Actually, any formula in  $\mathbf{L}$  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 Boolean variable that is possibly 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, then it exhibits at least one minimal (with respect to set-theoretic inclusion) inconsistent subset of clauses. Such a subset of clauses is called a *kernel*.

**Definition 1.** A *kernel*  $\Gamma$  of a SAT instance  $\Sigma$  is a set of clauses s.t.

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

An inconsistent SAT instance might contain several kernels. In several domains, it is sometimes assumed that at most one kernel exists. For example, in model-based diagnosis, such an assumption meets the intuitive idea that single faults occur most often, a fault being described by a kernel. The uniqueness of a kernel also entails less heavy computational techniques to locate it.

It is important to avoid any confusion between the clauses that are not satisfied by MAX-SAT in case of an inconsistent instance, and the kernels of this instance.

**Property 1.** Let  $\Gamma'$  be the set of unsatisfied clauses delivered by a MAX-SAT procedure on an inconsistent SAT instance  $\Sigma$ . Then, for each kernel  $\Gamma$  of  $\Sigma$ , there exists at least one clause in  $\Gamma$  that also belongs to  $\Gamma'$ .

Thus, each clause that is not satisfied by MAX-SAT takes part to at least one kernel. However, to get a complete understanding of what is actually contradictory in an instance, we need to find kernels.

#### 4 Failed local search to detect kernels

Computing kernels is a heavy task in the general case: indeed, checking whether a formula belongs to the set of kernels of an inconsistent instance or not, is in  $\Sigma_2^p$  (a consequence of theorem 8.2 of [8]). There exist several techniques to compute and approximate kernels. A first-one consists in finding the smallest resolution-trees proving inconsistency within the instance. In [14], it is shown how local search can be helpful for locating kernels. The basic idea is that clauses that are often falsified during a failed local search for satisfiability belong most probably to kernels, 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.

Such an heuristic has been studied in an extensive manner in [13,14]. It has also been extended in several ways to address NP-“harder” decision and optimisation problems [1,2,3,9,10].

#### 5 Taking the neighbourhood of the current interpretation into account

In the following discussion, we assume that the SAT instance is unsatisfiable. The heuristic by [14] increments the score of a clause, each time it is falsified by the current truth assignment during a local search process. In fact, this can require us to increment the score of clauses even when they do not actually belong to any kernel. Unless we solve the problem of finding kernels 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 kernel.

In this paper, we explore some relevant parts of the neighbourhood of the current interpretation to check 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 kernels.

To illustrate this concept, let us use the following example. Let  $\Delta = \{a \vee b \vee c, \neg a \vee b, \neg b \vee c, \neg c \vee a, \neg a \vee \neg b \vee \neg c\}$ .  $\Delta$  is unsatisfiable and is its own kernel. Let  $I = \{a, b, c\}$  an interpretation. Under this interpretation, only the clause  $\neg a \vee \neg b \vee \neg c$  is unsatisfied.

In the following, the *once-satisfied* clause concept will prove useful.

**Definition 3.** A clause  $C$  is *once-satisfied* by an interpretation  $I$  iff exactly 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 4.** A clause  $C$  falsified by the interpretation  $I$  is *critical* w.r.t.  $I$  iff the opposite of every literal of  $C$  belongs to another clause that is once-satisfied by  $I$ . These once-satisfied clauses that are not tautological ones are called *linked* to  $C$ .

In the example, the unsatisfied clause  $\neg a \vee \neg b \vee \neg c$  by  $I$  is critical w.r.t.  $I$ , and 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. 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 formula that was satisfied under  $I$ .

Now, in order to discriminate clauses belonging to kernels, the idea is to increment the scores of critical clauses during the search, together with their linked (satisfied) clauses, rather than incrementing the scores of all falsified clauses.

Such a technique can be easily grafted to a local search algorithm and the updates can be easily computed. Actually, it implements definitions that approximate a property of clauses belonging to kernels.

**Property 3.** Let  $I$  be an interpretation giving an optimal result for MAX-SAT on an inconsistent instance  $\Sigma$ . Then, any falsified clause  $C$  w.r.t.  $I$  belongs to at least one kernel  $\Gamma$  of  $\Sigma$  and is critical w.r.t.  $I$ . Moreover, at least one once-satisfied clause linked to  $C$  also belongs to a kernel of  $\Sigma$ .

Our technique is thus an approximation one in the sense that clauses and their linked ones are considered during the whole search, and not at the best step of a MAX-SAT procedure. Indeed, being a critical clause is neither a necessary nor a sufficient condition to belong to a kernel. As the following example illustrates it, 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 kernel. Let  $\Delta = \{a \vee d, \neg a \vee \neg b, \neg d \vee e, f, \neg e \vee \neg f\}$ . Clearly,  $\Delta$  is consistent.  $\neg e \vee \neg f$  is falsified under  $I = \{a, b, d, e, f\}$  and is critical w.r.t.  $I$ . Moreover, a clause from a kernel that is falsified under a given interpretation  $I$  is not necessary critical w.r.t.  $I$ , as the following example illustrates it. Let  $\Delta = \{a \vee d, b, \neg a \vee \neg b, \neg d \vee e, f, \neg e \vee \neg f\}$ . Clearly,  $\Delta$  is a minimal inconsistent 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.  $I$ , although at least one clause per kernel is critical.

## 6 Approximating and computing one kernel

In the following, we show that a meta-heuristic based on scoring critical clauses is viable to approximate or compute kernels.

**Vienna, Austria, August 22–26, 2005**

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 local search algorithm and tested benchmarks.

The main idea is as follows. Let  $\Sigma$  be the UNSAT instance. While  $\Sigma$  is inconsistent, the following operations are performed. (i) A local search algorithm (*LS*) is run and scores of critical clauses are computed (*Scoring*). (ii) Clauses exhibiting the lowest scores are dropped from  $\Sigma$ . Finally, the previous state of  $\Sigma$  before it is shown consistent is thus an upper-approximation of a kernel. Then a *fine-tune* procedure involves a step-by-step minimisation of  $\Sigma'$ , until the remaining clauses are proved to form a kernel.

**Procedure** *Find-one-kernel*( $\Sigma$ )

**begin**

**while**  $\Sigma$  is inconsistent **do**

$\Sigma' := \Sigma$  ;

$LS+Scoring(\Sigma)$  ;

$\Sigma := \Sigma \setminus Lowest\ scores(\Sigma)$

**done**

*Fine-tune*( $\Sigma'$ )

**end**

Table 1. Typical experimental results

<i>Instance</i>	<i>#vars</i>	<i>#clauses</i>	<i>zCore</i>	<i>Lynce-Ma</i> <i>rques-Silv</i> <i>a</i>	<i>Bruni</i>	<i>Find-one-kernel</i> without <i>Fine-tune</i> and with [14] <i>scoring</i>	<i>Find-one-kern</i> <i>el</i> without <i>Fine-tune</i>	<i>Find-one-ker</i> <i>nel</i> with <i>Fine-tune</i>
<i>aim-50-1_6-no-1</i>	50	80	22 (1,11)	22 (1,61)	22	22 (1,65)	22 (2,13)	22 (7,42)
<i>aim-50-2_0-no-4</i>	50	100	21 (1,29)	21 (3,49)	21	21 (2,97)	21 (2,85)	21 (9,31)
<i>aim-100-1_6-no-1</i>	100	160	47 (1,45)	47 (284)	47	47 (2,62)	47 (2,67)	47 (15)
<i>aim-100-1_6-no-2</i>	100	160	54 (1,12)	53 (224)	54	53 (2,37)	53 (2,82)	53 (28)
<i>aim-100-2_0-no-1</i>	100	200	19 (1,55)	time out	19	19 (1,91)	19 (3,01)	19 (9,46)
<i>aim-100-2_0-no-4</i>	100	200	31 (0,96)	time out	32	31 (2,28)	31 (3,25)	31 (12)
<i>aim-200-1_6-no-2</i>	200	320	81 (1,52)	time out	82	80 (1,79)	80 (2,94)	80 (37)
<i>aim-200-2_0-no-1</i>	200	400	53 (1,72)	time out	54	53 (2,29)	53 (2,96)	53 (16)
<i>jnh14</i>	100	850	91 (1,85)	time out	124	111 (45)	90 (89)	78 (818)
<i>jnh15</i>	100	850	119 (3,84)	time out	140	170 (42)	126 (78)	102 (1576)
<i>Jnh8</i>	100	850	90 (2,28)	time out	91	118 (65)	76 (102)	68 (801)
<i>Jnh9</i>	100	850	93 (2,23)	time out	118	123 (49)	102 (85)	83 (571)
<i>fpga10_11_uns_rcr</i>	220	1122	561 (27)	time out	-	565 (15)	561 (26)	561 (11538)

The parameters that were selected are as follows. WSAT [11] with the Rnovelty+ option was chosen as the LS procedure. The other parameters were fine-tuned based on extensive tests on various benchmarks. After each flip of the LS, the score of critical clauses is augmented 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  $min\text{-score} + (\#Flips / \#Clauses)$  are dropped, where  $min\text{-score}$  is the lowest score for a clause of  $\Sigma$ ;  $\#Flips$  and  $\#Clauses$  are the number of performed flips and the number of clauses in  $\Sigma$ , respectively.

This procedure was tested extensively on various UNSAT instances from several difficult benchmarks from DIMACS [7] and from the annual SAT competitions [16], and compared with other published approaches to compute kernels, as described in the next section.

## 7 Experimental results

All experiments have been conducted on Pentium IV, 4Ghz under linux Fedora Core 2. For efficiency reasons, we replaced the loop based on the inconsistency test by a loop where iterations are performed while the LS fails to prove consistency. As our results show it, this approximation proves to deliver a correct result most of the times. Moreover, the *Fine-tune* procedure ensures that a kernel is actually obtained. As most current approaches do not guarantee that the delivered inconsistent sets of clauses are actually kernels, we provide both the results of applying the *Find-one-kernel* with and without the *Fine-tune* routine. Without the *Fine-tune* routine, the approach delivers upper-approximations of kernels. However, on many instances, these approximations appeared to be actual kernels. We compared our approach with an adaptation of *Find-one-kernel* where *Scoring* is the basic heuristic of [14], which simply counts the number of times a clause is falsified. We also compared our approach with Zcore, the core extractor of zChaff [17]. zChaff is currently one of the most efficient SAT solvers. We also ran Lynce and Marques-Silva's procedure [12], and took Bruni's [4] experimental results into account. For Bruni's technique, we only mention the experimental results obtained by the authors, 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-kernel instances but failed to deliver good results when several kernels are present. Indeed, zCore does not concentrate on finding *one* kernel, but on finding proofs of inconsistency. Not surprisingly, our approach proved more efficient than the similar one where *Scoring* is based on [14] heuristic. Most often, it proved to be more competitive than all the other considered techniques when very large and difficult multi-kernels instances were considered. Noticeably, it was also the only technique to perform in a competitive way on all benchmarks.

**Vienna, Austria, August 22–26, 2005**

In Table 1, some typical experimental results are given. Except for Bruni's results which are just size results that we have extracted from [4], we provide both the experimental size of the discovered smallest inconsistent subsets, together with the average CPU time in seconds to get them. The approach based on the *Find-one-kernel* procedure without the *Fine-tune* routine was run 100 times, and average computing times are given. *Time-out* indicates that no result has been obtained within 1 hour CPU time. For example, for the *homer15.shuffled.cnf* instance, only our approach delivered a result (an inconsistent set of 561 clauses was obtained in 1104 secs.). Also, it can be seen e.g. on the *fpga* benchmarks that *Find-one-kernel* approach without the *Fine-tune* procedure delivered smaller inconsistent subsets than any other considered method, most often. Let us also emphasise that even on small instances like the *aim* ones, the *Find-one-kernel* method proved very competitive, as well.

## 8 Conclusions and perspectives

In this paper, a novel meta-heuristic-based approach to compute kernels 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 next future.

## Acknowledgements

This study has been supported by the EC under a FEDER grant and the *IUT de Lens*.

## References

- [1] Ansart D., "Utilisation et extensions de l'algorithmique pour SAT pour la résolution de différents problèmes en intelligence artificielle", Thèse d'Université, 2005. (*in French*)
- [2] Boussemart F., Hemery F., Lecoutre C. and Saïs L., "Boosting systematic search by weighting constraints", *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pp. 146-150, Valencia, Spain, August 2004.
- [3] Brisoux L., Grégoire É., Saïs L., "Checking depth-limited consistency and inconsistency in knowledge-based systems", *Int. Journ. of Intelligent Systems*, vol. 16(3), pp.333-360,2001.
- [4] Bruni R., "Approximating minimal unsatisfiable subformulae by means of adaptive core search", *Discrete Applied Mathematics*, 130(2), pp. 85-100, 2003.
- [5] Bruni R., "On exact selection of minimally unsatisfiable subformulae", *Annals of*

Vienna, Austria, August 22–26, 2005

- Mathematics and Artificial Intelligence*, 43(1-4), pp. 35-50, 2005.
- [6] Davis P., Logemann G., Loveland D., "A machine program for theorem proving", *Journal of the association for computing machinery*, 5:394-397, 1962
- [7] Dimacs benchmarks on SAT <ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/>
- [8] Eiter T. and Gottlob G., "On the complexity of propositional knowledge base revision, updates and counterfactual", *Artificial Intelligence*, 57 pp. 227-270, 1992.
- [9] Grégoire É. and Ansart D., "Overcoming the Christmas Tree Syndrome", *Int. Journ. on Artificial Intelligence Tools (IJAIT)*, vol. 9, n°2, pp.97-111, 2000.
- [10] Grégoire É., Mazure B., Saïs L., "Using failed local search for SAT as an oracle for tackling harder A.I. problems more efficiently", *Proc. of the Tenth International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'2002)*, LNCS 2443, Springer VeLSag, pp 51-60, Varna, Bulgaria, 2002.
- [11] Kautz H., Selman B., McAllester D., "Walksat in the SAT04 Competition", *International Conference on Theory and Applications of Satisfiability Testing*, Vancouver, 2004.
- [12] Lynce I. and Marques-Silva J., "On computing minimum unsatisfiable cores", *International Conference on Theory and Applications of Satisfiability Testing*, Vancouver, 2004.
- [13] Mazure B., Saïs L., Grégoire É., "A Powerful Heuristic to Locate Inconsistent Kernels in Knowledge-Based Systems", *Proc. Int. Conf. on Inform. Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-96)*, pp 1265-1269, Grenada, 1996.
- [14] Mazure B., Saïs L., Grégoire É., "Boosting complete techniques thanks to local search", *Annals of Mathematics and Artificial Intelligence*, vol. 22, pp. 319-322, 1998.
- [15] Oh Y., Mneimneh M.N., Andraus Z.S., Sakallah K.A., Markov I.L., "AMUSE: a minimally-unsatisfiable subformula extractor", *Proc. of the 41th Design Automation Conference (DAC 2004)*, pp. 518-523, 2004.
- [16] SATLIB : <http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html>
- [17] Zhang L. and Malik S., "Extracting small unsatisfiable cores from unsatisfiable boolean formula", presented at *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, S. Margherita Ligure – Portofino, Italy, 2003